UNIVERSITY OF
CAMBRIDGE

Resilient routing for MANETs

Marco Caballero Gutierrez

Churchill College

This dissertation is submitted on February, 2021 for the degree of Doctor of Philosophy

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or am concurrently submitting, for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or is being concurrently submitted, for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. This dissertation does not exceed the prescribed limit of 60 000 words.

Marco Caballero Gutierrez
February, 2021

Abstract

Resilient routing for MANETs

Marco Caballero Gutierrez

Mobile ad hoc networks (MANETs) are the core technology that provides the US military with adaptable and reliable battlefield communications. These self-organising networks are ideal for rapidly changing scenarios that require connectivity even under hostile conditions. The excitement and promise that these networks generated transferred to the civilian space as well, fueling over a decade of research. However, they only experienced limited success in this new setting, leading to a fragmentation into several application-oriented sub-fields that dealt with narrower sets of constraints.

In this dissertation, I postulate that the unique properties of these networks makes them much more error-prone than initially considered. Consequently, a focus on improving their capabilities rather than on mitigating their faulty nature made the design of general-purpose MANETs increasingly challenging. I support this argument through an extensive set of experiments that is informed by analysis of the literature, history, and properties of these networks.

Ultimately, my work contributes to the field in three fronts: Firstly, motivated by the multiple challenges that research in this area presents, I designed and built MeshSim, a real-time network simulator. This new platform focuses on code-fidelity and enables me to follow a data-driven experimental cycle. Secondly, I present the Reactive Gossip Routing family of protocols, designed to provide reliable and scalable routing by mitigating the MANET properties that lead to faults. Using MeshSim, I evaluate these protocols experimentally under increasingly harsher conditions and compare their effectivity against the incumbents in the literature. Finally, I demonstrate through experimentation that distance-vector routes are ill suited for MANETs due to a geographical-spreading effect they induce, a result that extends to many routing metrics when used in shortest-path algorithms.

Acknowledgements

First and foremost I would like to thank my wife, Laura Caballero, to whom I dedicate this work. Without her love and support none of this would have been possible. Secondly, I would like to thank my supervisor Jon Crowcroft for believing in me, letting me run wild with my ideas, and having my back. Likewise, I am grateful to my second supervisor Richard Mortier, for the many discussions, pub outings, and support.

During my PhD, I was helped by many people in various ways. Heidi Howard, my fellow PhD sibling and unofficial PhD mentor, thank you for all the advice and lunch conversations. Montserrat Larios, who saved me from my own awful aesthetics and helped produce decent posters. My dear brother, Alex Cuevas Gutierrez, who helped me with some of the software testing. Nathan Corbyn, my research assistant, thanks for all your help.

I'm also grateful to Alex Vetterl, Mario Cekic, and the rest of 2016 Computer Lab PhD class, for sharing this experience with me. To the rest of the Systems Research Group: Vadim, Andres, Chris, Matt, Justas, Ian, Ed, Carlos, Eva, Anil, Gemma, et al. thanks for the daily company and banter, you made coming to work a treat. Moving to a different country to pursue a PhD is not easy, but fortunately I was helped by meeting great people here. Jocelyn, Lakis, Sophia, Rodrigo, Sarah, Krittika, Kevin, Tiago, Marija, Fernando, Dushanth, Jannat, Angela, Ben, Charlie, Jessica, Aracely, and all my other friends from Churchill College and the Cambridge Mexican society, thank you for your friendship, the many shared drinks, and the laughs.

Lastly, I would like to thank the Consejo Nacional de Ciencia y Tecnologia (CONACyT), and the Cambridge Trust, who together funded my PhD.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the 1980s, the USA military developed a decentralised packet-radio system to provide dynamic communications in battlefields [11]. This system forewent the use of fixed infrastructure in favour of direct communication, where its nodes forwarded data for each other to increase their respective reach. By the following decade, mobile ad hoc networks (MANETs) that employed the same principles were introduced for civilian use. Lacking the military's specialised devices, researchers used personal computer (PCs) and repurposed the radios from the IEEE 802.11 standard (Wi-Fi). But, instead of connecting to a Wi-Fi access point (AP), these terminals established direct links with each other and collaborated to provide network services. MANETs enabled innovative solutions at no extra cost, such as on-the-fly networks, communication infrastructure for disaster recovery, and connectivity in remote locations. They were heavily studied during the '90s and early '00s, but research in the field slowed down afterwards, splintering into specialised fields based on application focus. However, a new generation of emerging applications as well as renewed interest in previous ones are bringing attention back to the study of MANETs. These applications include:

- **The Amazon Sidewalk project**[170], which seeks to create a MANET mesh for Internet of Things (IoT) and Amazon devices.
- **Unmanned Traffic Management (UTM)**[135][131], a new technology that aims to automatically manage airspace access for autonomous vehicles (such as drones), and relies on multi-hop machine-to-machine (M2M) communication.
- **Swarming robotics**[149][157], a field that is rapidly gaining attention in industry and academia, requires direct communication between the robots of a swarm to coordinate and achieve complex tasks that no single robot could on their own[134].
- The realisation of **self-driving cars** in urban life will largely depend on their integration to their environment. The use of vehicle-to-vehicle (V2V) and vehicle-to-

infrastructure (V2I) communications can facilitate this integration by coordinating cars among themselves[147] and smart roads[167] to achieve safer and more efficient navigation.

- **Community Networks** are not new, but the COVID-19 pandemic highlighted the need for internet connectivity for continued education and remote work. The Internet Society is looking into the use of community networks to address this connectivity gaps in Africa[142][169].

While all of these applications have their own set of characteristics and constraints which can lead to very different communication solutions, they all share MANETs as foundational work. That is to say, MANETs have shown to be a viable network architecture to solve these problems even if less efficiently [101] [114] [90] [117] , making them the baseline against which application-specific protocols compare. The problem with this though, is that many of these applications were in their infancy or did not exist at all when the original MANET research was published. This, in itself, invites a revisit to the assumptions of the foundational work.

One such assumption is the side effects of reusing technology for new purposes. Besides PCs and Wi-Fi radios, researchers also leveraged the broad library of routing techniques from wired networking as means to forward data between MANET nodes. While this decision sounds at worst, innocuous, it can not be overlooked either due to the significant differences between wired networks and MANETs. Wired networks have fixed point-to-point links, handle high volumes of data, and are concerned with latency, throughput, and low-cost routes. Conversely, MANETs are decentralised, mobile, self-contained, failure prone, and use an implicitly broadcast medium. Their algorithms require nodes to cooperate between themselves to provide network services that have to balance local and global good, while being mindful of power consumption.

The work present in this dissertation seeks to provide a new starting point for future research in this field. I pursue this goal by reframing the MANET routing problem to focus on properties of the problem space, rather than established routing techniques.

1.1 Problem statement

The difference in design objectives between wired networks and MANETs would suggest the need for drastically different routing protocols for each one. However, several MANET routing protocols are based on fixed networks protocols, for instance, DSDV [24] is based on RIP [20], AODV [43] is derived from DSDV, OLSR [54] is based on LSR [7], OSPF-MDR [91] is an extension of OSPF [97], and so on. The extent of this influence is not just a few examples, as these protocols are some of the most influential work in the field, with

hundreds of other protocols based on them or using them as benchmarks for performance comparison. The problem with this influence is that it affects how the routing problem is framed for MANETS, what techniques are used to solve it, and what metrics are used to measure the effectivity of these solutions.

Paraphrasing Richard Bellman [1], the routing problem for fixed networks is defined as *finding the minimal sum of edge weights that form a connecting path between two vertices in a weighted graph*. This problem framing implies a fixed topology, and point-to-point links between the nodes. There's a plethora of proven algorithms to solve this problem (Dijkstra's shortest-path [145], Bellman-Ford [145], etc.), and just as many network protocols that implement them (RIP, OSPF, etc.). And, on the implementation side, these protocols are evaluated in terms of their latency, jitter, throughput, packet loss, etc.

Similarly, the MANET routing problem is concerned with *the procurement of a sequence of wireless links that form a path between any two nodes in a dynamic graph*. This framing implies significantly different properties than those of fixed networks, such as dynamic topology (node mobility), a shared broadcast medium, power-constrained nodes, and a naturally noisy environment. However, as discussed above, several of the protocols designed to solve this problem draw heavily from the concepts and techniques of fixed networks. These properties also invite the prioritisation of evaluation metrics such as power consumption and efficiency/overhead, but most of the research that introduces these protocols uses the same metrics as their fixed counterparts.

Given all this, one can reasonably argue that the design focus of fixed-network routing protocols is unsuitable for MANETs, but what would the right focus then be? To answer that question, let's consider the MANET properties outlined earlier (figure 1.1), their interactions with each other, and the constraints they might impose on a networked system built on them. MANETs operate in a wireless, broadcast environment, making them susceptible to interference from multiple sources and preventing neighbouring nodes to transmit concurrently. Their mobility can induce unfavourable topologies and break previously established routes, spanning trees, and other logical structures. Lastly, they are limited in the number of calculations and transmission they can perform, and nodes can power off at any give moment.

All of these challenges add up to the fact that MANETs experience **frequent failures**. Of course, this is not to say that fixed networks do not have to deal with failures as well, but with MANETs, failures are not an occasional contingency that one plans, they are a constant.

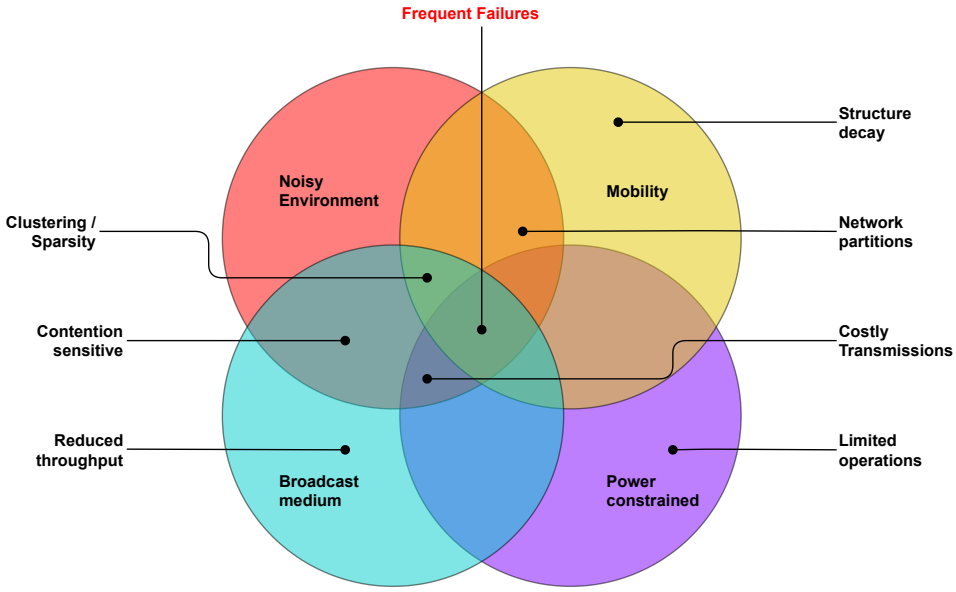


Figure 1.1: MANET properties and their derived constraints.

1.2 Research question and its substantiation

The grim conditions described in the previous section led me to postulate the following research question: *What is the performance effect of designing a MANET routing protocol that favours resiliency over other metrics?* After all, if frequent failures are the natural state of MANETs, then perhaps designing a routing protocol whose primary objective is resiliency could drive a compounding positive effect on traditional network metrics. If true, such a protocol would be much more effective at solving most of the motivating scenarios presented in this chapter than traditional MANET protocols, and as such, serve as a better benchmark for application-specific protocols.

I explore this question throughout this dissertation in different stages. Firstly, I review the existing literature on MANETs to inform and validate my assumptions, as well as learn of existing routing protocols and their failure-coping strategies. Secondly, I explore the research methodology and tools used in the field previously and analyse their relation to failures in MANETs. Thirdly, I present the *Reactive Gossip Routing* protocol family, a set of protocols that implement my ideas on routing resiliency. These protocols are evaluated against the literature through extensive simulations, measuring the effect that this design focus has on different networking metrics and answering this dissertation's research question.

1.3 Dissertation organisation and contributions

The contributions of this dissertation are presented sequentially, as each one builds from the previous. The first contribution of this dissertation is presented in chapter 2 in the form

of a historical recount of the field that classifies the work in generations. This classification allows me to contrast the motivating scenarios and properties of each generation of MANET technology. Chapter 2 also takes the ideas presented in this chapter and explores them in the context of the literature, analysing how established protocols adapt to the MANET properties, and how they cope with failure. A more focused literature review is also presented in chapters 3 - 5 as pertaining to the contributions present in them, which are the following:

1. **Introduction of the MeshSim tool (Chapter 3).** A significant challenge the MANET research community has faced is the lack of consistent tooling for experimentation and evaluation. Traditionally, experiments use either hardware-based experimental setups (Testbeds), or software-based abstractions (Simulations) to measure the effectivity of their work. Testbeds are sophisticated hardware setups used to test network stacks in conditions very close to those of a real-world scenario. These tools are ideal for performing high-fidelity experiments, but are usually expensive to set up and maintain. Simulations, on the other hand, use readily available software to speculate about the behaviour of a network stack for a given network model and experimental conditions. They are versatile and cost effective, which makes them very popular with researchers. However, they are merely abstractions of reality, and thus, require significant additional validation for their results to be credible [83].

   It is within this context that in chapter 3, I introduce *MeshSim*, a new simulator that combines the advantages of simulations and testbeds to facilitate MANET research. Written in Rust [127], a memory-safe programming language for systems development, it provides a simplified IP networking model that allows researchers to quickly and safely implement routing protocols. MeshSim's features are critical for the implementation of my experimental methodology, presented in section 1.4.

2. **Reactive Gossip Routing protocol family (Chapter 4).** The discoveries from the literature review led me to design the reactive gossip routing (RGR) protocol, which uses a variety of techniques to improve its resiliency, or, in other words, reduce the impact of the constraints that MANETs impose on routing protocols. The objective of this increased resiliency is to be able to provide satisfactory levels of operation (measured in various metrics) even under hash conditions. This protocol is later refined to use passive transmissions to inform a routing heuristic, which allow it to make better routing decisions (RGRII). The introduction of a secondary radio into RGRII as a mechanism to provide better data to the routing heuristic completes the protocol family design.

3. **Introduction of the distance-adjusted hop count(DAHC) routing metric (Chapter 5).** It is a long-held convention in the networking community that

shortest-path routes are highly desirable. This view also aligns with common sense, since these routes are intuitive to grasp and their advantages evident. After all, a shorter route requires fewer transmissions for a packet to traverse it, consuming fewer network resources, and potentially achieving a lower latency. Further, these routes have been studied extensively for wired networks, and shortest-path algorithms are at the core of some of the most used routing protocols for them.

Consequently, it is easy to see how these routes would seem like a natural fit for MANETs. Unfortunately, when mobility is present, the efficiency obtained by shortest-path routes comes at the expense of resiliency, which, as shown in chapter 4, increases the routing overhead, lowers the system stability, and paradoxically, lowers efficiency as well. To alleviate this effect, I present a new routing metric called distance-adjusted hop count (DAHC), which improves on the traditional hop-count metric by balancing the overall route length with the inter-node distance, leading to fewer route breaks due to mobility.

1.4 Methodology

The experimental methodology followed in this dissertation is motivated by the acknowledgement of the complex nature of wireless networking experiments, and a desire to produce reproducible experiments that adhere to best practices in the field. To this effect, I designed an experimental methodology called **Calibration-Simulation-Validation** (CSV), which is influenced by Kurkowski et al. [83] and the work done at the SURAN project [16]. Kurkowski et al. showed in their work "MANET Simulation Studies: The Incredibles" that a significant amount of published work in the area falls short of the standards required to make their results credible, even if the publication was peer reviewed. Likewise, the SURAN project, developed by DARPA in the '80s placed a strong focus on experimental fidelity, developing tools that allowed for simulated artefacts to be tested in real hardware, raising the quality and credibility of their results. With the development of MeshSim and the CSV methodology, I pursue the same objectives.

1.4.1 The calibration-simulation-validation cycle

This experimental methodology is designed to add an increased level of fidelity to my simulation experiments, as well as validation beyond that of following best practices. The CSV cycle consists of three stages that are executed sequentially and each one informs the next (figure 1.2), similar to the Waterfall method of software development [12]. When designing and running an experiment, I iterate through these stages as needed until I determine the experiment is calibrated enough to be representative of the phenomena I want

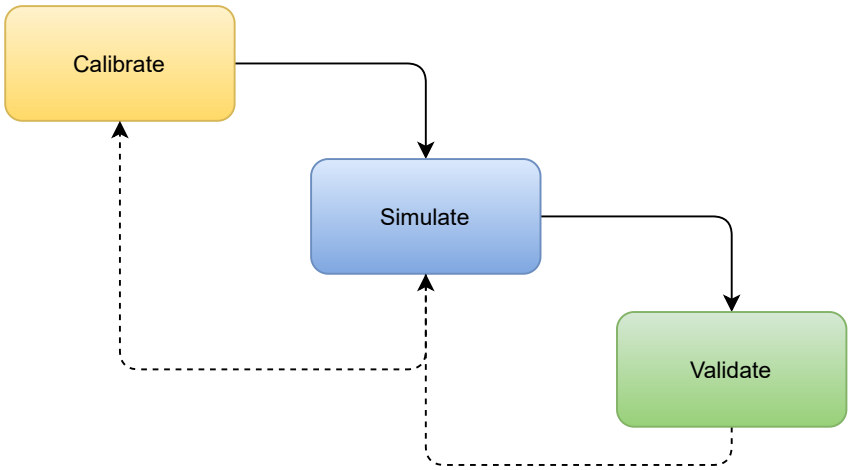


Figure 1.2: The Calibration-Simulation-Validation cycle.

to evaluate and knowing I have the tools to validate those results. Sections 1.4.1.1 - 1.4.1.3 describe each stage in more detail.

1.4.1.1 Calibration stage

This stage is concerned with the procurement of MeshSim configuration data that is representative for the experiment in question. While simulations are advantageous and convenient, they are at best, an approximation to reality, which is a known compromise of using these tools. That being said, not all approximations are the same, and by collecting real-world data I aim to make the results of the simulation a closer approximation to reality.

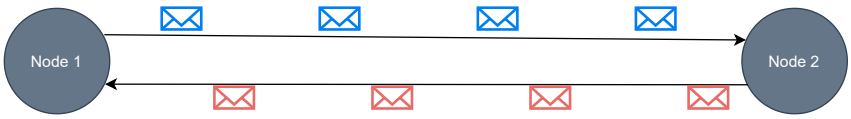


Figure 1.3: Simple two-node echo experiment.

To illustrate this process, consider an experiment consisting on two static nodes echoing packets to each other every second for a given period of time (figure 1.3). It would be trivial to set up and run in most network simulators, however, how consistent would it be with a real-world reproduction of that experiment? High-level metrics such as a packet delivery ratio, mean latency, or packets lost would likely be very similar, but obscure important factors. Latency, for instance, could be modelled in a discrete-event simulator using any of the delay spread models set out by International Telecommunication Union (ITU) [153]. Since the simulated transmission of a packet is practically instantaneous, it could be scheduled to occur in the future according to this estimated delay. However, latency is more than just the signal propagation time of a transmission, it also involves the overhead of moving the data from the network stack of each node into the application

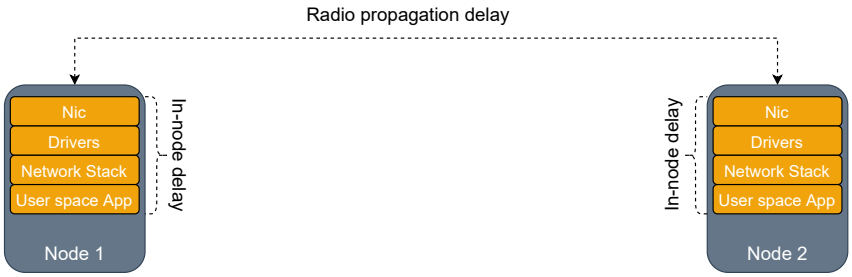


Figure 1.4: Breakdown of the transmission latency of a packet.

consuming it. Further, this overhead is not deterministic, as it is affected by the operating system's scheduler, and other load factors.

Differences such as these tend to scale linearly with the size of the experiment, which would make the validation stage very difficult, if not impossible. Instead of relying on these approximations, I select hardware nodes with specific characteristics that can be used to capture empirical measurements which are later fed into the simulator (see section 3.5). Thus, nodes in my simulations are in a way a virtual representation of the selected hardware nodes, which makes approximating their behaviour easier. What allows me to do this easily is MeshSim's operation modes (see chapter 3), which allow a node to either run using simulated or real network interfaces. My calibration experiments are detailed further in section 3.6.2.

1.4.1.2 Simulation stage

This stage is concerned with the actual execution of the simulation experiments that evaluate my contributions. The simulated scenarios I crafted for my experiments range from simple and general to complex and specific depending on their respective purpose, but they all use the data produced in the calibration stage. The results of these simulations, presented in chapters 4 and 5, form the body of evidence I use to draw conclusions regarding my research questions.

I chose to use simulations for my experiments because their flexibility allows researchers to represent complex scenarios with ease, but as stated earlier, they are abstractions of reality, which makes their results approximations. These abstractions can be adjusted via configuration parameters, modifying the behaviour of its subsystems and the entities simulated within, which makes the documentation of these parameters very important. Following the best practices for MANET simulations outlined by Kurkowski et al. [83], all the parameters from my simulations (summarised in table 1.1) are recorded and made available alongside this document. Each experiment in the subsequent chapters provides a link to a repository that holds all the source code, configuration files, data (raw and processed), and the scripts used to process the data. These artefacts are well documented and provided in good faith to encourage reproducibility practices in science. Further

details on how to craft and run a simulation with MeshSim are provided in chapters 3 and 4.

Policy	**Type**
State the simulator used for experiments	Reproducibility
Simulator code is available to others (open source)	Reproducibility
State the operating system used	Reproducibility
Provide the simulator version	Reproducibility
Address the PRNG used	Reproducibility
All experiment and configuration files are published	Reproducibility
All raw data is published	Reproducibility
Scripts used for analysis made available to others	Reproducibility
Address the simulation type	Best Practice
Address initialisation bias	Best Practice
Stated the number of nodes used in the study	Best Practice
Stated the size of the simulation area	Best Practice
Stated the transmission range	Best Practice
Stated the traffic send rate	Best Practice
Stated the traffic type	Best Practice
Stated the number of simulation runs (iterations)	Best Practice
Stated mobility model used	Best Practice
Stated the mean speed of the nodes	Best Practice
Stated the speed variance about the mean	Best Practice
Stated the mean pause of the nodes	Best Practice
Stated the pause variance about the mean	Best Practice

Table 1.1: Relevant policies in MANET simulation

1.4.1.3 Validation stage

Lastly, the validation stage closes the experimental feedback loop of my work. Each simulation provides a set of data in the form of structured logs, which document every event, action and reaction in the system throughout the entire simulation. These logs can be parsed and aggregated to provide all sorts of metrics regarding the behaviour of the routing protocol in question when executed under the conditions of each scenario. An ideal validation of how closely these results approximate reality would be to repeat the same experiment using a hardware node for each simulated node in the exact same configuration as that described in the simulated scenarios. Unfortunately, such validations are unfeasible as the simulated scenarios involve hundreds of nodes moving in particular directions. And, even if one had the resources to orchestrate such an experiment, this large-scale effort would likely be wasted as any area big enough to accommodate all the nodes and their movements would also be subject to external radio-frequency interference. The data obtained from such an experiment would still be useful, just too costly, especially

since the results of the simulation and this real-world experiment would not match due to the uncontrolled variables.

An alternative validation, however, would be to find a compromise that reduces the cost of the validation experiment. By creating a smaller-scale experiment that could be reproduced with real-world and simulated nodes, one could compare the metrics produced from each one and determine how close they are to each other. If one determine's these curves are similar enough, the experiment is considered properly calibrated and the abstraction is close enough to reality for the results to be significant. Therefore, one could then proceed to run the larger scale simulations with the same configuration parameters, with the confidence that produced results would also be significant. Of course, determining how close these experiments need to be is not trivial and requires further analysis in itself.

Should the results between the validation experiment and the simulated one differ too greatly, this would be indicative that further calibration is required, starting the cycle again. The nature of what needs to be calibrated would depend on the results of a given experiment, and might not be easily discernible from the difference in the aggregated metrics, which makes the raw simulation logs essential for any investigation. Given this, I would encourage any researcher using MeshSim in the future to publish all the raw data alongside their results as I do with this work.

1.4.2 COVID-19 considerations

As the reader might be aware, the world was hit by the COVID-19 pandemic at the end of 2019, affecting all aspects of our every day lives. Personally, this meant I had to spend the last year of my PhD working from home, writing this document in isolation while I tried to finish my experiments remotely. When it came to the simulation experiments, this was not a major problem as the technological infrastructure provided by the Computer Science & Technology department allowed me to login remotely to my simulation servers and do the work I required. For the experiments that required access to hardware nodes, however, this was a different story.

As mentioned in the previous section, the validation experiments consist in the deployment of twenty hardware nodes in a configuration that allowed for non-trivial routes to be created, but small enough that it could be handled by just myself. The first challenge of such a deployment is that nodes need be placed in locations such that the desired topology is created, which requires significant spacing between them, lest we end up with a fully connected graph for which routing is trivial. A deployment such as this thus implies having access to space in which the nodes can be safely placed for their operation. The careful placing of each node also means that a power supply might not be available for them, so each node needs to be able to operate on batteries. Therefore, to run a validation experiment, one needs to place the twenty fully charged hardware

nodes in their predetermined positions, turn them on, and then coordinate them to start their experimental payload at the same time. Such stringent requirements pushed me to develop a centralised experiment-management system (see section 3.5.1), which runs on my personal computer in my lab.

The result from this is that during my last year of PhD, I was unable to access my lab due to the government-mandated lockdown, which rendered me unable to run my validation experiments. Of course, I did run preliminary validation experiments (see section 6.1.3) which aided me in the early development of MeshSim and the experiment-management tool, as well as the required calibration experiments for the simulations, but the execution of last validation stage was out of my hands. I disclose this information upfront and in good faith for the consideration of examiners and readers alike, as the rest of this dissertation is written following the original experimental intent and design.

1.5 List of publications

During my PhD, I have published the following papers (in chronological order, relevant chapter indicated in bold):

- Marco Caballero and J. Crowcroft, 'Demo: MeshSim: A Wireless Ad Hoc Network Development Platform', in Proceedings of the 12th Workshop on Challenged Networks, New York, NY, USA, 2017, pp. 25–27. **Chapter 3**.
- Marco Caballero, Richard Mortier, Jon Crowcroft, 'Fault-tolerance-first routing in MANETs', under submission. **Chapter 4**.

Chapter 2

The rise of Mobile Ad Hoc Networks

The history of MANETs starts in 1972 when Professor Norman Abramson introduced the world to the ALOHANET [2], the first radio-linked computer network in the world. His system utilised random access radio-channel multiplexing to provide message switching over the UHF band [132]. The ALOHA channels derived from his work became foundational in the development of computer network technology such as Ethernet and computer radio networks. With the success of Abramson's work, the Defense Advanced Research Projects Agency (DARPA) from the US government became very interested in advancing this research. This interest would spark decades of research around computer radio networks that I group in the following four generations.

2.1 First generation: Packet-radio networks

Following the success of ALOHANET came a project known as Packet-Radio Network (PRNET). Funded by DARPA and led by its internal Information Processing Techniques Office (IPTO), the project brought together several industrial partners such as SRI International, BBN Technologies, Hazeltine Corp, and Rockwell Collins. The project goals were to extend the work from ALOHANET to produce a multi-hop radio system capable of supporting mobile nodes, which was a significantly more difficult problem.

Robert E. Kahn, one of the lead engineers in the project defined packet radio as "a technology that extends the application of packet switching which evolved for networks with point-to-point communication lines to the domain of broadcast radio" [5]. In the first of multiple publications about the project [4], Kahn outlined that the improvements they sought to implement over Abramson's work were:

- Distributed control of the networking functions.
- Use of spread spectrum signalling for anti-jamming and coexistence purposes.
- Anti-spoof and authentication mechanisms.

- The use of system protocols for routing, resource allocation, remote debugging and other networking functions.
- Power-management capabilities.

The initial system design (Fig. 2.1), which was called *Radionet*, consisted of at least two radio stations, a collection of geographically distributed radio repeaters, and a set of terminal clients. The packet radio units operated in the UHF band (1710-1850 MHz) in a single channel of 20 MHz, and they were the same for stations, repeaters and terminals. However, each unit type had different functionality. The terminal clients were considered the sources and sinks of transmissions. They were capable of initiating transmissions and receiving packets, but do not forward them. Terminals also required means for input (keyboards) and displaying data (screens). Repeaters were simple units outfitted with a packet radio and configured to forward all received traffic. Finally, stations consisted of a minicomputer connected to a packet radio, and acted as the control centre for the entire system.

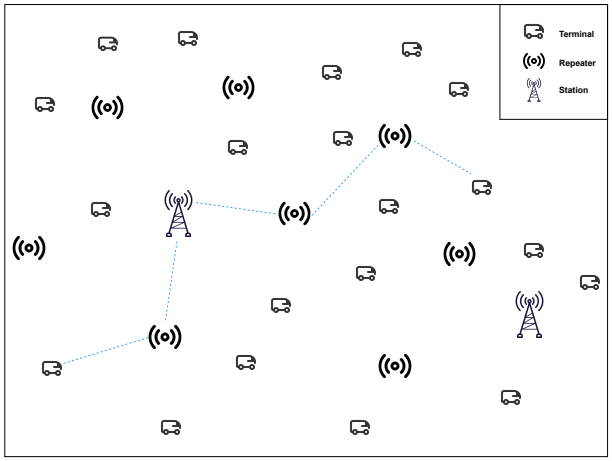


Figure 2.1: Radionet diagram.

The first iteration of the system was crude. Repeaters were dumb units, and control was centralised in one station, only using the other stations as backups. Routes could only be calculated at the control station, so all traffic flowed to it, and then to its final destination. However, they demonstrated it was functional when a test bed was built in the San Francisco bay area in 1976.

Further iterations on the system saw upgrades all around, such as:

- Established routing protocols [5] [9] [11].
- Distributed control among a larger number of stations [3].
- Inter-networking [6] with Ethernet-based LANs, ARPANET, and SATNET.

- Capability to do handoff between repeaters to continue providing coverage for a moving terminal.

For its time, the system ended up being quite sophisticated, and demonstrated without a doubt the feasibility and usefulness of packet-radio systems. So in the early 1980s, the US military was ready to take these concepts and try them out in military communications.

2.2 Second generation: Battlefield communications

The success of the PRNET project convinced the US military that packet radio networks were not only viable, but practical. So in 1983, they started the SURAN project [16]. The goals of the project were to expand the research from PRNET and produce a packet radio network of military grade. This meant a significant hardening of the software and hardware requirements, as well as a plethora of additional features.

The tactical focus of the system placed a greater importance on mobility, distributed self-management and inter-networking. Further, the project had the following operational requirements:

- Flexible and robust architecture that can accommodate dynamic changes in the number of nodes and their geographical distribution.
- Robust operation, even when dealing with changing traffic demands and link conditions.
- Provide resistance to data tampering and secure data and control traffic, despite electronic countermeasures (ECM).
- Signal-transmission adaptability, to minimise interference, error links and allow multiple simultaneous transmissions.

The project was very successful and yielded several advancements in the field. They developed specialised hardware such as the *Low-cost Packet Radio (LPR)*, the *Network Interface Unit (NIU)*, and the *Automated Network Manager (ANM)*; the *SURAN Protocols Packages (SURAP)*; and dedicated testing facilities dubbed the *SURAN Automated Laboratory Testbed (SALT)*.

The *LPR* was a sophisticated x86-based radio that featured spread spectrum signalling, forward error correction (FEC) in varying encoding rates, adjustable power output, and operation over 20 channels in the UHF band, all controlled by software. The *NIU* was an add-on for the *LPR* that provided TELNET, Gateway, Traffic generation, and end-to-end encryption services. Finally, the *ANM* provided fault detection, performance monitoring of the network, identification of compromised nodes, and encryption key distribution.

Throughout the project, several networking protocols were developed. As the project's iterations progressed, different versions of the system were released. Each release consisted of a bundle of the hardware and a given package of the protocols, which were chosen to align with the experimental goals of the release and in accordance to the protocol's maturity. The protocols addressed different layers of the networking stack and different needs, such as: Link adaptivity, channel access, link scheduling and buffering, adaptive network routing, routing in large networks, network security and management, and end-to-end protocols. Three main SURAP versions were released:

- **SURAP 1** - Distributed and dynamic routing for networks up to 50 nodes in size. Had an emphasis on quick route update when a link failure was detected.
- **SURAP 2** - Large node set support (up to 10,000 nodes). Uses hierarchical routing with dynamically formed clusters.
- **SURAP 3** - Builds up on previous work but enhances security features and adaptive transmission parameters for the LPRs.

Lastly, but not least important is the SALT facilities. The development and experimental procedures of SURAN depended on two platforms provided by SALT: The Packet Radio Integrated System Module (PRISM) [14] and a network simulator called PC-NETSIM [15]. PRISM consists of a matrix of coaxial cabling and programmable radio frequency (RF) attenuators designed to test the radio-link conditions in several scenarios. Each PRISM rack could house up to eight *LPRs* with their corresponding *NIU*. By configuring the attenuators, different network topologies such as string, star or dense groups could be achieved. The *NIU* unit accompanying each *LPR* would then be able to test different traffic patterns in the configured topology. Its counterpart, PC-NETSIM, was an x86-based simulator able to generate complex simulation scenarios. It had the added feature that the code that ran in each node of the simulation was the actual, unaltered code that ran on the *LPRs*, providing an additional level of validation to the simulation results.

The SURAN project was a wild success, both as a practical military network, and as the first wireless ad hoc network. The principles laid out in it, as well as the plethora of networking algorithms it developed would inspire academics in the upcoming decade when this research area would move from the military and into academia.

However, the military sector (government and private) was by no means done with research and development in this field. In 1997, the Joint Program Office (JPO) of the USA's Department of Defense (DoD) launched a project called Joint Tactical Radio System (JTRS) [52]. The objective of this project was to create the next generation of battlefield communications that could enable data, voice, and video communications for all branches of the military. Following the principles and success of SURAN, the JTRS

aims to replace the *LPR* with next-generation Software Defined Radios (SDR) capable of adaptive frequencies and forms (waveforms) via software update. This shift to SDR hardware instead of specialised radios aims to equip all armed forces with the same radio unit and be able to customise its functionality by software to adapt as necessary. However, unlike SURAN, this project is widely accepted as a huge failure. Even so, the DoD's quest for next-generation battlefield communications remains alive today [144].

On the other hand, the private military sector has seen great success in the development of MANET-based communication solutions. Such is the case of the company Persistent Systems, whose flagship radio, the MPU5 [159], is an example of successful MANETs in the military space in the current age.

2.3 Third generation: Civilian MANETs

Up until the SURAN project, research into MANETs was dominated by government agencies, the military, and their corporate contractors. This was natural, as the research relied on the use of expensive computational facilities and of specialised radio equipment that operated on regulated portions of the electromagnetic spectrum. It was shortly after the conclusion of this project, however, that the MANET research field opened up to the civilian sector. The catalyst for this transition was the advent of the IEEE 802.11 standard radios [161], commonly known as Wi-Fi. They were relatively inexpensive, designed to work with off-the-shelves personal computers, and operated on the unlicensed ISM band [133]. The availability of these radios in conjunction with the commoditisation of personal computers in the 1990s provided academics with the hardware building blocks necessary to build MANETs.

An initial obstacle to this goal was the fact that Wi-Fi radios were designed to operate with network infrastructure as a wireless extension of Local Area Networks (LANs), and not as peer-to-peer radios. What prevented them from being used as MANET radios was the Medium Access Control (MAC) functionality. Wi-Fi radios by default utilise *isotropic* or *omnidirectional* antennas. When they make transmission, all devices in radio range receive it regardless of who the intended recipient is. Thus, the electromagnetic (EM) waves from concurrent transmissions of neighbouring devices collide, rendering each other useless. The Wi-Fi standard addresses this problem with inclusion of a MAC layer. This layer runs multiple protocols that include a *coordination function*, which determines the transmission order of clients to avoid collisions. In a traditional Wi-Fi network, it is the Access Point (AP) who runs this coordination function.

In 1993, the Distributed Foundation Wireless Medium Access Control (DFWMAC) [19] protocol was added to the Wi-Fi standard, enabling these radios to coordinate transmissions without the need of a central control point. This addition to the Wi-Fi standard can

be considered the informal inception of civilian MANETs. Later, by 1997, the Internet Engineering Task Force (IETF) [21] created the MANET working group [33] to formally track the development of this field.

DFWMAC defined a new *distributed coordination function* called Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA). This function relies on medium sensing and a distributed *handshake* used by clients to determine if they can transmit or not. Nodes interested to make a *data transmission* first *sense* the medium for a period of time, and if they determine it is free, they either begin their transmission or start the Collision Avoidance (CA) handshake. This handshake involves sending a Request to Send (RTS) message to the intended receiver of the data transmission, and waiting for a Clear to Send (CTS) message reply from it.

Not all problems with the medium are solved by the MAC layer, of course. As the EM waves of a transmission traverse free space, they decay in a rate inversely proportional to the square of the distance they have travelled [69]. Past a certain distance, those radio waves can no longer be reconstructed into Wi-Fi frames and nodes from that point onward are considered to be *out of radio range* of the sender. However, those signals continue traversing and become what is known as noise, which can interfere with the successful reception of other transmissions, especially when this noise accumulates from multiple sources. Additionally, the CSMA process only coordinates access to the medium for nodes that are considered within radio range of the incumbent nodes, which creates the notorious hidden and exposed terminal problems [72]. However, the CA handshake significantly alleviates the hidden terminal problem.

Equipped with the necessary hardware, the only thing researchers needed to produce a workable MANET was a mechanism to effectively relay data packets across the network (more on that in section 2.5.1). The mechanism most commonly used for this purpose is known as *routing*. The first routing protocol for third generation MANETs was DSDV [24], which was published in 1994 and kicked off over a decade of intense research in this area. It was designed in line with the IEEE 802.11 [161] standard, but published before Wi-Fi cards were commercially available. DSDV is based on the *Routing Information Protocol* (RIP) [13] for wired networks, and as such, takes an aggressive approach to route calculations. Unlike most routing protocols, it can operate at the link layer of the OSI model [121], updating existing routing tables, while another routing protocol operates at the network layer. This mode enabled MANETS running DSDV to integrate with Wi-Fi base stations, a feature that was telling of the author's view of ad hoc networks: As extensions of existing infrastructure. Such views were consistent with those of several other authors at the time.

These views were a large departure from the SURAN project which was designed to support primarily mobile nodes that formed autonomous networks mostly isolated from any

infrastructure. However, they were not unexpected, as the equipment available to civilians at the time consisted mostly of desktop and laptop computers, and it would be years before the proliferation of mobile computing. Thus, most of the applications envisioned by researchers at the time implied low mobility and some form of integration with fixed networks, such as on-the-fly networks for business or academic settings, temporary offices, disaster recovery, or sensor networks [61]. In light of these low-mobility scenarios, it made sense to repurpose routing techniques from the fixed networks literature.

However, it was not long before the vision for MANET applications expanded and included scenarios with more numerous and mobile nodes. For instance, a very attractive scenario for MANETs in the early '00s was content delivery and communication between handheld computers (also known as PDAs). This scenario raised the number of potential nodes in a network from a few dozen in office settings to thousands on the streets, all of which were on the go. Researchers soon learned how some of the most sophisticated protocols published in the literature became nearly inoperable in the presence of such mobility.

Several coping mechanisms were developed to handle this scale and mobility with varying levels of success, with some protocols simply stating upfront they were designed for low mobility only. But a very successful strategy was to depart from the generalist approach MANETs had had so far, and focus either on a set of problem or constraints. This approach allowed researchers to discard some edge cases and utilise the properties of this constrained domain to their advantage. For instance, content delivery to PDAs does not require real-time delivery as opposed to other applications like live video streaming, thus, the requirements about the liveness of the data can be relaxed and *eventual delivery* techniques can be incorporated. This trend gave way to several specialised subfields of MANETs such as delay-tolerant networks (DTN), vehicular ad hoc networks (VANET), wireless sensor networks (WSN), wireless mesh networks (WMN), etc. This fracturing of the field saw increased interest and research activity in this subfields and a waning interest in general MANET research.

2.4 Fourth generation: Autonmous networks

As the research interest into generalised MANETs diminished in favour of application-specific ones, some of the applications previously considered for MANETs solidified while others evaporated. The big push to have MANETs integrate with existing infrastructure became successful with the emergence of Mesh networking [81], which has seen both commercial success [154][152][166] and acceptance within industry standards [109]. While MANETs are not yet widely deployed for natural disaster recovery as it was hoped, the technology has been adopted to create communication systems that work in the cases

of other infrastructure shutdowns. Such is the case of Bridgefy [156], Firechat [136] and Briar [151], which have been deployed during government-mandated infrastructure shutdowns due to protests in countries like Egypt, Taiwan, and Hong Kong [122] [125]. For other MANET applications it became simpler to use cellular data networks due to their wide adoption. Incidentally, the explosion in cellular data demand has grown so much that MANETs became an attractive alternative for offloading some of that traffic [119] [124].

A common factor in the applications of the previous generation is that they are human-centric. That is to say, humans directly consume the services enabled by these applications, and they also directly operate the devices they run on. For instance, Mesh networking provides better connectivity for human-operated devices like smart phones, tablet and laptop computers; Early VANETs were envisioned to provide traffic reports and warnings for human drivers [116][107][98], or distribute content to car devices such as map data for navigation [108]; And, content distribution applications were often modelled on human social connections [99][96].

What distinguishes the fourth generation of MANETs, is a departure from this human-centric objective. The consulting firm PWC estimates that over 76,000 drones will be flying daily over the UK by 2030 [165]. Most of these drones will be autonomous and airspace access will be orchestrated by Unmanned Traffic Management (UTM) systems[135][131]. Early WSNs provided actionable data directly to humans, such as the state of the soil on a farm [94][128]; while fourth generation WSNs provide real-time data that is combined at multiple levels to trigger autonomous systems in smart cities [141] and industrial settings [163]. New generation VANETs no long aim to give information to the driver, but rather, provide the vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication necessary for autonomous and coordinated driving [147][167]. By removing the human factor, many of these applications will scale to larger sizes, with nodes moving faster than before. While exceptional work has been done in all these subfields, plenty of the solutions in them were deemed appropriate by benchmarking against general-purpose MANET protocols that were unsuitable for dynamic environments then, and even more so now. To further discuss the suitability of a MANET-based network architecture for a given application, however, it is necessary to first dissect their anatomy.

2.5 Building a MANET

As stated in previous sections, a mobile ad hoc network (MANET) can be defined as a collection of independent nodes that communicate through wireless links and collaborate to provide network services and forward data for each other. The high-level anatomy of a MANET node, depicted in figured 2.2, consists of a collection of wireless links, a network stack, and a computing unit. It is important to dissect its composition to understand the

needs of its subcomponents, such that we can evaluate how well do different solutions meet these needs.

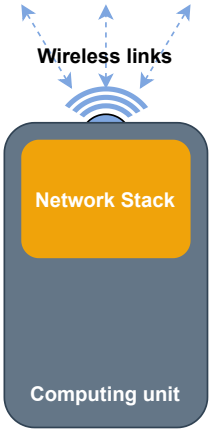


Figure 2.2: High-level anatomy of a MANET node.

A wireless link is a virtual connection between two nodes established through a radio-frequency channel provided by a wireless network interface card (NIC) and managed by a medium access control layer (MAC). The MAC layer allows semi-concurrent medium access to neighbouring nodes by establishing an access order, thus preventing collisions of their transmissions. It can also provide several features that are commonly used by higher level components, such as adaptive power control and various link-layer statistics. The most commonly used wireless links for MANETs are Wi-Fi links, which operate in the ISM band [182]. These links provide relatively high bandwidth, a medium transmission range, and power consumption low enough that even battery-powered systems can use it. The suitability of a given wireless technology for a MANET will depend on how well the radio-link budget (bandwidth, transmission range, power consumption) matches the application requirements. Other alternative radio technologies include Bluetooth [59], Zigbee [53], LoRa [155], and WAVE [100].

The network stack refers to the software run by each node to provide its networking functionality. This stack ranges from the NIC drivers to the networking system calls of the kernel, but since these components are provided for all network activities (not just MANETs), we focus on the delivery mechanism used by nodes to forward data between each other. This mechanism generally involves a set of protocols that, through different operations, determine the best way to relay data from a source node to a destination(s), passing through multiple intermediary nodes. The effectivity of these mechanisms is crucial to the success of a MANET, and its performance depends on multiple factors, such as the application in question; the bandwidth and latency requirements; the mobility, quantity, and concentration of nodes, etc.

Lastly, the computing unit of a MANET node refers simply to the actual computer

that controls the wireless NIC and runs the code for the network stack. The computing capacity of these units tends to be diverse and is difficult to control for, as any node with the appropriate hardware and software can join a MANET. While power consumption increases linearly with computing capacity, it is generally desirable to have as high a computing capacity as possible as this allows for more complex route calculations. However, this capacity is generally not factored heavily into the design of a MANET because it is fixed once chosen. Conversely, the soft nature of the data delivery mechanism makes it easy to switch at any moment, often with dramatic changes to the performance of the system. It is because of this, that the majority of MANET research focused on these mechanisms.

2.5.1 Data-delivery mechanisms

The ultimate goal of any network is to deliver data between its endpoints. The basic premise of data delivery in MANETs is that, since not all nodes are in direct transmission range of each other, intermediary nodes will relay data packets between them. The way in which these packets can be relayed is hugely varied and their effectivity is dependant on multiple factors, starting with the overlaying application using the network.

For instance, one naïve way to deliver a data packet from a source to a destination, is for the source node to transmit the packet and have every receiving node forward it. This mechanism creates multiple *replicas* of the original packet which should reach all of the *connected* nodes of the network and eventually reach the destination, provided there is a *path* to it. A more sophisticated mechanism would consist on *forwarding* a packet one node at a time through a sequence of suitable intermediary nodes until it reaches its destination. This mechanism is predicated on being able to determine which intermediary nodes are suitable for forwarding the packet, and what is the cost of acquiring this information. No data-delivery mechanism is perfect for all applications, which calls for a clear delimitation of the scope of the literature review where my work fits. To this end, it is important to review these techniques, their properties, advantages and disadvantages.

2.5.1.1 Replication

Replication-based mechanisms involve the creation of copies of a packet at different stages of its traversal through the network, with each replica following a different path towards their destination. The techniques based on this mechanism usually have little information about the network state and thus rely on the replicas to probe different potential paths to the destination concurrently.

The simplest replication-based technique is called naïve flooding, and it involves the forwarding of all received data packets between the nodes of a network.Since all transmissions

in a broadcast medium are received by every node in range, each transmission effectively creates a replica of the packet for each receiving node. Of course, additional control logic is necessary to keep nodes from re-forwarding packets from their neighbours and avoid creating infinite loops between them. The packet replicas it creates at intermediary nodes add a high level of fault-tolerance and likelihood of delivery, since they will probe all available paths in the network. It is also not affected by the mobility of the nodes, making it suitable for data delivery in high-mobility scenarios. However, it is considered to be very inefficient due to its high overhead and the contention it induces in the medium [42].

More sophisticated replication-based techniques improve on the efficiency of naïve flooding through different means. Gossip-based flooding [60][130] utilises a probabilistic approach to determine whether to forward a packet at each node or not. Since most of the replicas created by flooding end up being useless, the removal of a significant portion of them through the *gossip factor* would improve network performance while keeping its reachability properties intact. However, probabilistic approaches can not deterministically guarantee packet delivery.

A different approach to improve the performance of flooding comes from determining Minimum Connected Dominating Set (MCDS), which is the minimum set of nodes that would have to broadcast a packet for it to reach all nodes in the network. Since this problem is proven to intractable (NP-Complete) [56], approximations to this set are instead used through different heuristics such as self-pruning and dominant pruning [56].

While inherently inefficient, replication-based techniques can be quite useful in MANETs if applied judiciously. They are commonly used by routing protocols to probe the topology of the network, and are useful when transmissions are either infrequent, require to reach all nodes, or nodes have a very high mobility rate. These techniques remain an active area of research, as recent publications (2021) find uses for them in new settings such as content distribution in MANETs with low-bandwidth links [171].

2.5.1.2 Forwarding

Contrary to their replication counterparts, forwarding-based techniques seek to utilise knowledge about the network topology to forward each packet only by the necessary nodes in order to reach their destination. Based on this knowledge, packets are forwarded sequentially in a node-by-node basis utilising unicast transmission. That is to say, each transmission has an intended destination, so even if neighbouring nodes are to receive it, they would normally discard it rather than queue it up.

Routing is the most common technique in this family and in MANET data delivery in general. It is defined as the process through which a source and destination determine the most desirable subset of nodes in the network to use as intermediaries to relay data packets between them (Fig. 2.3). The chosen nodes relay the *data packets* sequentially

node by node, and this sequence is known as a *route*. The criteria to determine a node's suitability for a route is a significant design decision, and several protocols are mostly differentiated between themselves by these criteria.

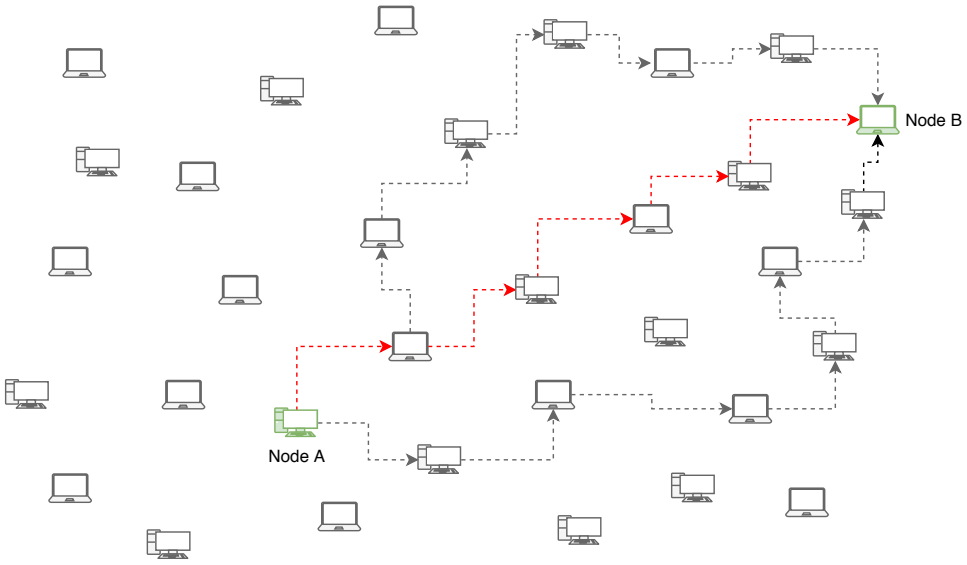


Figure 2.3: A wireless ad hoc network displaying three potential routes between nodes A and B (shortest route highlighted).

To create and maintain routes as well as support other protocol operations, nodes exchange *control packets*. The *overhead* of a protocol is then defined as the ratio of control to data packets, and it is considered an important measure of a protocol's efficiency. Since the capacity of a wireless network is limited [46], an *efficient* protocol spends more of it relaying data packets than control packets. A poor selection criteria for adding nodes to routes also contributes to overhead, since suboptimal routes produce unnecessary transmissions [62].

Since only a relevant subset of nodes are involved in data transmission between each source-destination pair, the network-wide resources consumed by each data flow is significantly lower than that of flooding, making it generally much more efficient. However, it is not without its drawbacks. Depending on the nature of the routing protocol, it might incur a delay when a new route is required, or spend significant resources to have routes readily available. Further, the acquisition of said routes is dependant on a path existing to intended destination, which might not always be the case (e.g. partitioned networks). Lastly, the utility of these routes is affected by all factors that can cause a link to disconnect, such as mobility, medium contention, power downs, or interference. Nevertheless, routing is the most popular data-delivery technique in MANETs given its suitability for most applications.

Another forwarding-based technique that is used in MANETs is *greedy* forwarding. In this scheme, a full route from a source to a destination has not been calculated in

advance. Instead, routing decisions are made at every node according to the minimisation of a given set of metrics, with nodes choosing the best next hop for the packet based on their local view of the network. The advantage of this technique over end-to-end routes is that it doesn't depend on having a fully accurate view of the network at every node. Since topology updates are received faster by closer nodes than to further ones, as a packet progresses closer to its destination, better forwarding decisions can be made. Protocols based on this technique benefit from added resiliency to topological updates, but can still be subject to forwarding errors and require the overhead of propagating the necessary information for forwarding decisions to all nodes. Same as routing, they also depend on the existence of a connected path to the destination.

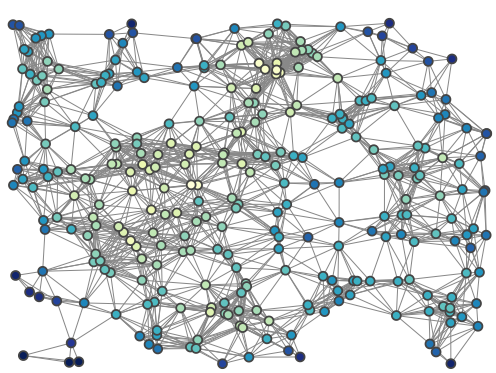
2.5.2 Connectivity and Mobility models

As discussed earlier, when designing a MANET, one should be mindful of choosing computing nodes, wireless links, and networking protocols that align with the needs of the application that will utilise it. Thus, the properties of the application are of paramount importance. While an in-depth exploration of all the potential application properties is out of scope for this document, there are two exceptions: The expected distribution and mobility of nodes.

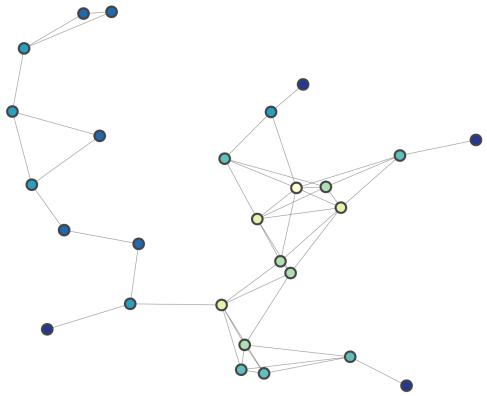
The distribution of nodes refers to the expected quantity of nodes and their geographical concentration. This property factors heavily into the choice of network protocol to use for several reasons. Firstly, some network protocols perform occasional flooding operations which might complete in an acceptable time for medium networks, but not so much for large ones. Secondly, the concentration and distribution of these nodes also has an impact in the network design. While dense MANETs (figure 2.4a) present many alternative paths, link-state protocols may struggle to keep up with the state of so many neighbours; conversely, sparse configurations (figure 2.4b) impose a low overhead in terms keeping track with link-state, but provide few paths between nodes. Lastly, we most consider that this node distribution also affects the connectivity of the nodes and could be split the network (figure 2.4c), causing a lack of paths between its various subsets. This split could be the natural network formation from the application (e.g. two different sites or user groups), or induced by the mobility of nodes.

The mobility pattern of the nodes (i.e. the mobility model) is tightly coupled to the nature of the application, and can inform the selection of network protocols as well. A model that expects a high degree of mobility from its nodes might require a protocol that converges quickly, that is unaffected by mobility, or that is fault tolerant. Alternatively, a model that presents some degree of predictability in the node trajectories could open up different avenues for connectivity, like the *store-and-carry* paradigm.

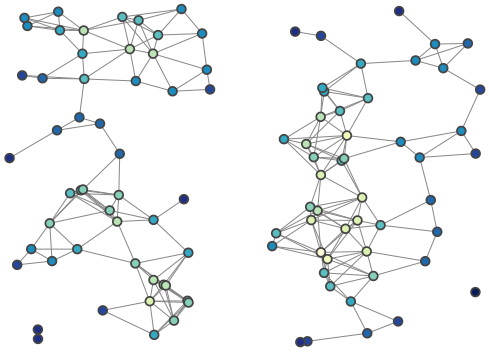
The store-and-carry paradigm involves the addition of a packet buffer to each MANET



(a) A dense MANET.



(b) A sparse MANET.



(c) A split MANET.

Figure 2.4: Different node distributions.

node such that, in the case that a path is not currently available to the destination, the packet would be stored and transmitted at a later time. To illustrate this principle, consider figure 2.5 in which we can see the orbits of two geo-synchronous satellites. If these satellites needed to transmit data to each other, they could keep track of their respective orbit times and wait for the next transmission window (e.g. when they are close enough to establish a wireless link).

The store-and-carry paradigm can be used to enhance both replication-based (e.g. Spray-and-wait [84]) and forwarding-based (e.g. CarTel [87]) protocols to overcome disconnected network scenarios. Of course, this paradigm is not infallible and for it to be applicable, two pre-requisites must be met. Firstly, the mobility model must guarantee a future transmission window, which is more complex than just having some degree of predictability in the mobility model. Consider the Manhattan mobility model [63] as depicted in figure 2.6. While the model can predict that at time t_0, node A will have a transmission window with the network in the future (juncture β), link conditions (duration, bandwidth, quality) might not be appropriate to transmit the payload. Further, should node A choose to turn left at juncture β, there is no guarantee a new transmission window will emerge in the future. Secondly, the application must be able to tolerate the

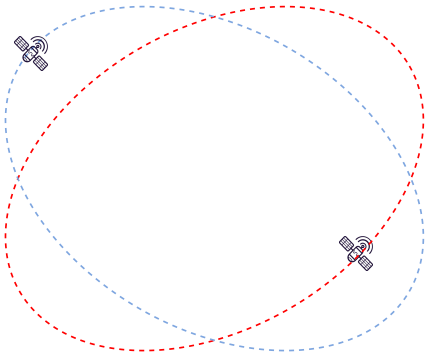


Figure 2.5: Geo-synchronous orbits of two satellites.

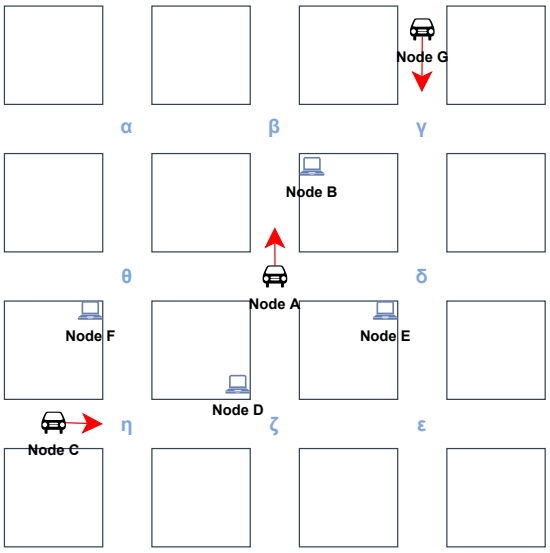


Figure 2.6: Manhattan mobility model example.

delay incurred by waiting for a transmission window. While this might be fine for some applications, real-time applications such as audio/video streaming or swarming robotics are unlikely to be able to cope with this delay.

2.6 Research Scope

MANET research has several dimensions and layers, and no one solution is ideal for all the combinations of these variables. Thus, it is fundamental to define the scope of my research to identify the relevant variables and understand where it fits in the literature. To address the research question I postulate in section 1.2 within the context of the problem statement defined in section 1.1, and in line with the motivating scenarios of Chapter 1, I intend to design a network protocol more adept than those in the literature at solving

the *general* MANET routing problem. This protocol will be evaluated extensively by simulation experiments, designed to test it and the protocols it will be benchmarked against under a variety of relevant conditions to determine their suitability to solve the aforementioned problem.

The choice of focusing on routing protocols is motivated by the fact that they have shown to be suitable for general MANETs as well as application-specific ones with different levels of effectivity (see chapter 1). Within this context, my focus is on routing protocols that produce end-to-end paths. While other solutions like the use of the store-and-carry (SC) paradigm can enhance these protocols to support disconnected paths to some degree, they are not suitable for all scenarios. There are two reasons for this:

1. For SC to be beneficial, there must be future contact opportunities between nodes, and the more information that can be inferred about them the better.

2. The use of SC requires a relaxation of the delay bounds for packet delivery to accommodate for the carry phase of the paradigm.

While some applications might have information about contact opportunities and might not be sensitive to this added delay, that is not the case for all. For instance, audio/video streaming and the transmission of navigation data for swarming robotics are very delay-sensitive. Since my work aims to improve on the general-purpose routing protocols in the literature to serve as a better benchmark for application-specific ones, it would not make sense to incorporate this technique in my solution or the evaluation. One could argue that regardless of delay sensitivity, an end-to-end routing protocol can not deliver packets through a disconnected network, while protocols the implement SC can at least eventually deliver them. However, eventual delivery might not be sufficient for all applications. Conversely, early failure detection is something that most applications can benefit from, and something that end-to-end routing protocols can provide in the case of a disconnected network.

Considering this scope, the rest of this chapter will explore the most prominent routing protocols in the literature and provide an analysis of their suitability to solve the MANET routing problem.

2.7 Routing protocols

Given the impact of routing protocols to the performance of a MANET, a significant portion of research in this area has been dedicated to them. This abundance of work makes the task of performing a comprehensive literature review of MANET routing protocols challenging. However, one can still focus on the most significant work in the area and narrow down the reviewing scope as pertinent.

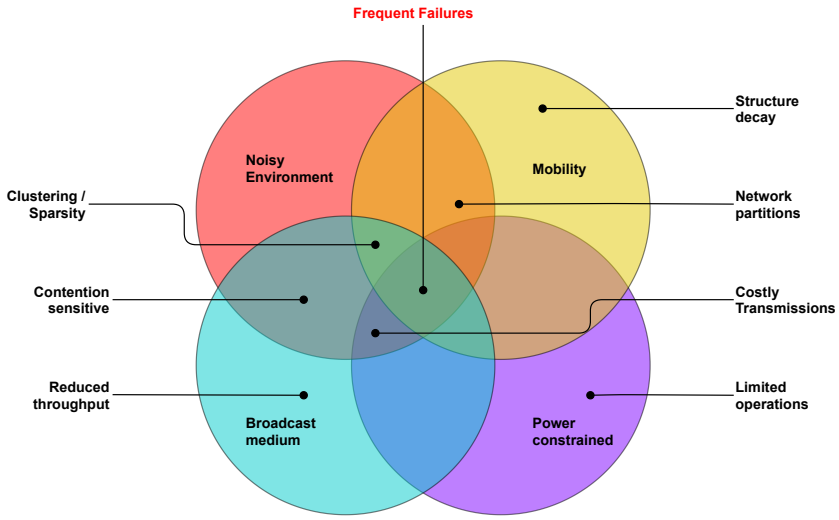


Figure 2.7: MANET properties and their derived constraints.

The protocols present in this review are grouped according to the protocol family they belong to. This logical grouping is important as it allows for direct comparison between members of the same family, which share similar design objectives, and with members of the other families for contrast. By the end of this section, it will become obvious that no protocol is infallible, and all of them have tradeoffs. Nevertheless, some of them have been preferred by both academia and industry to study, extend, compare against, and even implement, making them the de facto benchmarks in the field.

Of course, for a review to bring value, it needs to be more than a simple recount of what has been done, but offer an analysis of each protocol and characterise their potential to solve the general MANET routing problem accordingly. Several patterns emerge during this review, such as the techniques used as building blocks or common design tradeoffs. These patterns can be used to discern the preferred scenarios or applications for each protocol, or used to predict their performance under a given set of metrics. This analysis, however, requires me to first define the criteria on which I will compare them.

The most import measurement of a routing protocol's effectiveness is *packet delivery*, but that can only be assessed experimentally. Analytically, however, we can still look at how the different parts of a protocol may promote or hinder its goal. In other words, we evaluate how well its design fits with the properties of the problem domain.

The problem domain at hand was initially introduced in section 1.1 and depicted in figure 2.7. Each MANET property presents different challenges to routing protocols which are amplified at their intersections, leading to frequent failures.Thus, it would be reasonable to hypothesise that mitigating the impact of each individual property on a protocol could lead to fewer failures, promoting fault-tolerance through the reduction of faults. This hypothesis is supported by the work of Jabbar et al. [113], who identified mobility and radio range as the primary operational parameters that predict resiliency

of MANETs at the topological level. By this reasoning, the evaluation criteria I will use through the rest of this chapter is based on how well each protocol adapts to these properties

Requirement	**Definition**	**Scale**
Mobility	The extend to which the protocol can continue its normal operations under varying degrees of node mobility.	See table 2.2
Power usage	Design considerations built into a protocol to mitigate its power requirements.	See table 2.3
Contention	Design considerations built into a protocol to mitigate the medium contention it generates	See table 2.4
Scalability	A protocol's ability to adapt to changing conditions in the number and distribution (density) of nodes it supports.	See table 2.5

Table 2.1: Summary of the MANET requirements.

I call this criteria the *MANET requirements*, which are summarised in table 2.1. Each requirement is graded on a scale with three possible ranks, from lower to higher, given in accordance of how well a protocol fulfils it. A higher fulfilment is desirable for all requirements, but different wording is used for the scales as semantically appropriate. These requirement scales are provided in tables 2.2 - 2.5.

Grade	**Definition**
Limited	Only supports a small degree of mobility before its operations are severely hindered.
Viable	The impact caused by mobility is constant, limited, or there are mitigations in place.
Friendly	Mobility assumed as the norm. Protocol operations do not rely on any order. Continues working properly under constant mobility.

Table 2.2: Grade scale for mobility.

When considering a protocol's suitability for **mobility**, we consider how the mobility of the nodes may directly affect the mechanisms used by the protocol to do its job. For instance, mobility of a node might break an established route, trigger transmissions to update the network's topology, invalidate caches, etc. Thus, a protocol designed for mobility would have mechanisms (implicit or explicit) that limit these effects. The grading scale of this requirement is detailed in table 2.2.

When it comes to the **power consumption** of a protocol, there are two main sources for it: transmissions and calculations. While the power-consumption implications of a protocol's calculations are implementation dependant, one can identify expensive mechanisms in their design such as frequent recalculations or lack caching. Likewise, the power

Grade	Definition
Unrestricted	No considerations at all are designed into the power consumption of the protocol.
Controlled	Some mitigations are in place. The scaling of power consumption with other metrics is limited or capped.
Efficient	Several mitigations are in place. Power consumption is one of the primary metrics that drive the logic of the protocol.

Table 2.3: Grade scale for power-consumption.

consumption of radio operations (transmissions and receptions) are dependant on the hardware, drivers, and OS used on devices, and are he dominating component in power consumption of a protocol. But by estimating the number of radio operations performed by a protocol based on its design, one can get an idea of its power efficiency. Table 2.3 explains the criteria used for this requirement.

Grade	Definition
Unrestricted	The routing overhead is quite large or it scales up with other metrics (like mobility or scalability). There are no mitigations in place for the contention.
Controlled	The protocol is aware of the contention of its operations. It makes efforts to mitigate it, or limits the rate at which it scales up with the other metrics. Avoids transmissions when possible.
Efficient	The protocol is fully oriented towards minimising contention. Several mitigations are in place. Often relies on cross-layer operations, working with the MAC and PHY layers.

Table 2.4: Grade scale for medium contention.

Medium contention in MANETs is caused by neighbouring nodes (e.g. within radio range of each other) attempting to make transmissions concurrently. MAC protocols address this by creating a sequence of ordered accesses to the medium between the incumbent nodes. However, it is routing protocols who initiate these transmissions and who can determine how traffic patterns affect contention for better or for worse. Further, most transmitted packets have an expectation of being forwarded by other nodes, potentially introducing further contention in different regions of the network, Thus, a good predictor of contention of a protocol is its routing overhead, since fewer transmissions result in fewer opportunities for nodes to contend over the medium. Table 2.4 details the grading scale for this requirement.

Grade	Definition
Low	The protocol only operates well within a limited range of node densities or network sizes.
Medium	Protocol operations are somewhat affected by network size, node density, or both.
High	Protocol performance and operations are independent of network size, and are not especially affected by node density.

Table 2.5: Grade scale for scalability.

Lastly, we consider the **scalability** requirement. While this one does not match directly to one of the MANET properties from figure 2.7, it is a consequence of the interactions of these properties. The need for this requirement is due to the fact that several common techniques in routing protocols are affected by network size, making them suitable for smaller networks, but completely unviable for larger ones. Further, plenty of protocols are also designed under the assumption that a plethora of different paths between the nodes will be available, and their effectiveness collapses under sparse configurations. The grade scale for this requirement is detailed in table 2.5.

Having defined the criteria for the analytical evaluation of the protocols, the following sections will present them one by one, grouped by their respective protocol family. This chapter is completed in section 2.7.4, which provides a conclusion of the protocol review. A summary of this evaluation is present in table 2.27, which provides a head-to-head comparison of how the protocols measure up in the MANET requirements.

2.7.1 Proactive protocols

Also known as table-driven, this family of protocols favours spending network resources (transmissions, computations, and power) to avoid having an initial delay in procuring a suitable route to any given destination. They generally have one or more routing tables which they can use to produce any route required, and actively work to keep these tables up to date. This strategy often results in the production of a significant amount of control traffic for route procurement and maintenance, thus, they are best suited to support latency-sensitive applications. Their shortcomings are generally around the computational, spectral, and energy resources spent in maintaining their routing tables.

The means through which their routing tables are built and maintained are varied, as are the metrics used for route calculations. These protocols commonly rely on either *distance-vector* or *link-state* metrics to model the network, and estimate the *shortest route* between sources and destinations. They rely heavily on having an accurate vision of the network, so they actively exchange information about network topology at different rates and conditions. These techniques are function as building blocks for protocol design, and

one can see them being mixed and matched all over the literature. The following are some of the most relevant proactive protocols in the literature.

2.7.1.1 Destination-Sequenced Dynamic-Vector (DSDV)

DSDV frames the routing problem between two nodes in MANETs as a distributed version of the *shortest-path problem* [8]. As such, it implements a variation of a known algorithm for this problem called the *Distributed Bellman-Ford algorithm* (DBF) [104]. Its core mechanic is keeping an up-to-date routing table, with entries for all destinations in the network. Each entry in the table has a destination, a preferred neighbour for reaching it, a cost metric (in *hops*), and a route sequence number.

When a node receives a data packet, it extracts the destination identifier from the header. Then it looks it up in the routing table and forwards the packet to the listed preferred neighbour. This process continues in the next node of the route until the data packet reaches its destination. Of course, a crucial aspect of the protocol is how that routing table is constructed and maintained.

All nodes periodically broadcast their entire routing table to their *immediate neighbours* (i.e. within radio range). Upon reception of such message, nodes process the incoming table, adding new routes as needed, and updating existing routes if appropriate. A route is updated when the cost of reaching a destination d through a neighbour n ($Cost_{dn} + 1$) is lower than the current node's cost of reaching that destination. Further, when a node updates a route, it advertises this change to its neighbours in order to disseminate up-to-date routing information through the network. The same updates happen when a node is no longer reachable, due to mobility, interference, or power failure.

Route advertisements can be either updates to the routing table or the full table, as each node deems appropriate. Nodes can also control the rate at which it advertises route changes it detects. These measures seek to prevent *control traffic* from overtaking the network. *Control traffic* refers to the packets produced by a protocol in order to establish and maintain routes. It is differentiated from *data traffic*, which is the actual data the protocol intends to deliver across the network. A deeper exploration of these concepts will be done in chapter 4.

DSDV implements route-tagging as a solution to the route loops often created by DBF. Each node maintains a route sequence number that it includes in all table broadcasts, and that it increases periodically. When a node contemplates updating a route due to received information, it first checks the received sequence number. If it is higher than the stored sequence number, the route is updated regardless of the cost metric. If the sequence numbers are equal, the cost metric is evaluated. And if the sequence number is lower, the message is discarded and the route remains unchanged. This method allows nodes to distinguish between stale and current route data, which aids in the avoidance of

DSDV	
Mobility:	Limited
Mobility breaks links, and link breaks trigger topology updates in this protocol. A few nodes in constant, moderate mobility would be enough to trigger a broadcast storm that would take down the entire network.	
Contention:	Unrestricted
Its routing overhead scales quadratically with the network size, creating more contention. Further, unintended synchronisation events can cause high, localised contention.	
Power:	Unrestricted
Given the large routing overhead of this protocol, its power consumption is dominated by radio transmissions.	
Scalability:	Low
Only supports small network sizes before the control traffic collapses the network, due to its heavy routing overhead.	

Table 2.6: DSDV's MANET grade.

route-loop creation, as well as keeping fresh and usable routes. DSDV's MANET grade is given in table 2.6.

2.7.1.2 Wireless Routing protocol (WRP)

Another protocol based on the DBF algorithm is WRP [28], by Murthy et al. Its overall design and mechanisms are very similar to those of DSDV, but with some added refinements. In order to minimise the time of convergence of the network when the topology changes, it also keeps track of the next-to-last hop to each destination. If a *source* node is transmitting to a particular destination, but the topology is slowly changing (due to mobility perhaps), the source depends on receiving a new control message from the destination in order to update its routing table and adjust its routes. However, by keeping track of this next-to-last hop, which is slightly closer to the source, and can report on the link to the destination due to its immediacy, the source can learn about relevant topology changes from two nodes, and converge its routes faster. This faster convergence makes the temporal route loops that are commonly present in DBF-based solutions shorter. Further, it places great importance on reliable data transmission, using both *passive* and *active acknowledgement (ACK)* at each hop, as appropriate.

When an intermediate *node i* forwards a packet, and it expects the next *node j* downstream to forward it too, *node i* can listen on the radio channel. Upon overhearing *node j* forwarding the packet, it can then confirm the packet was received at the other end. This *passive acknowledgement* is "free", as it does not require further computations or messaging to be obtained. However, a lack of it does not guarantee the packet was not received either, as *node j* might have chosen to queue it, drop it, or *node i* was simply

WRP	
Mobility:	Limited
Mobility triggers topology changes that are flooded through the entire network.	
Contention:	Unrestricted
High overhead that leads to significant contention. Unintended synchronisation of operations among neighbours leads to more contention events.	
Power:	Unrestricted
Power consumption is dominated by its routing overhead and beaconing. A network running this protocol could drain the batteries of all its nodes, without ever transmitting a single data packet.	
Scalability:	Low
Only supports small network sizes before the control traffic collapses the network, due to its heavy routing overhead.	

Table 2.7: WRP's MANET grade.

unable to overhear its retransmission due to packet collisions or interference.

When a stronger guarantee of delivery is needed, protocols can use *active acknowledgements*. This is usually achieved by setting a field in the packet header that indicates to *node j* that *node i* requires confirmation of reception. *Node j* can then unicast a short packet to *node i* providing this confirmation. That ACK packet can either be empty, or contain a given *packet id*, in case many pending packets are waiting for confirmation. Both of these ACKs can be provided at the data link or network layers. WRP's MANET grade is practically identical to that of DSDV (see table 2.7), as their mechanisms are very similar.

2.7.1.3 Source-tree adaptive routing (STAR)

Departing from the traditional DBF approach, Garcia-Luna-Aceves et al. introduced the STAR protocol [40]. In this paper, the authors remark that *proactive* protocols produce notoriously more control packets than their *reactive* counterpart. This created a notion in the community that they were significantly less efficient. The goal of STAR was to design a *proactive* protocol that rivals *reactive* protocols in their efficiency of control traffic. Unlike previous protocols, STAR uses a *link-state* metric to guide its routing decisions, rather than *distance vector*. Protocols that rely on this technique track the properties of virtual links between nodes, such as signal strength, per-hop latency, and packet drop rate.

To achieve its efficiency goals, each node in STAR only keeps partial topology information: A list of links to its immediate neighbours, and a *source tree*. This source tree is the set of preferred links along all possible paths from the current node, to every destination. This partial topology is periodically beaconed using *Link State Update* (LSU) packets. To maximise the utility of these packets, but reduce their size, most LSU packets contain as many link *updates* as possible (partial update). If too many updates are detected for a

single LSU packet, it can be further broken down as needed. Periodically, nodes transmit their full source tree (full update), even if no updates are detected. This is necessary to break potential routing loops, to onboard new nodes joining the network, and specially useful for initialising the protocol.

STAR (ORA)	
Mobility:	Limited
Mobility causes topology changes, triggering LSU updates, which can quickly overtake the network.	
Contention:	Unrestricted
The use of partial LSUs diminishes the protocol's bandwidth requirements, providing slight improvements in its overhead and contention, but not significant enough to improve its grade in this metric.	
Power:	Unrestricted
The reduced overhead also implies reduced power consumption. These changes, however, are minimal to the overall operation of the protocol, which can still drain the batteries from its units without transmitting a single data packet.	
Scalability:	Low
Topology updates are flooded through the entire network, preventing the protocol from supporting anything but small size networks.	

Table 2.8: STAR's (ORA) MANET grade.

STAR (LORA)	
Mobility:	Viable
Topology updates are no longer reported immediately. However, they still need to converge for the protocol to operate correctly, and this convergence is negatively affected by mobility.	
Contention:	Controlled
The tracking of partial topology, use of partial LSU packets, and reduced control traffic improve its contention grade with respect to the ORA version.	
Power:	Unrestricted
With the improved topology update rate, and smaller updates due to keeping partial topology, I consider this metric to be borderline controlled. However, no direct power considerations are made in the protocol to upgrade it.	
Scalability:	Low
On the higher-end of low, close to medium. Even if just partially, it still tracks the network topology, causing the overhead to grow with the node count. Further, node clustering can cause contention hot zones.	

Table 2.9: STAR's (LORA) MANET grade.

On initialisation, the source tree of every node is just the collection of links to its neighbours, as detected by the link layer (a one-hop tree). As nodes receive the first LSU from its neighbours (containing the links to its neighbours' neighbours), it learns about new destinations and choses the best links to reach them, thus growing its source tree (now

a two-hop tree). This process continues until every node learns about every destination, and has produced a tree of routes routed at itself. With this tree at hand, STAR uses Dijkstra's SPF algorithm [104] to calculate the best routes to all destinations.

The efficiency gained from keeping partial topology information (as opposed to full topology) is marginal at best. STAR's authors recognised that the real source of inefficiency from previous DBF-based algorithms is the rate of control traffic updates. To address this, STAR can operate in two different modes. *Optimum routing approach* (ORA) mode, which seeks to update the routing tables as quickly as possible, and produce optimal routes based on any of the *link-state* metrics. And *least-overhead routing approach* (LORA) mode, which seeks to provide viable paths, regardless of their optimality, and produce as little control traffic as possible. The selected operation mode changes the performance of the protocol drastically. Under ORA, link changes to the partial topology are reported immediately (like DSDV and WRP). Under LORA, however, a node only reports changes to its source tree when either a destination is no longer reachable due to them, when a new destination becomes available, or when it detects these changes can induce a routing loop. The authors also recognise that all table-driven protocols up to that point adhered to the ORA model, as they have been adaptation from wired-network protocols. The MANET grade of both versions of the protocol can be found in tables 2.8 and 2.9.

2.7.1.4 Clusterhead Gateway Switch Routing (CGSR)

The protocols discussed so far had what is considered a "flat view" of the network. That is, all nodes are considered to be peers, of equal importance and no real distinction in their roles. Chiang et al. had a different approach in mind, which they introduced in their CGSR protocol [30].

The design goals of CGSR are to reduce routing overhead and spectral pollution, in order to support larger network sizes. The primary mean used to reach this goal is the use of *clusters*: non-overlapping virtual areas that divide the entire network. Each node belongs to exactly one cluster and can only communicate directly with members of the same cluster. If a mobile node crosses cluster boundaries, it switches membership to the new cluster. Conversely, each cluster has a *clusterhead* node, which is elected among its peers, and oversees the cluster's operation. Nodes at the border of a cluster area are called *gateways*, and they coordinate with the clusterheads for inter-cluster communication.

Ideally, adjacent clusters are configured to non-overlapping radio channels to prevent interference. Inside the cluster, all nodes communicate with a protocol very similar to DSDV and flood topology data regularly. The flooding, however, does not cross cluster boundaries, which enables CGSR to support larger network sizes.

All clusterheads keep a member's table, which they exchange with other clusterheads periodically. So when a node wants to communicate with a node in another cluster, its

CGSR	
Mobility:	Viable
Even though the intra-cluster protocol is DSDV, the propagation of changes induced by mobility is capped at the cluster limit. Further, while nodes remain within their cluster, their inter-cluster reachability remains unchanged.	
Contention:	Controlled
This metric is dominated by node density. A high node density in any cluster would collapse it. However, if all clusters are in a workable state, the use of multiple radio interfaces/channels localise the medium contention at the cluster level.	
Power:	Unrestricted
As with other proactive protocols, the power consumption is dominated by the radio operations. CGSR still runs DSDV on top of its own inter-cluster protocol, making it very power hungry.	
Scalability:	Medium
The cluster-level isolation allows the protocol to extend over large geographical areas and supporting a high number of nodes, but it is very susceptible to node-density contention. Further, the clusterhead is a significant bottleneck for inter-cluster communication.	

Table 2.10: CGSR's MANET grade.

clusterhead can calculate the route. These are shortest-path routes made of clusterhead and gateway nodes.

Further, CGSR supports (and almost recommends) the use of multiple radio interfaces per node. This extension allows nodes to belong to more than one cluster simultaneously, and enables gateway nodes to more effectively forward traffic across clusters. Without multiple interfaces, gateway nodes need to switch their configured radio channel dynamically to communicate with adjacent clusters, which the authors admit is not ideal. CGSR's MANET grade is given in table 2.10.

2.7.1.5 Fisheye State Routing (FSR)

Similar to CGSR, the FSR protocol [51], by Pei et al. takes a *hierarchical* (i.e. non-flat) view of the network. FSR is a *link-state* protocol uses hierarchies to control the dissemination of topological data. It is based on the concept of the eye of a fish, which captures a great level of detail near its focal point, and the detail dilutes the further away from it. This concept is implemented with concentric areas/hierarchies around every node. For a given *node n*, the focal point is the hierarchy that contains all nodes one hop away. The next hierarchy consists of all nodes that are at most two hops away. Finally, the last hierarchy consists of all nodes more than two hops away from *node n*.

Nodes in FSR disseminate topological information periodically, rather than as a reaction to changes. These update packets contain the *link-state* data of a given hierarchy only, and use sequence numbers to avoid routing loops. The frequency for which a hierarchy's

data is disseminated is directly proportional to how close it is to the node. Further, these packets are not flooded. They are broadcast to all immediate neighbours, who process the updates and then discard the packet. Thus, every node has very detailed and fresh data regarding the topology close to it, and a less accurate view of nodes further away.

FSR	
Mobility:	Viable
On the higher end of viable, closing to friendly. Mobility triggers no operations, and its greedy-routing-based routes do not break because of it. However, topology information is slow to reach the outer hierarchy. At higher mobility rates, routing would be unfeasible.	
Contention:	Controlled
The protocol's hierarchies severely mitigate the contention created by flooding the topology updates. However, it is still susceptible to high contention in node-dense scenarios.	
Power:	Unrestricted
While the protocol's overhead is heavily optimised when compared to other link-state proactive protocols, all nodes still constantly flood their topology information.	
Scalability:	Medium
Most control traffic is unaffected by the size of the network, as it is localised. However, node density is still a major limiting factor.	

Table 2.11: FSR's MANET grade.

Route calculation in FSR is done with a *shortest-path* approximation. That is, nodes calculate *shortest-path* routes normally with the topological data they have, even though they know it might not be entirely accurate. This approach works because routing decisions are performed *greedily*. At every node, a decision is made regarding which is the best next hop for reaching the intended destination. So while the first hop might not be ideal, the closer the packet gets to its destination, the more accurate data gets to make routing decisions.

Simulation results by the authors show FSR scales well with network size and mobility, even though it's a *proactive* protocol. This performance, however, is subject to some observations. Firstly, the number of hierarchies and their radius need to be adjusted to network size. Secondly, while lower overhead is great, if *link-state* data propagates too slowly, routing will not work. The *greedy* forwarding depends on the assumption that *link-state* data at all nodes is accurate enough to get the packet closer, and not farther away from its destination with every hop. See table 2.11 for more details.

2.7.1.6 Greedy Perimeter Stateless Routing (GPSR)

Another notable *greedy* protocol is GPSR [48], by Karp et al. In this paper, the authors make the observation that the dominant factors in scaling a routing algorithm for MANETs

are the number of nodes, and the rate of change in the topology of the network. In order to remove this susceptibility from their protocol, they chose a *location-aided* approach. By assuming each node is equipped with a Global Positioning System (GPS) device that can inform it of its real-time position, they forego the dissemination of topology data.

Instead, this protocol depends on a location service called the Grid Location Service (GLS [50]). Under GLS, each node choses a set of nodes from the network to act as *location servers* for it, to whom they report their location periodically. The algorithm for choosing them relies on the id of the nodes and concentric hierarchies (much like FSR). At each hierarchical level, three nodes are chosen as location servers, such that the closer one gets to a node, the more location servers can be queried for its position, but even nodes far away from it can learn its location. When *node A* wants to find out the location of *node B*, it uses the same selection algorithm to find B's location servers, and queries the closest one for the location. This enables all nodes in the network to find out the position of every other node at any point in time.

GPSR	
Mobility:	Viable
While the routing protocol is practically unaffected by mobility, a high mobility rate would severely affect the underlying GLS protocol.	
Contention:	Controlled
The contention created from the routing protocol is minimal, however, the GLS still uses a form of controlled flooding.	
Power:	Unrestricted
No considerations are made for power consumption, and GLS performs periodic flooding.	
Scalability:	High
While the number of location servers per node grows with the node distribution, which increases the overhead, it does so at a sub-linear rate. No other operations are affected by node count or density.	

Table 2.12: GPSR's MANET grade.

The state kept by each node to support GPSR is quite small, as they only keep track of the position of their immediate neighbours, which they obtain regularly through *beaconing*. Thus, route calculations become very simple. When a source node (s) wants to send a packet to a destination node (d), it queries the GLS for the coordinates of *d*. Using Euclidean-distance calculations, s determines which of its immediate neighbours is closest to *d*, and forwards the packet to it (*greedy forwarding*). The process repeats at each node until the packet reaches its destination. The protocol is conceptually quite similar to FSR, but it uses location data instead of link-state data. The experiments compare it against the reactive protocol DSR (see section 2.7.2.2), and indicate that GPSR incurs less overhead than DSR in most scenarios. See table 2.12 for further analysis of its performance.

2.7.1.7 Distance Routing Effect Algorithm for Mobility (DREAM)

Another notable *location-aided* protocol is DREAM [37], by Basagni et al. Like GPSR, all nodes are expected to have a GPS device that they use to get their real-time position. Unlike GPSR, though, it follows a more traditional *proactive* approach, in which all nodes periodically flood their own location through the network. Instead of building a connectivity map of the network to make routing decisions, nodes utilise the locations flooded by each potential destination. In order for this approach to work, DREAM relies on two key observations about the temporal-spatial distribution of nodes in the network:

- **Distance effect** - The further apart two nodes are, the slower their positions change relative to each other.
- **Position propagation rate** - The rate at which a node broadcasts its own position to neighbours must be proportional to its own velocity.

The *distance effect* means that nodes further away from a given *node n* do not need to be updated about its position as frequently as those closer. Therefore, two *position update* packets are used. *Short-lived packets*, that have a smaller lifetime and traverse only a short distance away from its origin. And *long-lived packets*, which conversely have a lifetime and are meant to inform nodes further away of the general direction of *node n*. Both types of packets are designed to be very small in order to have short transmissions, which drives radio collisions down and aids in keeping power consumption low.

All nodes keep a Location Table (LT), which is populated by the reception of location update packets. Upon reception of such a packet, an entry in the LT is created or updated with the node identifier, its position, its distance to the current node, and its *direction*, which is calculated via its polar angle. This direction is subsequently used for routing decisions.

When a node n wants to send a data packet to node d, it consults the LT to get the distance, direction, and optional velocity of d. Then, node n estimates a "cone" of angle α rooted at itself and bisected by the line segment of the Euclidean distance between n and d, such that given d's estimated velocity, it is very likely within that cone. Using α, n gets a list of immediate neighbours in that direction, and forwards the data packet to them. The process repeats at each node, *greedily*, until the packet reaches its destination. This forwarding principle is very similar to that of FSR, in which the initial decision is known to be an approximation, with an expectation to improve at each hop.

It is worth noting that, unlike other location-aided protocols, DREAM has a natural resistance to GPS inaccuracies, since the position of a node is not as crucial as its relative direction from other nodes. However, the protocol does depend on nodes knowing their own velocity. If this is estimated by sampling the GPS over time, that may lead to

DREAM	
Mobility:	Friendly
While the routing protocol is practically unaffected by mobility, a high mobility rate would severely affect the underlying GLS protocol.	
Contention:	Controlled
On the lower end of controlled. While most control packets are localised, the rate at which they are transmitted grows directly with mobility rate. However, the short transmission time of these packets lowers the collision window for each one.	
Power:	Controlled
The control messages are designed to be short bursts that consume less power. The periodic beaconing also slows down as needed when the nodes slow down.	
Scalability:	Medium
On the higher end of medium. Most control traffic is localised, so neither node density nor count have a strong impact on the network's performance. Network size is limited by the flooding of position updates.	

Table 2.13: DREAM's MANET grade.

significant protocol misbehaviour. This, of course, can be mitigated by equipping the nodes with accurate velocity sensors, which are more common today than they were when the protocol was published. DREAM's MANET grade is provided in table 2.13.

2.7.1.8 Optimised link-state routing (OLSR)

One of the most prominent proactive protocols in existence is OLSR [54], by Jacquet et al. OLSR goes back to the core premise of proactive protocols by performing aggressive route calculations based on link-state topology information, but making the right design choices to mitigate the shortcomings of this approach. In a nutshell, the protocol calculates *shortest-path* routes by using the network topology information kept at all nodes. The topology information is periodically flooded by *most* nodes (but not all). The key about the protocol is how that topology data is flooded.

All nodes in the network periodically exchange *hello* messages with their immediate neighbours. These hello messages contain a list of the *known* immediate neighbours of those neighbours. Which means all nodes are informed of the topology of their respective neighbourhood up to two hops away. Equipped with this information, each node selects a minimum set of its one-hop neighbours such that all two-hop neighbours are reachable by them. These nodes are called Multi-Point Relay (MPR) nodes. In each *hello* message, nodes indicate which neighbours they have selected as MPRs. The selected nodes keep track of this as well, and the set of all nodes that have selected a given node as an MPR is called its *MPR selector set*. This is useful information because it can be seen as "the set of nodes through which a given node can be reached".

The second type of periodic messages sent by nodes are called *topology control* (TC)

OLSR	
Mobility:	Limited
The MPR nodes create a network spine through which most traffic flows. Link breaks cause by mobility of any of those nodes carries a significant overhead.	
Contention:	Controlled
The use of MPR nodes reduces the propagation of control traffic significantly. However, link breaks of these nodes still need to be propagated, and may even lead to synchronised contention events.	
Power:	Unrestricted
No power-consumption considerations are made.	
Scalability:	Medium
Hello messages are localised, but their rate and size are a function of node density. The number of MPR nodes and their maintenance overhead is affected by network size.	

Table 2.14: OLSR's MANET grade.

messages. These messages contain the nodes *MPR selector set*, and are only forwarded by MPR nodes at each hop. Thus, nodes can construct a connectivity graph of MPR nodes that guarantee all other nodes are reachable by. When constructing routes, these routes contain only MPR nodes.

Further, OLSR seeks to control its overhead by sending control messages periodically, rather than as a reaction to link breaks. These control messages can be split into smaller updates as needed to save bandwidth, and make radio transmissions shorter. Further analysis of OLSR's performance is provided in table 2.14.

OLSR has been very well received by academia and industry through the years. The protocol reached a level of maturity sufficient to be accepted as an internet standard [65], and has several implementations for various operating systems. However, it is not without faults, and has been reported [101] to not work well with high mobility or in scenarios in which the MPR set of nodes changes frequently. It has received several updates through the years to address its shortcomings, culminating in a second major version of the standard (OLSRv2) [123] published in 2014.

2.7.1.9 Better approach to mobile ad hoc networking (BATMAN)

The last proactive protocol to be reviewed in this section is BATMAN [101], by Johnson et al. This is a very interesting protocol for various reasons. Firstly, instead of using simulation as most academic work, this protocol was developed using a large testbed of 300 hardware nodes in the city of Berlin. Secondly, it was created as a direct reaction to the shortcomings of OLSR experimented by its authors when attempting to create a community network. And finally, its design is quite simple, yet sophisticated, and very different from all other protocols reviewed so far in this chapter.

Instead of distance vector, link state, or location-aided techniques, this protocol uses a *stigmergic*[1] approach for route building, inspired by other processes in nature such as ant colonies. The core of its operation is the flooding of *originator* packets. These packets are produced periodically by all nodes, and include the id of the originator node, the id of the last node to forward it, and a sequence number for freshness. Upon reception of these packets, nodes keep count of how many of them they have received for a given destination in the current *sliding window* (configurable time window), and the node through which it was received.

BATMAN	
Mobility:	Friendly
Mobility does not trigger any actions or affect the protocol's operations. A high degree of mobility can make the forwarding decisions at each node less accurate, but this is not a problem as long as they get the packet closer to the node after each hop.	
Contention:	Controlled
On the lower-end of controlled, since the overhead is significant; however, no network conditions cause the overhead to grow or scale.	
Power:	Unrestricted
Control traffic grows quadratically with size, although node density is not an issue.	
Scalability:	Low
Hello messages are located, but their rate and size are a function of node density. The number of MPR nodes and their maintenance overhead is affected by network size.	

Table 2.15: BATMAN's MANET grade.

When data packets need to be sent, instead of calculating full routes, BATMAN uses a *greedy* approach and forwards the packet to the best *candidate node* to reach the intended destination. The best candidate in this case would be the immediate neighbour through which the highest count of originator packets have been received for the intended destination in the last sliding window. This process is repeated at each node until the packet reaches its destination.

This simple design does not specifically account for edge cases in topologies that could create temporary routing loops, or what happens in periods of congestion when originator packets do not flow as freely. Their experimentation in the 300-node testbed comparing BATMAN with OLSR indicate the former has a higher throughput, support for a larger number of nodes, more efficient use of resources and a constant routing overhead (as opposed to OLSR). BATMAN's MANET grade is given in table 2.15.

[1] A nature-inspired mechanism of indirect coordination between agents through an environment.

2.7.2 Reactive protocols

The tradeoffs made by this protocol family are opposite to those of their proactive counterparts, as they favour resource efficiency over packet latency, and thus, only calculate routes for destinations when they are needed. These protocols are characterised by the use of the *query - response* mechanism for route procurement. This mechanism consists of two stages: A route discovery (query), and a route response.

A *route discovery* is the process through which a node finds a suitable path from itself to an intended destination. It is commonly achieved by *flooding* a route discovery packet, which allows the protocol to probe all available and connected paths in the network from the source. The *response* phase of the route procurement operation is triggered once the discovery packet reaches its intended destination. This phase involves the complimentary process of finding a route from the destination to the source, which can be done by recording the path taken by the packet to reach the destination (*route accumulation*), flooding again, or other means. A route between two nodes is considered to be established when the source node receives the *route response* notification from the intended destination, at which point data packets can begin to flow.

Route discoveries are considered expensive operations. They require computational and spectral resources and add additional latency to data packets, which incentivises protocols to maximise their usage for as long as possible. To this end, some protocols rely on *route-maintenance* processes to avoid an early teardown. These processes usually involve the use maintenance control packets to make sure the expected links are still valid. Once a route is no longer needed or it becomes unavailable, these protocols usually initiate a *route teardown* process.

Similarly to their proactive counterparts, the main difference among reactive protocols is how they procure, maintain and teardown routes. The design of these phases can vary significantly, leading to different levels of resource usage and performance. The main drawback on these protocols is around the route discovery process. While the initial latency cost introduced by the discovery process is amortised through the extended use of the route, some applications might not be able to tolerate it. Further, concurrent flooding-based discovery process can lead to high medium contention and network congestion, both of which are highly undesirable. Thus, the suitability of these protocols to any given application will depend on how judicious they are in mitigating both these factors. The following are some of the most relevant protocols in this family.

2.7.2.1 NP

One of the earliest published reactive protocols is NP [25], by Corson et al. The protocol itself is one of the most basic implementations of how reactive protocols operate, as described in section 2.7.2. It operates based on two principles: nothing is calculated

before it is needed, and shortest-path routes are not that important, since all routes can break at any moment due to mobility or a changing environment. It also uses link-state measurements for route calculation, and it keeps track of the state of links to its immediate neighbours (which is unassigned by default). Further, this state is assumed to be provided by the link layer protocol in place, and does not define any packets to probe its neighbourhood.

The protocol defines three different control packets: query (QRY), reply (RPY), and failure query (FQ) packets. When a node needs a route to a destination, it floods the network with QRY packets that include the id of the destination. All nodes processing a QRY packet check first if they have a route established to the destination. If they do, they construct a RPY packet and send it over the link through which it received the QRY, pruning the route discovery. If not, they forward the packet over their unassigned links. As the QRY packets are forwarded, the link states to those nodes are changed to downstream (upon confirmation of reception, provided by the link layer). When a QRY packet reaches its destination, a RPY packet is constructed and sent to the originator. There is no assumption about the directionality of the links, so the RPY can flow through a different path altogether, changing the direction of the links as it flows. When the originator receives a RPY packet, data packets can start flowing.

FQ packets are broadcast by a node when it has lost all of its routes due to link failures. It is used to signal to other nodes that they should no longer route through it, but it can also be replied with a RPY to indicate to that originator node that it does in fact have remaining downstream neighbours.

NP	
Mobility:	Viable
Mobility does not trigger any operations. However, the routes have a strict sequence, and having two paths for the same route (downstream and upstream) means mobility in either side can break the route, even if packets are only flowing on the other side.	
Contention:	Controlled
The reactive nature translates in naturally lower overhead in most scenarios, reducing the window for contention. However, if multiple routes are being queried simultaneously, the concurrent flooding can be problematic, and only gets worse with size.	
Power:	Unrestricted
No power considerations are present.	
Scalability:	Medium
Query operations are flooded, which limits the size to which the network can grow.	

Table 2.16: NP's MANET grade.

The protocol is simple in nature, but covers all the basics of a reactive protocol. While it does not make attempts to create shortest-path routes, they do mention that the shortest path is often the route created, as it emerges naturally. Finally, while the RPY packets by

intermediate nodes might reduce the overhead of route discovery in some cases, it can also be a false positive for a route that is no longer available and in the processes of winding down. NP's MANET grade is provided in table 2.16.

2.7.2.2 Dynamic source routing (DSR)

The DSR [27] protocol, by Johnson et al. is one of the most influential reactive protocols published, and one that is often used as a baseline comparison. At large, the protocol follows the general *query - response* model for building routes. However, it makes adjustments through the entire protocol operation to achieve its two design goals: Lower routing overhead as much as possible, and support high levels of mobility among its nodes. The low overhead objective translates into operations that extract as much data as possible from all packets, to avoid making unnecessary transmissions. The high mobility one translates into two things: Knowing the data collected is only temporarily valid, and that hop-to-hop transmissions need validation.

When a node *orig* needs to send a data packet to a given destination *dest*, it first checks its *route cache*. Entries in the route cache are indexed by destination, contain the sequence of nodes needed to reach it, and expire after a threshold. The *route cache* is filled by standard *query - response* operations, but also by forwarding operations. When a route has been established, and data packets are flowing, the full route is embedded in the header of the packet. Thus, all forwarding nodes can extract this information to update their routes to all downstream nodes. Optional optimisations to the protocol enable it overhear all packets, and update its route cache based on the information contained in them.

In *route discovery*, DSR uses a technique called *route accumulation*. When a route discovery packet is flooded, all forwarding nodes append their id to the *route record* contained in the packet. The *route record* contains the exact route followed to reach any node receiving it. When the discovery packet reaches *dest*, if it assumes bidirectional links, it could send back a reply by *winding back* the *route record*. Otherwise, it either checks its own *route cache* or starts a route discovery process back to *orig*, in which it appends the route record from *orig* to *dest*. The route is established when *orig* receives the reply message with the route record it can use to reach *dest*. Just as with the NP protocol, the discovery process can be pruned by an intermediate node with a valid route to *dest*.

DSR does not specify route maintenance packets. However, given the volatile nature of routes in a setting with mobility, it performs its own maintenance via *ACKs* at every hop. Depending on the particular network configuration, it can use link layer, passive, or active *ACKs* as appropriate.

Finally, DSR encourages *piggy-backing* operations as appropriate. This technique refers to the act of appending data or control packets to another packet that had to be sent, in

DSR	
Mobility:	Viable
Mobility does not trigger any operations. However, the routes have a strict sequence.	
Contention:	Controlled
On the higher end of controlled. No control packets are used other than those for route queries (which are flooded) and eventual route breaks. Some overhead might be introduced by not guaranteeing route optimality, although the shortest path route is often the first one to be discovered.	
Power:	Unrestricted
While the overhead is low, no considerations are made to lower the power consumption.	
Scalability:	Medium
Flooding of query packets limits the network size.	

Table 2.17: DSR's MANET grade.

order to avoid a second transmission. This can be used both for control purposes, such as checking the end-to-end validity of a route by piggy-backing on a data packet. Or accelerating data transmissions by appending a TCP SYN packet into a route discovery packet. However, most of these optimisations are application dependant, and need to be configured and tested for validity. DSR's MANET grade is given in table 2.17.

2.7.2.3 Temporally-ordered routing algorithm (TORA)

Shortly after DSR was published, came TORA [34], by Park et al. Based on the NP protocol (section 2.7.2.1), but with the objective of minimising the control packets used in its operation, and with a different approach than DSR. TORA also uses the *query - response* technique and keeps track of the directionality of links to form a DAG between source and destination. However, it adds a new parameter: Each node has a *height* value associated with it.

This new height parameter changes the way the entire protocol operates. Making an analogy with the way water flows, data packets in TORA only flow downstream, from a higher height to a lower height. The height values are calculated and set on route discovery, and route failure operations. When a route discovery process succeeds, the appropriate values for all nodes in the route are calculated and transmitted. Those calculations take into consideration existing routes such that, all routes flowing downstream from a node have lower heights. In order to make sense of these multiple topological changes, all packets are timestamped, using either a logical clock, or a relying on synchronised hardware clocks. When a node receives multiple updates messages, it orders them by timestamp in order to arrive at the right height.

According to the authors, this protocol is best suited for large and dense mobile networks. When link failures occur, only adjacent nodes are notified and react accordingly, in order to keep mobility viable. Further, the protocol is not concerned with route

TORA	
Mobility:	Viable
Mobility does not trigger any operations. However, the routes have a strict sequence.	
Contention:	Controlled
A flooding-based query process is used initially to establish routes, but it can later use longer routes to new destinations (when possible) without the need of flooding again. This, however, adds to the overhead of the non-optimality of paths.	
Power:	Unrestricted
Low overhead, but not a single consideration to power consumption is made.	
Scalability:	Medium
The main route establishment mechanism is flooding, which limits its scalability. It is on the higher side of medium, though, as most of the other control operations are fairly localised.	

Table 2.18: TORA's MANET grade.

optimality (e.g. shortest path), given that its assumptions of mobility mean that all routes are short-lived, regardless of their length. TORA's MANET grade is provided in table 2.18.

2.7.2.4 Associativity-based routing (ABR)

Another protocol from the early days, but one that is quite interesting, is ABR [35], by Toh et al. One of the things that makes this protocol quite interesting, is not only the techniques it uses, but its design goals. Its primary goal is to produce routes that are as *stable* as possible, arguing that a longer but *stable* route is preferable to a shorter but *unstable* one. In this context, *stable* refers to the estimation of a route's prolonged ability to forward traffic as needed. This prediction is informed by the quality of all links, the workload, and the *associativity* of each node in the route.

The *associativity* of a node is a metric of its relative position stability with respect to its neighbours. This associativity is calculated with the use of "ticks", which are periodic link-layer messages that simply contain the id of the sender. All nodes keep track of the ticks they receive from all neighbours. The higher the tick count for a node over a recent period of time, the higher its associativity is considered.

The protocol's route discovery process uses a *query - response* technique with *accumulation*. The *broadcast query* (BQ) packet used for this purpose is flooded, and at each node accumulates: Node id, associativity ticks, relaying load, and link propagation delay. The associativity ticks and link propagation delay are both in relation to the previous node in the route, and relaying load refers to the number of packets in the current node's queue. Because of flooding, the intended destination can get multiple copies of the BQ packet that followed different routes. After receiving the first BQ packet for a given route, the destination will wait for a period of time for further packets. Finally, it will choose the more stable route among those discovered a send a route reply (REPLY). The

REPLY packet traverses back the discovered route, activating the relevant nodes. All other discovered routes remain inactive.

ABR	
Mobility:	Viable
Bordering on friendly. The overhead cause by mobility is constant, moving nodes are avoided when forming new routes, but they can still use other nodes for transmissions. However, the routes formed still follow a strict sequence, and after being selected, those nodes can move and break the routes without any mitigation.	
Contention:	Efficient
On the lower end of efficient. The route establishment mechanisms, which avoid overloaded nodes, lead to a natural spacial distribution of the traffic which should heavily aid in avoiding contention. However, the ticking mechanism can cause problems on dense portions of the network.	
Power:	Unrestricted
No power-consumption considerations are made. The ticks can drain the batteries of all nodes without data being transmitted.	
Scalability:	Medium
Route discovery relies on flooding, which does not scale. Further, the tick mechanism can cause high contention in dense scenarios.	

Table 2.19: ABR's MANET grade.

By means of the associativity metric, nodes that are less likely (in theory) to move and thus break the route are chosen. While this would translate to lower overhead due to fewer route breaks, it is unclear if the associativity is a good metric for mobility prediction. Further, while most link-layer protocols try to avoid poor links (just like ABR), few look into spreading the routing load. By including the relying load in the metrics, nodes that are ideally placed for multiple routes are not overloaded. This measure could also spread the radio contention, leading to fewer missed transmissions. ABR's MANET grade is provided in table 2.19.

2.7.2.5 Location-aided routing (LAR)

Proactive protocols are not the only ones that looked into *location-aided* techniques, and LAR [38], by Ko et al., is one of the most prominent examples. As with all previous protocols, it is not just the techniques used that matter, but the intended goal to reach with them. In the case of LAR, it aims to reduce the propagation of route request packets.

LAR works in a standard *query - response* manner, but instead of flooding the entire network looking for a route to the destination, it only floods the *request zone*. The *request zone* is a superset of the *expected zone*, which in turn, is a geographical zone in which the source node expects the destination to be in. This estimation is dependant on the last known position and velocity of the destination. If the destination D was last seen at

position L at time t_0 with a velocity v, then at time t_1 (with $t_1 > t_0$), D is expected to be within a circular area centred at L and of size $\pi(\Delta v)^2$, where $\Delta = t_1 - t_0$. Given this, the simplest algorithm to calculate the request zones would be a rectangle that fully contains the *expected zone* of D, as well as the position of the source node.

When a route request is initiated, if the source node knows the last position and velocity of the destination, a *request zone* (its coordinates are embedded in the packet); otherwise, the request is flooded. All intermediate nodes calculate if their current position in within the zone described in the packet, and if so, they forward the packet. Once the request reaches its destination, it emits a reply that contains its current position and velocity, to use for future route calculations.

LAR	
Mobility:	Viable
Routes created have a strict sequence, and high mobility rates break the expected-zone heuristic.	
Contention:	Controlled
On the lower end of controlled. If the expected-zone heuristic is working, the contention of the flooding can be significantly lowered. However, that also means rediscovery of routes, which creates further overhead and contention.	
Power:	Unrestricted
No power-consumption mitigations are in place. However, having no passive traffic, this is on the higher end of unrestricted.	
Scalability:	Medium
The main route discovery mechanism is still flooding, which does not scale well.	

Table 2.20: LAR's MANET grade.

Technically, this strategy to mitigate the overhead of the *query - response* process would alleviate one of the main weaknesses of reactive protocols. However, it has a significant weakness, as its usability and the performance of the network are at odds. The initiating node has no way to know the intended destination's position and velocity before having flooded the network at least once. If a future discovery process is done, it now has the relevant data to make an efficient route discovery. However, the data are only useful if the time between discoveries is short enough that it is still relevant, and that would mean the previous route was short-lived. If the time between discoveries is too long, the data is useless to estimate the current position of the destination, and the expected zone might encompass the entire network. The authors discuss the possibility of extensions to the protocol to passively listen for all control packets in order to learn the positions of nodes across the network, but there is no guarantee of the freshness of the data. As such, the usability of the heuristic to reduce the route request propagation remains in question. LAR's MANET grade is listed in table 2.20.

2.7.2.6 Ad hoc on-demand distance vector (AODV) routing

And lastly, the most cited protocol in MANETs: AODV [43], by Perkins et al. A departure from his previous work (DSDV), Perkins recognises that a protocol with frequent control message flooding does not scale well with network size, and that having a route for every node almost always exceeds the needs of most nodes. However, he still holds the core belief that route acquisition latency should be kept at a minimum, and he seeks to implement that with this protocol.

As most other protocols in this family, AODV operates in standard *query - response fashion* and utilises a distance-vector routing metric. For route discovery, it floods *route request* (RREQ) packets until they reach the intended destination. As packets flow down the network, intermediate nodes create *upstream* routes (i.e. back to the originator of the route request). Route table entries at each node keep track of the next hop to that destination, the sequence number (used in the same way as DSDV), and a precursor list, which refers to all known nodes that use the current node to forward traffic to the destination in question. All entries are only valid while active (i.e. actively forwarding packets) and expire after an *active timeout* period.

Once an RREQ reaches its intended destination, an RREP packet is unicasted upstream back to the originator. As this packet traverses down the network, intermediate nodes perform the complementary actions to the RREQ and create downstream routes to the destination. Once the RREP reaches the source node, all intermediate nodes have appropriate route entries for upstream and downstream traffic, and the data flow can commence.

An important aspect of the route procurement process of AODV is its distance-vector metric. Every time a control packet is processed, the node checks the sequence number of the originating node to make sure it only processes fresh packets. Further to that, though, it checks the distance of the route in question, and if a shorter route to an existing destination emerges, it updates its table to always use that shorter route.

AODV implements various optimisations for performance gains. The route discovery process has a *ring search* feature to control the flooding of RREQ packets. Every RREQ packet will have a *time to live* (TTL) set in its header (in number of hops) that starts small, and decreases every time it is forwarded. If an RREQ fails, the source will retry eventually with an expanded ring radius. While this feature can reduce the overhead of an RREQ, it also goes counter to the objective of reducing route acquisition latency. AODV also implements route discovery pruning, and any intermediate node with a valid (i.e. active) route to the destination can produce an RREP, and optionally notify the intended destination of this action. Finally, AODV utilises periodic HELLO messages for route maintenance. While it supports passive and active ACKs to detect link breaks, it also takes a proactive approach to link-break detection in order to mitigate its effect. Each

AODV	
Mobility:	Viable
Its mobility potential is limited by the strict sequencing order of the routes it creates.	
Contention:	Controlled
On the lower end of controlled. It uses flooding for route discovery, although it can be somewhat mitigated by the search ring. It also uses HELLO messages to maintain routes, which adds to the overhead.	
Power:	Unrestricted
No power-consumption mitigations are in place.	
Scalability:	Medium
Flooding does not scale well with size.	

Table 2.21: AODV's MANET grade.

node keeps track of the last time it has received a packet for all active routes. All nodes with at least one active route, periodically broadcast a HELLO message to its neighbours. Thus, even if an active route has paused data traffic temporarily, nodes can verify the link is still available. If a node has not received a packet from an active link (HELLO or otherwise) for a given period of time, it assumes the link is broken. At which point, it can attempt to repair the route with a partial RREQ, or report the breakage to all relevant nodes with a route error (RERR) message. AODV's MANET grade is given in table 2.21.

AODV has been immensely popular over the years. It has become one of the default baseline protocols to compare against and matured enough to be standardised into an RFC [71]. Further, relevant extensions have been created for it, such as AODV-PA [67], which leverages the *path accumulation* technique from DSR. The interest in AODV remains to this day, with an experimental RFC currently under review for its second version (AODVv2) [150].

2.7.3 Hybrid protocols

The third and final family are the *hybrid* protocols, which, as the name suggests combines aspects of both *reactive* and *proactive* protocols. This combination seeks to either mitigate the shortcomings of one protocol family, enhance its effectivity, or both. Unlike proactive and reactive, this definition is slightly looser, as no particular set of behaviours suffices to qualify or disqualify a protocol as *hybrid*. Given this loose definition, *hybrid* protocols do not exhibit common behaviours that make them better or worse suited for a given set of scenarios.

It could be argued that most hybrid protocols follow a pattern of being either mostly *proactive* or *reactive*, while adopting a technique from the other family to mitigate an identified weakness of its own. An interesting question in this space would be what to compare these protocols against, or what the right baseline is. Without a definitive answer,

most protocols tend to compare against what suits them better and highlights their own strengths. The following protocols are some of the most notable in this category.

2.7.3.1 Zone routing protocol (ZRP)

The first hybrid protocol for consideration in this section is ZRP [31], by Haas et al. This protocol is loosely based on the concept of hierarchical protocols like CGSR, but implemented in a "flat" topology to avoid the overhead of clusterheads and gateways. The design goal of the protocol is to support networks that cover a large geographical area, and support any number of mobility velocities.

In order to achieve its goals, ZRP implements the concept of a *zone*, which for any node, it contains all nodes r hops away or less from it. All nodes keep track of the full topology of their respective zones by running a truncated version of DSDV within it. Control messages do not propagate outside the zone to prevent the scalability issues of DSDV.

When a node *src* needs a route to a node *dest*, it first checks within its zone and uses the existing route created by the zone protocol. If it is not in its zone, then ZRP switches to a *query - response* approach. Node *src* sends a query packet to the nodes at the periphery of its zone (nodes r hops away). If *dest* is found in the zones of the periphery nodes, a reply is sent back to *dest*. Otherwise, those nodes forward the query packet to the periphery of their own zones, and so on until *dest* is found. Through the forwarding process, route accumulation is used to learn the forwarding route, and it is included in the payload of the response packet.

ZRP	
Mobility:	Viable
Mobility triggers topology-update packets, but their propagation is limited to the zone.	
Contention:	Unrestricted
Bordering on controlled grade, this protocol suffers from the contention issues of both proactive and reactive protocols. Unlike CGSR, this protocol does not implement the use of separate radio channels/interfaces to isolate the contention of neighbouring zones, so the truncated DSDV still produces massive amounts of contentions. Further, while the query/response mechanism does not fully flood the network, it still does it partially.	
Power:	Unrestricted
No power-consumption considerations are made.	
Scalability:	Medium
The slimmed-down version of flooding and controlled DSDV allow for support of larger networks than DSDV. However, its performance can collapse with a high node density.	

Table 2.22: ZRP's MANET grade.

Their experiments compare ZRP to DSDV, and given that all nodes run a truncated version of DSDV, it, of course, shows much less overhead than it. And while the zone concept prevents the control packets from the *truncated DSDV* to propagate through the entire network, it is unclear if that is enough cover large geographical areas. Further, running a variation of DSDV, which implies broadcasting all changes to network topology, seems to run counter to supporting high mobility rates. ZRP's MANET grade is listed in table 2.22.

2.7.3.2 Zone-based hierarchical link state (ZHLS) routing

The ZHLS [41] protocol, by Joa-Ng et al. expands on the concept from ZRP. The protocol can be summarised as a hierarchical version of a common Link-State Routing (LSR) [78] protocol that is location aided. It depends on the core concept of a *zone map*, which divides all geographical locations in non-overlapping zones with distinct ids. By using a GPS device and having a copy of the zone map, all nodes can identify the zone they currently belong to.

General operations of ZHLS are quite similar to ZRP: proactive routing is done within a zone, and reactive routing is used across zones. Within each zone, all nodes run a variation of an LSR protocol. Nodes periodically broadcast a *link request* packet, and get a *link reply* from all immediate neighbours. These packets contain the node's id and zone id. With it, each node builds their LSP, which is a list of the ids of their same zone neighbours, and the zone id of their different zone neighbours. These LSPs are periodically flooded within their respective zones, in such a way that all nodes in a zone have full connectivity information of their zone, and what neighbouring zones are reachable through it.

After the initialisation phase of a zone (i.e. when all nodes in the zone have the full node LSP data), all nodes build a *zone LSP*. This zone LSP is simply the list of all the reachable neighbouring zones of this zone. Once the zone LSP has been created, all gateway nodes (i.e. nodes that neighbour with at least one zone) broadcast this zone LSP. This zone LSPs are forwarded across zones, in such a way that eventually all zones know the full inter-zone connectivity. Further to this, the shortest-path algorithm is used to create both intra-zone and inter-zone routes.

When a node needs to send a packet outside of its zone, it sends a *location request* packet to all other nodes specifying the node id. The packet will be forwarded at the inter-zone level to all other zones, until it reaches the zone containing the intended destination, and a reply is generated. When the zone id is learned, data packets containing both zone and node ids can flow using the inter-zone and intra-zone routing tables.

Just as with ZRP, ZHLS evaluates the performance of their own protocol against the pure version of their intra-zone protocol (LSR). And just as with ZRP, the results favour ZHLS. And while the inter-zone routing is flexible as long as the inter-zone connectivity is

ZHLS	
Mobility:	Viable
Mobility is limited by its routes' strict sequences.	
Contention:	Controlled
If the zone map contains sufficiently small zones, the overhead from running the LSR protocol should be contained. Still on the lower end of Controlled, though.	
Power:	Unrestricted
No considerations are made in this regard. Further, the constant control overhead that happens even in the absence of data packets can drain all nodes.	
Scalability:	Medium
Depends heavily on the size of the zones. Very dense zones will cause the LSR protocol to collapse.	

Table 2.23: ZHLS's MANET grade.

not broken, it is unclear how the performance or overhead of this protocol fare against a purely reactive protocol. The MANET grade for ZHLS is given in table 2.23.

2.7.3.3 Core-extraction distributed ad hoc routing (CEDAR)

The CEDAR [45] protocol, by Sinha et al., is quite different from the previously analysed protocols. The design goal for CEDAR is to provide routes for all nodes that can offer *quality of service* (QoS) guarantees. In order to meet this goal, it runs a core-extraction algorithm that forms the basis of all routes.

In a nutshell, the core-extraction algorithm consists in finding a (ideally minimum) dominating set among all nodes to act as the core. A dominating set refers to a set of nodes such that they can form a connected graph among themselves (no isolated nodes) and that all other nodes of the network are neighbours of at least one core node. Upon startup, the first stage of the protocol is to extract this core. Once the core nodes are selected, they periodically flood the state of their links, testing for loss rate, bandwidth and stability. From these links, they select the good links (according to the configuration), and flood that data to other core nodes. The bad links, on the other hand, are only advertised locally, such that they can be used if needed, but that nodes further away do not rely on them. The core-extraction algorithm is rerun periodically.

All non-core nodes are assigned a *dominator* node. This *dominator* node is a neighbour that is part of the core, and all traffic to and from the given node will flood through its dominator. The *dominator* node for a node can be switched at a later point in time if link-quality changes, or the *dominator* becomes overloaded. When a node n needs a route to a node *dest*, it queries its dominator ($Dom(n)$). $Dom(n)$ will then proceed to do a *query - response* search for the dominator of *dest* ($Dom(dest)$). This route search through the core may produce multiple routes, and the selected route will be the one that suffices for the bandwidth requirements specified by n.

CEDAR	
Mobility:	Limited
The protocol depends entirely on the existence of a stable CORE. Link breaks from any of those nodes trigger recalculations, and if several of them are on the move simultaneously, the protocol becomes inoperable.	
Contention:	Controlled
The QoS properties of this protocol mean that it can avoid building routes through overloaded nodes. However, topology changes can trigger massive amounts of synchronised control traffic.	
Power:	Unrestricted
No power considerations are present.	
Scalability:	Medium
Depending on the core-extraction algorithm used, the decisions can be either fairly local, or extend to the entire CORE. The larger the network, the larger the core needed, and the more complex its calculation becomes.	

Table 2.24: CEDAR's MANET grade.

While the proposed solution may certainly work to offer QoS guarantees, it is easy to see how in an environment with relative mobility that produces multiple link breaks, the core-extraction algorithm might easily overtake the traffic from the network. CEDAR's MANET grade is provided in table 2.24.

2.7.3.4 AntHocNet

AntHocNet [76], by Di Caro et al. is protocol inspired by natural processes, that relies on a stigmergic data collection strategy (like the BATMAN protocol). As the name indicates, this protocol is inspired by the way ants do path exploration and leave pheromones on their way to signal to other ants what path to follow. Unlike the other protocols analysed in this section, which consistently run a *proactive* process in order to later make better reactive routing decisions, AntHocNet is primarily *reactive*.

As other (primarily) reactive protocols, AntHocNet makes no calculations or transmissions until it is part of an active route. When a node *src* requires a route to a node *dest*, it starts by sending *reactive forward ants* (RFA). These "ants" are a route discovery packet that is flooded through the network. RFA packets use path accumulation, but at each node they perform further calculations. Specifically, they calculate a goodness value at each hop, which is an estimation of how long it takes to relay a packet through the recently passed link. This estimation uses the number of packets in the queue, as well as the measured transmission delay. When an RFA reaches *dest*, it becomes a *backwards ant* (BA), which traverses back through the accumulated path, once again updating the goodness values of the links in the backward direction. Due to the nature of flooding, multiple RFA may reach *dest*. Unlike other algorithms, AntHocNet accepts all RFAs,

which start their traversal back to *src* (the *multipath* technique).

AntHocNet	
Mobility:	Viable
On the higher end of viable. While routes still have a sequence, it can have multiple options at each hop (due to its multipath nature), which makes the impact of mobility on them lessened.	
Contention:	Efficient
On the lower end of efficient. The protocol is held back in this regard by the use of flooding, which can become expensive with multiple concurrent route discoveries. However, its multipath technique also spreads the contention around the route branches. Further, the PFA can create additional route branches as a data flow progresses, that ultimately might prevent a route from breaking, and triggering a new route discovery.	
Power:	Unrestricted
No considerations regarding power consumption, although given the low overhead, it should be close to controlled.	
Scalability:	Medium

Table 2.25: AntHocNet's MANET grade.

Once the *query - response* process finishes, a route tree seeded at *src* and finishing in *dest* is created. At this point, *src* can start sending data packets to *dest*. However, the data packets do not follow a deterministic path. Given that at each node, there can be n different paths to follow to *dest*, a data packet will choose a path i with probability $G_i^2 / \sum_{i=0}^{i=n} G_i^2$. This ensures data packets heavily favours those paths with a higher goodness value, but leave open the opportunity to spread the traffic among all available links, providing some degree of load balancing.

Further, for every (configurable) N^{th} data packet transmitted, *src* will transmit a *proactive forward ant* (PFA). This PFA is a route maintenance packet that is transmitted to recalibrate the goodness values of all nodes in a path, as well as potentially create new alternative paths. It has probabilistic-forwarding rules similar to data packets, and updates the *goodness* values at every hop. The difference in its forwarding rules are that it does not use the squared value of the goodness (to make it more likely to select alternative paths), and it always has a small probability to be broadcast, rather than unicasted downstream. When a PFA is broadcast, it may reach nodes that are not part of that route. When that happens, it probes them for an existing route to *dest*. If it finds one, it traverses back to *src* to add this newly discovered path to the existing route to *dest*.

When evaluated against AODV, AntHocNet has higher delivery ratios, supports larger workloads and a lower delay for route finding in non-trivial scenarios. The MANET grade for AntHocNet is given in table 2.25.

2.7.4 The battle of the protocols

Having reviewed the most significant MANET routing protocols from the literature, one can finally compare them head to head in table 2.27. The requirement grades of each protocol are presented with symbols for readability purposes. These symbols are explained in table 2.26, and for all requirements, a higher grade is more desirable.

Symbol	Meaning
▼	The lowest grade in the requirement.
●	The intermediate grade in the requirement.
▲	The highest grade in the requirement.

Table 2.26: Protocol grade symbols.

Protocol	Family	Year	Mobility	Contention	Power	Scalability
DSDV	Proactive	1994	▼	▼	▼	▼
NP	Reactive	1995	●	●	▼	●
WRP	Proactive	1996	▼	▼	▼	▼
DSR	Reactive	1996	●	●	▼	●
CGSR	Proactive	1997	●	●	▼	●
TORA	Reactive	1997	●	●	▼	●
ABR	Reactive	1997	●	▲	▼	●
ZRP	Hybrid	1997	●	▼	▼	●
DREAM	Proactive	1998	▲	●	●	●
LAR	Reactive	1998	●	●	▼	●
STAR (ORA)	Proactive	1999	▼	▼	▼	▼
STAR (LORA)	Reactive	1999	●	●	▼	▼
AODV	Reactive	1999	●	●	▼	●
ZHLS	Hybrid	1999	●	●	▼	●
CEDAR	Hybrid	1999	▼	●	▼	●
FSR	Proactive	2000	●	●	▼	●
GPSR	Proactive	2000	●	●	▼	▲
OLSR	Proactive	2001	▼	●	▼	●
AntHocNet	Hybrid	2004	●	▲	▼	●
BATMAN	Proactive	2008	▲	●	▼	▼

Table 2.27: Head to head protocol comparison.

An easy pattern to pick up from this table is the fact that most protocols make little effort to mitigate their power consumption. While is doubtful the authors are unaware of the importance of this metric, its effects are not as immediate as those of the other metrics, making its relegation natural. However, the reduced medium contention associated to protocols with lower overhead is quite likely to provide more power efficient operation. It is also worth mentioning that a range of protocols exists whose design goal is to be power efficient, but they are not in scope for this review.

Another interesting question raised by these results is the challenge of mobility. Not only do very few protocols excel at this requirement as the protocol table indicates, but the review shows that this core aspect of MANETs can collapse even highly sophisticated protocols. Protocols like OLSR and CEDAR that build complex logical structures to achieve their goals can be very effective in the right circumstances. But they become inoperable in the presence of mobility, as the overhead of recalculating these structures overtakes the network.

The last pattern from these results that I present to the reader is that of tradeoffs. We can clearly appreciate in this table that no protocol excels at more than one thing. While this analysis is subjective, it is fairly evident after this review that the focus on some features that makes a protocol excel in some metric requires they sacrifice another. The design choices that lead to prioritising one metric over the other should be aligned with the applications intended for the protocol. However, a general-purpose MANET protocol should be able to operate reasonably well in most scenarios. That is to say, it should be able to delivery packets under varied conditions and not fully collapse when one of the MANET properties is dialed up. Therefore, a general-purpose routing protocol should aim to be more balanced in all MANET requirements.

In chapter 4 I will present my own routing protocol called Reactive Gossip Routing, whose design is informed by the lessons learned from the analysis in this section. However, before presenting my contributions, we take one last detour in the literature. It should be clear to the reader by now just how important fault tolerance is to MANET routing protocols. Naturally, I am not the first researcher to consider this, so in the last section I will present a review of specialised MANET protocols focused on providing fault-tolerance to the network.

2.7.5 Fault-tolerant protocols

As we reviewed the protocols in the last few sections, we considered them under the premise I introduced in section 2.7 of promoting fault-tolerance through fault reduction. That is to say, that a protocol would experience fewer faults the more it adapts to the properties that cause them. However, that is not the only approach one can take to tackle the problem of reliability in MANET routing protocols. Many researchers devoted their efforts to designing solutions that would make their protocols resilient in the face of errors or prevent them altogether. There are two general strategies that researchers take to fault-tolerant protocols: Topology control, and logical redundancy. Each of these strategies works on different premises, making them suitable for different scenarios.

2.7.5.1 Topology control protocols

Fault-tolerance by topology control is predicated on the notion that routing faults in MANETs come from the lack of diverse paths that is induced by undesirable topologies. These protocols then aim to improve the effectivity of routing by removing the root cause. Most of them focus on creating topologies that have plentiful diverse links between its nodes, as this naturally creates diverse paths through the network. Naturally, MANET topologies are transient, as they are affected by mobility, RF interference, obstacles from the environment (vehicles, people, etc.), or even by power failure in nodes. Thus, topology control algorithms face the constant challenge of handling these scenarios.

Basu et al. [75] present two such algorithms based on node mobility, which aim to reconfigure an autonomous robot network in such a way that all nodes have links to at least two different nodes. This configuration ensures that all nodes can support the failure of at least one link without becoming disconnected (e.g. they are 2-connected). Their first algorithm is called *Contraction* and works based on the premise of a naïve heuristic. By having nodes disseminate their position through link-state updates, they can calculate the centroid of the network and iteratively move towards it until their connectivity needs are met. This simple but effective algorithm is better suited for networks running a link-state routing protocol like OLSR or HSLS, and in which nodes are equipped with localisation devices (e.g. GPS). However, it limits the mobility range of each node and requires a supervisory layer to balance mission and connectivity needs.

Their second algorithm called *MakeBiconnected* is more sophisticated. It addresses the problem by targeting the removal of *cutvertices* from the network, nodes whose connectivity is such that it can partition the network if it moves or fails. This strategy involves moving specific nodes closer to the cutvertices to produce redundant links in that portion of the network. This reposition is done iteratively while being careful not to produce new cutvertices.

Aside of the algorithms themselves, there are two interesting things to note in Basu's work [75]: Their choice to implement both algorithms as heuristic-based solutions, and the application domain (mobile robots) that makes their solutions viable. The heuristic nature is a consequence of the problem's difficulty, as finding a minimum-cost k-connected subgraph is an NP-hard problem [32], which is why many researchers in this area turn to approximations to make this problem more tractable. The application domain focus, while necessary for their case, is quite constrained for the many different applications of MANETs. However, mobility is not the only way to control the connectivity graph of a wireless network.

A different approach for topology control relies on a variable transmission range of the nodes via power adjustments. Li et al. [70] follow this approach in their work, where they present a model to calculate the required radio range for all nodes in a MANET such that

the network is *k-connected* with high probability. A k-connected network is one in which all nodes can withstand the failure of up to $k-1$ of their links and remain connected to the network. This model is based on the number of nodes (N) in the network and predicated on them having a random distribution (Uniform or Poisson) at all times. The resulting network topology offers multiple node-disjoint paths between any two nodes in the network which provide an implicit degree of fault tolerance. An optimisation of their model is also presented which allows the calculated subgraph to remain k-connected while maintaining fewer links than the general model.

Li and Hou [88] tackle the same problem with a different strategy. They seek to maximise the *lifetime* of the network (e.g. lower power consumption and medium contention) by finding a minimum spanning tree (MST) that connects all nodes using the lowest aggregate power consumption possible. They recognise the need for fault tolerance in their goal, as such an MST would be liable to partition the network at the cutvertices. To address this, they propose their Fault-Tolerant Local Spanning Subgraph ($FLSS_k$) protocol, which produces a k-connected MST. It achieves this by having nodes exchange HELLO messages periodically that they use to adjust their transmission power. Each of these messages includes the geographical position of the node, its maximum transmission power, and the list of its current neighbours. Nodes use this information locally to maximise their own reach while minimising power consumption and meeting their k-connectivity goals.

The ideas behind these topology control algorithms are interesting, since they seek to provide the network conditions necessary for a routing protocol to operate with minimal faults. However, their applicability is limited by physical constraints. If there are not enough nodes in the network to provide the required redundant paths, these protocols might not converge at all. The same applies to the node distribution, as there is a limit to range of the node's radios, and they might not be able to provide the desired topology.

2.7.5.2 Logical redundancy

A different approach to fault tolerance comes from the creation of logical constructs that can withstand errors. Unlike the work presented in section 2.7.5.1, entries in this section are actual routing protocols that aim to be fault tolerant by design. The aforementioned constructs are based on techniques such as fault *detection* and *estimation*, and *redundancy*. While these techniques are common, it is their correct use and combination that can lead to significant improvement in a routing protocol's effectivity.

Protocols that rely on *fault estimation* utilise the data they have gathered from the system to make decisions that they expect would lead to fewer routing errors. An example of such work is the End-To-End Estimation-Based Fault Tolerant (E^2FT) routing protocol [80]. Authors of this work postulate that designing an efficient fault-tolerant routing algorithm with low overhead and delivery-rate guarantees is an NP-complete

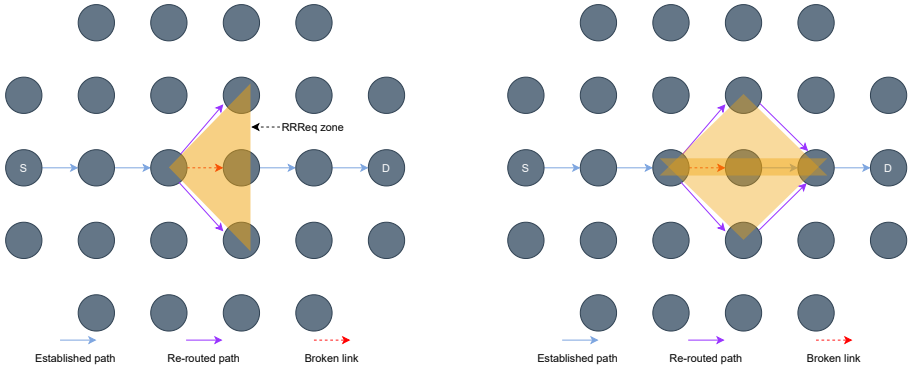
problem, and therefore a solution can only be approximated. This on-demand protocol consists of two stages, *route estimation* and *route selection*, and assumes nodes can acquire multiple paths to a destination through mechanisms similar to those of DSR [27]. During route estimation, a source node iteratively transmits multiple data packets through all of its routes to a destination until the delivery rate of at least one route meets the required delivery rate guarantee. Afterwards, the route selection process chooses the shortest path among those that meet the delivery rate guarantees for all future transmissions to the destination and drops the others.

Their simulations show delivery rate to be superior to DSR and on par with and unspecified "blind" multipath routing protocol. However, their protocol is inferior to DSR in terms of overhead but superior to the blind multipath protocol, likely a consequence of their expensive estimation period. Lastly, they disclose their evaluation scenarios only cover static networks as they acknowledge the effectivity of their protocol suffers under mobility.

Other estimation-based work includes the WEFTR [89] algorithm, which uses a binomial distribution and an adjustable learning parameter to make the estimation phase smoother; the BFTR [80] protocol, which addresses several of E^2FT's issues by discarding the estimation phase in favour of a rolling estimation process during a route's lifetime; and the LAFTRA [118] algorithm, which also employs a rolling estimation procedure but enforced by a learning automata (LA) that avoids routes with faulty nodes. All of these protocols make considerable estimation improvements but have their own shortcomings, such as expensive estimation phases, false-negative node classifications, and significant susceptibility to mobility.

Protocols that focus on *fault detection* accept the fact that errors will occur during their operation, and instead focus on detecting them as quickly as possible and mitigating their impact through different means. An example of such work is the Local Self-Recovery Routing (LSR) protocol [93] protocol. This on-demand protocol based on DSR [27] uses passive and active ACKs to constantly monitor for faulty nodes/links. Upon fault detection, it buffers all pending packets for the route in question and starts a local redirection around the fault.

This is achieved by defining a cone-shaped repair zone with its apex at the detecting node and its bisector passing through the faulty node, which is then broadcast in a route repair request (RRReq), as depicted in figure 2.8a. Nodes receiving this request reply with an alternative path to the destination if they have one or forward the RRReq if they are located within the repair zone (see figure 2.8b). This process effectively propagates the RRReq around the faulty node/link, incurring only local overhead. To calculate if they belong to the repair zone, nodes can either rely on location services (e.g. GPS) or use relative distance estimations such as Time of Arrival [68]. In simulation results, it

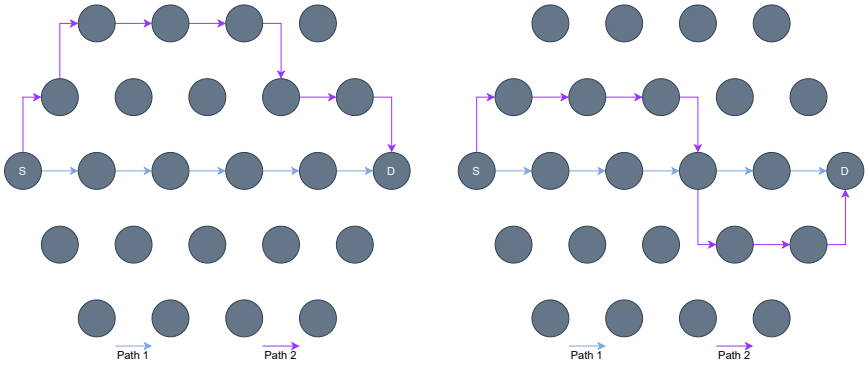


(a) Route redirection process starts. (b) Process continues around the fault.

Figure 2.8: LSR route redirection process

outperforms DSR in both throughput and overhead. However, both protocols are known to be susceptible to faults through stale cached routes, a problem that is amplified linearly with the rate of topology changes.

Lastly, the most common technique to improve the fault tolerance of a routing protocol is to add *redundancy* to it in such a way that it could tolerate different types of failures. The most common way this redundancy is implemented is through the addition of multiple redundant paths to a route (multipath routing). However, redundancy can also be implemented at the packet level by having the source node transmit each packet more than once, in case one of these replicas fails to be delivered. While many mainstream protocols feature a multipath variant, it should be noted that the incorporation of this technique into a routing scheme does not guarantee fault tolerance in itself, as it can be implemented in many ways to different effects.

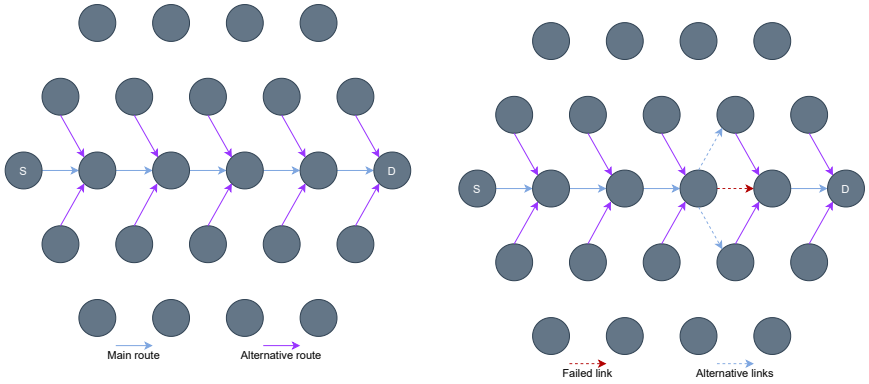


(a) Node-disjoint AOMDV route (b) Link-disjoint AOMDV route

Figure 2.9: Possible AOMDV routes

This diversity in implementation and results can be appreciate in analisying AOMDV [57] and AODV-BR [49], two multipath variants of the popular AODV protocol. While both seek to add fault-tolerance to their operation by implementing multipath routing, their design choices vary significantly. On one hand, AOMDV modifies the AODV

route discovery process to establish two link-disjoint paths (figure 2.9). The source node selects the shortest of the two paths as the main one and forwards data packets through it until a link error is detected, at which point it immediately switches to the backup path without having to wait for a route repair or a new route discovery. These alternative paths not only provide fault tolerance, but also promote spatial reuse of the medium by spreading out the flows in the network. AODV-BR on the other hand adds multiple backup links adjacent to the main path (figure 2.10), effectively creating multiple backup paths for it. Thus, should the main path become disrupted at any level, the route has many available alternatives to leverage.



(a) The fish-like structure of AODV-BR routes.

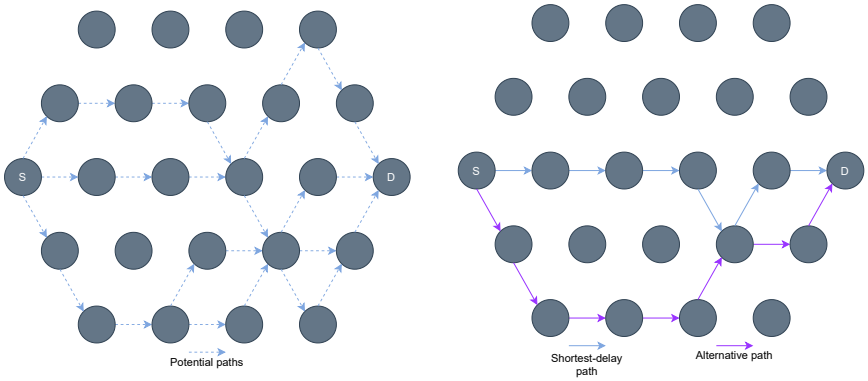
(b) Use of alternative-route links.

Figure 2.10: AODV-BR routes.

While both variations improve on the fault-tolerance of their base protocol, their design choices strongly affect their tradeoffs. AOMDV gains spatial reuse on top of fault tolerance to one link break, but both paths are as susceptible to disruption as regular AODV paths. AODV-BR routes can tolerate multiple link breaks but features a high overhead, as the support links are only used temporarily while the source node starts a new full route discovery.

The effect of design choices between multipath implementations can also be observed between SMR [55] and MP-DSR [58], both variations of the DSR protocol. Similarly to AOMDV, SMR improves the route discovery of its base protocol by establishing two paths in the same operation. The first query response to reach the source node is always deemed to be the primary path, while the secondary path chosen is the one that is maximally disjoint from the primary path (figure 2.11). Further, intermediate nodes no longer reply with cached versions of routes to the destination, but they can now forward the same query packet more than once, provided each one arrives through a different link. The source node purges all routes from its cache that used a broken link after it learns of it, and switches data traffic to the secondary path to the destination if it remains available.

MP-DSR [58] on the other hand seeks to add an alternative path to the destination to



(a) Available paths after searching. (b) Selected paths.

Figure 2.11: SMR route discovery.

every intermediate node of a route, rather than just to the source. Similarly to SMR, the first query response for a given route discovery is established as the main path. However, the destination only responds to subsequent queries for that route if they provide a link-disjoint path to an intermediary node of the primary path (see figure 2.12). In this regard, this protocol is quite similar to AODV-BR, as they both make every node resilient to a one downstream link failure. However, the number of alternative paths created depends on the topology, as it might not always support link-disjoint paths to the destination for every node of every route.

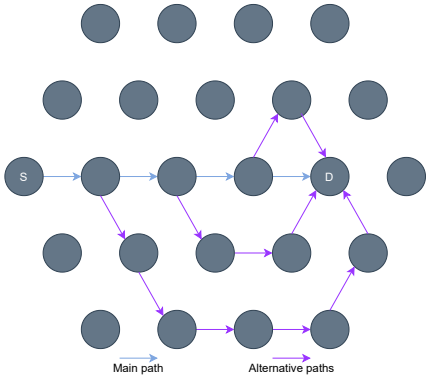


Figure 2.12: MP-DSR route.

Both of these variants improve on the fault-tolerance of DSR to a different degree and with different effects. SMR's modest improvements come not only from the backup path but from discarding the query responses from intermediate nodes, given that their freshness can not be guaranteed. It pays for this with a more expensive route discovery process that could escalate into a broadcast storm, given that query packets can be forwarded multiple times. MP-DSR boasts more significant improvements to its fault tolerance, given that its routes now feature multiple redundant paths which it utilises for as long as it can. However, this gains are contingent on whether the topology can accommodate the backup

paths or not.

In conclusion, we can appreciate that while specialised solutions for MANET fault tolerance exist in the literature, they are far from infallible. They all bring different guarantees that are predicated on having the network and application meet specific conditions. Further, many of them are as susceptible to mobility as some of the routing protocols reviewed in section 2.7. This discovery not only informs the design of the routing protocol I present in chapter 4, but the experimental conditions I use to evaluate the fault-tolerance of various protocols. The next chapter describes the experimental procedure and tooling I developed to this end.

Chapter 3

MeshSim: A new approach to MANET protocol research

An essential aspect of all research endeavours is the choice of methodology and tooling appropriate for the problem at hand. While I introduced my research problem and the state of art around it in chapters 1 and 2 respectively, adopting their tooling and methodology is not a clear decision. The source of this problem is that experiments in wireless networking research are notoriously difficult. The breadth of techniques available for these experiments go from purely analytical to complicated and expensive deployments, all of which are met with mixed results.

Consequently, research in this area has long been done predominantly using simulation, which seemed like an ideal balance between convenience, cost, and experimental quality. However, as I mentioned in chapter 1, the work of Kurkowski et al. [83] raised serious concerns with this practice. In this paper, they discovered that the results of the majority of the peer-reviewed papers they considered in their study were simply not credible due to poor experimental practices. Naturally, this is something I wished to avoid in my own research.

This led me to consider what is it that made a MANET project successful? The PRNET [5] and SURAN [16] projects were considered very successful and form the basis of military communications technology to date. While it would be easy to attribute this success to their massive funding, not all military MANET projects had the same fate [144]. But what the SURAN project did have, was comprehensive tooling and experimental procedures that relied on multiple paradigms for success. This realisation motivated me to look deeper into the matter, in order to make an informed choice for my own research.

3.1 MANET research approaches

The reason why the tooling issue is so significant in MANET research as opposed to other sub-areas of computer science is the nature of MANETs themselves. Ideally, when doing systems research, one would measure activity of a real-world system deployment in order to characterise it, understand it, and postulate theories about it. For research in wireless communications, however, this can be prohibitively difficult, as any sizeable deployment would involve dozens of participants, equipment, development of specialised tools, etc.

A common trade-off in this field is to forgo a degree of experimental *fidelity* in exchange for more achievable experiments and higher experimental control. In this experimental context, fidelity refers to how close is the chosen abstraction to the actual phenomena (e.g. a real-world deployment), and control refers to the ability to isolate and manipulate the variables affecting it. Thus, one can characterise the *research approach* taken in a project by the choices made in the axes of experimental fidelity and control.

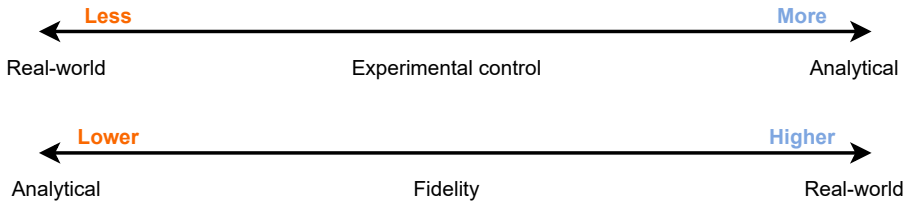


Figure 3.1: Research approach axes.

The scale of these axes is relative and their limits are defined by other research approaches, as seen in figure 3.1. Intuitively, one can see that the highest score in the fidelity axis would be the observation and measurement of real-world phenomena. Consequently, observation of real world phenomena scores the lowest in experimental control, as any intervention in it would interfere with the results. On the opposite end of both axes, we find theoretical formulations of a problem, likely given by a set of mathematical expressions. This approach provides full control of the experimental settings but lacks severely in fidelity, as it sits on top of multiple layers of abstractions of the natural phenomena.

When placed orthogonally, these axes form a Cartesian plane that provides a useful visual representation of the trade-offs provided by the most common research approaches for MANETs, as seen in figure 3.2. The properties of each approach carries different implications and might make them more or less suitable for different research endeavours. In the following sub-sections, I will provide a more detailed view of the most common approaches used in MANET research.

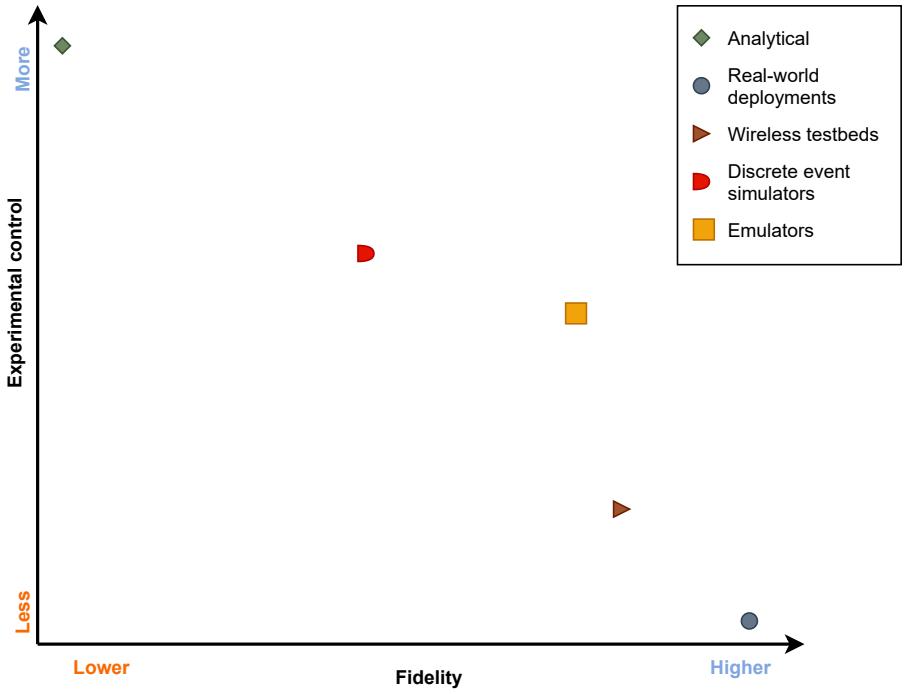


Figure 3.2: Comparison of the most common MANET approaches.

3.1.1 Wireless testbed approach

Wireless testbeds are experimental setups in which an array of configurable wireless nodes are used to test various layers of a networking stack. This research approach favours a high-fidelity abstraction close to real-world deployments by using the same communication links, but with a higher degree of control. Experimental fidelity suffers, however, as the patterns of communication and topologies are synthetic and may not be representative of of natural phenomena. Further, even sophisticated testbeds struggle to scale scale well, and both mobility and natural background noise are very difficult to incorporate.

Testbed nodes vary significantly, ranging from the use of commodity hardware (e.g. laptops with Wi-Fi cards), to custom-built computing units with *software-defined radios* (SDR). The capabilities and complexity of these setups grow proportionally with cost. The simplest testbeds use commodity hardware, are composed of a few nodes only, and are used for simple network tests. These setups can usually be managed by a single person. On the opposite end, the most complex setups can have hundreds of nodes and be used for complex hardware and firmware evaluation, requiring a full team of people to manage. These expensive nodes offer advanced capabilities such as support of variable radio frequencies, power control schemes, and different radio-modulation techniques.

The benefits of using this approach are varied. But mainly, they provide high-fidelity experiments within a controlled environment. This is a massive advantage over real-world deployments as vehicles for research, not only because testbeds provide a faster experimental turnaround, but because they allow for the creation of specific test conditions.

For instance, a sophisticated testbed could be configured in such a way that most of the radio links established are unidirectional. While this edge case would be very rare to encounter in a real-world deployment, it would be very valuable to test a network stack under such extreme conditions.

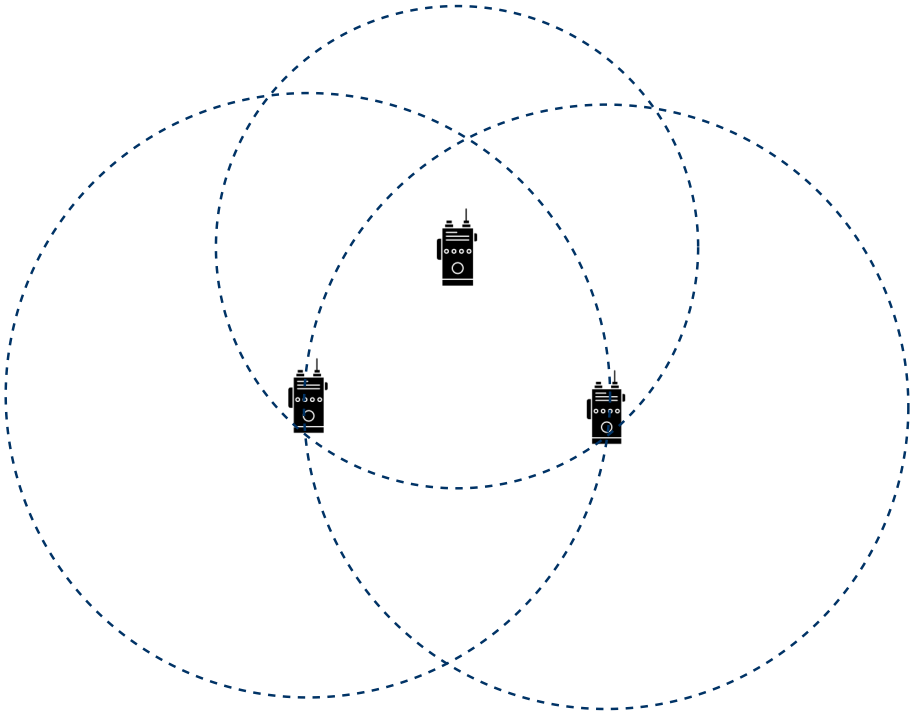
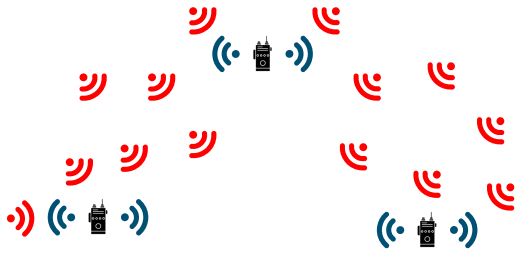


Figure 3.3: Radio ranges in wireless testbeds overlap significantly.

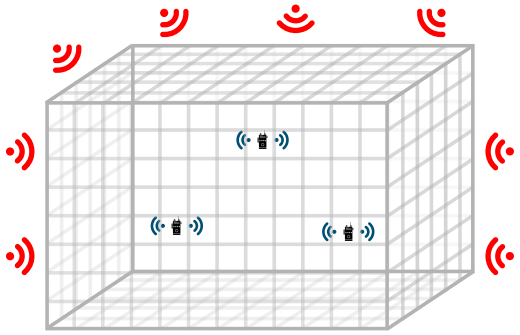
On the other hand, their primary drawbacks are that they can be costly and time consuming to set up, they are subject to unexpected sources of experimental error, and can generally be difficult to operate. In particular, topology control is a major complication. Setups that utilise nodes equipped with SDRs can use power-control and radio-modulation techniques to generate various topologies, but these units are costly and complex to use. But when using commodity hardware, nodes have to be spread out geographically in positions that lead to the desired connectivity lest their radio ranges overlap (see figure 3.3). Given that the ranges of these radios tends to be between tens to hundreds of metres, it becomes impractical to have testbeds with routes that are multiple hops long. A way to address this is to configure the radios to lower transmission power in order to reduce their transmission range. This, however, has its own issues, as some effects of radio propagation and interference are hidden at this lower power output, which would reduce the experiment's fidelity. Of course, topology control becomes even more difficult when introducing mobility.

Another major issue to consider in testbeds is radio interference. This interference can be external (figure 3.4a), which can come from radio signals in the same band, or even from adjacent radio-bands. This type of interference is particularly troubling when using

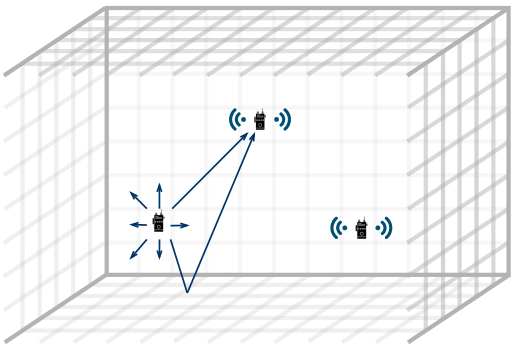
Wi-Fi radios, as they operate in the ISM band [137], which is shared with other scientific equipment and even household appliances. External interference can be mitigated by using devices like a Faraday cage (figure 3.4b), but the radios inside the cage can produce interference themselves via radio multipath propagation [69] (figure 3.4c).



(a) External interference in wireless testbed.



(b) Faraday cage keeps external interference at bay.



(c) Internal interference caused by signal multipath.

Figure 3.4: Grid topology depictions.

Lastly, it is important to consider the drawbacks of testbeds when compared against real-world deployments. While they both suffer from interference, these problems affect testbeds disproportionally due to their design objectives. Real-world deployments assume the environment is completely out of their control, and thus seek to collect information and internal state of the nodes during their deployments, without external concerns. The purpose of testbeds, however, is to introduce experimental control to high-fidelity experiments, which forces them to account for interference in some way. Regardless of their drawbacks, testbeds are invaluable tools for MANET research, and something that all researchers would likely use to some degree if cost and complexity where no objection.

3.1.2 Simulation-based approach

Simulation-based research refers to the use *simulation* software to design the experiments that validate or disprove a theory. These simulators are tools that abstract several

aspects of a complex system to produce informed speculations about its behaviour under different conditions. Simulators provide users the means to describe different scenarios and conditions in which the entities of the system interact. Network simulators in particular also propagate packets between the nodes over simulated radio channels, to which all nodes react according to their configuration of protocol. These interactions are recorded and aggregated into high-level metrics that dictate the success level of an experiment.

Simulations are incredibly helpful in the research of complex systems whose state is too difficult or expensive to set up, execute, and analyse. A large-scale MANET deployment experiment (real world or testbed) consisting of hundreds of nodes would be expensive, difficult set up and execute, and almost impossible to replicate. But using a simulator, not only is this easily achievable, but the experiment can be executed multiple times with varying simulation parameters to examine the boundaries of the system. Further, simulations can be written to test scenarios with complex interactions or a very large scale, both of which would be unfeasible in deployments. They provide a convenient and cost-effective way to test the interactions within a complex a system.

Of course, simulations have their own shortcomings as well. Simulators abstract a system (natural or otherwise) in a way that it becomes describable by some language (programming, markup, etc.). Thus, the simulation is only an approximation of the real phenomena, and the quality of the approximation is dependent on multiple factors. This approximation can also make the results questionable depending on the fitness of the abstraction, and the multiple experimental configurations simulations can have.

Thus, the simulation-based MANET approach seeks to gain a significant degree of experimental control and the capabilities of expressing larger and more complex test scenarios, at the expense of experimental fidelity. No longer are the nodes of the network independent physical devices establishing wireless links over radios, but coordinated and simulated entities whose behaviour is abstracted away as a set of properties and rules. Lastly, not all simulations are made equal. The abstractions used to describe the components of a simulator are varied and alter the fidelity and control of a simulation experiment.

3.1.2.1 Simulation techniques

There is a large range of techniques for executing simulations, such as discrete event, fixed and variable time steps, and real-time simulation. Time-step-based simulations update the state of all entities in the system according to its configuration once every time step. The amount of *work* an entity can perform in a time step is calculated at the start of a simulation and depends on its configurable granularity, which can be dynamic. An example of such simulator is Matlab Simulink [176], for which various modules exist to perform ad hoc networks simulation [138].

Real-time simulators offer the capability of simulating a physical system in which the simulation time matches clock time. That is to say, events occur at the same rate as they do in the actual physical system. This property is favourable in terms of fidelity, as this real-world clock ensures no artificial coordination effects are induced on the simulation. But likewise, it is also unfavourable in terms of experimental control, as a real-time computer program is then subject to the conditions of its environment, and can be affected by it. These types of simulators are usually favoured by hardware designers, and rarely used in MANET research. Examples of these simulators include RT-LAB [178] and Simulink [23].

Lastly, there are discrete-event simulators, which are very popular among network researchers, and the most used type of simulator in the MANET literature. They operate similarly to time-step-based simulators, updating the state of all entities at each sequential simulation step, but they abstract away *time* and use simulation steps. Instead of executing the code of each entity in real time, propagating their inputs and outputs as appropriate, all entities produce events. These events are ordered linearly according to the simulation step in which they are produced, and executed in that order. All events executed during the same time step are considered to happen concurrently, even if the calculations are performed minutes apart in real life. This abstraction of time allows researchers to pause simulations and examine intermediate states of its entities, not just the final results.

A very important property of discrete-event simulators is that their output is deterministic. Since the processing of events occurs in logical, linearisable events, they should not be subject to interference by their environment. If the simulator controls for all variables, like random number generators (RNG) seeds, running the same experiment should always produce the exact same output, regardless of when it is run, and in what environment.

These simulators also abstract away the collective computing power of all the simulated entities. To execute a large-scale MANET (hundreds to thousands of nodes) in real time, a very powerful computer would be needed to equate the collective computing capacity of all nodes being simulated. Lacking this computing power, any of the nodes could drag in their computations and responses, significantly altering the fidelity of the simulation. But with discrete event simulators, each entity executes its logic when it is given processing time by the machine running the simulation. Of course, this means that running such simulation on a system with lower computing power would take longer, but it is still possible.

3.1.2.2 Simulation abstraction levels

Another major difference between simulators (even those that use the same technique) is the MANET abstraction level at which they operate. This abstraction dictates the granularity at which the components of a MANET are represented, and in turn, the nature of the rules used for the interactions of such components. For instance, a simulator can choose to abstract at the packet level, focusing on how packets are built and moved around

the network. In such simulators, the users are given the means to describe the rules for moving the packets. These rules can be used to write routing protocols, or various other networking services. Examples of such simulators include the Maisie language [22], Parsec [36] (Maisie's successor), and earlier versions of NS [26][47]. In particular, the Maisie and Parsec can be used not only for describing routing rules, but as building blocks to build simulators with different abstractions.

A different (and interesting) level of abstraction is that of components, rather than entities. In such simulators, no single entity of a MANET system is abstracted, but rather, several building blocks that can be used to describe nodes, packets, radio channels, and other relevant components. These types of simulators favour the use of Graphical User Interfaces (GUI) to chain together the building blocks, given their design-oriented nature. Notable simulators that follow this model are Matlab Simulink [176] and OMNET++ [102].

Finally, I consider simulators that make a domain abstraction at the node level. These simulators focus on providing the means to define and configure nodes, which are independent from each other and can only communicate via the means they use for interacting in the real world (i.e. wireless radios). The models built for these simulators then describe the properties of each node, which govern their interactions and behaviour during the simulation. Inside the simulator, nodes are provided with virtual resources such as network interfaces or protocol stacks. When the simulation is executed, these virtual resources are instantiated to concrete types according to the configuration of the simulation. Some of the simulators that employ node-level abstractions are the *Maryland Routing Simulator* (MaRS) [18], the Global Mobile Information System Simulator (GloMoSim) [39], NetSim [85], my own MeshSim [140], and the hugely popular NS-3 [92].

3.1.3 Emulation-based approach

One of the primary drawbacks of simulations, is their low fidelity. Through the introduction of its many abstraction layers, it is very difficult to verify that the obtained results are actually representative of the phenomena being modelled. However, one technique that can mitigate this problem is *hardware emulation*. Emulation seeks to implement the full functionality of a hardware device in software. This can be achieve in several ways, such as creating virtual devices that load the same firmware modules as their hardware counterparts.

The incorporation of virtual devices into a simulation not only improves its fidelity level, but it changes the research approach taken with it. Since researchers no longer have to be concerned with modelling the devices, they instead focus on simulating the propagation of input signals and environmental factors of the simulation. Thus, not only can emulation increase the fidelity level of a simulation, but can produce a cascading effect that raises the quality of the system as a whole.

Naturally, this approach is not without its own drawbacks. Since device firmware is not designed as a discrete-event component, its incorporation to such simulators can be difficult. Firstly, the integration of a component that runs in real-time to a system with controlled time means that very careful synchronisation is required for it to even begin working. Secondly, this integration might create simulation artefacts in which all operations by an emulated device are completed either too slowly or too fast, depending on how fast time propagates in the simulation. Further, the fact that these virtual devices run in real time also means that they require real-time computing power to perform its operations, raising the computational requirements of discrete event simulators. These reasons make emulation a technique better suited for use in real-time simulators. However, much care is required even then because a consequence of the real-time nature of the system is that host machine running the simulation must be able to satisfy the aggregate computing demands of all components, lest the simulation starts to drag. Example of MANET emulators include NetSim [85], Mesh Network Lab [180], MANE [105], eMANE [111] [110], CORE [95], and NS-3 [92].

3.2 Introducing MeshSim

MeshSim (short for Mesh Simulator) is a real-time MANET simulator with emulated nodes and testbed capabilities I developed to aid in my research, and published a paper on it in 2017 [140]. Its unique properties make it an attractive option for researchers working on MANET routing protocols, which is why I open sourced it under the MIT license. Its latest release and source code are available at Github, and the version used for the experiments of this thesis is archived at the university's repository. The motivation for building this platform came to me early in my PhD while reflecting on the conclusions reached by Kurkowski et al. [83]. While it was clear to me that I wanted to produce experimental results that were credible and representative of the phenomena being evaluated, it was not clear this was feasible just with careful use of simulators.

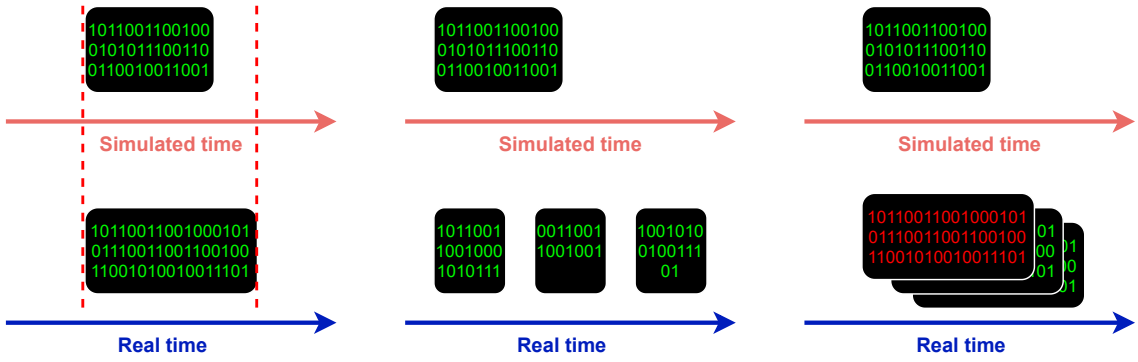
My first intuition was to compliment simulation data with other experimental techniques for validation, such as testbeds. But I quickly concluded that a better approach would be to design, implement and evaluate my protocols in a feedback loop of simulation and validation such as the one used by the SURAN [16] project. After considering the available platforms I could use for my work, it seemed that NS-3 with its emulation framework [86] was the one closer to my requirements. However, this simulator has two properties that ultimately made me decide to implement my own platform:

1. **Complexity level** - NS-3's design has evolved over the years to support a myriad of use cases between wired and wireless networks with different levels of simulation, emulation, and virtualisation. While this makes the software very versatile, it also

makes its configuration complex and error prone. But more than that, it makes extending it a challenging endeavour that is only aggravated by the fact that it is implemented in C++ [173]. This is problematic because C++ is both well known for its speed and by how easy it is to produce erroneous code with it even by professional developers, due to its memory model [44][168][174].

2. **Discrete-event nature** - As discussed in section 3.1.2, a simulation is a reduction of a real-world phenomenon to a set of key properties and subsystems that describe its behaviour. These abstractions are what allow us to produce computationally-feasible models of natural systems that have seemingly infinite complexity and run them in a finite amount of time. However, not all abstractions are the same. While NS-3's discrete-event nature allows it to control simulation time, serialise events, and make simulations reproducible, it also subtracts fidelity from the model at the code-execution level in different ways (see figure 3.5), which is problematic for my use-case.

   When comparing the performance of routing protocols in a MANET simulator, we can consider the simulated environment as a static component and the protocol in turn as the payload. Since all protocols are evaluated under the same conditions, what we are comparing is each protocol's operation. Thus, higher fidelity of the execution of a protocol's tasks takes precedence in this use-case. Of course, all subsystems of a simulation are important and one also has to ensure the environment meets a minimum set of requirements. But we always have to make trade-offs in system design, and NS-3 did not provide the right ones for me.



(a) Not aligned. (b) OS scheduling. (c) Concurrency effects.

Figure 3.5: Differences between simulated and real-time task execution.

Naturally, my claims of how discrete-event simulators substract fidelity from the model at the code-execution level warrant further explanation. Firstly, as illustrated in figure 3.5a, the simulated time it takes to complete a task within a model can differ from its real-world counterpart, and it is not trivial to calibrate each task to representative values. Secondly,

this synchronisation issue in execution times is aggravated by the OS scheduler of real-world tasks. Even detailed simulation calibration can not account for the execution delay a real task might suffer when fragmented by the scheduler (figure 3.5b). The effects of this problem on simulation fidelity range from mild (extended running time of a task) to severe (partial state updates in shared resources). Lastly, we have the issue of task concurrency (figure 3.5c) and its associated resource contention. In real-time code execution, a task might run flawlessly the first time and suffer delays or failures the next, due to concurrent tasks competing for the same resources (file system, IO devices, kernel structures, system buses, cache misses, etc.). Access to some of these key resources could (and should) be captured in a simulated model, but it would be nearly impossible to capture the entire complexity of these interactions and their outcome. Thus, I chose to implement my own simulation platform that addresses these issues.

MeshSim aims to fill the gap in the tooling state-of-art I just described by providing a simulation platform with emulated nodes, high-fidelity in task execution, and testbed capabilities. It achieves these goals by running the same networking stack on both simulated and real nodes, running at real time, which it accomplishes by having each node bind its communications to either a real or a simulated wireless NIC depending on configuration. This binding occurs at initialisation, afterwards, nodes enter the same main loop in both cases, oblivious to the nature of their NIC.

MeshSim addresses the concerns I listed above regarding NS-3 in the following ways:

- **Complexity and extensibility** - Instead of trying to provide all the features that NS-3 does, MeshSim focuses on the use-case of supporting MANET routing protocol research. This prioritisation allows for a lean software design that is easy to use and to extend. Further, MeshSim is implemented in Rust [127], a memory-safe programming language created by the Mozilla foundation that is aimed at facilitating system's level development. The choice of programming language used to implement the platform and write new protocols for it is quite relevant for various reasons. Firstly, safe Rust code is very resilient to the most common bugs found in C and C++ code (memory safety errors) [172][181]. Secondly, its type system allows programmers to detect complex thread-safety issues at compile time [139]. Adding all of this up, MeshSim enables researchers to write MANET protocols expediently and efficiently, sparing them of the significant debug time that comes with existing tools, which in itself, is a contribution to the field that was sorely needed.

- **High-fidelity task simulation** - MeshSim's real-time nature provides a very high level of fidelity in task simulation. In fact, the code for every task in a protocol is not simulated, throttled, or manipulated at all. It is the exact same code that runs on device deployments, unaltered, and one could even use the exact same binaries if

the processor architecture of the simulation machine and the chosen devices match. However, this is still not perfect fidelity, as all nodes in a MeshSim simulation would run on the exact same OS, with the exact same configuration, and the exact same hardware (e.g. those of the host machine), which of course is a very unlikely condition in a real-world deployment.

- **A sturdier experimental cycle** - As I stated at the beginning of this section, early in my PhD I was preoccupied with coming up with a robust experimental procedure similar to the one used by the SURAN project. My solution to this problem was the creation of the *Calibration-Simulation-Validation* methodology that I described in section 1.4.1. I bring this up again, however, because this methodology was enabled by MeshSim and its real-time nature. It is these properties which allow me to capture the difference of a protocol's performance between simulated and real-world deployments with the confidence of knowing that the only thing that changes is the environment, and not the code.

In the next sections, I present the design challenges involved in building MeshSim as well as its limitations, followed by an overview of the system architecture, and finish with the experimental results used to validate it.

3.3 System design challenges

As one would expect, designing and implementing a system with MeshSim's objectives has a unique set of challenges at different levels. Some of these challenges are a direct result of the features I want MeshSim to have, and others a consequence of design trade-offs. Likewise, I address these challenges at different levels of the solution stack by means of design, implementation, or experimentation changes. The following subsections detail the high-level challenges I faced during MeshSim's development.

3.3.1 Simulation artefacts

A simulation artefact refers to an unintended effect a simulator might have on the models it runs. That is to say, an imperfection in the system that does not allow it to reproduce a model perfectly, that alters the results, or that introduces some bias to it. From a design perspective, one of the most important aspects of building a simulator is to ensure there are as few simulation artefacts present in the system as possible.

Two things are important to mention in this regard. Firstly, simulation artefacts and fidelity are tightly intertwined, as a lack of fidelity in any part of the system might introduce an artefact. Secondly, because of the previous point, there are no simulators

without artefacts, as that would require a simulator with perfect fidelity. Just as with simulation abstractions though, not all artefacts affect a system with the same magnitude. Consequently, the simulation artefacts that emerge from a design's trade-offs can be treated as system design challenges to overcome. The following are the main simulation artefacts to overcome in my design:

- **Modelling of radio channels** - Modelling a radio channel for simulation with perfect fidelity is a complicated endeavour. Not only does it require to use of a propagation model for radio waves, but also to consider interference from external sources and from itself, which in turn necessitates ray tracing the radio wave, modelling of obstacles and their reflective properties, etc. Thus, very few (if any) simulators are free from simulation artefacts in this regard. However, MeshSim is particularly susceptible to these because of its real-time nature.

  Running in real time means that the simulated NICs that each node is equipped with have a limited amount of time to perform all the calculations involved in determining the outcome of a radio operation. Thus, I opted for a simple radio-channel model based on empirical measurements. This model is described in section 3.4.3.1, and the experiments performed to calibrate the channel are found in sections 3.6.2.1 and 3.6.3.2.

- **Timing of radio operations** - Similar to the previous point, we have the importance of proper timing in radio operations. As described in section 3.4.3.1, the simulated NICs in the system are modelled as isotropic (omnidirectional) radiators. Thus, when a node makes a transmission, all other nodes in range should receive it at *roughly* the same time. Otherwise, this could introduce an artefact in which some nodes in the system would see new transmissions before the others, react to them faster, and potentially modify the outcome of the simulation.

  Further, while a real-world radio transmission requires the same amount of "work" regardless of the number of receivers, that is not the case in a simulated environment. Irrespective of the implementation choices for transmission, the simulation host will perform more work to deliver a packet of 200 bytes in size to 3 nodes, than a 50 kilobytes packet to 100 nodes. This is particularly relevant in a real-time simulator, as that extra work could make a simulated transmission *too slow*, which would be another simulation artefact. Thus, this effect most not only be minimised within the system, but its operating limits characterised for safe configuration. More details on how I address and measure this effect are available in section 3.6.3.2.

- **Movement granularity** - The rules of movement of nodes in a simulated model should follow different rules depending on what these nodes represent (people, vehicles,

etc.). MeshSim's extensible design allows for the support of multiple mobility models which handle these rules. However, the core mobility functionality that updates a node's position based on its velocity vector is part of the platform, and the same for all mobility models. If the time granularity (periodicity) at which node positions are updated is too coarse, it could introduce a simulation artefact, as nodes would apparently teleport between updates. This effect would be more pronounced in models with high velocity. The way I address this is detailed in section 3.6.1.

3.3.2 Experimental control

A consequence of gaining task-execution fidelity by running the code in real time is a loss of experimental control, which prevents MeshSim from having deterministic outputs even when controlling for all meaningful variables. This is because the simulation becomes exposed to the environmental conditions of the host machine such as available hardware resources, cache hits, OS-level contention, and the scheduler, all of which can alter the order and timing of operations. Thus, running the same simulation multiple times will likely offer slightly different results. While this difference tends to be minimal, sometimes even a small event such as the delivery (or lack of) of a packet can trigger a chain of events in a protocol's operations that significantly affect the output.

While these conditions might seem very detrimental to the usefulness of the system, they are unsurprising and quite manageable. After all, testbed-based research faces the same experimental conditions. Nevertheless, it remains of paramount importance to mitigate these effects in all experiments. The main challenges in this regard are the following:

- **Resource availability** - Another effect of running the code of each node in real-time, is that this requires the simulation host to have available at least the aggregated computational capacity of all nodes. If this was not the case, then one or more of the nodes, which run as independent processes, would *drag* behind the rest of the simulation. This would make it slower to react to incoming messages, internal state, and environmental conditions, potentially altering the results of the simulations. The was I control for this effect is described in section 3.6.3.

- **Reproducible simulations** - As stated above, exposing the simulation to the variability of its environment makes it impossible to have a fully reproducible simulation. However, that does not mean this effect can not be mitigated, a goal I achieve through two different strategies.

  Firstly, several of the test suites in the platform are dedicated to ensure the stability and reproducibility of core operations of the simulator (more on that in section 3.6.1.

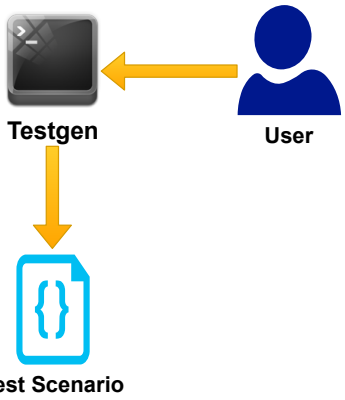
Secondly, I isolate the variability introduced between simulation runs by running each scenario multiple times and aggregating the results. The rationale behind this is simple. While the individual results may vary, I can observe the tendency of each metric and its variability rate by using the mean, median, and standard deviation of the data across simulation runs. As it turns out, most metrics tend to remain quite stable, which provides me with the confidence that the true value is close to the observed one. Further, by following this method, I was able to detect anomalies in the models or the platform when I observed high variability in some metric. Of course, some times the variability is perfectly reasonable given the experiment and metric in question, but often enough they pointed to a flaw in the the experiment.

3.4 System architecture

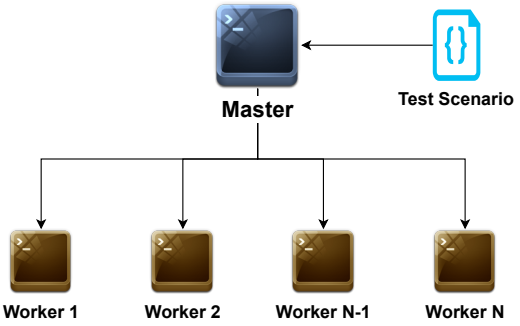
The MeshSim system consists of a set of executable binaries and configuration files that can be used together to either execute a *simulation experiment* or a *device deployment*. Regardless of the operation mode selected, the system relies on the same components, and the behaviour switch is driven by the configuration files. The primary components of the system are the *master*, the *worker*, the *test generation tool*, and the *configuration files*.

The *master* is a command-line program that acts as the main entry point for simulation mode. When invoked, it takes a *test scenario* file as input which describes the necessary components for the master to orchestrate the simulation, schedules all necessary actions, and finally, collects the produced data (figure 3.6b). Much like the MaRS system [18], a test scenario primarily contains the description of the network topology, the selected routing protocol, a workload specification, and a mobility model (figure 3.6a). A full description of the test scenario specification is provided in section 3.4.2. Test scenario files can be written manually, but the preferred way to create them is through the *Testgen* utility. This utility is an interactive command-line program that acts as a helper utility for the master. When executed, a series of prompts ask the user questions about the type of simulation scenario they are trying to create, and at the end produces an appropriate test scenario file to execute such a simulation.

The *worker* is a command-line tool that encapsulates the behaviour of a single node, which is driven by an accompanying *worker-configuration* file. Its configuration file determines, among other things, if it will operate in simulated or in device mode. If running in device mode, it will try to bind itself to the specified radio interfaces for communication (figure 3.7); otherwise it is provided with virtual resources that act the same as the physical ones. Once started, a worker in unaware of whether it is in *simulated* or *device* mode, and operates in the same way. When in simulation mode, it is started by the master, which provides it with the required configuration. When in device mode,



(a) Users produce test scenarios.



(b) Test scenario is used by the Master to drive a simulation.

Figure 3.6: MeshSim's simulation mode

only one instance of the worker can run per device, and it can either be started manually by a user or registered as a background process to provide its networking functions. Orchestrating a device deployment can be difficult, given that it can involve numerous devices spread over a geographical area. To aid this task, an external helper tool simply called *Cluster* was developed (further details in section 3.5.1).

Figure 3.7: MeshSim's device mode deployment.

3.4.1 The Master

The master is the coordinator program used to run simulation experiments. It takes a test scenario specification file as input, parses it, allocates all the required resources for the simulation, and starts it. While a simulation is being run, individual nodes are oblivious

to the fact that they are in simulated mode, and thus are unable to perform *simulation tasks*. It is the master's job to schedule and execute all said tasks, such as turning a given node into a data source, kill a node that has been specified to fail at a certain point, or end the simulation. The master has two primary subcomponents: the *master shared library* (or master library) and the *master_cli* binary. The master_cli binary is a thing wrapper around the shared library, used to handle and validate input before calling onto it, while the master library encapsulates the primary functionality of the master. The master library has in itself two main subcomponents: the TestSpec and Mobility modules. Figure 3.8 illustrates these modules and their most relevant properties.

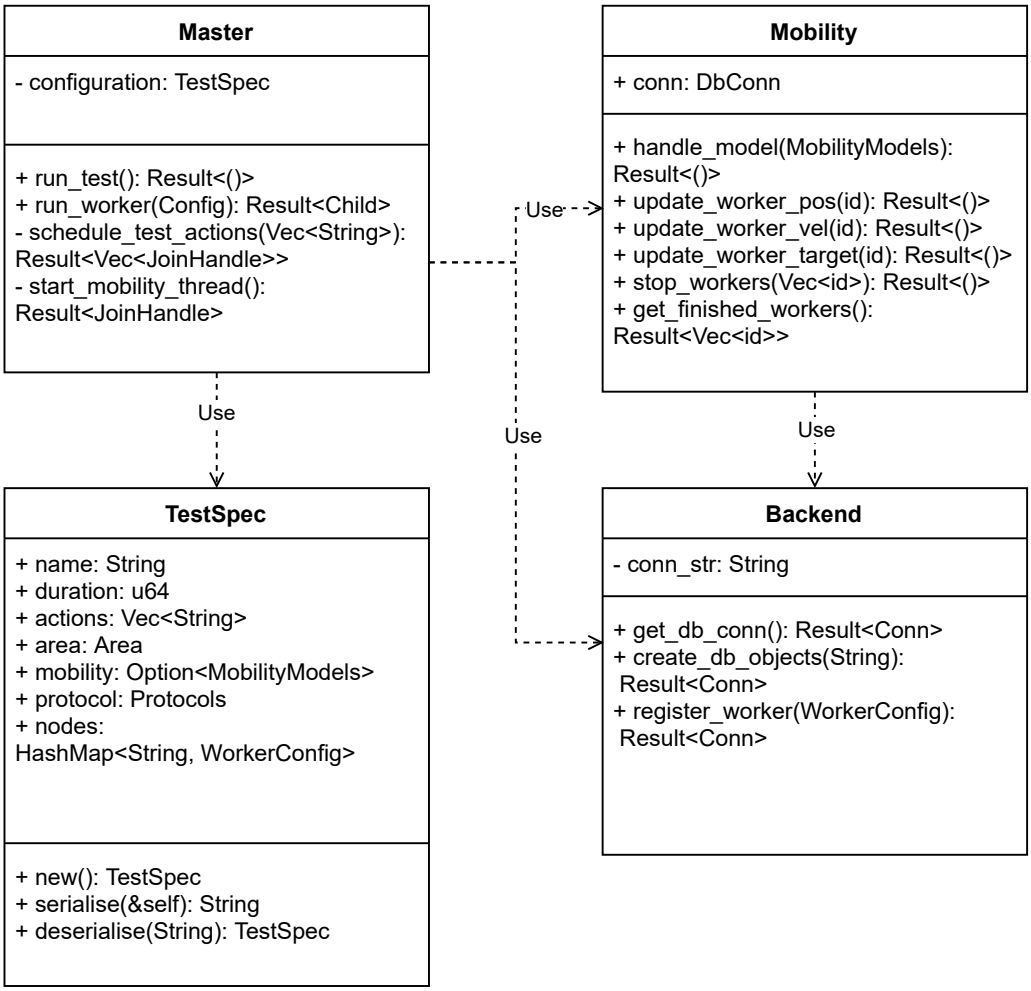


Figure 3.8: Partial class diagram of the master library.

3.4.1.1 Running a simulation

When running a simulation, the master_cli tool is called from a shell, passing the test scenario as a parameter (see listing 3.1.) Right away, the file is validated to exist and to conform to the correct format by leveraging the `deserialise` method of the `TestSpec` module. Once the configuration is validated, a few housekeeping activities are performed

(checking the working directory is writable, etc.), then, the logging infrastructure is activated. Finally, the master_cli instantiates the `Master` struct by passing the logger and validated `TestSpec`, and hands over control to it. The `run_test` method of the `Master` struct is the entry point for executing a simulation, and where the control of the program remains for the reminder of the simulation. The method performs four large tasks: Setup, nodes initialisation, test-action scheduling, and starting the mobility handler.

Listing 3.1: Starting MeshSim in simulated mode.

```
$ ./master_cli -t /path/to/specs/scenario1.toml
```

The *setup* stage of `run_test` consists of preparing the environment for the simulation. For this task, the `Master` relies on the `create_db_objects` function from the `Backend` module. MeshSim utilises a relational database to support its operations. Specifically, it uses the database to keep track of the positions of all nodes at all times (mobility), and to calculate medium contention (MAC layer). While multiple DBMS providers are supported by MeshSim, the default is Postgres [162].

Next, the `Master` proceeds to the *node-initialisation* stage. It iterates through the `nodes` collection of its `configuration` (figure 3.8), and performs three tasks per entry:

1. **Register the new node** - Using the configuration for the current node, it inserts its basic data, position, destination, and velocity into the database using the `Backend::register_worker` function.

2. **Create a worker configuration file** - Utilising its own `serialise` function, the `WorkerConfig` object is written into a file in the simulation's working directory, to be consumed later by its matching worker process.

3. **Start the worker** - Finally, a worker process is forked that executes the *worker_cli* program and takes the newly-create configuration file as input. A handle to the processes' *standard input* is saved for later use.

Next comes the *test-actions scheduling*. During this stage, the `Master` parses the `actions` collection from its `configuration` into a collection of `TestAction` for the simulation to execute, discarding improperly formed ones. Each `TestAction` is dispatched to a handler function for that type of action, along with all of its parameters. This dispatching involves the creation of a new thread, which after validating its parameters, suspends itself according to the passed *start_time*, ensuring the *TestAction* is executed at the designated time. All simulations have at leat one default test action called end_test, which is created upon parsing of the test scenario file according to its *duration* parameter.

Listing 3.2: Format of workload test actions.

```
add_source TYPE SOURCE DESTINATION [ARGS] START_TIME
```

Of special note in this stage are the *workload* test actions. These are the actions that determine the data to be transmitted during the simulation, and follow the format shown in listing 3.2. At the specified `START_TIME`, the associated test action thread utilises the *standard input handle* of the `SOURCE` node to feed it data to be transmitted to the specified `DESTINATION`. This interaction is akin to receiving a request for transmission from an upper layer in an OS networking stack. The number of packets, their size, the rate of transmission etc. are part of the extra `ARGS` passed to the test action, and they are variable depending to the specified source `TYPE`. Currently supported source types are *constant bit-rate* (CBR), and *ping*.

Finally, the `Master` proceeds to start the *mobility handler*. Similar to the handling of test actions, it dispatches a mobility handler function in a new thread, according to the *mobility_model* passed to it through configuration. This thread updates the position of each node according to its velocity vector at each mobility step, once per second. Regardless of the model, uniform motion is assumed when updating a node's position (e.g. acceleration of zero), although specific mobility models can easily modify this behaviour by changing a nodes velocity vector before its position is updated. The supported mobility models are defined in the `Master::MobilityModels` enum, which at the moment only supports two models: Stationary and Random Waypoint.

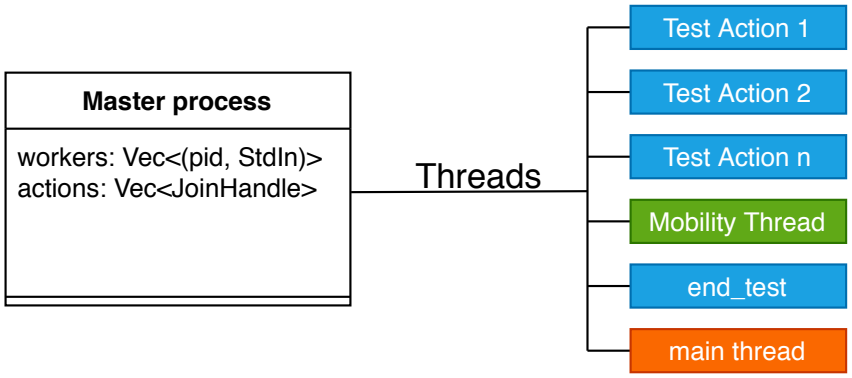


Figure 3.9: Internal state of the Master object once the simulation has been started.

Once all stages have been completed, the simulation is considered to be started, and the state of the `Master` object is similar to the the depiction in figure 3.9. At this point, the main thread of the `Master` waits on the `JoinHandle` of the *end_test* action. Once this handle returns, the `Master` sends `SIGTERM` signals to all the worker processes to terminate them, makes sure all logs are written fully, and the simulation ends.

3.4.2 Testgen

A significant aspect of any simulation tool is not only the features it supports, but the way in which its users describe the simulation scenarios. MeshSim utilises a test scenario file

that follows a certain specification. This specification is what ends up being the "language" in which users communicate the simulator what they want it to do. The format followed by the specification is quite *descriptive*, which makes cumbersome crafting by hand scenarios with large topologies. This is where `testgen` comes in. It is an interactive command-line tool that asks the user questions about the general type of simulation scenario they wish to create, and produces a a valid test scenario file for them.

After running the command inside a shell, it asks for the basic information all test scenarios require: Scenario name, duration, simulated area, mobility model, and protocol to run. Next, the user is greeted by a prompt that accepts commands to continue crafting the scenario. The tool is quite basic and does not a large range of commands, but is sufficient to get any researcher started with a viable scenario file. The supported commands are:

- **add_nodes** - Allows the user to add nodes to the scenario. The command follows the format `add_nodes TYPE NUM`. There are three types of nodes supported: *Initial*, which are nodes that will be started upon simulation start; *Available*, which are nodes that are not automatically started, but are part of the pool of nodes the simulation can later start via a test action; and *Grid*, which places the specified number of nodes in an $n \cdot m$ grid arrangement. All non-grid nodes are randomly placed along the simulation area.

- **add_source** - Adds a test action to turn a node into a data source at a specified time. The command follows the format `add_source TYPE SOURCE DEST [ARGS] TIME`. This test action makes sure that after `TIME` milliseconds have elapsed from simulation start, `SOURCE` node transmitting data to `DEST`, given the passed `ARGS` and the selected routing protocol. Two types of sources are supported: *Ping*, which produces a single packet with the text *"HELLO"* encoded as UTF-8 for payload; and the CBR type, which produces `RATE` packets per second of `SIZE` bytes, for `DUR` milliseconds.

- **add_action** - Adds a test action to the scenario. Each action has a unique format. Other than the ones already covered, two more test actions are supported: `start_node NAME TIME`, which starts node `NAME` after `TIME` milliseconds have elapsed from simulation start. An entry for `NAME` must exist int he pool of available nodes. And `kill_node NAME TIME`, which causes node `NAME` to crash-fail after `TIME` milliseconds have elapsed.

- **finish** - Lastly, this command finishes the execution of the testgen tool and prompts for a location to write the scenario file.

3.4.2.1 Test-scenario file specification

The testgen utility is useful a useful starting point for creating simulation scenarios, but it does not allow for the fine-tuning of simulation parameters. That task must be done manually at the discretion of it users. The file format used for these files is the TOML markup language [177], which allows for a rich set of key-value pair types, while remaining more readable than other formats like XML or JSON. Of course, within the format, there is a certain set of expected key-value pairs that drive the behaviour of the simulation. These expected values are given by the TestSpec format specification. The source code of the declaration of this struct can be appreciate in listing 3.3.

Listing 3.3: Simulation scenario specification.

```
#[derive(Debug, Deserialize, Serialize, PartialEq, Default)]
pub struct TestSpec {
    /// Name of the test
    pub name: String,
    /// Duration of the test in milliseconds.
    pub duration: u64,
    /// Vector of actions for the Master to take.
    pub actions: Vec<String>,
    /// Size of the simulation area.
    #[serde(flatten)]
    pub area_size: Area,
    /// The pattern of mobility the workers will follow
    pub mobility: Option<MobilityModels>,
    /// The maximum number of queued packets a worker can have
    pub packet_queue_size: Option<usize>,
    /// The protocol that this test will run.
    pub protocol: Protocols,
    /// Collection of worker configurations for the master to start at
    /// the beginning of the test.
    pub initial_nodes: HashMap<String, WorkerConfig>,
    /// Collection of available worker configurations that the master
    /// may start at any time during the test.
    pub available_nodes: HashMap<String, WorkerConfig>,
    /// Table used for storing any additional key-value pairs.
    /// Initially added to keep track of the grep patterns used for
    /// data analysis when running experiments.
    pub metadata: HashMap<String, String>
```

```
}
```

While the declaration of `TestSpec` in conjunction with its implementation of the `Serialize` and `Deserialize` traits from the `Serde` crate provide all the information needed, an example works better. Listing 3.4 provides a sample simulation specification file. The basic information about the simulation is placed at the top. It indicates this simulation is design to run for six seconds, in a 200 by 200 simulation area. All nodes follow the Random Waypoint mobility model with 10 second pauses, and run the RGR-III protocol. Finally, *node1* is supposed to send a ping packet to *node2* 1.5 seconds after the simulation starts.

The `initial_nodes`, `available_nodes`, and `metadata` sections of listing 3.4 are kept empty for readability purposes. In a real scenario file, both nodes sections would have an entry like the one depicted in listing 3.5 for each node in their respective section. The available configuration for each node gives a lot more control on the parameters of the simulation, especially in the performance of each of the radios a node holds. Further detail on node configuration is given in section 3.4.3.

Listing 3.4: Sample scenario specification.

```
name = "rgrII␣sample␣test"
duration = 6000
actions = [
  "Ping␣node1␣node2␣1500"
]
width = 200.0
height = 200.0
[mobility]
Model = "RandomWaypoint"
pause_time = 10
[protocol]
Protocol = "RGRIII"
k = 1
p = 0.5
q = 0.2
beacon_threshold = 6000
[initial_nodes]
[available_nodes]
[metadata]
```

Listing 3.5: Sample configuration for a single node.

```
[node1]
worker_name = "node1"
worker_id = "8bec26ee"
work_dir = "/tmp/test"
random_seed = 3433106158
operation_mode = "Simulated"
packet_queue_size = 500
stale_packet_threshold = 3000
[node1.position]
x = 45.40279508341434
y = 47.934522097623635
[node1.destination]
[node1.velocity]
[node1.radio_short]
timeout = 100
interface_name = "wlan0"
range = 38.0
mac_layer_retries = 8
mac_layer_base_wait = 16
```

3.4.3 The Worker

The *worker* is component of MeshSim that encapsulates the behaviour of a single node in a simulation, and acts as a network stack in a device deployment. The worker can operate in either *simulated*, or *device* mode. When operating in simulated mode the worker utilises virtual network interfaces to communicate with other nodes. This mode is reserved for when the *master* starts the worker, as the virtual interfaces require the backend to be setup accordingly. When operating in device mode, the worker attempts to bind the network interfaces indicated by name in its configuration file. Thus, the main difference between its two operating modes is whether it uses real or virtual network interfaces.

Regardless of its operation mode, once the worker has been initialised, it follows the same code path and goes into its main control loop. Inside it, it switches between listening for packets on all configured interfaces and executing periodic maintenance-callback functions required by the protocol. Every time a packet is received, the Worker hands it over to its *protocol handle* for processing. When a reply is necessary, the protocol handle returns a fully formed packet for the Worker to transmit.

The protocol handle is a *Rust trait object* implementing the `Protocol` trait (more in section 3.4.3.2). These objects are created at run time according to the program's configuration and are guaranteed to implement a trait (Rust parlance for interface), but whose underlying type has been erased. This abstraction makes it imposible for the Worker to know which protocol it is actually running, and keeping its operation protocol-agnostic, which is ideal for benchmarking protocols in equal conditions.

Listing 3.6: Starting a worker.

```
$ ./worker_cli --config /path/to/config/node1.toml
```

Listing 3.7: WorkerConfig struct

```
#[derive(Debug, Default, Serialize,
 Deserialize, PartialEq, Clone)]
pub struct WorkerConfig {
    worker_name: String,
    work_dir: String,
    random_seed: u32,
    operation_mode: OperationMode,
    packet_queue_size: Option<usize>,
    stale_pkt_threshold: Option<i32>,
    position: Position,
    destination: Option<Position>,
    velocity: Velocity,
    protocol: Protocols,
    radios: Vec<RadioConfig>,
}
```

Listing 3.8: RadioConfig struct

```
#[derive(Debug, Default, Serialize,
 Deserialize, PartialEq, Clone)]
pub struct RadioConfig {
    timeout: Option<u64>,
    interface_name: Option<String>,
    range: f64,
    mac_layer_retries: Option<usize>,
    mac_layer_base_wait: Option<u64>,
    frequency: Option<u64>,
    spreading_factor: Option<u32>,
    transmission_power: Option<u8>,
    max_delay_per_node: Option<i64>,
}
```

Much like the Master, the Worker is divided into two major components: The *worker_cli*, and the *worker* shared library. The worker_cli is a thin-wrapper command-line tool that provides argument parsing and validation, as well as logging functionality, but its main job is to call into the worker shared library. It is called from a shell (listing 3.6) and takes a configuration file as an argument (example in listing 3.5). The configuration file indicates what protocol it will run, how many network interfaces it has available (virtual or real), and what mode it will run on (simulated or device). The full specification of the configuration file is given by the `WorkerConfig` struct (see listing 3.7), and the format validated by its `deserialize` function. Further, the configuration of each radio interface is provided through the `RadioConfig` struct (listing 3.8), which provides a fine-grained configuration for both simulated and real radio interfaces. Once a worker process has been fully initialised and it is running, its internal state is similar to the one depicted in figure 3.10.

The *worker library* has a main structure simply called `Worker` that fully encapsulates the functionality of a node (simulated or otherwise). At the same time, the `Worker` depends on two subcomponents: The `Radio` and `Protocol` traits. Figure 3.11 shows a high-level vision of how these components interact. The `Worker` holds a collection of `Radio` objects, and a reference to its `Protocol` handle. After the worker_cli performs validations, it creates the `Worker` object and calls its `start` function. This function performs the necessary protocol and radio initialisation operations and then goes into the main loop, as described above. It relies on its `Radio` objects to receive and transmit packets, and on its

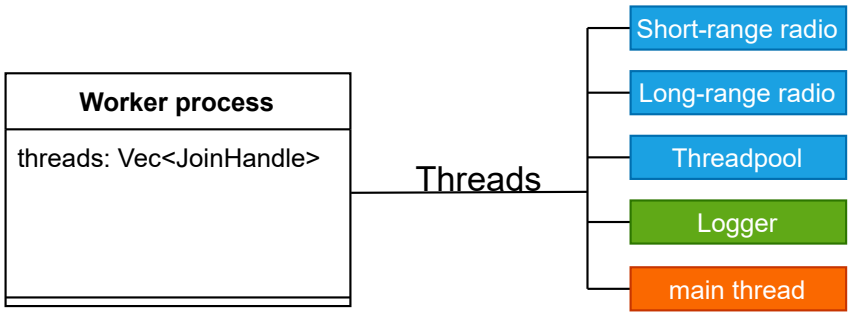


Figure 3.10: Worker state and threads.

`Protocol` to process said packets.

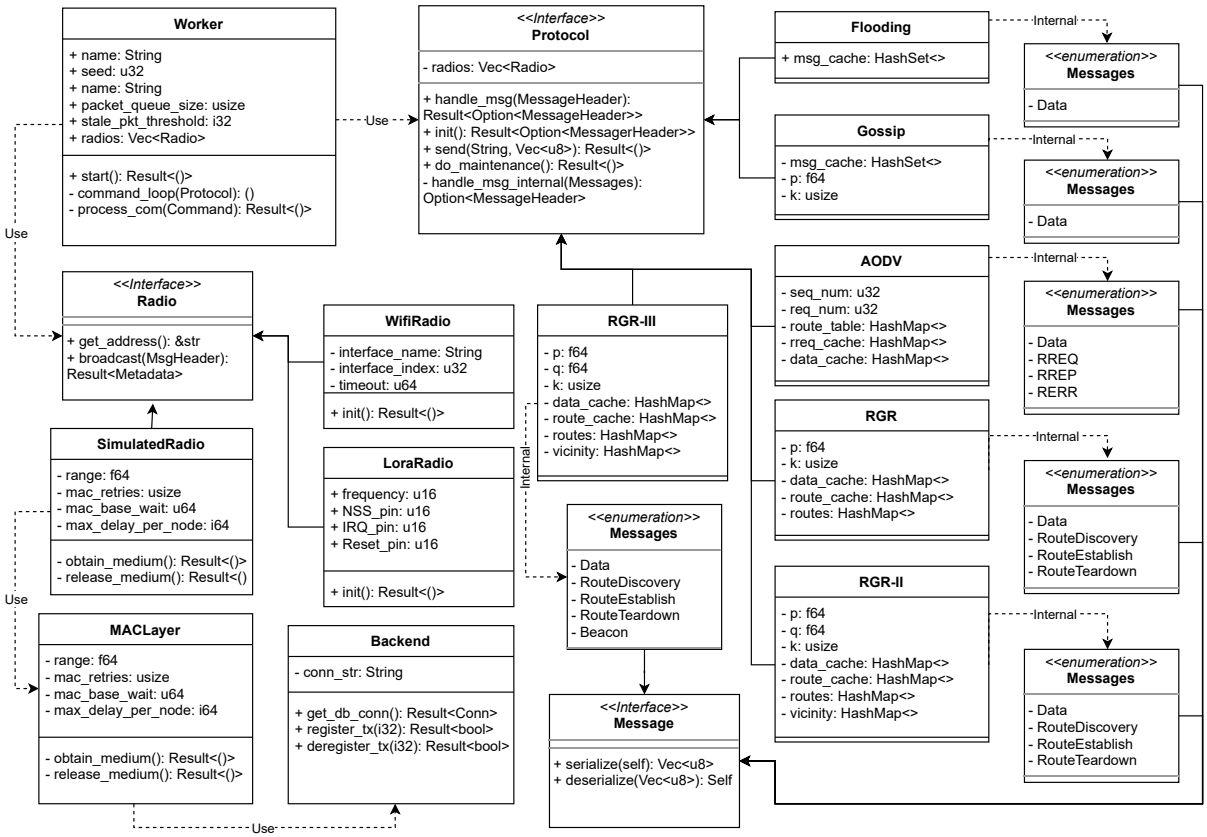


Figure 3.11: Architecture of the Worker.

3.4.3.1 The Radio trait

The radio trait encapsulates all network interfaces used by the system (real and virtual), and is what allows the code from the simulated nodes to be deployed into physical devices without modification. As illustrated in figure 3.11, this is implemented by three different structures: `WifiRadio`, `LoraRadio`, and `SimulatedRadio`. The trait is quite simple as it only contains two functions: `get_address` and `broadcast`. For the `WifiRadio` and `LoraRadio` cases, their respective implementation simply involve making sure the

appropriate network interfaces are present in the system and *UDP sockets* can be created with them. For the `SimulatedRadio`, however, it gets a bit more complicated.

The `WifiRadio` and `LoraRadio` both represent real interfaces (e.g. used only in device mode), and are used as short range and long-range radios respectively in device deployments. `SimulatedRadio` abstract radios of all ranges and properties to be represented in the simulator. The implementation of this feature assigns a socket address (e.g. local address plus a port) in the machine for each active interface, as well as a set of rules for each such radio (given by its configuration). All simulated radios depend on a MACLayer module to provide medium contention functionality, which is based on the frequency of the radio, so simulated WiFi and LoRa radios would not interfere with each other.

When a simulated node wants to make a transmission, it calls its `broadcast` function. Subsequently, `broadcast` calls into `MACLayer::obtain_medium` to perform the simulated RTS/CTS operations, since the MACLayer module simulates CSMA/CA functionality. This module then checks on the backend, and based on the position of the calling node and the range and frequency of the radio in question, it returns a list of the nodes in range if it succeeds to acquire the medium. Upon failure, it performs exponential random back off (as configured by the `mac_layer_retries` and `mac_layer_base_wait` parameters) and tries again until the medium is acquired, or gives up. With the medium acquired, the broadcast function proceeds to iteratively transmit the packet in question to all nodes in range one by one. This method can be problematic, as some nodes would obtain the transmission much sooner than others (which would not happen in real devices). To attenuate this effect, the transmitting node calculates a `wait_time` for each packet based on their respective transmission order and the total number of nodes involved in a single broadcast operation. All receiving nodes extract this `wait_time` from the `MessageHeader` structure, and the receiving thread sleeps for that number of microseconds before returning from the function. This mechanism not only mitigates the problem of the iterative transmission, but ends up translating as the propagation delay of a real transmission (more on that in section 3.6.2).

3.4.3.2 The Protocol trait

The protocol trait encapsulates the behaviour of a protocol in regard to its interactions with the `Worker` struct. It was designed such that a MANET routing protocol can be implemented as an independent module, and the "glue" that connects it to the rest of the system is the `Protocol` trait. It contains four methods: `handle_message`, `init`, `send`, and `do_maintenance` (see figure 3.11).

The `init` method ensures that before a protocol receives a packet for processing, it has had the opportunity to perform any operations it requires to handle (neighbour scanning, procuring membership, etc.). Conversely, a lot of protocols require periodic maintenance operations to be performed, such as purging old entries in routing tables or

transmitting beacon messages. The `do_maintenance` method ensures that does operations are performed regularly as needed.

The reminding two methods are where the core of the worker-protocol interaction occurs. When a new packet is received by the Worker on any interface, it is represented by the `MessageHeader` struct. This struct acts as a simplification of an IP packet header with very similar fields (source, destination, hops, TTL, etc.) and holds a binary payload (a `Vec<u8>`). When such a struct is read from a network interface, the worker passes it to the protocol for processing by calling unto its `handle_message(MessageHeader)` method. This method extracts the MessageHeader payload and deserialises it into a variant of the protocol's own `Messages` enumeration. This enumeration holds an entry for each of the message types defined by each protocol. This message variant is then passed to a `handle_msg_internal` function, that dispatches the message its corresponding handling function based on the message type, and optionally, produces a response in accordance.

Finally, the `send` function acts as the entry point for upper network layers to signal the need for data transmission. When a simulation requires a node to transmit a certain workload to another node, the data is first generated at the Master module. For each data packet, the Master writes into the `stdin` handle of the source node, which reads and decodes the data, and then passes it to the protocol via the `send` method. Depending on the protocol in question, this method determines if it has a viable route to the destination or if it requires to procure one, and then starts the data transmission.

3.5 Device deployments

One of the features enabled by using MeshSim is the Calibration-Simulation-Validation experimental process I describe in section 1.4.1. Naturally, this process requires not only the flexibility of the software to run unaltered on both simulation and device mode, but it actually requires devices to run it on. To this effect, I built a set of devices capable of running a MeshSim worker, that are mobile, can collectively serve as wireless testbed, and are easy to control.

Although an initial consideration was made to acquire off-the-shelf units for this purpose such as smart phones, I ended up moving away from this option shortly after. The main problem with them was the fact that they are semi-closed platforms. While software development kits exist for the most popular smart phone platforms (Android and iOS), they are also designed only for apps to run on them, and not to give direct access to the hardware or operating system. Further, while all smart phones come equipped with multiple radio interfaces as I wanted, the transmission of the cellular bands is regulated in most countries, which would prevent me from using that interface for transmission. Lastly, even though smart phones have been commoditised, the cost of a single unit is still in the

range of hundreds of pounds.

Given the closed nature of the smart phone platforms, I ended looking into open platforms that would allow me to put together a device with the characteristics required. In the end, I ended putting together a device based on the Raspberry Pi 4 [148] system-on chip (SoC) computer. This platform ended up being a perfect replacement for mobile phones. Their 64-bit ARM-based processors are on par (performance-wise) with the most expensive smart phones in the market, and have the same architecture. They come equipped with 8gb of memory, have expandable storage, a UART (serial) interface for expansion, and run a simple Linux-based operating system (Raspbian). In short, they provide equivalent computing power to that of a smart phone in a compact form, with expandable hardware and running on an open operating system.

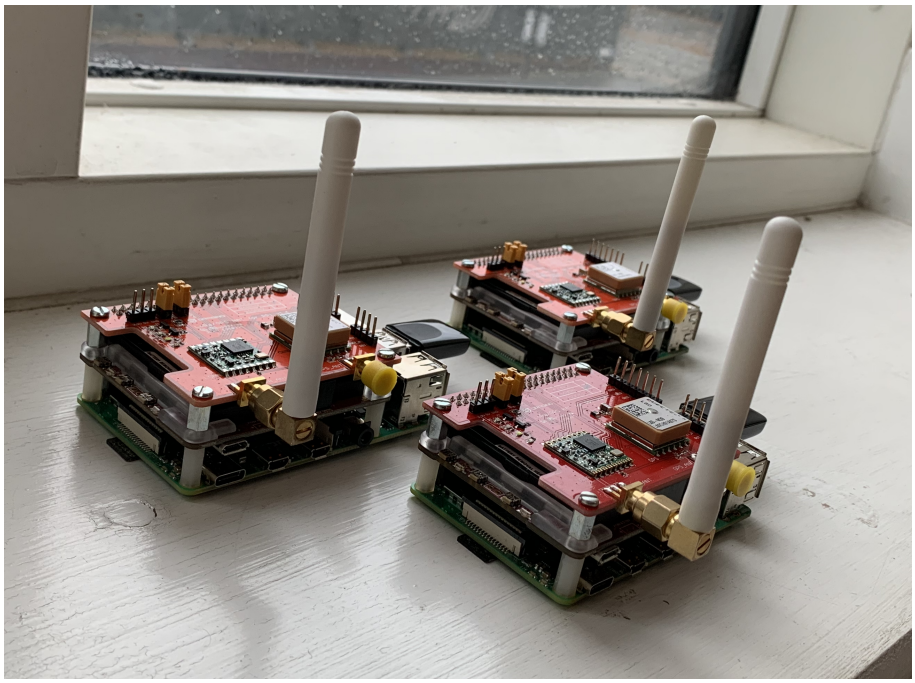


Figure 3.12: The device-deployment units.

Figure 3.12 depicts three of the twenty full units I built for my experiments. The full unit consists of a Raspberry Pi 4 computing board, an energy-management board mounted on the computing board, a Dragino LoRa/GPS HAT [160] board on top, a second WiFi interface, and a 64gb SD card for storage. The energy management board comes equipped with a 5000 mA battery, that allows the units to be run untethered from a power outlet for a limited amount of time. Finally, the LoRa/GPS HAT board is used in lieu of a traditional 4G/5G cellular radio. While they have very different radio properties, the purpose of both for my research was to have access to a heterogeneous radio band with a larger radio range and lower bandwidth than WiFi.

The repurposing of a LoRa radio for use as a two-way communication channel was not trivial, though. LoRa radios were developed for IoT networks, envisioning situations

in which the client nodes only transmit data a few times a day, and they rarely need to receive data themselves. However, their low-power consumption and range were attractive for this use case, so I decided to use them. This required the development of a Linux driver for the radio for the Raspberry Pi. The development of this driver and the Cluster tool were done by my undergrad research assistant, Nathan Corbyn, as part of his research project in our lab in the summer of 2019.

3.5.1 The Cluster tool

The compact form and battery-powered nature of the device units afford significant flexibility when performing device experiments, as they can be placed anywhere with network access to their control server. This flexibility allows for several topologies to be formed with the units, but it also places an unfortunate overhead on the execution of these experiments. While the nodes do not need to power up at exactly the same time, they do need to coordinate the start time at which their worker process starts listening for packets. This coordination is orchestrated with a software tool written for this purposed by my research assistant, called the *Cluster tool* [158].

Listing 3.9: Sample deployment file

```
name = "lora_wifi_beacon_placement"
log_dir = "results/lora_wifi_beacon_placement/log"
command = "cargo"
args = ["build", "--release"]
[hosts]
[hosts.rogue1]
command = "./target/release/worker_cli"
args = ["--config", "experiments/lora_wifi_beacon_placement/rogue1.toml"]

[hosts.rogue2]
command = "./target/release/worker_cli"
args = ["--config", "experiments/lora_wifi_beacon_placement/rogue2.toml"
```

The operation of deployment system is quite simple. The deployment client runs on the units as a Debian service that auto-starts with the OS. Upon start up, it issues a registration command to the control server, which then lists the unit as available in its dashboard. After the required units have registered, a new experiment can be started from the dashboard by selecting one of the available experiment files (see listing 3.9).

The control server then sends the command to all registered units to start a new experiment. Upon receiving this command, all units perform the following tasks:

1. Perform a `git pull` operation on the MeshSim Git repository to obtain the latest version of the source code.

2. Start a distributed-build process of MeshSim with the release profile (top-level command of the spec).

3. Execute the command defined in their respective configuration section.

It is with the help of these devices that I was able to perform the calibration experiments described in the next section.

3.6 System Evaluation

Evaluating a system is the process through which researchers determine if it performs the job it was designed to and benchmark how well it does it in relation to some baseline. While this process is straightforward to describe, its execution is rarely so, as the intricacies of each system, its context, and so forth can make comparisons difficult. For instance, one can be confident in the results of comparing two MANET routing protocols in the same simulator since we know they are subject to the same environment and conditions. But, when comparing two MANET simulators to each other, what do we use as ground truth? Relative measurements between the platforms are pointless since they use different abstractions and trade-offs, leading to different results.

Thus, it would be futile for me to try to compare MeshSim's performance directly to other simulators given their different abstractions. Instead, I answer the system evaluation questions of MeshSim in this section through two different mechanisms: Correctness validation, and system profiling. The first such mechanism (section 3.6.1) presents the work I did to make sure MeshSim's design correctly addresses its research challenges (section 3.3) and that its implementation corresponds to the design. The second mechanism (section 3.6.3) then presents a series of experiments that further address the research challenges of section 3.3 and also provide the operational range of MeshSim's parameters. That is to say, the operational limits under which I can guarantee MeshSim's results are accurate in my simulation environment.

3.6.1 System tests and validation

When it comes to validating a software system design, the more straightforward way to do it is through traditional software testing, and MeshSim is no different. These tests are there not only to ensure high quality in the code, but also to ensure that changes in the simulator do not introduce any of the simulation artefacts discussed in section 3.3.

Following software engineering best practices, MeshSim features two different types of tests:

1. **Unit tests**, which focus on a specific component and test its behaviour and properties in isolation through different ranges of inputs.

2. **Integration tests**, which test the end-to-end behaviour of the system under different conditions.

The entire test suite is comprised of over 70 tests, adding up to 5,122 test source code lines and 33,948 configuration lines, describing different scenarios. While source code lines are not a synonym of quality, it does speak to the breath of testing and validation that were part of the MeshSim development process. The source code for the entire test suite is available as part of the MeshSim repository, either under the `tests` folder, or as part of tests modules that accompany some source modules. Thus, any person can clone the repository and, provided they followed the setup instructions detailed there, can run all tests to confirm they pass by using the command `cargo test -- --test-threads=1` and get an output such as the one depicted in listing 3.10.

The tests themselves are organised in the following modules:

- **Backend** - The tests in this module are in charge of ensuring the simulation's data is stored consistently and safely, and that it can be accessed in the same way. As the described in section 3.4, both the `Mobility` and `MacLayer` modules utilise this `Backend` to store and query the positions of all nodes. Thus, this test module also ensures that these data is accessed in the way its meant to.

- **Radio** - This extensive test module is likely the most important of the system, as it checks the correctness of all functionality in the `Radio` module, where most of the simulator's abstractions reside. Further, it also ensures that all radio operations are well within the system calibrations and that no simulation artefacts are being introduced. More details on those specific tests are provided in section 3.6.3.

- **Worker** - These tests validate that all operations of the worker are responding as they are expected. These checks range from the validation of configuration files, to the timing of each of the worker's operations.

- **Protocols** - This extensive test module defines a submodule for every routing protocol supported by the platform. Within each, the tests ensure that each protocol's basic operation, and that they respond adequately to different network conditions.

- **Mobility** - As the title implies, this test module ensures that all mobility operations within the simulator function correctly. Such tests range from validation of the

core mobility operations (e.g. node positions are timely and correctly updated), the specifics of the different mobility models supported, and to corner cases and decimal round up inaccuracies.

- **Master** - This module is in charge of testing the basic functionality of the `Master`, such as starting worker processes, scheduling test actions, and so forth. Given that the `Master` is the entry point for all MeshSim experiments, and consequently, all integration tests, these also carry the load of testing it. An important aspect of this module is that it ensures that the `Master` does not introduce any simulation bias or artefacts in its operations. For instance, the test `test_process_start_sequence` checks that all workers of a test specification are always started and registered in the database in the same order. This is important as altering this order could introduce unintended variability in the system.

Listing 3.10: Extract of test execution

```
worker::worker_config::test_workerconfig_write_to_file ... ok
CBR::test_cbr_basic ... ok
aodv::aodv_rerr ... ok
device_mode::integration_device_mode_basic ... ok
increased_mobility::one_node ... ok
master::test_process_start_sequence ... ok
random_waypoint::three_node_movement ... ok
reactive_gossip::test_packet_counting ... ok
reactive_gossipII::test_route_discovery_optimization ... ok
reactive_gossipIII::test_route_discovery_optimization ... ok
mobility::increased_mobility::test_increased_mobility_basic ... ok
mobility::random_waypoint::test_random_waypoint_basic ... ok
mobility::worker_positions::test_worker_positions_basic ... ok
radio::test_broadcast_delay ... ok
radio::test_broadcast_device ... ok
radio::test_mac_layer ... ok
radio::test_signal_loss_calculations ... ok
radio::test_tx_bandwidth ... ignored
```

Lastly, as the most observant readers might notice, some of the tests depicted in listing 3.10 have a status of `ignored`. This is no coincidence, as these tests are not ran as part of the normal test suite, but rather, they are used to perform *parameter sweeps* of MeshSim. More details on these tests are available in section 3.6.3

3.6.2 Simulation calibrations

In section 3.3, I mentioned how the real-time nature of MeshSim make it difficult to simulate a complex radio channel, given the limit on how many operations can the simulator perform in the short time a "real world" transmission would take. While this trade-off is expected due to my design, the radio channel remains one of the crucial components in a simulator that can not just be taken for granted. The approach I adopted to mitigate this effect was the use of empirical measurements to elevate the fidelity rate of my simple radio-channel model.

As I mentioned already in section 1.4.1.1, the radio channel model of any simulator is in charge of determining *which* nodes receive every transmission and *when* they receive it, in accordance to a given propagation model. However, any such model is in itself an abstraction, and one that does not consider the overhead of the software running the network stack. In the experiment depicted in this section, I outline how I perform the empirical measurements necessary to calibrate MeshSim and my experimental scenarios to adopt the properties of a "real-world" radio channel. While this measure was something I initially considered a mitigation measure to compensate for the trade-offs of my design, I eventually came to realise the value it brings to the table. This is because the empirical results I obtained did not match the expected results from the theoretical models.

3.6.2.1 Radio-calibration experiment

In order to address the issues stated above, I designed an experiment to capture empirical data of relevant radio metrics that I could use to calibrate my simulations. Using my deployment devices (see section 3.5), I measure the maximum optimal transmission range of their radios as well as the end-to-end latency of a radio transmission. These measurements are performed in an indoors environment and are therefore representative of this particular configuration.

To perform this experiment, I created a simple network protocol called the *Beacon* protocol. Upon initialisation, this protocol enumerates the radio interfaces it is configured to use, and then proceeds to transmit periodic *Beacon* packet on each NIC. Each Beacon is numbered, and upon reception of such a packet, the protocol responds with a *BeaconResponse* packet containing the same sequence number from the Beacon. All sent and received packets are logged, and later used to calculate the transmission delay of the interface.

Equipped with this protocol, the design of the experiment is quite simple. Using two device units (See 3.5), I place one of them in a known location and power it on (the base node). The second node (the roaming node) is connected to a laptop via wired Ethernet, and powered on. This connection allows me to log into the device via `ssh` and see its live log via `journalctl`. The most important information displayed in this log is the reception

of packets from the base node, for as long as I receive packets from it, the nodes are still in range from each other.

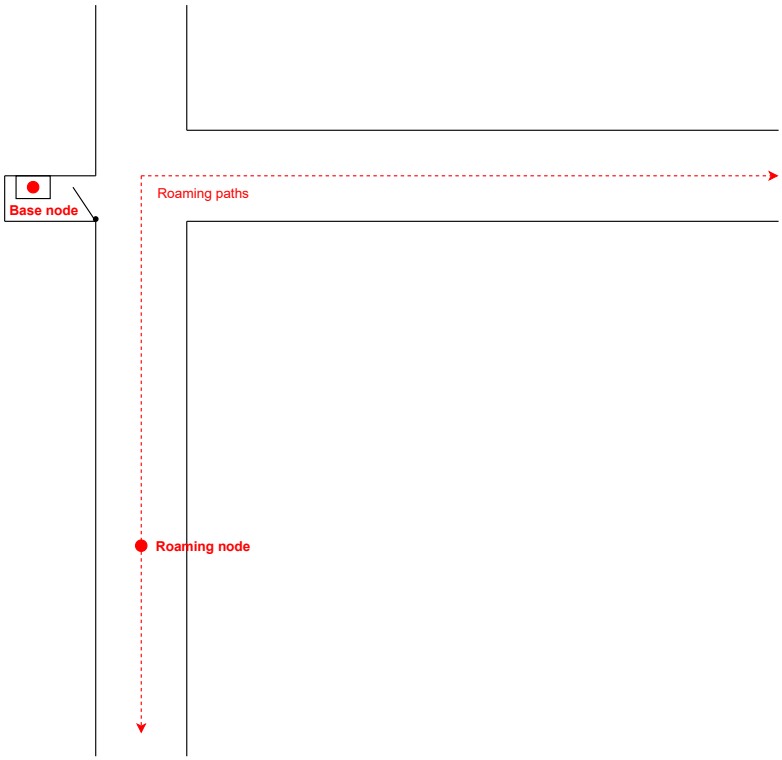


Figure 3.13: Diagram of the calibration-experiment area.

Starting from the base node's position, I follow a straight path inside the Computer Laboratory building (see figure 3.13) and walk down it slowly until I stop receiving packets from the base node. The goal of this movement is to find the maximum distance before I start losing packets from the base node. In order to know I missed a packet, I use the sequence numbers from the received beacons, as they are monotonically increasing. Once the maximum distance of 100% packet reception has been found, I mark it down, and proceed to calculate the total displacement from my starting point, using a laser distance meter. Using these annotations, and an architecturally correct map of the building at a scale of 1:200 (figure 3.14), I calculate the maximum distance. This procedure is repeated using the same trajectory, and in the perpendicular direction from the base node, and the data is added together. The results can be seen in table in 3.1. Based on this data and for simplification, simulated WiFi and LoRa radios are calibrated to have ranges of **38.0 metres** and **70.0 metres** respectively in the rest of the experiments in this thesis.

Radio	Iter 1	Iter 2	Iter 3	Iter 4	μ	σ
WiFi	37.5 m	39.1 m	38.3 m	36.8 m	**37.925 m**	**0.8613 m**
LoRa	68.9 m	71.4 m	72.1 m	68.2 m	**70.15 m**	**1.6379 m**

Table 3.1: Radio range measurements.

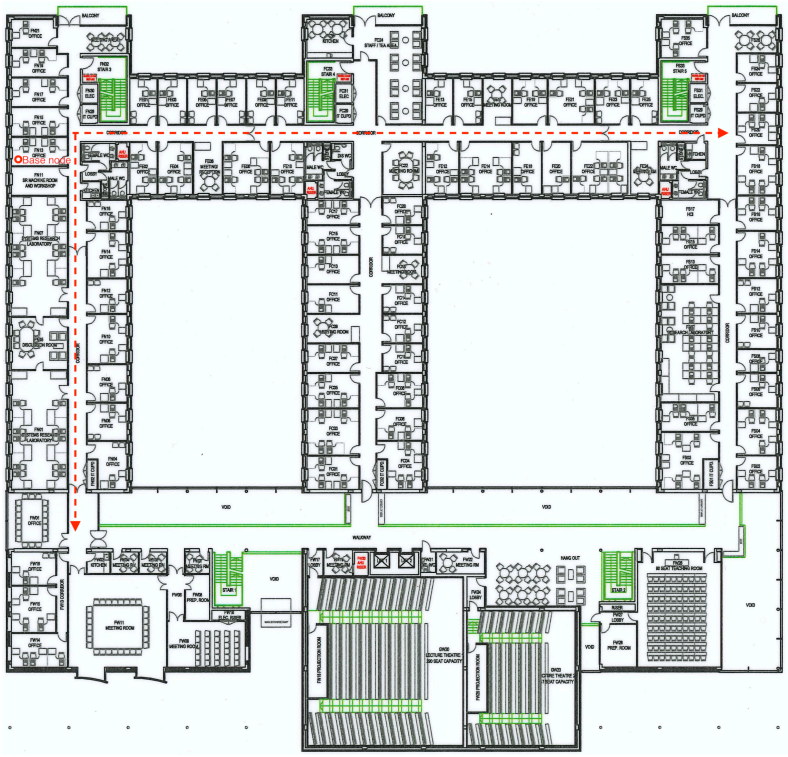


Figure 3.14: Map of the area used for calibrating the radio range.

The other important measurement obtained from this experiment is the propagation delay of a single radio transmission. Given the design choices used for MeshSim, a broadcast operation from a simulated radio translates into a round-robin unicast transmission to all n nodes in simulated range. The unicast transmission translates into a sending a UDP6 packet from one socket to another in the simulation server. Such an operation is significantly faster than a real-world wireless transmission, as it ultimately results in copying data between two local memory locations. However, if the time to transmit one of these UDP6 packets is δ, then the broadcast transmission time (B_t) would take $n * \delta$ µs. On the other hand, a real-world wireless broadcast is omnidirectional and all n nodes in range would receive it at the same time (D_t). Thus, the problem becomes that for a very large n, $B_t \gg D_t$, and for smaller n, $B_t \ll D_t$. By finding out the value of D_t, I can come up with a solution that for all cases ensures $B_t \approx D_t$.

Fortunately, the data collected from the range-estimation experiment is ideal for this purpose. As mentioned before, each of the *Beacon* packets transmitted carries an identifier, which is echoed back in a *BeaconResponse* packet. By subtracting the timestamps of matching Beacon and BeaconResponses, I get the total round-trip time of that packet, and can thus reasonably estimate that half of that would be equal to D_t. Figure 3.15 illustrates the distribution of round-trip times collected for all packets in the experiment.

Given that the mean of the collected data is 5.6117 ms, the simulated radios are configured on the assumption that $D_t \approx 2.8059$ ms. The simulated broadcast algorithm

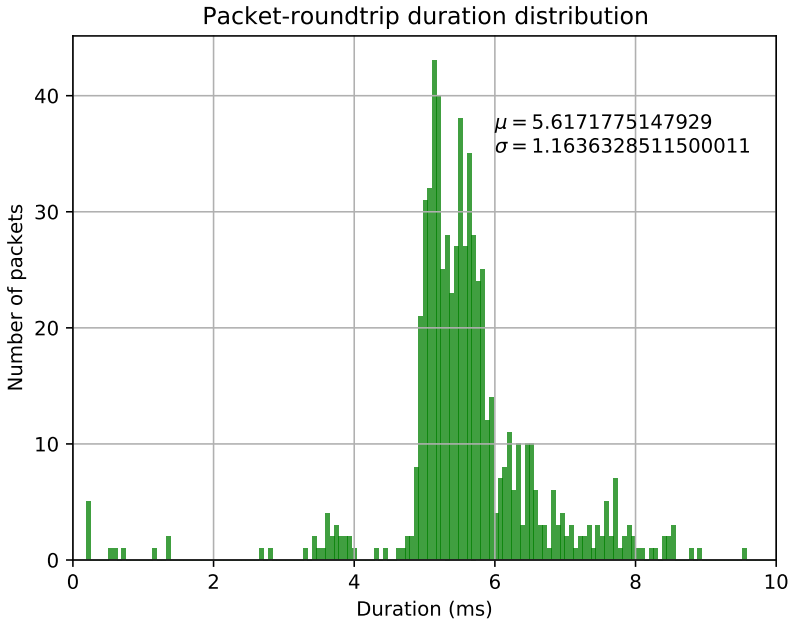


Figure 3.15: Distribution of round-trip times for all beacons.

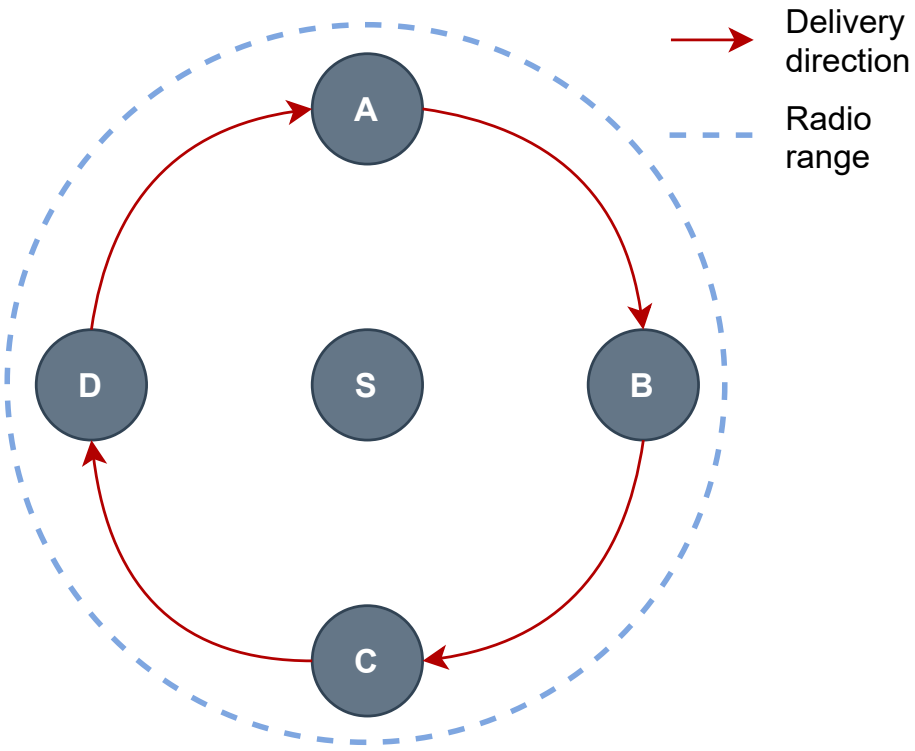


Figure 3.16: Distribution sequence of a simulated broadcast.

incorporates these data by inducing an artificial delay into the receivers of a transmission. When the simulated radio of a source node acquires access to the medium via its MAC layer, it also receives a list of nodes within its range. Since Wi-Fi radios use omnidirectional antennas, all those nodes should receive a transmission made by the source node at that

time, and they should receive it almost simultaneously.

To implement this feature, the simulated radio does a round-robin unicast transmission to each of the incumbent nodes, adding an expected delay parameter in the `MessageHeader` (see figure 3.16). The initial value of this expected delay is 2.8059 ms, with an added jitter of $\pm$ 0.581 816 4 ms, and it decreases proportionally for each node involved in the transmission. Upon reading and decoding a MessageHeader, all receiving nodes wait for the indicated delay time, thus ensuring they all receive it in a reasonably synchronous manner, and as close as possible to the calibrated transmission time.

3.6.3 Operational limits

Since MeshSim simulations run each node as real time process, it lacks the abstraction of computational resources that discrete-event simulators provide. This makes it of paramount importance that the computer running the simulation must have sufficient computational resources to emulate the aggregated demand of all nodes in the simulation. Failing to meet this requirement would alter the results of the simulation, by means of *process drag* in some nodes. That is to say, lacking the required CPU time to update their internal state and process inputs in a timely manner would result in a delayed reaction to network events, something that would very likely affect the operations of a protocol.

In this section, I detail what are the important metrics to consider to make sure MeshSim's operation is correct. Further, I provide a characterisation of MeshSim's capabilities and limits for a fixed set of hardware (my simulation server) and a given radio-channel configuration. The purpose of this data is twofold: Validate that my experiments in chapter 4 were performed under adequate circumstances, and to inform the readers on what they can expect from MeshSim for their own purposes.

To this effect, the next section presents the details of the hardware used to perform this measurements. This is followed by the experiments used to determine the limits in the radio-channel given the calibration data from section 3.6.2, and lastly a general evaluation of resource consumption of MeshSim as a function of the simulation's size.

3.6.3.1 Hardware specs

The simulation server I utilised for all my experiments is a Linux-based virtual machine in the infrastructure hosted by the Computer Science & Technology Department at the University of Cambridge. This machine was allocated exclusively for my experiments, which is noted to discard variation in the results by concurrent workloads of other users. Its hardware and operating system specifications are summarised in table 3.2. Further, table 3.3 contains the relevant kernel parameters and their values for supporting a large computational workload such as the ones performed in these experiments.

Parameter	Value
Machine type	Xen HVM ver 4.13
OS	Ubuntu 18.04.5 LTS
Kernel	4.15.0-70-generic x86_64
CPU(s)	30 Dual core Intel Xeon Gold 6142s arch: Skylake rev.4 cache: 675840 KB clock speed: 2594 MHz
Memory	80,522.2MB
File system	Ext-4

Table 3.2: Hardware specification of the Simulation Server.

Parameter	Value
threads-max	643847
pid_max	131072
nf_conntrack_max	2621440
max_map_count	65530

Table 3.3: Kernel configuration parameters of the Simulation Server.

3.6.3.2 Radio-channel limits

As described in section 3.4.3, my `SimulatedRadio` performs broadcast operations by unicasting a packet sequentially in a round-robin fashion. To ensure these packets are received by all nodes roughly at the same time, each *MesssageHeader* contains a delay parameter that is applied at the receiving end. Further, from we know from the Radio Calibration experiment (section 3.6.2.1) that the end-to-end delay for the transmission of a packet is 2.8059 ms with an added jitter of ± 0.581 816 4 ms. Given this information, there are three natural limits to consider in regards to the timeliness of my simulated radio operations: correctness, accuracy and workload size.

The first such limit is simple to postulate: no simulated transmission should ever take longer to complete than 3.387 716 4 ms, or less than 2.224 083 6 ms. This behaviour is ensured by the test `test_broadcast_delay` in the radio test module. This test performs hundreds of transmissions on a simulated channel with different conditions, and ensures they all meet the correctness bound.

When it comes to accuracy, I refer to the fine-grained delay calculations for each receiver of a packet. These delays are calculated by the transmitter in real-time, and could therefore be subject to inaccuracies. Fortunately, I ensure my algorithm for such calculations is robust enough using the test `test_broadcast_timing`. This test performs hundreds of broadcasts by a single transmitter in three different receiver setups: Single receiver, ten receivers, and 50 receivers. Each such transmission is timed and aggregated within its group, and then the test validates that the variability in average transmission times for all three groups is minimal (less than 5%).

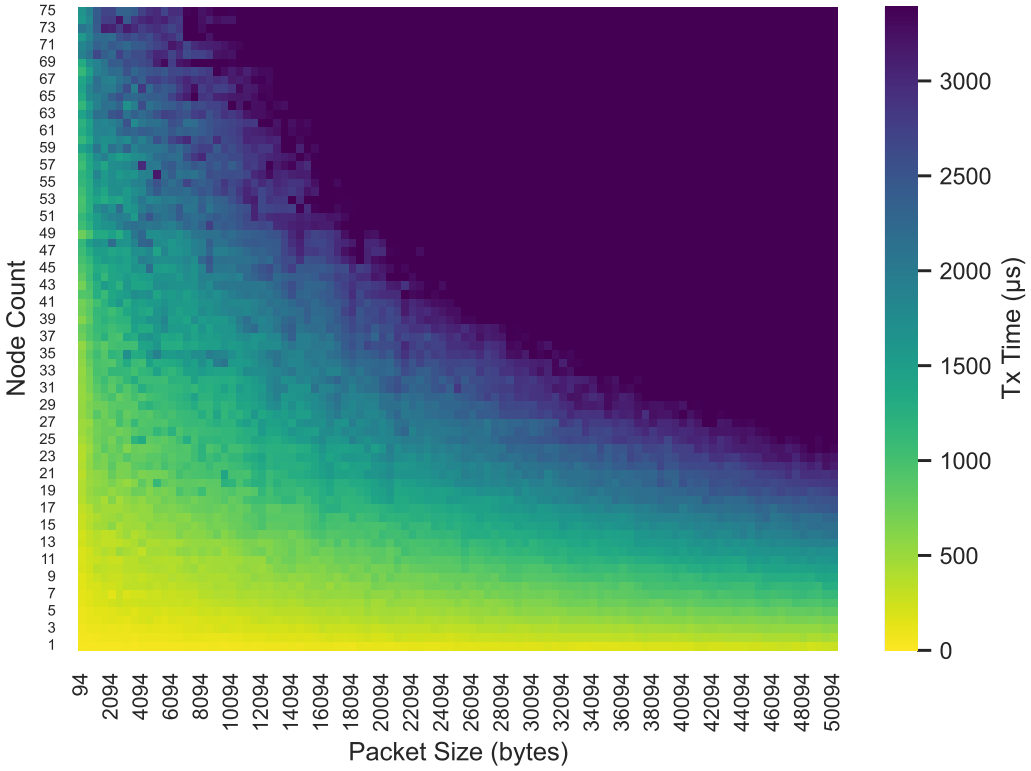


Figure 3.17: Mean transmission time per receiver count and packet size.

Lastly, we have the limits of workload size. Intuitively one can see that both the number of receiving nodes and the size of the packet would have a direct effect on the time it takes to complete one of my simulated broadcast operations. Thus, it is important to characterise these workload limits to ensure that all simulated radio transmissions of a simulation can be performed within its expected calibration time. This characterisation is performed in the test `test_tx_bandwidth`, which sweeps across a range of values for number of receivers and packet sizes and performs 25 transmissions for each configuration. The mean transmission time for each of this configurations is presented in figure 3.17. In this figure, we can appreciate the safe region for maximum number of receivers and packet size. With the structured logs provided by MeshSim, it becomes trivial to write a script that ensures these parameters are not violated at any moment during a simulation.

3.6.3.3 Simulation size limits

To close the chapter, this section presents a study that measures MeshSim's resource consumption as a function of the simulation size. Once more, these data is presented both as validation that my experiments were performed well within the operational limits of the platform and to inform researchers of MeshSim's capabilities. The experiment itself is quite simple. I repeatedly run a simple simulation of N nodes for a set period of

time, increasing N at each step. To add reliability to the measured data, the full set of simulations is run multiple times and all data added together.

During each iteration, I measure MeshSim's resource consumption using the standard Linux tools contained in the `sysstat` software package. This package provides a program called `sar` (short for System Activity Report), which can run as an independent process and collect several different system statistics at a configurable sample interval. By running the `sar` program as a background process just before starting the execution of any test, I capture the resource demand placed into the system by them. I present the results from these measurements in figures 3.18 - 3.20.

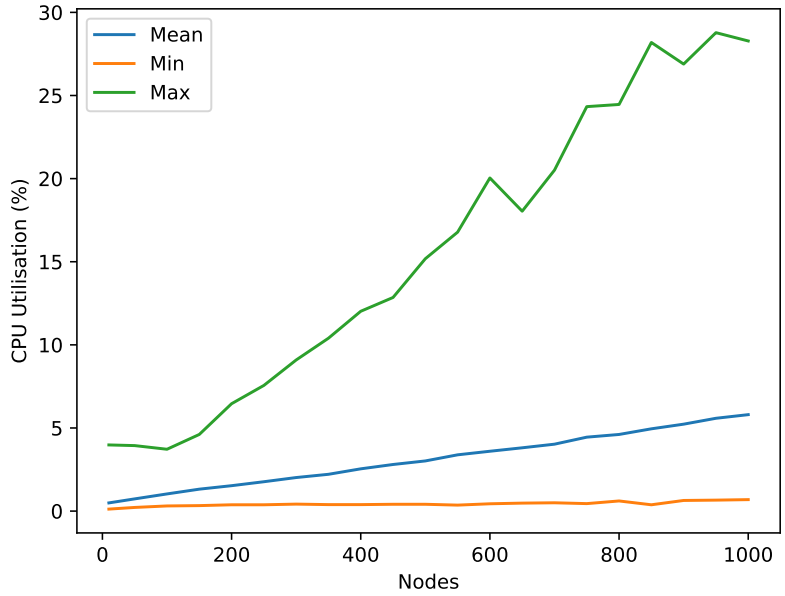


Figure 3.18: Aggregated CPU usage.

Naturally, one of the most important metrics to look at is CPU utilisation. Figure 3.18 shows the minimum, maximum, and average aggregated CPU demand of the system for the duration of the test. Notably, the max usage for all test runs corresponds to the startup period of the simulation, where all nodes require CPU time to perform their own initialisation tasks. For most of the tests, however, CPU utilisation remains close to the minimum as expected, since no workload is being processed.

However, this is not enough to demonstrate the nodes are not constrained by CPU demand. While evidently my server had more than enough CPU capacity to allocate the demand of the tests, that is not the end of the CPU demand analysis as the main adversary for a real-time simulator, is the OS scheduler. Since the number of worker nodes created (10-1000) quickly outpaces the number of CPU cores (60), all nodes need to share the cores to meet their processing demands, putting them at risk of lagging. Which begs

the question, how can one guarantee that all of them received as much processing time as they needed in a timely manner?

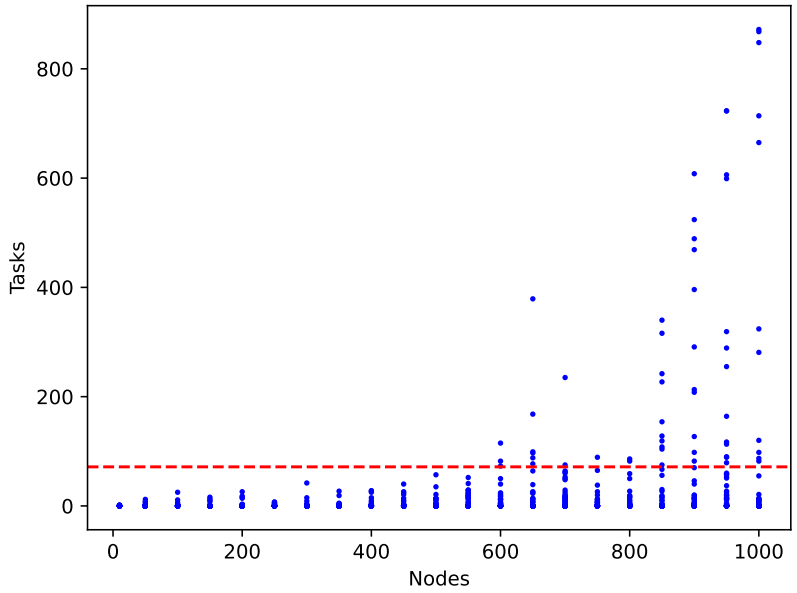


Figure 3.19: Run-queue occupancy.

The best way to measure this supply problem is to look at the run-queue occupancy of the OS. Contrary to popular belief, this queue keeps track of all the processes in the running and runnable states [112], not just those waiting to run. The rule of thumb is then, that as long as the run-queue occupancy remains less than twice the number of CPU cores (120), one can be certain that no process was starved for processing time [164]. Thus, by looking at figure 3.19, we can see that server's capacity is more than enough to meet the CPU demands of all nodes as long as the total node count remains smaller than 600 nodes. This number would likely be increased significantly by improving the Worker's design to use fewer threads.

Lastly, figure 3.20 shows the memory usage for the experiment in relation to the node count. As it can be appreciate, this demand is grows linearly with the number of nodes and remains stable, since Rust does not have a garbage collector or other memory-managed subcomponents. We can also see that the maximum demand (approximately 1.75GB) is far less than the available memory in the system, and hints at the fact that each worker node requires around 1.75MB of memory,

In conclusion, this chapter not only presented MeshSim, a new hybrid simulation/emulation platform with testbed capabilities for MANET protocol development; but it also evaluated the system's capabilities and limits, and showed that it can adequately provide all the required building blocks to build and evaluate MANET routing protocols with it. Thus, the following chapter will be dedicated to show the work I did using MeshSim.

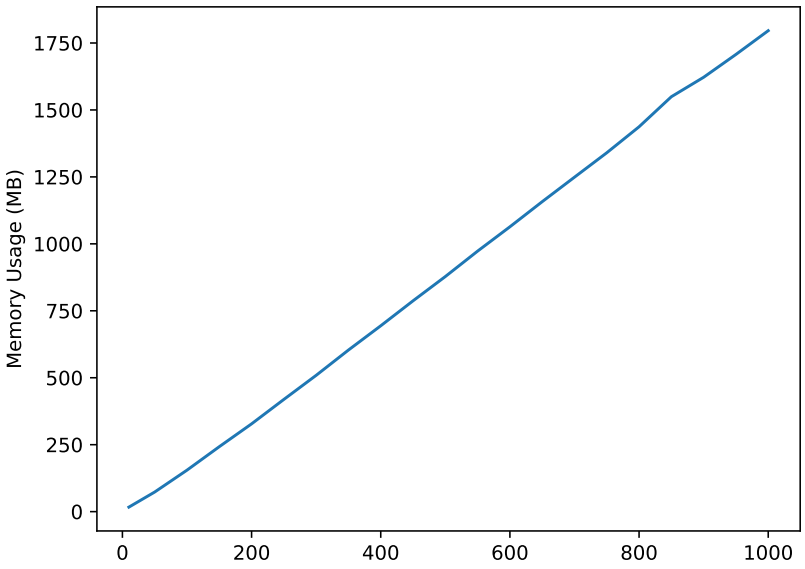


Figure 3.20: Memory utilisation.

Chapter 4

Reactive Gossip Routing

After introducing the problem of routing in MANETS (chapter 1), postulating the impact of fault tolerance in this domain (chapter 2), and presenting the tools I built to attack this problem (chapter 3), I dedicate this chapter to describing my contributions to the field. The protocols presented in this chapter used the lessons learned from the literature to improve their fault-tolerance even under harsh conditions, while keeping a low operational overhead. But first I dive into their details, allow me to contextualise the design choices that went into these protocols by recapitulating the problem space.

In section 1.1, I postulated that the interaction of the unique properties of MANETs lead to frequent failures, and postulated that avoiding such failures should be a driving principle of designing protocols for these networks. Afterwards, I narrowed my attention into a subset of these properties that I called the *MANET requirements* (summarised in table 2.1). I consider these four requirements (Mobility, Power-consumption, Medium Contention, and Scalability) as the most impactful properties to a routing protocol's performance, and that researchers should always account for them when designing a new protocol. And now, I would like to momentarily bring your attention to what is likely the most disruptive of all MANET requirements: *Mobility*.

The contributions of mobility to the faulty operation of MANET routing protocols is notable because it interacts strongly with all the other constraints. For starters, the *power-consumption* constraint only exists because mobility prevents nodes from being tethered to a continuous power supply. It also causes link breaks as nodes either move out of range from each other or break line of sight. These breakages can lead to network partition, clustering issues, and increase *medium contention* as nodes attempt to recover from them. But the most important that mobility imposes to MANETs is *transitivity*, as all logical structures created in the network such as routes, spines, neighbour sets, etc. are only temporarily valid because of it. In short, it makes difficult to calculate effective routes, but also to know when it is worthwhile to do so as many routes become invalid before they are used.

Given these difficulties, it is only reasonable to question if it is reasonable to build routes in settings with mobility. After all, as discussed in section 2.5.1, there are other alternatives available to accomplish data delivery for MANETs such as flooding. The basic flooding technique is quite simple, when a source node wants to deliver a message to another node, it broadcasts it to all nodes in range. All receiving nodes forward the message to their neighbours at most once, a process that is repeated until the message finally reaches its intended destination. While functional, one can immediately see this mechanism's inefficiency as it would create many replicas of the message throughout the network, even by nodes far removed from the source and destination. However, it is precisely this property what allows flooding to deliver messages under high mobility rates [62].

The problem with flooding lies in its cost. Assuming a connected network of N nodes, every node in the network but the destination forwards the message exactly once resulting in a *transmission cost* of $C_m = N - 1$. When multiple nodes transmit simultaneously using flooding, it can lead to a *broadcast storm* [42] in which medium contention, collisions, and retransmissions can collapse the capacity of a network. However, even when the worst-case scenario is avoided, *naïve flooding* is still a very costly operation in terms of network capacity, which is a precious and limited commodity [46].

Naturally, several ways have been devised to improve the message efficiency of flooding such as *gossiping* [60]. Also known as *epidemic propagation*, gossip is a forwarding technique that aims to reduce the number of replicas generated by flooding while keeping its powerful reach. It accomplishes this goal by taking a probabilistic approach in which every node decides to forward a message or not in accordance a *gossip factor* P $(0 \leq P \leq 1)$. This gossip factor modifies the cost function of flooding a message m in a network of N nodes to $C_m = P \times (N - 1)$, thus, reducing it by a factor of $1 - P$. Given a common value of $P = 0.72$ to maintain a network reachability close 100% [60], Gossip does not improve the cost of flooding nearly enough to make it more attractive than routing. However, given that flooding is also a common operation performed by routing protocols, it does provide me with the initial inspiration to design a routing protocol that can remain operational under high mobility.

4.1 The Reactive Gossip Routing algorithm

In this section, I describe the design and operation of the Reactive Gossip Protocol (RGR) I propose as a solution for fault-tolerant routing in general-purpose MANETs. My intuition to achieve this goal was that by mitigating the four MANET requirements, the resulting protocol would be naturally fault tolerant. This mitigation is effected via RGR's unique set of features: low-cost directional-gossip-based route discovery, flexible braided routes,

and secondary long-range radio.

Given the state-of-the-art review I provided in section 2.7, I leaned strongly towards starting off with a reactive blue print in mind, as proactive protocols have too high of an overhead to scale out efficiently. However, I also knew that reactive protocols can become inefficient as well due to the expensive flooding operations involved in route discovery. Although the use of gossip provided me with an initial clue on how to mitigate this cost, I also suspected that would not be nearly enough.

The problem with reducing the cost of route discovery is that it is in essence a distributed search operation which requires that at worst, we visit all the nodes in the network. But even in the average case, it also burdens us with finding a way to control the propagation of search messages even if the destination is found early in the process. Following the principles of dynamic programming, I decided that if I could not avoid performing an expensive operation, then I could at least reuse its result as much as possible. For route discovery, this meant that I could take advantage of all paths to the destination discovered by a single query.

By fusing some of these paths together I create a type of multi-path route that I call a *braided route* which are naturally fault tolerant. Further, by leveraging the message replication principle of flooding within my braided routes, I achieve a natural load balancing within the paths as well as increased resiliency to mobility for a small overhead price. Having these long-lasting routes also means that the overall cost of route discovery is amortised over time, making it cheaper to discover new routes when needed than to spend control packets for maintenance. This allows it to cope well with failure and handle mobility with ease.

I dedicate the next subsections to describe each of RGR's four phases: route discovery, route establishment, data transmission, and route teardown.

4.1.1 Route Discovery

The route discovery process starts when an application wants to send data from the current node to another node on the network. At that point, the *source* node crafts a *RouteDiscovery* message and broadcasts to its immediate neighbours. Each *RouteDiscovery* has a unique *RouteID*, and all nodes keep a cache of processed routes in order to discard duplicates.Table 4.1 describes the anatomy of a RouteDiscovery packet.

RouteSource	RouteDestination
RouteID	Route Length
Route[ ]	

Table 4.1: RouteDiscovery message format.

When a node receives a *RouteDiscovery* message, it makes a gossip-style forwarding

decision. If successful, route accumulation is used and the forwarding node appends its ID to the tail of the *Route* field. If not, it drops the message and the gossip simply dies out at this node. The gossip is adjustable via two parameters:

- P, the probability that this node will forward a route-discovery message.
- K, the minimum number of hops before the P factor is applied.

The gossip process shows a bimodal operation based on the success of the *early* gossiping decisions. If too many early gossiping decisions fails, the packet will almost certainly fail to reach most nodes in the network. Otherwise, it almost certainly will reach all nodes. A way to ensure the gossip does not die out too early is to propagate it for at least K number of hops.

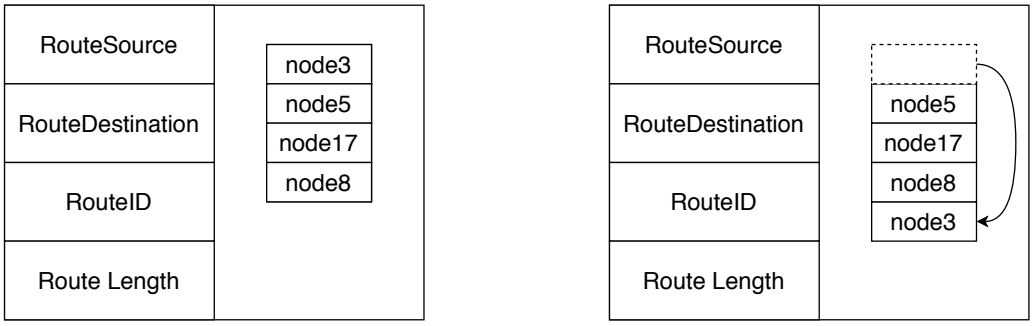
If the *RouteDiscovery* message reaches its intended destination, a *route establish* process will begin, described in the next section. If it does not, the *source* node can decide whether to retry or give up after a *route discovery timeout* period. Of course, this discovery process can result in a destination being reachable via many different routes, and RGR takes advantage of this.

4.1.2 Route Establishment

Once a RouteDiscovery message has reached its intended destination, it will reply with a *RouteEstablish* message. The *RouteEstablish* message has the same format as *RouteDiscovery* (see Table 4.1), except that it is tagged as RouteEstablish. The route establish process is essentially an unwinding of the route collected in the discovery, and as such, it relies on bidirectional links being available between nodes. The protocol could support operating over unidirectional links by performing a reverse route discovery from the destination to the source, but that feature is not considered in this initial version.

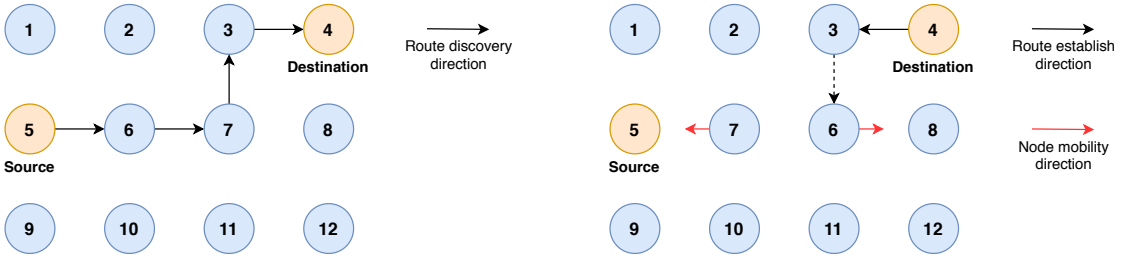
As illustrated in figure 4.1, when a node receives a *RouteEstablish* message, it examines the top node of the route. If it matches the current node, it subscribes to this route by adding its ID to its *KnownRoutes* list. Afterwards, it shifts the route list to place itself at the bottom of it and forwards the message. If the IDs do not match, it drops the packet.

This processing of a *RouteEstablish* message may seem arbitrary, but it follows a specific rationale. While the processing node could simply search the route list and subscribe to it if it finds its ID at any position of the list, this would imply the route is being processed in a different order in its upstream trajectory that was downstream. The out-of-order processing could be innocuous, but most likely it is a sign of mobility in the nodes that comprise the route (see figure 4.2). Given that a successful route is bound to have multiple paths in it, I chose to not let this potentially unstable path be established to favour better paths.



(a) Before processing (b) After processing

Figure 4.1: Processing of a RouteEstablish message by *node3*.



(a) RouteDiscovery node order (b) RouteEstablish node order

Figure 4.2: A failed route-establish process due to unstable/mobile nodes.

When a RouteEstablish message reaches its original source, the discovery process is considered complete. The source node subscribes to this route, and maps it to the destination, such that if new data requests arrive for that destination, it can leverage the same route. Afterwards, the node signals the upper layer that is ready to start sending data, or starts transmitting any queued *DataMessages* for the route's destination.

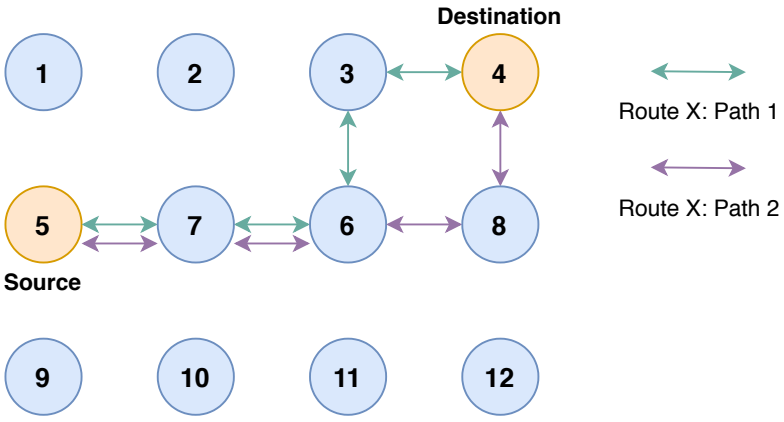


Figure 4.3: A braided route with two different paths.

As mentioned in the previous section, multiple *RouteDiscovery* messages may reach the destination. While the node could simply drop repeated messages or choose the shorter route, it actually crafts a response message for each one. The reason for this

is path redundancy, as all successful *RouteDiscovery* messages must have reached the destination via a different path. The resulting *braided route* is composed of all the paths that successfully winded back to the source (see figure 4.3). In a braided route such as this one, if either *node 3* or *node 8* were to become unavailable, the route could still forward data messages to the destination.

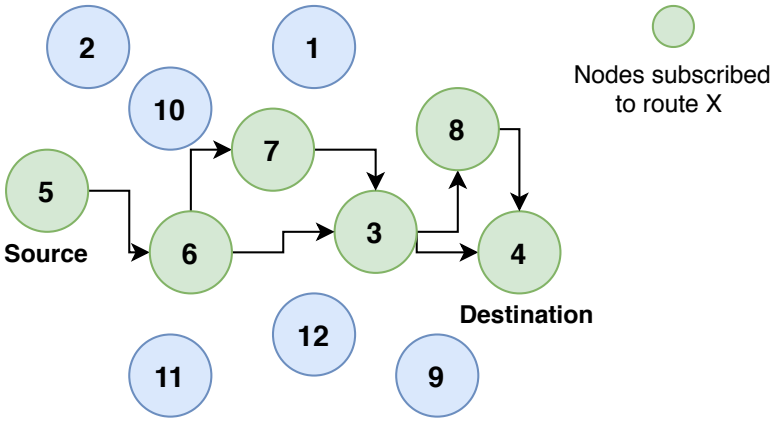


Figure 4.4: Route X has become shuffled over time due to mobility of its nodes.

While the route establishment considers the order in which the discovery happened, this order only matters at this stage for the purpose of discarding unstable paths due to mobility. Once the route has been established, however, the assumption is that mobility is the norm, and not only can it break routes, it can also shuffle them (see figure 4.4). However, this is of no consequence for RGR, as the data forwarding process is not affected by it. This out-of-order forwarding gives the routes further resiliency to mobility.

4.1.3 Sending Data

Data messages (see table 4.2) can start flowing once an appropriate route is established, and remains available. Upon reception of a *Data* message by a node, it is accepted if meant for it, forwarded only if the current node subscribes to the associated route, or dropped otherwise. If the intermediate node only subscribes to a different route to the same destination, the packet is discarded, since there is no guarantee that alternative route is still valid. Each *Data* message contains a message id, which is calculated by the source node by hashing the source, destination, timestamp and payload. This id is used to ensure all nodes process each *Data* message only once.

Destination	RouteID
PacketID	Payload Length
Payload	

Table 4.2: *Data* message.

RGR's route discovery process is very similar to that of various multipath routing protocols, but differs significantly from them when it comes to data forwarding. When a packet reaches a route junction in the forwarding process, instead of choosing a path, it creates packet replicas, similarly to the technique used by the Duct protocol [10], developed for PRNET [4]. The purpose of these replicas is to introduce resiliency through redundancy to the routes, a tradeoff that might seem out of place for RGR, given its low-overhead philosophy, but is one that pays off. While a more conservative protocol would first attempt to forward a packet through one of the available paths, and upon failure, try another one, RGR takes advantage of all available paths simultaneously, bypassing error detection (see figure 4.5). This allows the packet to proceed downstream faster, makes it more responsive to dynamic topologies (due to mobility), and reduces the management overhead at the junction nodes.

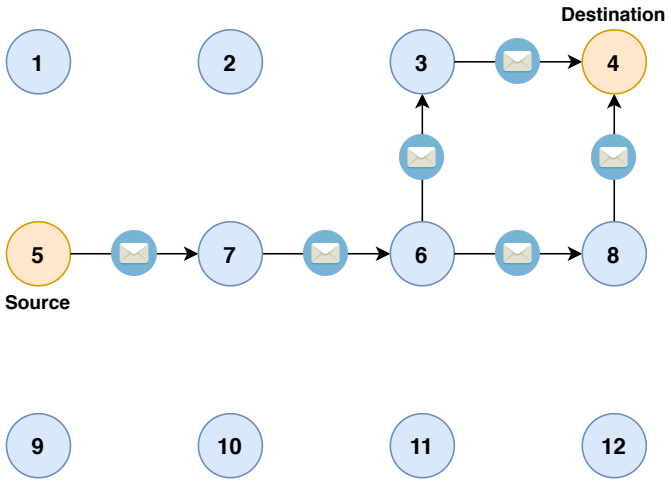


Figure 4.5: A packet in flow gets duplicated at node 6.

Further to the redundancy gained by the braided routes, they also provide a degree of *load balancing*. Given a network state in which all nodes have the same workload, and equivalent link qualities as far as propagation delay and reliability go, the shortest path between two nodes should be the first one to be established. When *Data* messages start to flow, they are likely to favour the shortest path as well. However, as the data flow continues, the load in shortest-path nodes may increase, leading to an increased forwarding delay. In this scenario, the replicas through the alternative paths can deliver the *Data* messages faster, balancing the traffic through the paths and eventually alleviating the load of the shortest path (see figure 4.6).

Finally, RGR keeps a cache of recently transmitted messages (MC), which it uses to discard replicas and audit the health of the route. This auditing consists in utilising passive acknowledgements to ensure *Data* messages are progressing successfully downstream. If a message is not confirmed to have moved downstream after a *message retransmission timeout*, the node can either retransmit it, or decide the route is broken and start a tear

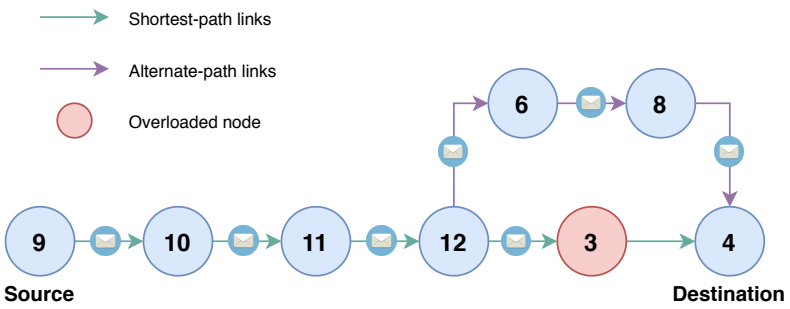


Figure 4.6: Load-balancing provided by alternative path.

down process.

Now that the main mechanism to transmit data packets between nodes in RGR has been established, I can analyse its cost function in relation to Gossip flooding. Since the process of sending a message m to destination d entails first discovering a route to d, and then have all the nodes in the route forward the *Data* message, the cost function would be:

$$C_m(d) = C_R(d) + \mid R_d \mid -1 \tag{4.1}$$

where

$$C_R(d) = \begin{cases} 0 & \text{if } d \in \text{KR} \\ (\mid R_d \mid -1) + P \cdot (N-1) & \text{if } d \notin \text{KR} \end{cases}$$

$$\begin{aligned} KR &= \text{Known-routes set.} \\ R_d &= \text{Set of nodes in the route to } d. \\ N &= \text{Number of nodes in the network.} \\ P &= \text{Gossip factor.} \end{aligned}$$

As appreciated in equation 4.1, the cost of transmitting a single message from a node to a new destination is $2 \cdot \mid R_d \mid + PN - P - 2$, whereas the cost function for gossip flooding ($P \cdot (N-1)$) is lower. However, when transmitting a second message to the same destination, RGR's cost is reduced to $\mid R_d \mid$. To answer the question postulated at the beginning of this chapter, routing remains a more desirable delivery mechanism due to cost. For all the adaptability flooding provides (gossip-based or not), the transmission cost per packet compounds very rapidly and surpasses the overhead of building routes.

4.1.4 Route Teardown

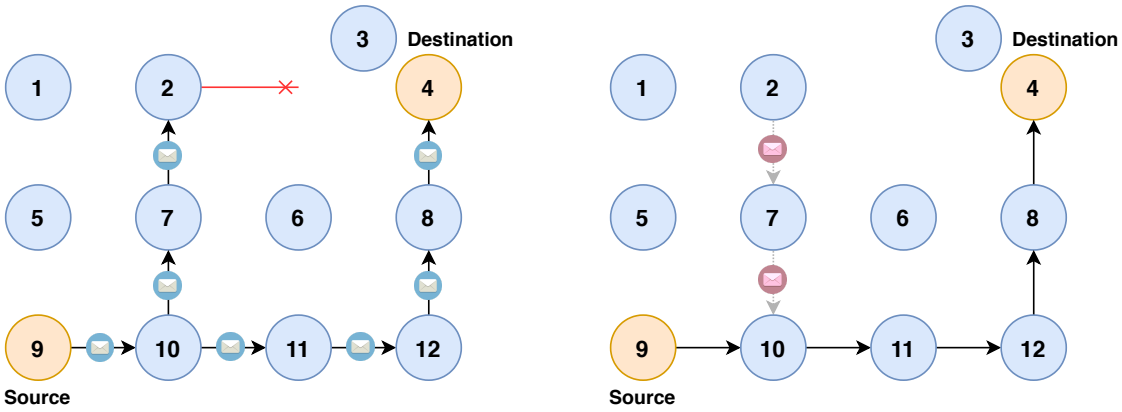
RGR's philosophy is to keep control packets to a minimum. As such, it avoids operations such as tearing down a route once it is done being used. However, when an *active route* is found to be broken, it does start a tear down process. A route is deemed to be active when at least one data packet is in transit through it. An active route is considered broken

when no confirmation is received that a *Data* message progressed downstream, at any hop of the route. When a breakage occurs, the detecting node unsubscribes from the route and crafts a *RouteTeardown* message (see table 4.3). This message contains the id of the route and the packet that failed to be confirmed. Optionally, it can also add the ids of other packets pending confirmation for that same route. This message is then propagated upstream.

RouteID	PacketID
Add_Packets_len	Additional Packets

Table 4.3: *RouteTeardown* message.

When a node receives a *RouteTeardown* message, it checks if it is subscribed to the route in question. If so, it unsubscribes from the route, checks if it has any unverified packets for it and appends them to the list, and finally forwards the message. This teardown process continues back to the source, at which point it can choose to start a new route discovery process. Further, it will obtain a list of the ids of all packets that were not delivered, which it can use to retransmit them if applicable. If the teardown process somehow does not propagate all the way back to the source, then the next time a *Data* message is sent from the source, a new teardown process will start at the last accessible node in the route.



(a) A broken link is detected in one path. (b) Path torn-down to the junction node only.

Figure 4.7: Route teardown process downstream from a junction node.

A special case is considered for the processing of RouteTeardown messages at junction nodes (see figure 4.7). If the route break occurs at one of these paths, but others remain available, it is undesirable to break the full the route. In order to detect this scenario, the *message cache* used for passive ACKs has a counter associated with each entry. If a node receives more than one passive ACK after forwarding a data packet, it can assume it is a junction node and the message has progressed downstream through multiple paths. As such, if a route break occurs downstream from it, the junction node must receive as

many route *RouteTeardown* messages as passive ACKs it received for the data message in question.

4.2 RGR-II: Probabilistically reducing routing overhead

The RGR protocol has the goal of providing low-overhead routing for MANETs while coping with mobility as well as possible. To achieve this, it uses a gossip-based query/response mechanism for route discovery, and creates braided routes that are more resilient than traditional ones. Finally, it attempts to keep control traffic to a minimum by not performing route maintenance, and opting to rediscover routes instead, since the cost of this operation is reduced from traditional flooding.

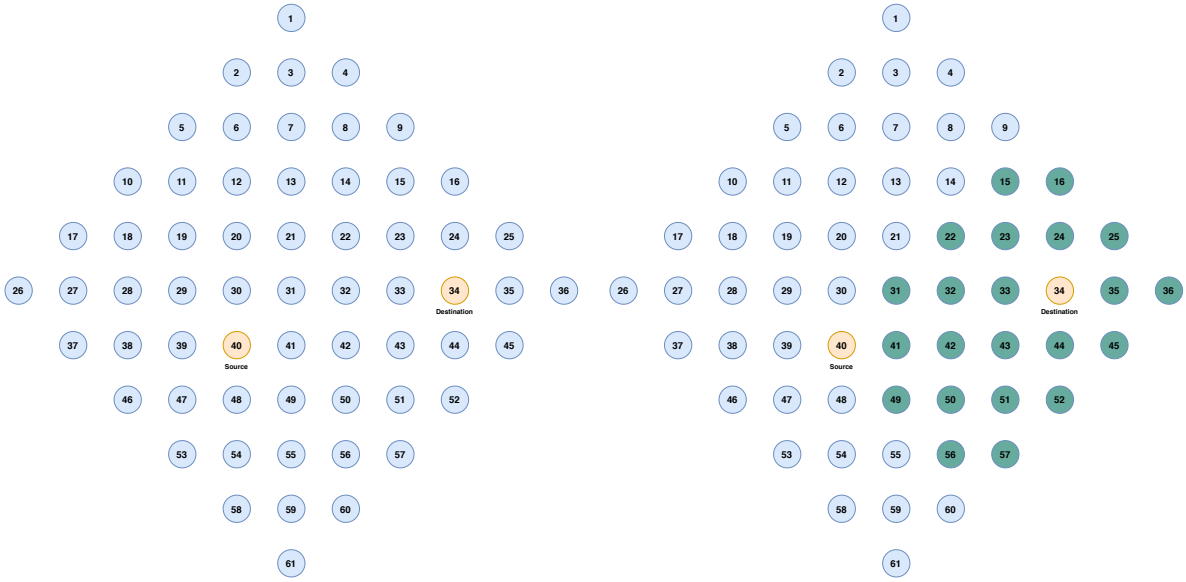
The cost of a single RGR route discovery ($C_{RD} = P \times (N-1)$) is intuitively cheaper than a flooding-based discovery ($C_{RD} = N-1$), since $0 \leq P \leq 1$. However, it clearly still grows linearly with network size, and its effects on contention are compounded with concurrent discoveries. This puts into question whether the route discovery is cheap enough for RGR to leverage it instead of a route maintenance procedure.

4.2.1 Leveraging bimodal operation

In section 4.1.1, it was mentioned that gossip propagation has a bimodal operation. If it succeeds to catch on early, it reaches all nodes with high probability, but if the gossip fails to propagate early, it almost always dies out quickly. To escape this bimodal operation, Haas et al. propose the use of the P and K parameters, which make sure the gossip always catches on. Is that necessary, though? While the rationale behind this is that the intended destination could be *anywhere* in the network, and therefore it is necessary to ensure all nodes are reachable by the discovery process, the destination is not *everywhere* at once.

Consider figure 4.8a. In this example, node 40 wants to discover a route to node 34. Several potential paths are available for this route, and by gossiping, RGR can explore most of them. However, not all nodes are equally useful in this discovery process. Through several iterations of this scenario, it is highly likely that only the nodes highlighted in figure 4.8b would participate in any routes between nodes 40 and 34.

Based on how useful a node is considered for forwarding data messages between two nodes, they are categorised as either *active* or *inactive* for route discoveries between them. The main idea behind RGR-II is to leverage gossip's bimodal operation such that the gossip only progresses in the *active region* of nodes, and dies out early in the inactive portion of the network. This is achieved by dynamically adapting the *gossip factor* (P) of each node based on the conditions of the network. When successful, not only does this



(a) Node40 wants a route to Node34.

(b) Useful routing nodes highlighted.

Figure 4.8: Subset of useful nodes for a routing scenario.

significantly reduce the overhead of the protocol, but *localises* the route discovery traffic as well. Note that if the network in figure 4.8 grew by hundreds of nodes added to its edges, the set of useful nodes would remain *mostly* same. Such a change would decouple the discovery process from the size of the network, and tie it only to the useful region between source and destination.

4.2.2 Updates to RGR

The *active region* of nodes introduced in the previous section is akin to what the DREAM [37] protocol achieves, as they are all the nodes in the *direction* of the intended destination, as viewed by the source node. However, the DREAM protocol uses a GPS device and constant flooding of the network to produce these calculations, which contravenes RGR's philosophy. In lieu of that, RGR-II introduces the proxy concept of a *vicinity*. Not to be confused with a neighbourhood (nodes at most i nodes away), the vicinity of a node n refers to all nodes that n considers to be reachable through itself. As such, when a *RouteDiscovery* message reaches a node, the probability that it forwards it is affected by whether it considers the intended destination to be in its vicinity.

The reachability estimation for the vicinity of a node is obtained by passive examination of the traffic it receives, without incurring further communication costs. Of course, given the dynamic nature of MANETs, this information is only assumed to be valid for a given period of time. The rules for updating the vicinity set of a node are:

1. For every message received: Add the *sender*.

2. For every RouteDiscovery message received: Add every node in its *route field*.

3. For every RouteEstablish message received: Add every node from the top portion of its *route field* (from the destination to the current node).

All updates to the vicinity set are timestamped and purged periodically after a *freshness threshold*. The gossip factor incorporates the vicinity data in the following way:

$$P = \begin{cases} P_0 & \text{if } d \notin \text{V} \\ P_0 + P_V & \text{if } d \in \text{V} \end{cases}$$

Where

$$V \quad = \quad \text{Set of nodes in the current node's vicinity.}$$

Where P_0 is a base probability, and P_V is the vicinity probability. Although both parameters are configurable, their default value follows that RGR-II's $P_0 + P_V$ equals RGR's P. This configuration ensures that the gossip propagation only decreases if a node is not considered in vicinity, and at most, the overhead is as large as that of RGR.

While this new metric is quite promising as a solution to reduce the cost of route discoveries, it suffers from a bootstrapping problem since its effectivity is directly linked to the temperature of the network. That is to say, the ability each node has to determine if another node is in their vicinity is dependant on having a fresh and well populated vicinity set. Given that this set is populated from network activity, this might prevent the network from reaching a state in which the heuristic shows its full potential. Fortunately, there are several ways to mitigate this effect, or even circumvent it altogether.

4.3 RGR-III: Warming the network

For the *vicinity* heuristic to work properly, each node requires data about the other nodes reachable by it. When a node considers another node to be reachable, it implies that it is relatively close to it, that at least one path between them has good-quality links, and that the intermediate nodes are not too overloaded. Further, this information is only temporarily valid. While it seems difficult to devise a way to infer all these data from other means, the first requirement for it is topological closeness.

Several of the protocols I reviewed in chapter 2 relied on topological data for route calculations. The methods to obtain such data are varied, but most of the time rely on some form of *beaconing*. These beaconing protocols periodically transmit a packet with different types of data, which are used by all nodes to learn about the topology of the network and make routing decisions. Allowing nodes to learn about their local topology in this way could help RGR-II to solve the cold-network problem, it would severely impact its goal of being efficient in both routing overhead and power consumption.

The main problem with learning about the network's topology through beaconing is that it has to be at least partially flooded. A beacon that is never forwarded can keep the impact to overhead localised and constant, but it is only useful to learn about immediate neighbours. Other techniques could be used to control the propagation of a beacon, such as the FishEye [51], but these mitigations are not enough to meet RGR's efficiency goals.

For most of the design decisions of RGR, I have used a lower protocol overhead as a proxy for lower radio-contention. After all, if nodes make significantly fewer transmissions, there are fewer chances for them to compete for the medium. However, the contention only happens because these concurrent transmissions are all made on the same frequency. Using radios on the same frequency is a necessity for communication, but there is no reason why only one radio should be used.

4.3.1 Multi-radio systems

This concept of integrating multiple radio interfaces in a single system was already introduced in this thesis in chapter 2. The CGSR [30] protocol recognises radio interference as a major obstacle to network scalability. As such, authors strongly suggest the configuration of adjacent clusters to different radio channels to reduce interference, or even use multiple radio interfaces. Multi-radio systems were also notoriously studied by researchers at Microsoft in the early 2000s.

In the first of those papers, Adya et al. [73] introduced the Multi-radio Unification Protocol (MUP). This link-layer protocol utilises *homogeneous* radio interfaces set to *orthogonal* (non-interfering) channels to reduce interference between adjacent nodes and maximise network throughput. This follows the concept that by having two non-interfering interfaces in the same node, throughput can theoretically be doubled by enabling receiving and transmitting simultaneously.

In a following paper, Draves et al. [79] introduce the Multi-Radio Link Quality Source Routing (MR-LQSR) protocol. In MR-LQSR, *heterogeneous* radio interfaces are supported (802.11a and 802.11b), noting that the diversity in bandwidth, range, and fading characteristics of their respective bandwidths can be exploited to increase robustness. This link-state protocol introduces a routing metric called *Weighted Cumulative Expected Transmission Timeout* (WCTT), which favours channel-diverse paths to intended destinations. The effect of this channel diversity allows it to form paths in which each hop can utilise links from either radio interface, both to achieve high bandwidth and minimise the systemwide interference.

Finally, Bahl et al. [74] presents a summary of the findings by the Microsoft Research team into multi-radio systems. In this paper, they consider the use of heterogeneous radios to separate control and data traffic. In their Control-On-LPR system, they incorporate a low-powered radio (LPR) and a high-powered radio (HPR) into a single system. These

bandwidth capabilities of each radio correspond to their power consumption, and as such, the HPR is preferred for data transmissions. However, instead of having the HPR in idle state waiting for responses, it is turned off and the LPR listens instead, waking up the HPR when it is time to receive data. They remark that the integration of the LPR significantly reduces power consumption of their system. All of the Microsoft protocols were designed for community networks with no mobility, and demonstrated the viability of integrating multiple radio interfaces into a single protocol.

4.3.2 Integrating a second radio into RGR-III

RGR-III uses a combination of the techniques presented in the previous section to ensure the network always has enough data for the *vicinity* metric to work well. By moving the *beaconing* to a secondary radio, it can ensure it does not incur further radio contention in the network. However, this separation of radio frequencies for the new control packets does not mean overhead is not added to the protocol, or that power consumption is not affected. In order to mitigate both of these effects, the secondary interface chosen is a LoRa radio [143].

LoRa (Long Range) is a spread spectrum technique based on chirping spread spectrum (CSS) developed by Semtech Corporation [179]. Radio interfaces that implement the LoRa modulation (or simply LoRa radios) have a long-range and low power consumption, as they are originally designed for Internet of Things (IoT) devices which are severely constrained in power reserves. The working range for LoRa radios in rural areas is reported to be between 15 km-30 km [129]. Those extreme ranges, however, are subject to very specific configurations (antennas, power gain, deployment space, etc.) My experimental calibration (see section 3.6.2) measures the working range my LoRa NICs around twice as that of the onboard Wi-Fi cards of the experimental nodes.

The radio properties of LoRa are then ideal for the purposes of RGR-III. Without the need to forward any packets, nodes can learn about their *medium range* neighbourhood. Further, given this range, it is unnecessary to send these beacons too often, which minimises the contention on the LoRa medium. Finally, the low-power nature of the radio minimises the impact it has on the power consumption of the protocol.

4.3.3 Updates to RGR-II

The RGR-III protocol consists of the incorporation of a secondary radio into the operations of RGR-II. To utilise the secondary radio, a new type of control message is introduced, named the *HELLO* message. Every *hello threshold* milliseconds (configurable), the protocol will build a HELLO message and broadcast it over its *long-range* radio (the LoRa radio). The only thing contained in the HELLO message is the id of the transmitting node. Upon

reception of such a message, nodes update their *vicinity* set to increase the count of messages received by that node.

In the next section, I will evaluate all three variations of RGR to test their individual features.

4.4 Evaluation

In this section, I describe the experimental procedure I used to evaluate RGR and present the respective results. As I established earlier in this chapter, RGR's goal is to provide effective routing services under faulty conditions, and I highlighted mobility as one of the main contributors to such conditions. Therefore, through a series of experiments, I measured the effectiveness of all RGR variations to route packets under varying mobility settings. Since a protocol's ability to route packets is known to become compromised as mobility increases, I determine how apt a protocol is at this task by measuring how much its performance diminishes with this increase. However, for those values to be meaningful, they must be put in context.

To this effect, in each experiment I test all RGR variations as well as three other protocols that I use as baselines for comparison. The first two such protocols are *naïve* and *gossip* flooding. I include them not as real competitors to RGR but as upper bounds in terms of efficiency, since they are both well known to be able to deliver packets under high mobility but at a very high overhead cost. The choice of which main protocol to compare RGR against, however, was complex for multiple reasons.

As I established in previous chapters, while research in MANETs has splintered in the last couple of decades based on application domains, RGR is a general-purpose protocol. This design choice was deliberate, as the gap in the literature it intends to fill is to serve as a better baseline for application-specific protocols to compare against. This is an important role to fill, as some of the most influential protocols in domains such as Vehicle-to-Vehicle (V2V) communications [103][115][120] and Flying Ad-Hoc networks (FANET) [126][146] first established their ground by comparing their performance against general-purpose MANET protocols such as AODV, OLSR, DSR, etc. Thus, it makes more sense to compare RGR against these established incumbents rather than any particular application-specific protocol.

Given all of this, I believe AODV [43] is ideal as a comparison point for multiple reasons. It is a reactive routing protocol that shares many of RGR's goals, since it attempts to keep a low overhead and cope well with mobility by reacting to topology changes quickly. However, in a sense, it also spouses many of the design aspects opposite of RGR, such as a expensive route discovery and maintenance, complex state management, and a very state-dependant operation of its nodes. Further, it's influence in all application-specific

protocols is undeniable, since it is the most cited MANET routing protocol, has an established [71] and an upcoming [150] RFC, and researchers keep trying to adapt it to many new scenarios.

All three experiments under which I test these six protocols are simulation-based and ran with MeshSim (see chapter 3). Each experiment contains a simulation scenario designed to test different specific conditions. Within the experiment, the scenario is executed independently ten times per protocol to filter out outlying data points. Afterwards, high-level performance metrics are extracted from the accumulated data of all runs of each protocol and used to compare them with each other.

Each simulation scenario is defined by four main components:

- **The general simulation parameters** - This includes things like duration of the simulation, area that nodes can move through, and most notably, the **protocol** that will be run, which is the only thing that differentiates test specifications for different protocols in the same experiment. The full list of parameters is given in section 3.4.2.1.
- **The network topology** - The network topology refers to the number and position of all nodes as well as the links between them, which is given by a radio range parameter calibrated in accordance to the results of the experiment in section 3.6.2.
- **The mobility model** - This parameter controls how fast each node moves and under which rules.
- **The workload** - The workload is defined as a series of constant bit-rate (CBR) sources with a predetermined destination. At a time configured per source (but all after simulation warm-up), each source starts producing packets of 64 bytes at a constant rate of five packets per second. The workload is identical for all experiments in this section.

The details of each experiment are given in the next three subsections, followed by all the results.

4.4.1 Stable grid experiment

The purpose of this experiment is to establish a performance baseline-measurement for each of the tested protocols under near ideal conditions. The topology of this scenario consists of 100 nodes arranged in a ten by ten grid as depicted in figure 4.9 with a stationary mobility model (e.g. no mobility). In this grid-like arrangement, nodes are placed like cells in a grid and only have links to adjacent nodes (left, right, top, and bottom). This

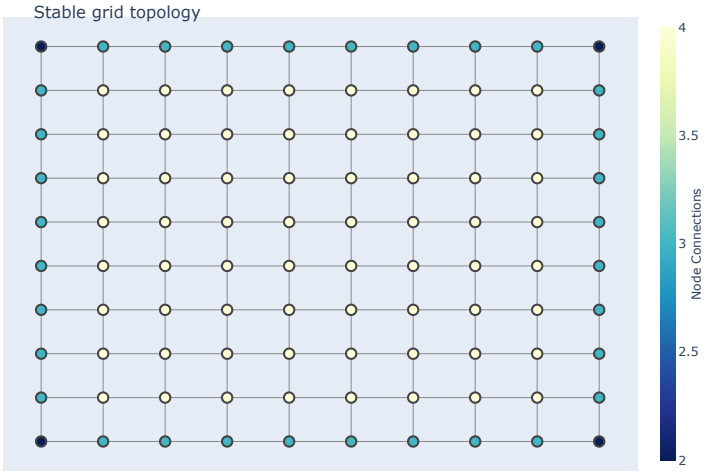


Figure 4.9: A grid-like arrangement of nodes.

ensures all nodes have between two and four links each, which provides variety in paths, but also isolates the effects of contention from clustering.

Each simulation run lasts 60 seconds and is configured to have a workload of ten CBR sources. These CBR sources act as an upper layer that requests the network stack to transmit data packets of 64 bytes each at a rate of *five packets per second* for a duration of *20 seconds*, producing a total of *100 data packets* per source. Likewise, the network stack of each node utilises the protocol it was configured with to figure out how to deliver each of these packets to their intended destination. The start times of all CBR source are distributed randomly and uniformly, such that none of them start before the *warm-up* period finishes or after the *cool off* period begins, and they all have enough time to transmit their workloads. While this topology is unrealistic for any real world deployment of a MANET, it has many advantages. The size of the network was chosen to be large enough to create non-trivial routes and observe the difference between active vs passive nodes, while remaining small enough to be easily visualised. The visualisation allows me to make predictions about the performance of the protocols, such as the available paths between nodes, or the potential contention hot zones for some given some workload.

4.4.2 Random Mobility experiment

With the performance baseline established by the Stable Grid experiment, the Random Mobility experiment seeks to explore how this baseline is affected by mobility. As such, only the mobility model and topology change from the previous experiment. In this experiment, nodes are placed following a uniformly distributed random topology (see figure 4.10).

After all nodes have been initialised, they begin to move through the simulation

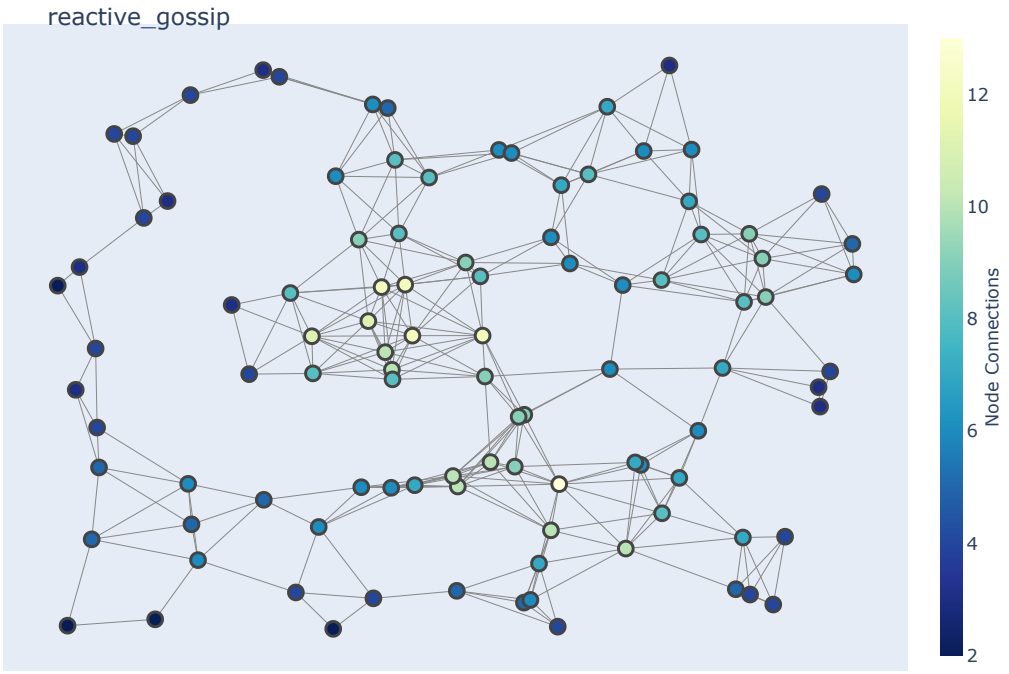


Figure 4.10: Initial topology of the random mobility experiment.

area following a random waypoint model (RWP) [64]. The speed and direction of each node is dictated by their velocity vector, which is randomly sampled following a normal distribution around the average walking speed of humans [29]. Once a node reaches its destination, it remains there for a configurable *pause time* before resuming mobility with a newly calculated random velocity and destination This process continues until the end of the simulation.

I chose this mobility model because it allows me to get an idea of the performance trend of the protocol for arbitrary network sizes and mobility patterns, which is well suited for general-purpose MANET routing protocols like RGR and AODV. In contrast, *realistic* models (e.g. trace-based) do not offer this flexibility [106], and further, the mobility of their nodes is affected by complex interactions with each others. For instance, mobility models based on car traces follow the rules of the road, and models based on the mobility of people follow social interactions and so forth, which makes these models for suitable for application-specific protocols.

4.4.3 Increased mobility experiment

In this experiment, I explore more deeply the effect that mobility has on the performance of routing protocols by iteratively increasing the mobility rate of the simulation scenario presented in the random mobility experiment. To do this, I present a variation of the random waypoint mobility (RWP) model that I call the *Increased Mobility model*. This

model functions very similarly to RWP with one exception: When a node reaches its destination, instead of sampling a new random velocity, it calculates it by multiplying the magnitude of its current velocity by a configurable factor (10% for this experiment). However, the next destination of the node is still chosen at random.

Two more changes in this experiment has to the with the experimental procedure and configuration. For starters, the pause time of nodes is reduced from the 10 seconds used in the random mobility experiment to 5 seconds. Secondly, unlike the previous two experiments, simulation runs for a given protocol are not identical as each subsequent receives a 10% increase to the initial velocity of all nodes. The rationale behind these changes is that, although the random mobility experiment gives a good preview of the effects mobility has on routing protocols, any faults observed are still the result of the interactions of many variables, not just mobility. But as mobility slowly increases with this model and inflicts ever harsher conditions on the routing protocols, it eventually overtakes all other variables and allows us to see just how much it can affect their performance.

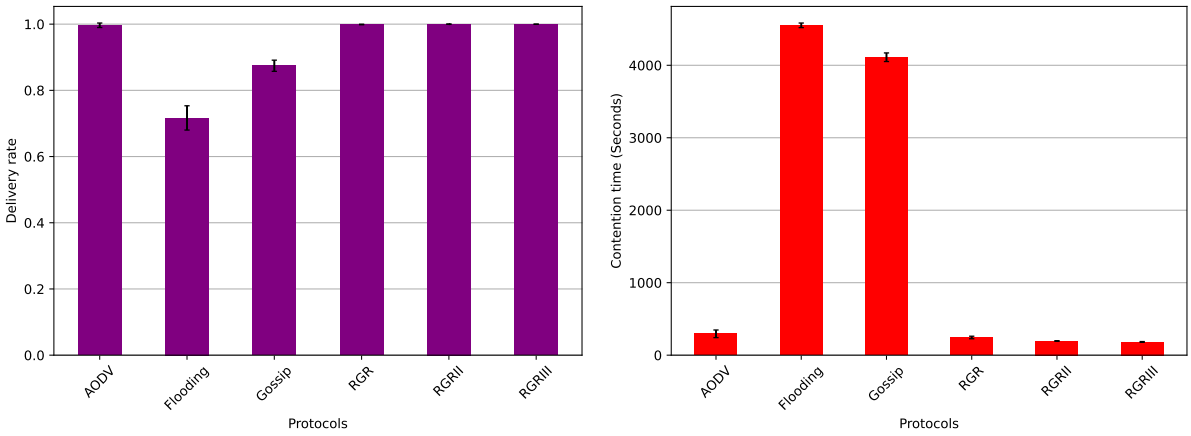
4.4.4 Results

Through all three experiments, the same metrics are extracted from the simulated data and used to compare all six protocols: delivery rate, efficiency (overhead), medium contention, and latency. These metrics were selected as they represent some of the primary aspects that all networking protocols should provide, and as proxies to measure the effectiveness of the protocols to deal with the MANET constraints. The source code, configuration files, and instructions to reproduce all experiments are available in the thesis companion repository, alongside all raw and processed data.

4.4.4.1 Stable grid

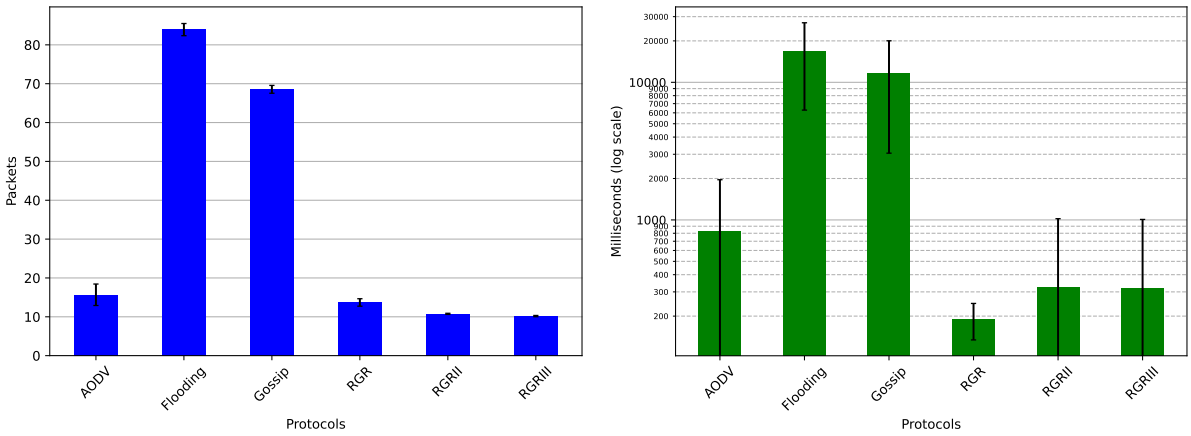
The first set of results we will look at, are those for the stable grid experiment, depicted in figure 4.11. The most important metric to consider is *delivery rate* (figure 4.11a), as the primary job of routing protocol is to in fact, deliver data packets. As we can appreciate here, all routing protocols (RGR variations and AODV) have no problem delivering close 100% of their packets, as the ideal conditions of the grid are conducive to that. The upper-boundary baselines (Flooding and Gossip) are not as effective in this regard, which is clearly explained by looking at their high levels of medium contention in figure 4.11b, which leads to dropped packets.

Another significant contrast between the baselines and the routing protocols can be appreciated in figure 4.11c, which depicts the total number of packets needed to deliver a single data packet. So, on top of having a lower delivery rate, Flooding and Gossip achieve their respective numbers at a much higher cost than the routing protocols. Lastly



(a) Data packets delivery rate.

(b) Aggregate contention time.



(c) Per-packet overhead.

(d) Per-packet latency

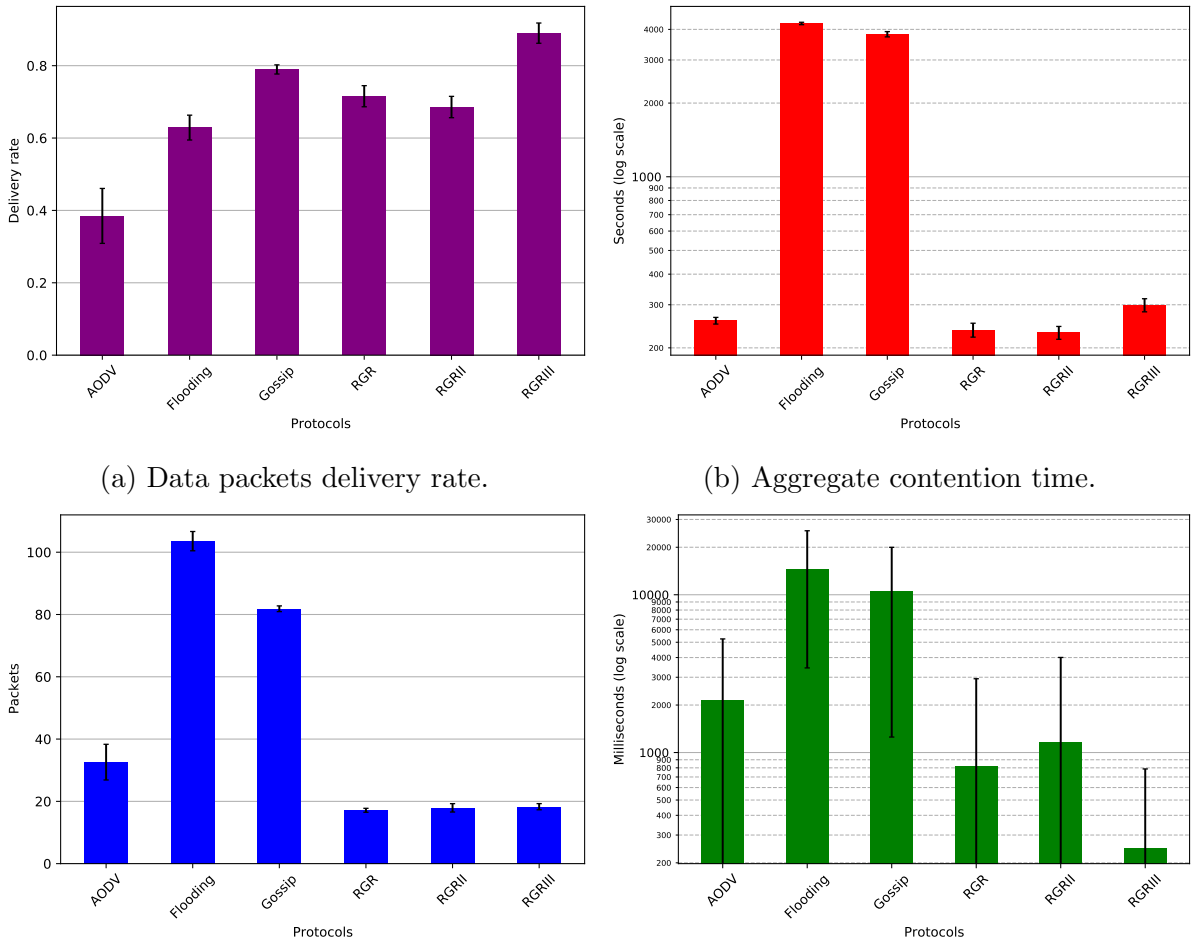
Figure 4.11: Stable grid experiment results.

figure (4.11d) shows the average latency per data packet, from the moment its payload is received from the upper layer, to the moment it is received at the target destination. While this value is much lower for the routing protocols, they display a much larger standard deviation, something that is to be expected since the first packet for a route will wait much longer to be delivered than subsequent packets as it waits for a route to be acquired. Also to be noted, this is the first metric in which the RGR variations start to show a notable improvement over AODV.

As for the RGR variations themselves, their performance is almost identical on every metric with the exception of latency, which is to be expected under such ideal conditions. These ideal conditions are also more conducive to RGR's basic path finding algorithm, preventing the heuristics from RGR-II and RGR-III from showing their potential. All in all, however, all of these results are exactly what we expected to see and in-line with my theoretical framework.

4.4.4.2 Random mobility

Things start to get interesting when we look at the results for the *random mobility experiment* depicted in figure 4.12. This time, figure 4.12a (delivery rate) is packed with interesting information. Firstly, it is evident that the effectivity of all protocols decreased as we expected. But that is hardly the case for the baselines, which very low drops in their delivery rates, confirming their ability to deliver packets under mobility without issues. But most notably, are the delivery rates of RGR-III and AODV. Under this harsher conditions, RGR-III's heuritic allows it to keep its delivery rate around 85%, while all the other variations drop into the 70-80 percentile. On the other hand, AODV's performance plummets dramatically under 40%, confirming my theoretical analysis from chapter 2 that contrary to what their authors expected, it does not handle mobility well. The reason behind this performance drop are explored in depth in the next chapter.



(a) Data packets delivery rate.

(b) Aggregate contention time.

(c) Overhead per packet.

(d) Protocol latency

Figure 4.12: Random mobility experiment results.

Moving on to overhead per packet (figure 4.12c), we can see the efficiency of all entries increase notably. However, this is understandable as they all need to produce further control

traffic to account for the failures that mobility induces. Total contention (figure 4.12b) actually goes down for all entries, which is to be expected since all transmissions carry a degree of medium contention and, since they deliver significantly fewer data packets, they also experience a reduced aggregated contention.

Lastly, we can also observe notable changes in latency per packet (figure 4.12d) for all protocols but RGR-III. The reason for this increase is simple: None of the CBR sources is now able to delivery all 100 of its packets using a single route before it breaks, pushing them to re-discover and adding further delay to the packets in the queue. Surprisingly, RGR-III's average latency actually improves slightly from 300ms to 250ms, showing the effectivity of its heuristic.

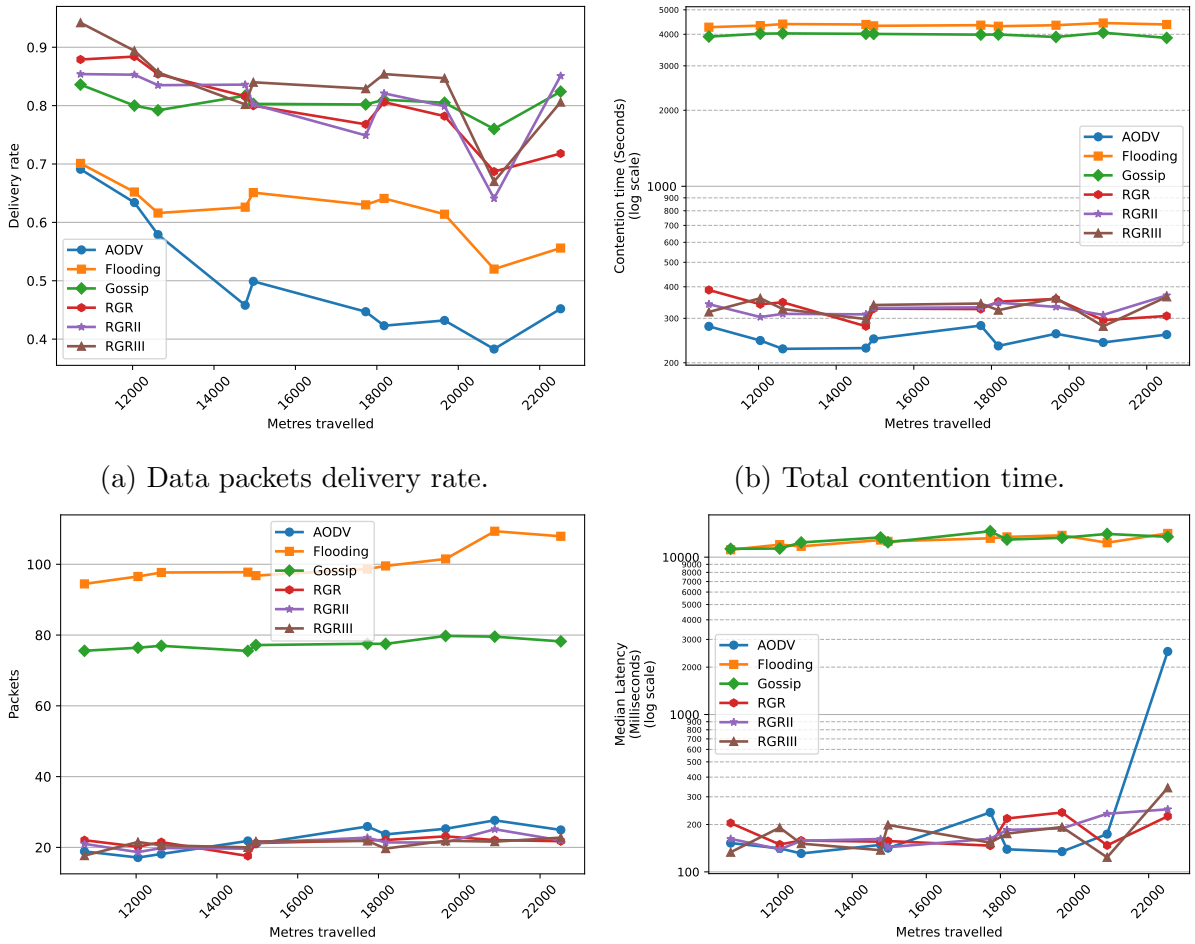
4.4.4.3 Increased mobility

Moving on to the last experiment, we take a look at the results for the increased mobility experiment presented in figure 4.13. As described in section 4.4.3, the objective of this experiment is to observe how as mobility increases, it affects a protocol's ability to deliver packets. Consequently, the average speed of all nodes is increased at every iteration. However, since these speeds are still randomly sampled, rather than classifying the iteration by an average speed, I do it by the collective distance travelled by all nodes, since its a direct measurement of how much the nodes were moving during a fixed set of time. The higher the total distance travelled, the higher the *mobility rate* was during the simulation.

Once more, we start this analysis by looking at the delivery rate (figure 4.13a), were we can see the clear downwards tendency of the delivery rate of all entries, but much more pronounced for AODV and Flooding. While all RGR variations remain more resilient to mobility's effect, we can see how their delivery rate starts to descend in the higher mobility rates and they actually become even with pure Gossip. This is unsurprising, since we knew from the start that Gossip and Flooding were very effective at delivering packets under mobility. Nevertheless, the RGR variations achieve their delivery rates at about 25% the cost of Gossip, as we can see in figure 4.13c. As for the other protocols, AODV's overhead increases but very slowly, while Flooding shows a clear upwards trajectory.

Lastly, we consider the effect of this increased mobility in latency (figure 4.13d). Although this metric seems to remain relatively constant as the mobility rate increases, we see a clear upwards trajectory for all entries, with some of them (AODV and RGR-III) even shooting up rapidly in the last iteration. This effect is observed since at these high speeds, it takes longer for route discovery processes to succeed (many more retries) and many more route discoveries over all are required, as more route breaks occur due to mobility.

To summarise this chapter, my initial hypothesis that inspired this work was that the unique properties of MANETs made it so that the most important aspect in protocol



(a) Data packets delivery rate.

(b) Total contention time.

(c) Overhead per packet.

(d) Protocol latency

Figure 4.13: Increased mobility results per mobility rate.

design should be fault tolerance. I further speculated that by designing a protocol that was naturally fault tolerant, all other classical network metrics such as delivery rate, latency, and efficiency/overhead would benefit. Lastly, I postulated an approach to fault-tolerant protocol design not present in the literature: By mitigating the individual factors leading to faults (medium contention, complex state management, expensive and strict control operations, etc.), I could produce a fault-tolerant protocol that did not require to specifically keep track of errors or try to force error-free network conditions. In this chapter, I presented the Reactive Gossip Routing (RGR) family of protocols that embody my previous hypothesis. By exposing my protocols to increasingly error-prone conditions, I demonstrated their ability to cope with these environments and keep a much higher level of service than both the baseline measurements and the incumbent protocol in this area (AODV), which is often used as a benchmark point of reference.

Chapter 5

The woes of shortest-path routes

In the previous chapter, I demonstrated through experimentation the effects that fault-tolerance can have in MANET routing protocols. These experiments (sections 4.4.2 and 4.4.3) compared my Reactive Gossip Routing (RGR) family of protocols which take fault-tolerance-first approach to their design to a state-of-the-art protocol (AODV) that favours other metrics. The effects observed from the data favour the fault-tolerant design, and range from improved delivery rate to lower medium contention, and even somewhat counter-intuitively, improved latency.

However, even though the results were inline with my expectations, I was still puzzled by the severity of the performance drop AODV suffers once mobility is introduced. Before delving into the analysis, however, the possibility of human error in the experiments must be considered. In particular, two factors should be taken into account: Potential misconfiguration of the protocol, and errors in the protocol implementation. AODV has multiple configuration parameters that can potentially alter its performance, such as `ACTIVE_ROUTE_TIMEOUT`, `NODE_TRAVERSAL_TIME`, or `HELLO_INTERVAL`. The values used for these parameters are the recommended defaults indicated in the RFC [71], and they can be audited in the corresponding configuration files for each experiment[1]. The version of AODV used in MeshSim was fully coded by me in order to conform with the MeshSim interface, and to be able to test it against the other protocols in the same conditions. This implementation was done based on AODV's RFC [71], all relevant sections of the algorithms are documented with the corresponding text and sections of the RFC, and it is fully open source for auditing[2]. So, although the possibility of misconfiguration or errors in the code is always latent, I am confident both are true to specification. Finally, all data (raw and processed), experimental artefacts (scripts for orchestrating the experiments and processing the data), and the source code are open sourced for verifiability.

To start this analysis, let us consider AODV's delivery rate across all three experiments

[1] See the experiments repository for more details: `https://github.com/Dash83/PhD_Research`

[2] The MeshSim repository contains all versions of the code, and the university repository has the version used for all experiments in this thesis.

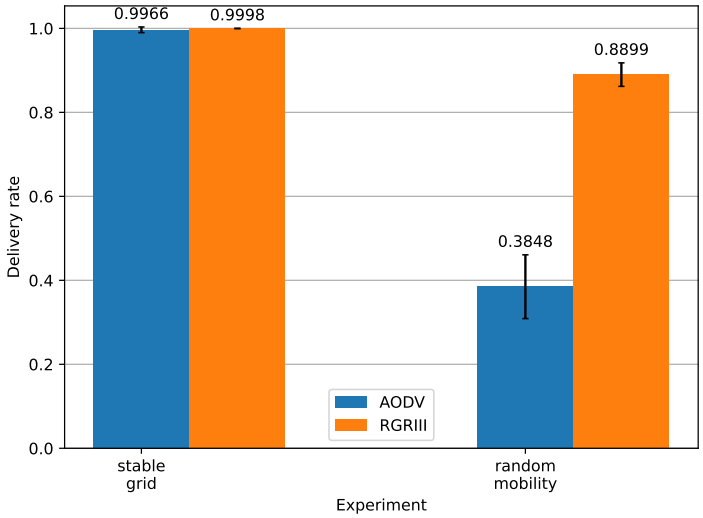


Figure 5.1: Mobility's effect on Delivery Rate.

from chapter 4. As we can see in figure 5.1), AODV's delivery rate decreases significantly once mobility is introduced, an effect that is only accentuated when compared to RGRIII's performance. And, although AODV's aggregated delivery rate seems to improve slightly in the increased mobility experiment, figure 4.13a showed a clear downward tendency in delivery rate as mobility increased.

It is intuitive to see how a higher mobility rate would make any path created in the network be shorter-lived as the rate of change of the topology (e.g. mobility) increases. However, this is the case for all protocols, not just AODV. What then makes AODV so susceptible to mobility? My theory: Inflexible node sequences and shortest-path routes.

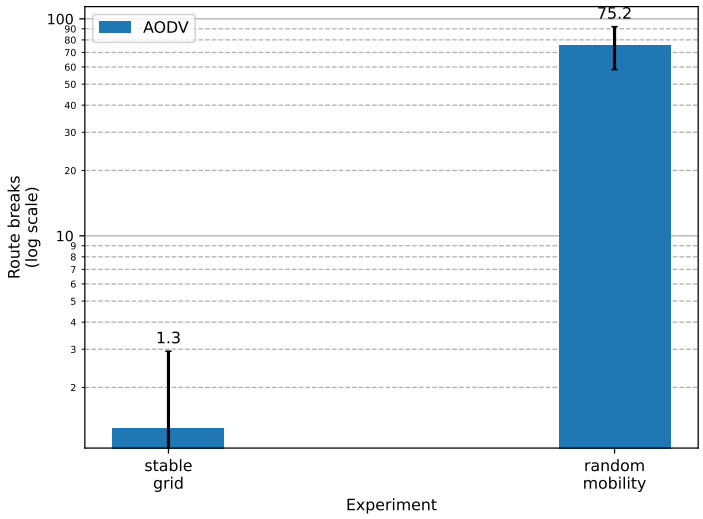
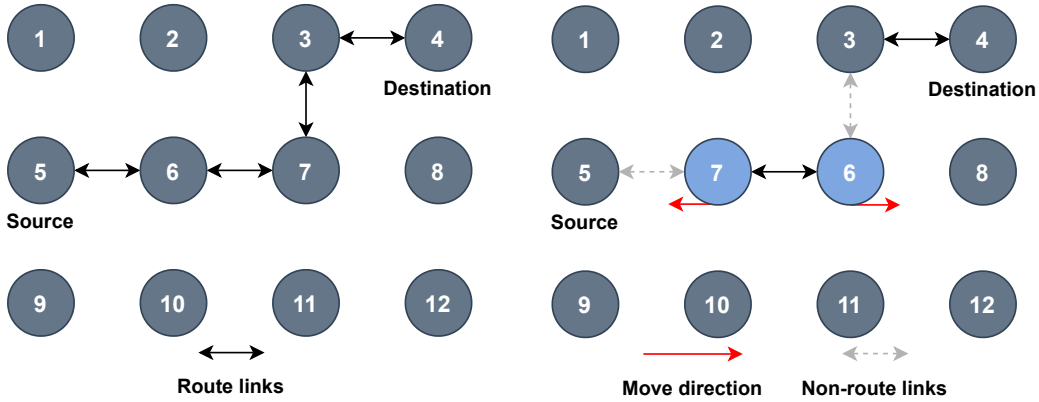


Figure 5.2: AODV's route breaks across experiments.

At each hop of the route-query process AODV uses for route establishment, intermediate nodes record the previous and next hops along the path that they will use when forwarding

data packets between the source and destination. By *inflexible node sequences*, I refer to the fact that packets will only flow in the strict sequence in which the route was established, regardless of viable alternatives. This effect is illustrated in figures 5.3a and 5.3b. An AODV route is established in figure 5.3a and packets flow through it in the specified node sequence. But, as time elapses, mobility of nodes 6 and 7 has them switch places, breaking their previous links. As illustrated by the grey arrows of figure 5.3b, they remain within range of the rest of the route nodes. Nevertheless, since packets are unicasted in an expected order, this route will be reported as broken once either node 5 is unable to reach node 6, or node 3 is unable to reach node 7.

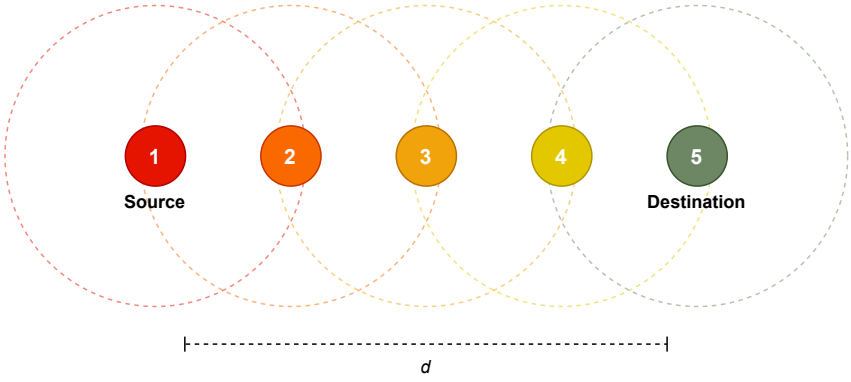


(a) A common AODV route.

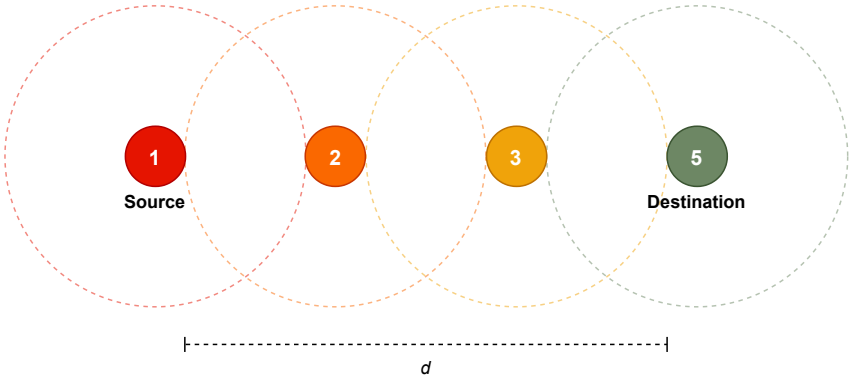
(b) Mobility breaks the route.

This strict node sequence is not arbitrary, as it may be useful for a number of purposes, but it can also become a liability as explained above. For instance, knowing which are the downstream nodes for all known routes can help a node react quickly to a broken link, and either attempt to repair the route or report it as broken. However, this imposes an overhead regardless of the action taken as well as a pause to data-packet flow, both of which could be avoided. Another justification for the node sequence is the fact that not all links are equal and exchangeable. Loss rate and bandwidth can vary significantly between links due to a plethora of factors, even with identical wireless NICs. And while the default hop-count metric used by AODV does not take link properties into consideration, keeping the link sequence in all routes opens AODV for adaptation to more sophisticated routing metrics.

In regards to shortest-path routes being partially at fault for AODV's plummeting delivery rate in the face of mobility, what I refer to is the *distance-minimisation* effect that shortest paths induce when used in conjunction with route metrics such as hop count. As it was described in section 2.7.2.6, AODV is a reactive protocol based on distance-vector routing. The premise of distance-vector routing is to favour the creation and use of routes that minimise the distance travelled by packets. To this effect, most protocols do not actually minimise the geographical distance between the two endpoints of a route, but rather, a proxy metric used for distance: hop count. And while there is good reason for



(a) Sub-optimal route.



(b) Shortest-path route.

Figure 5.4: Two hop-count-based routes.

trying to minimise this metric (see section 5.1), it also induces an unexpected effect into the routes: shorter path duration.

To illustrate this effect, consider the route in figure 5.4a, with a fixed distance d between source and destination, three intermediate nodes, and the radio-ranges of all nodes depicted by dotted concentric circles. In such a route, the latency of transmitting a packet between source and destination is dominated by the number of times it has to be relayed by intermediate nodes, as each hop carries some overhead. In contrast, figure 5.4b covers the same distance d but with only two intermediate nodes, lowering the relay overhead of moving packets between source and destination. While they both cover the same distance, it is the second route that is considered shorter due to the hop-count metric.

The problem with distance-vector routing and mobility arises when considering the radio-ranges and positions of the nodes on both routes As nodes in the second route clearly sit at the edge of their respective radio ranges, even the slightest mobility of any intermediate node would break the route. This would naturally trigger additional control traffic to be generated to cope with the event, and potentially the loss of some data packets in the process. In contrast, the nodes of the first route have more flexibility for some local movement without breaking the route. In short, shorter-path algorithms using a distance-based metric favour the creation of routes with nodes spread further apart,

making them more susceptible to breakage by mobility.

This analysis led me to the research question that motivates this chapter: *Are shortest-path routes that use distance-based metrics suitable for MANET routing protocols?* This question is in itself both controversial, as the desirability of shortest-path routes is dogma in networking; and a well-known result, as other impactful work has hinted to this, but did not outright conclude it. However, it is clear to me that a case for it still needs to be made as even modern MANET routing protocols such as OLSR2 (2014) [123] and AODVv2 (2019) [150] still use hop count as their default routing metric. Naturally, before I present my proposal to address this problem, in the next sections I will provide further detail in the problem space and look at the state-of-the-art in this space.

5.1 Shortest-path routing

The shortest-path routing technique comes from the solution to the shortest-path problem in graph theory [1], which consists of finding a path between two nodes that minimises the sum of the weight of the edges. This problem is commonly used as an abstraction in several realms (such as computer networks), and at its essence is an optimisation problem that seeks to minimise the resources consumed by a path. In wired networks, the main resource they look to minimise is time (latency). Wired networking lines are much more reliable than wireless transmissions and are less concerned about actual distance between nodes, as it has little effect on packets. Thus, the primary contributors to the latency of a transmission are hop count, bandwidth, and propagation delay at each node (due to queuing and load). And while modern routing algorithms for wired networks have sophisticated metrics that take all these (and more) factors into consideration, hop count remains central to the algorithm's calculations.

Plenty of these factors are even more important for wireless networks in general and for MANETs. When a wireless node makes a transmission, it occupies the medium around it, preventing its neighbours from transmitting concurrently. Thus, a longer route would potentially prevent a larger number of nodes from transmitting, lowering the throughput of the whole network. Further, MANET nodes are battery constrained, so minimising the power consumption of the network as a whole (e.g. fewer transmissions per packet) is desirable. Thus, not only are shortest paths desirable for the source and destination of a route (since it provides less latency for their packets), but they are globally desirable for the network. In short, the constraints of MANETS expand the scope of the optimisation pursued by shortest-path routing from local only, to local and global.

The problem, however, is that the properties and constraints of MANETs introduced in section 1.2 negate these theoretical advantages. The following are the main reasons for this:

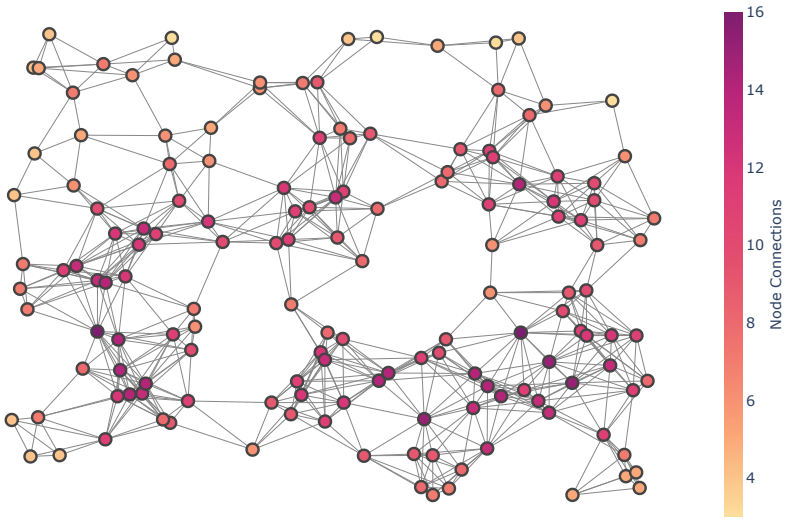


Figure 5.5: Node temperature by edge count.

1. Unlike wired networks, the distance covered by a wireless link is important, as the signal strength of wireless transmissions decays with distance due to path loss. This would be a counter-indication to the desirability to spread nodes as much as possible over distance to traverse it with fewer hops (see figures 5.4b and 5.4a).

2. For any given geographical distribution of a network's node positions, the nodes in the centroid of the distribution are likely to be part of most non-trivial shortest hop-count routes, as illustrated in figure 5.5. This presents two problems in itself. The oversubscribed nodes are likely to have longer queuing times, which increases packet latency; and a lack of spatial reuse, as the spectrum around them becomes congested while the periphery of the network is likely less contented.

3. As mentioned in the previous section, the mobility of MANETs can turn the asset of shortest paths (geographical spread) into a liability.

Several of these issues are well known in the literature. De Couto et al. do a good analysis of some of these issues in their publication "*Performance of multihop wireless networks: shortest path is not enough*" [66]. In this paper, the authors argue about the quality of the link over hop count as better metric for route calculation. They perform a set of experiments in a static wireless ad hoc test bed, in which they compare the throughput of shortest-path routes between nodes calculated with DSDV, and longer, static routes chosen manually based on link quality. Their experiments show significant throughput difference in favour of link quality, arguing that since hop count does not take link quality in consideration, it can lead to less than ideal route performance. Further, they argue they

found no correlation between signal strength and delivery rate, and their ideal metric for route calculation would be one that maximises signal-to-noise ratio (SNI).

As stated earlier, shortest-path routes do not necessarily build paths shorter in any terms of distance, but rather, minimise an accumulated metric along a path. It is the choice of metric, hop-count in this case, that provides the proxy for distance minimisation that as I have outlined above, becomes problematic for MANET routing protocols. Naturally, this begs the question, if not hop-count, then what metric should MANET routing protocols use (if any) to address the problems described in this section?

5.2 Routing metrics

Having established that shortest-path with hop-count technique is insufficient to consistently provide reliable and efficient paths for MANET routing, we know consider the alternatives in the literature. The motivation for some of these routing metrics align with my own, which is identifying hop-count a inadequate for the problem at hand, while other researchers were motivated by finding efficient ways to quantify the *goodness* of a wireless channel.

5.2.1 Round-trip time

One attempt to come up with a metric to quantify the quality of a link is the round-trip time (RTT) metric [73]. The concept behind RTT is simple: All nodes broadcast probe packets at fixed intervals and expect a response from their neighbours, calculating the total time it took for the round trip, and infer properties about the link's quality based on the delay. In a contended environment, nodes may experience delay in acquiring the medium, incurring a delay in the RTT. The same logic applies to having to retransmit a probe or probe response packet due to a lossy environment.

$$SRTT = \alpha * RTT_{new} + (1 - \alpha) * SRTT \tag{5.1}$$

Every RTT received from a given link is weighted against the weighted average of the previous RTTs, as described in equation 5.1. While the intuition behind RTT is good, the expectation of capturing all details of the link quality, and only the link quality, is flawed at best, since the RTT can be dominated by queuing delays at every node. Further, it suffers from a self-interference effect, as the overhead of processing all the probe packets can cause further queuing delays, completely skewing the metric away from the channel properties.

5.2.2 Packet-pair probe

An improvement over RTT is provided by the adaptation of the packet-pair probe (PktPair) [17] from wired networks into MANETS, as suggested by Draves et al. [77]. In order to minimise the sensitivity of RTT to queuing delays, PktPair suggests the periodic broadcasting of two back-to-back probe packets, followed by a measurement of the delay between them at the receiver's end. Since any load-induced delay applies to both packets, it is factored out by calculating the difference in arrival times between them, finishing with a reply of this value to the sender. Further, by making the first packet very small, and the second packet very large, an estimation of link-bandwidth can be incorporated into the measurement, as larger packets would take longer to be transmitted on links with poor bandwidth. Finally, the PktPair metric is smoothed out against its weighted average, just as with RTT.

A critique of PktPair, as pointed out by Draves et al. [77], is that it incurs an even larger control overhead than RTT, and it induces a different type of self-interference into the algorithm. While calculation of a link's PktPair is unaffected by queuing delays, it does incur an overhead on all nodes, which is measured in the PktPair metric of the other links to that same node.

5.2.3 Expected transmission count

Following their work in which they outline some of the limitations of shortest-path routes, De Couto et al. published their seminal work introducing the ETX routing metric [82]. Given that shortest path over hop-count can lead an algorithm to select poor-quality links, De Couto's further implies that these lossy links trigger retransmissions of lost packets since the link layer never receives acknowledgement of reception. Thus, the number of transmissions required to route a packet from source to destination is not actually the hop count, but a multiplier of it based on the number of retransmissions needed. The ETX metric incorporates the need for these retransmissions into its calculations, allowing an algorithm to choose paths formed with better links.

$$ETX = \frac{1}{d_f \times d_r} \tag{5.2}$$

where

$$\begin{aligned} d_f &= \text{Forward delivery-ratio} \\ d_r &= \text{Reverse delivery-ratio} \end{aligned}$$

$$r(t) = \frac{count(t-w, t)}{w/\tau} \tag{5.3}$$

where

$$\begin{aligned} t &= \text{Time} \\ w &= \text{Rolling time-window} \\ \tau &= \text{Probe-packet rate} \end{aligned}$$

The ETX calculation of a link is given by equation 5.2, and it is based on the bidirectional delivery ratio between the two ends of a link. Under ETX, all nodes produce probe packets at an τ interval (1 s by default). All nodes keep track of the number of probes they have received from their neighbours, and since they know to expect a probe packet per second, can therefore at any time t estimate the delivery ratio from a neighbour over the last w seconds. By considering both the forward and reverse delivery ratio of a link, the ETX link disfavours heavily asymmetrical links. Thus, the ETX of a route is the sum of the ETX metric of all its links. Algorithms using ETX as its route metric can then apply well known-shortest path algorithms on it, and obtain short and efficient paths.

5.2.4 Weighted Cumulative Expected Transmission Time

The weighted cumulative expected transmission time (WCETT) routing metric was published by Draves et al. [79] with the objective of procuring high-throughput paths for multi-radio systems. WCETT introduces the Expected Transmission Time (ETT) link metric as an extension of ETX that addresses one of its main weaknesses: Bandwidth considerations. While ETX does a good job at selecting high-quality links, it provides no guarantees regarding their bandwidth capacity. ETT's calculation is defined in equation 5.4

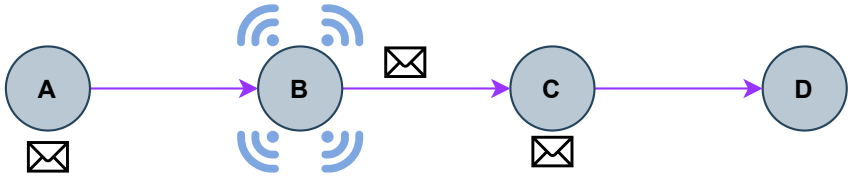
$$ETT = ETX * \frac{S}{B} \tag{5.4}$$

where

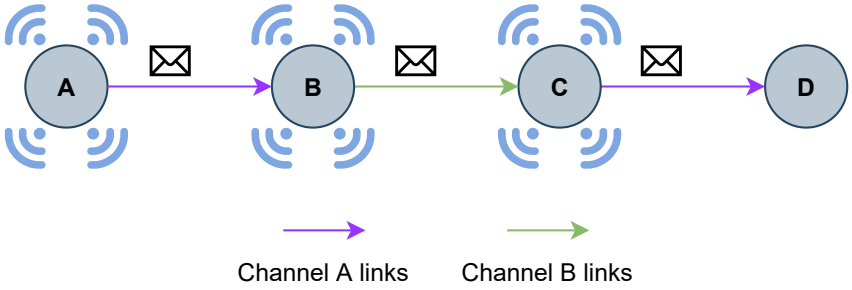
$$\begin{aligned} S &= \text{Size of the packet (in bytes)} \\ B &= \text{Bandwidth of the link (raw data rate)} \end{aligned}$$

By adding the bandwidth component, a link with 50% loss and 100 mbps would be preferred over a link with only 10% loss rate, but only 10 mbps.

WCETT was designed with multi-radio systems in mind, and aims to take advantage of all NICs in every node to produce radio-diverse paths in order to reduce interference. As packets flow downstream in a route, adjacent nodes contend for the medium and reduce the throughput of the route (see figure 5.6a). WCETT aims to address this self-interference by choosing paths with links transmitting in non-overlapping radio channels (see figure 5.6b). Thus, in a system with k different radio channels, the bottleneck channel of a route is the one with the higher count of links in the path. Taking this in consideration, the full WCETT route metric is a combination of the sum of the ETT metric of all selected links in the path, weighted against the bottleneck channel (equation 5.5).



(a) Nodes A and C are ready to transmit, but have to wait for node B to release the medium.



(b) Throughput increased with multiple radio channels

Figure 5.6: Self-contention effect in routes.

$$WCETT = (1 - \beta) * \sum_{i=1}^{n} ETT_i + \beta * \max_{1 \leq j \leq k} X_j \tag{5.5}$$

where

$$\begin{aligned} \beta &= \text{Configurable weight parameter} \\ k &= \text{Max number of channels} \end{aligned}$$

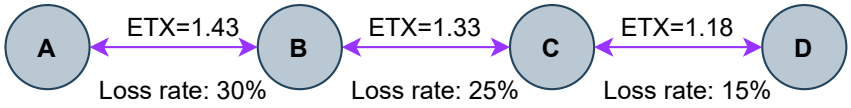
Using this path metric, an algorithm chooses the minimum WCETT among available paths to obtain a high-throughput route. Depending on the selected value for β, this metric can choose how much it favours channel diversity versus the best links for its own path. Authors of the metric describe this as a balance between choosing the selfishness and global good. If all nodes generated routes only based on the ETT sum, they would get the best path for itself, but at the cost of potentially introduce more contention and interference in the network. By choosing radio-diverse paths, the throughput of single routes might suffer slightly as longer routes might be selected, but it favours spatial reuse.

5.2.5 What is the better metric?

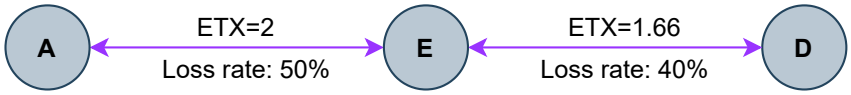
In the previous sections, I considered four alternative metrics to hop count for shortest-path routing: ETT, PktPair, ETX, and WCETT. Each of these metrics makes different decisions about how to infer further information regarding neighbouring links, and use that to make informed decisions about routing. Something else they have in common is that they are all increasing metrics (e.g. they do not produce negative values) and they

all grow numerically as links become less desirable. Therefore, any of them is suitable replacements for hop count in shortest-path routing, in which the minimisation of the given routing metric yields the most desirable route. But which one is best?

Draves et al. [77] perform a very thorough job of comparing most of these metrics in their wireless ad hoc network testbed. Through extensive testing, they ultimately conclude that ETX outperforms hop count, RTT and PktPair in most scenarios, except in their one mobile scenario, where hop count reacts better to topology changes. On the other hand, RTT and PktPair are too load-sensitive, and do not outperform hop count in most scenarios. In a following paper [79], Draves and his team also conclude that WCETT outperforms both ETX and hop count in single-radio scenarios, and significantly outperforms both in a two-radio scenario.



(a) Route ETX metric=3.94



(b) Route ETX metric=3.66

Figure 5.7: Comparison of two ETX route metrics

Nevertheless, I maintain that none of these metrics addresses AODV's susceptibility to mobility. As noted by Drave et al.'s own comparative study [77], while ETX outperforms all other metrics in most scenarios, it lags behind hop count in the only mobile scenario they tested, something that is already quite telling. The reason for this is, that while ETX and WCETT are quite sophisticated and provide a plethora of improvements over bare hop count, they have two weaknesses:

1. They both benefit from more stable topologies as that allows them to collect better link measurements.

2. But most importantly, **they still act as proxy metrics for distance**!

As described in their respective sections, both ETX and WCETT metrics of a link grow as the link becomes less desirable. And, since shortest-path routes always look to minimise the accumulated metric in question for a given path, it will naturally be biased to choose the paths with the smallest hop count as illustrated in figure 5.7. While both metrics consider link quality, they will still favour routes with nodes spread further apart, even at the cost of poor quality links. This selection makes them vulnerable to node mobility in the same way that hop-count-based paths are.

In the next section, I present my proposal to address this issue.

5.3 Adjusted-distance metrics

The routing metrics considered in section 5.2.5, while very sophisticated, are not sufficient to produce reliable routes in the presence of mobility for protocols like AODV. Although these metrics have some issues regarding the overhead they produce and the fact their calculations work better with more stable topologies, that is not the entire story. Initially, I blamed the hop count metric for producing routes with nodes spread apart as much as possible, making them susceptible to breakage by mobility. But in the last section I showed that none of the alternative metrics even considers distance in their calculations and they all end up likewise susceptible. Thus, it would be natural to conclude that the nature of shortest-path route calculations is partially at blame as well, as it drives this geographical-separation effect by favouring paths with the smaller cumulative metric sum. However, I believe this can be fixed by incorporating a measurement of distance between nodes to the calculation of most metrics.

The 802.11 standard [161] defines a field in the Wi-Fi headers called the Received Signal Strength Indication (`RSSI`), which is represented by a single byte, thus supporting values from 0 to 255. This field is a relative measurement of the strength of the radio signal of a transmission upon reception at the receiver, as reported by the wireless network interface card (NIC). The RSSI is used with a manufacturer-defined `RSSI_MAX` constant to provide the so-called `Link Quality` measurement observed in networking programs such as `iw`, or the deprecated `iwconfig` in the Linux operating system (see listing 5.1). These relative metrics are in arbitrary units, and are a proxy to report the received signal strength (P_{Rx}), expressed in decibel milliwatts (dBm).

Listing 5.1: Sample output of iwconfig.

```
wlan0   IEEE 802.11  ESSID:"Eduroam"
        Mode:Managed  Frequency:5.66 GHz  Access Point: 38:3B:C8:3E:D4:3A
        Bit Rate=867 Mb/s   Tx-Power=18 dBm
        Retry short limit:7   RTS thr:off   Fragment thr:off
        Encryption key:off
        Power Management:off
        Link Quality=46/70  Signal level=-64 dBm
        Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
        Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

The received signal strength is the power change caused in a receiver's antenna by a transmitter's signal after path loss. That is to say, the transmission power (P_{Tx}) of

an outgoing signal decays progressively until it reaches a receiver's antenna, becoming the P_{Rx}. The decrease in signal power caused by its traversal between transmitter and receiver can be estimated estimated with different models such as the Free Space Path Loss (FSPL), Log-Distance Path Loss (LDPL), or the ITU's General Delay Spread models [133] depending on whether the transmission occurs in open space, an urban environment, or indoors. Regardless of which model applies, however, both models calculate the power loss as a function of the distance traversed by the signal. Thus, both the $RSSI$ and the P_{Rx} of a signal can also be used as a proxy to estimate the relative distance between the transmitter and receiver as function of their radio range.

$$AD\text{-}Metric = \frac{\beta}{RSSI/255} \cdot (1-\beta)Metric \tag{5.6}$$

where

$$\begin{aligned} RSSI &= \text{Received Signal Strength Indication} \\ \beta &= \text{Configurable weight parameter} \end{aligned}$$

By incorporating either of these proxies into a routing metric, I believe it is possible to cancel the geographical-spread effect that shortest-path calculations induce. The general model of this adjustment which is call the *Adjusted-Distance metric* is given in equation 5.6. By multiplying the value of any of the existing metrics of a link by the reciprocal of the signal-strength percentage, it discourages the selection of links whose signal strength corresponds with a higher distance between the nodes. The strength of this effect is modulated by the β factor to support scenarios in which different weights are desirable for the adjust-distance (AD) and base-metric components. The ideal β value might be different for each use case and is thus left open for protocol configuration while using a default value of 0.50, giving each component equal weight.

This model two properties that I should not about this model. Firstly, the it is compatible with any of the metrics explored in the previous section, so one could easily to implement an AD-ETX, or AD-WCETT. Secondly, it does not incur any additional overhead, since the required data for calculation is already present in all packets.

5.3.1 AODV Adjusted-Distance

To test my adjusted-distance model, I decided to apply it to the hop-count metric. While clearly, there are better options, this choice allows me to test directly against the AODV results from chapter 4 with the knowledge that any difference in results is a direct effect of my model and not of a sophisticated base metric. Applying my model from equation 5.6 to hop-count produces equation 5.7, as the value of the base metric is simply 1 for each link.

$$AD\text{-}Hop\text{-}Count = \frac{1}{(RSSI/255)^2} \tag{5.7}$$

As one can appreciate in equation 5.7, the denominator of the fraction is changed to be the squared of the one depicted in equation 5.6. This adjustment came after early experiments showed me (see figure 5.8) that the base model was in fact too insensitive to distance, which prompted me to modify it. An interesting property of the AD hop-count metric is that it grows slowly until $RSSI$ approaches 0 at the limits of the transmission range. This means that links to nodes that are not close the transmission edge are barely modified, whereas nodes close to it are actively avoided. This is a highly desirable property, as those nodes near the edge are the ones that are more susceptible to break a route due to mobility.

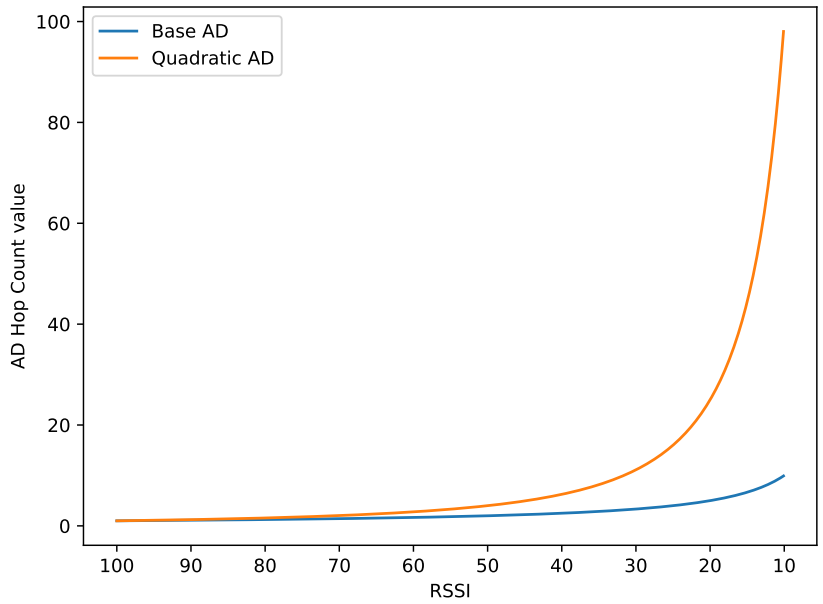


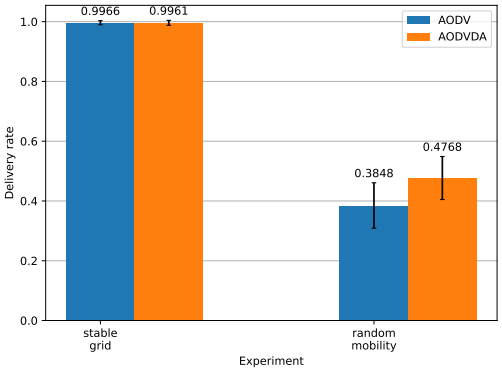
Figure 5.8: Plotting of the adjusted-distance hop count function.

I incorporate this new metrics into AODV and use it instead of hop count in all relevant operations, calling this variation AODVDA (Distance-adjusted). To support this new metric, I also modify the format of all control packets to use a new 4 bytes `message_cost` field instead of the 1 byte `hop_count` defined in pages 7-9 of the RFC [71]. This new field stores the calculation of the new route metric. Likewise, I modify the format entries in the `route_table` (page 13 of the RFC) to hold this new 4 bytes value.

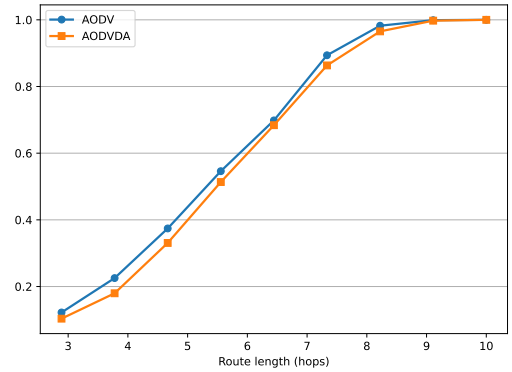
Other than that, no modifications are made to the operation of the protocol. In fact, the main modification to AODV's codebase in MeshSim came in the form of allowing the route metric to be chosen via the configuration file. Thus, AODVDA's implementation in MeshSim is actually just a wrapper around AODV that ensures the new routing metric is used, but the uses the exact same code as the AODV base version for everything else.

5.3.2 Evaluation

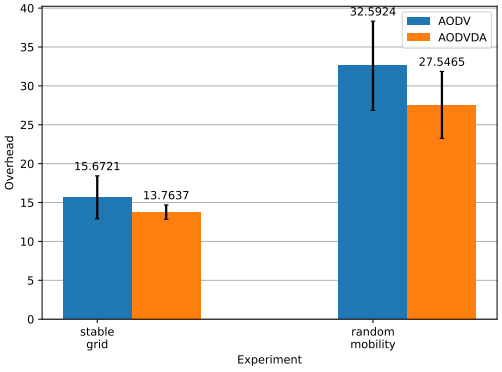
My motivation to explore this space lies in figuring out why AODV's performance suffers so vastly once mobility is introduced and to answer the research question I postulated earlier in this chapter. Thus the evaluation of AODVDA is straightforward. First, I run it in the stable grid experiment to obtain baseline measurements and compare it against AODV. Then, I use it in the mobile scenario described in the random mobility experiment, where I expect it perform better than AODV due to its improved metric. All these results can be seen in figure 5.9.



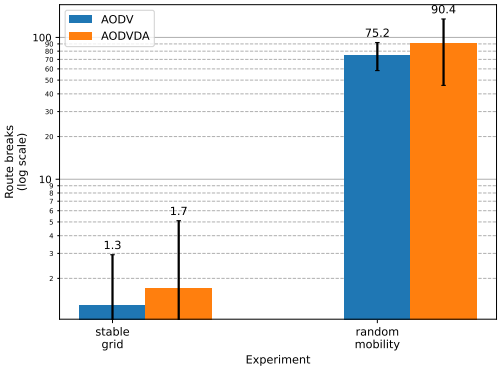
(a) Delivery rate across experiments.



(b) Path lengths CDF (random mobility).



(c) Overhead per data packet.



(d) Route breaks per experiment.

Figure 5.9: AODV vs AODVDA evaluation results.

Starting with figure 5.9a, we can see that the new metric improves AODV's delivery rate by almost 10% when mobility is introduced. Further, its longer routes do not seem to have any negative effects in the stable grid baseline. Moving on, figure 5.9b shows a CDF of the route lengths created by both protocols in all the runs of the random mobility experiment. AODVDA produces longer routes as expected but by a surprisingly low margin. In all likelihood, most routes are almost the same length in hops, but AODVDA avoids the nodes that sit at the edge of the transmission range, affording it a bit more resilience to breakage.

Another important aspect of having longer routes, is that each data packet requires

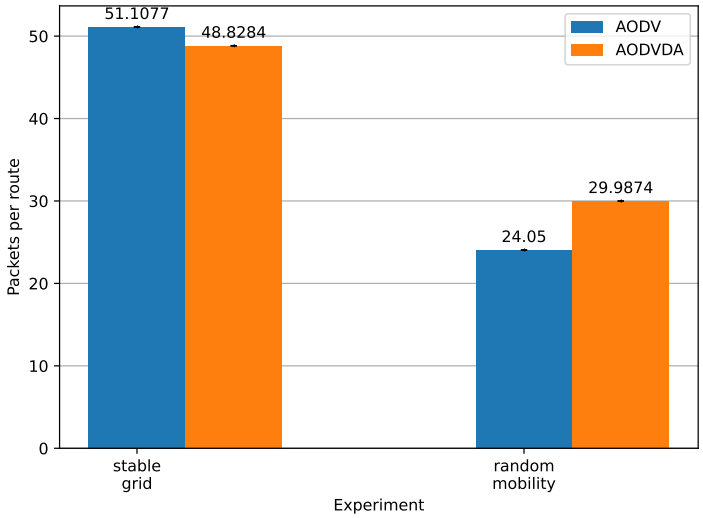


Figure 5.10: Data packets per route.

extra transmissions to be delivered. Contrary to this intuition, however, figure 5.9c shows that AODVDA actually has a lower overhead per packet than AODV. A potential explanation for this reduction in control traffic would be that the longer routes break less often, however, figure 5.9d shows there is actually a slight increase in route breaks for AODVDA. Thus, the explanation for AODVDA's increased performanced is given by figure 5.10, which shows us the average number of data packets delivered per each route that was successfully established. As we can appreciate in this figure, AODVDA's routes deliver in average 30 packets per route to AODV's 24. This added efficiency in the use of routes naturally leads to fewer route discoveries being needed, and consequently lowering the control traffic emitted.

This initial foray into my proposed *Adjusted-Distance* metric shows promise in mitigating the effects of mobility on MANET routing, but this is only the beginning. Further experiments and measurements are needed before we can rule-out the use of metrics that do not account for node-separation in MANETs, but the preliminary data supports my initial intuition sufficiently to warrant further exploration in follow up work.

Chapter 6

Conclusions

Through my years as a PhD student, I learned in conversation with many academics that the doctoral thesis that most people start with is not the one they end up with, and that was definitely the case for me as well. My initial proposal involved studying a system I designed which merged the capabilities of MANETs with onion routing to provide anonymous communications that did not rely on infrastructure. Early on my PhD process, however, a professor in my department asked me about the overall capacity of MANETs to provide these communication channels, which led me to deeper dive at how these networks operated, their protocols, their challenges, and limitations. It turned out I never emerged from this dive, and I dedicated the rest of my PhD to studying the many aspects of MANETs.

As the reader might recall, the initial hypothesis that inspired the work contained in this document was that the unique properties of MANETs suggested that fault tolerance should be at the core of protocol design. This led to the research question I postulated in chapter 1: *What is the performance effect of designing a MANET routing protocol that favours resiliency over other metrics?* I later measured this effect by designing a set of fault-tolerant protocols and testing them in increasingly harsher simulated conditions against an incumbent from the literature and two baselines. The results confirmed my speculations that designing for fault tolerance would ultimately increase performance in the other classical networking metrics, answering my research question.

The first conclusion that I can offer from this journey can be tracked back to when I was formulating my research question, and I anticipated two things about the work that would follow: That it would take many iterations to find relevant sources of faults, and that I would need to come up with a *composite* metric to show that my protocols were indeed better. I was wrong in both accounts. Surprisingly, my proposition of isolating the multiple properties of MANETs that lead to faults was not already covered extensively in the literature, and my protocols outperformed the incumbents in all metrics. I believe both of these phenomena are explained by the fact that **the importance of fault tolerance**

is overlooked too often.

Further, I think this phenomena occurs due to the the nature of academic research itself. While it often pushes us to look for research avenues that show bigger, faster, and better results all around, fault tolerance invites us to think more about production code, about the engineering challenges of deployment, etc. In short, it insinuates itself as a concern of maintenance, and not of innovation. But, as my research shows in this document, it is overlooked at everyone's peril as it can provide unexpected performance benefits that extend far beyond notions of maintainability.

My second conclusion is that **shortest-path routes are not always desirable**. The desirability of shorter routes is often treated as dogma in networking, but perhaps we should re-think this. As I showed in chapter 5, the effect of using shortest-path routing algorithms alongside a metric that does not account for geographical-spacing effects can result in fragile paths, which are more vulnerable to mobility. But these routes can also cause negative effects in other realms such as data centre (DC) networks.

When a top-of-rack (ToR) switch calculates a route to another server in the same DC, it almost always uses some protocol with a shortest-path algorithm at its core. Depending on the traffic patterns of the DC, this can create congestion among many of these shorter paths that could be avoided by taking longer routes further up the DC topology. Naturally, this effect is well known and addressed in many ways, such as using composite routing metrics that look at more than the length of the route. However, since these routing metrics are still used in minimising algorithms (e.g. shortest-path), it only pushes the problem away as different traffic patterns could again lead to congestion despite the existence of longer paths with more available capacity. Questioning the universality of the application of these routes could lead to different and more effective schemes.

The last conclusion I offer, is that **matching research methodology and tooling are essential**. It goes without saying that all research is hard, but this is especially true in a field like this in which the analytical models are vastly removed from real-world deployments, which themselves are too costly and do not scale. Thus, choosing the right level of abstraction for study and experimentation is crucial. In this sense, I believe the Calibration-Simulation-Validation experimental cycle promoted MeshSim can be a valuable contribution to the community. While it obviously lacks the maturity of the established toolsets, the simplicity of its design coupled with its powerful features might attract community members to use it and extend.

Lastly, I would like to say that even though the direction my PhD took was very different to what I anticipated, I consider it a privilege to have had the opportunity to work on it. Ever since I was a child I dreamt of becoming a scientist, but I never anticipated I would get to prepare for this journey in an institution such as the University of Cambridge. Through my work, it is my hope that future researchers interested in this field can use this

thesis as a starting point, and that it might inspire prospective scientists from developing nations to apply to their dream school, I promise you it is achievable.

6.1 Future work

I was once told by a dear friend of mine, Professor Isaac Schearson, that half of what makes a PhD thesis is the contribution it makes to science, and the other half is the amount of work it opens up. The contributions this thesis makes have been described in the previous chapters and summarised in this one. Thus, this last section is dedicated to outlining future directions that my research could take, as well as potential avenues to improve on the work I have done.

6.1.1 Power consumption modelling

Out of all the things I measured in my experiments in chapter 4, power consumption is unfortunately not one of them. The reason for this is simple: It is quite difficult, if not impossible, to provide an accurate power-consumption reading from a simulation experiment. In section 2.7, I describe that the power consumption of a routing protocol is given by the power consumed by calculations and by radio operations, with the latter heavily dominating the measurement. However, neither is easily measured. Using software tools like Powerstat [175] to measure the power demand of the protocol's calculations would be futile, as they are not only dependant on the workload but on the hardware, and server-grade CPUs are unrepresentative of what a MANET node would use. Further, since the radio operations are simulated as well, they can not be measured.

$$Pow = C_{p\mu} \sum_{i=1}^{|Pkt|} p_i + \psi \sum_{i=1}^{|Tx|} t_i + \psi \sum_{i=1}^{|Rx|} r_i \tag{6.1}$$

where

$$\begin{aligned}
Pow &= \text{Total power consumption of the protocol} \\
C_{p\mu} &= \text{Average CPU power consumption (mW) of the node} \\
Pkt &= \text{Set of time measurements of all processed packets} \\
p_i &= \text{Time it took to process a given packet; } p_i \in Pkt \\
\psi &= \text{Power consumption (mW) of the node's wireless NIC} \\
Tx &= \text{Set of time measurements of all transmissions} \\
t_i &= \text{Time it took to perform a given transmission; } t_i \in Tx \\
Rx &= \text{Set of time measurements of all radio receptions} \\
r_i &= \text{Time it took to perform a given radio reception; } r_i \in Rx
\end{aligned}$$

However, all of the above can be approximated by utilising proxy measurements which are easier to collect. Equation 6.1 provides a model to describe the power consumption of a MANET routing protocol in a single node. Since the MeshSim model relies on the use of a concrete testbed node for calibration and verification, its properties such as radio and CPU power consumption can be obtained from its specifications. These values used in conjunction with the measured time for the transmission, reception, and processing operations of each packet, which are already captured by MeshSim, can be used to obtain a reasonable estimate of the power consumption of the protocol. Alternatively, a simplified version of this model would involve performing a new calibration experiment. By obtaining average values for the power consumption of each type of operation in a testbed node, one can then just multiply this value by the number of corresponding operations performed in a simulation.

6.1.2 Adjusted-Distance metric implementations

In section 5.3 I proposed the notion of *Adjusted-Distance* routing metrics as a way to address AODV's vulnerability to mobility. The experiments in that section showed the promise of such notion, as an adjusted-distance hop count metric improved AODV's delivery rate and efficiency. However, hop count is hardly the most sophisticated of metrics, and as I postulated in that chapter, I believe other metrics such as ETX are also susceptible to the same geographical-spreading effect. Thus, it would be very interesting to implement adjusted-distance versions of these metrics to further study the relationship between geographical spreading of nodes and faults caused by mobility.

6.1.3 RGR verification experiments

As I previously discussed, the MeshSim experimental method relies on the calibrate-simulate-validate cycle to perform high-fidelity simulations which are further validated by their counterpart testbed experiments. Unfortunately, I was unable to fully finish the validate cycle for my experiments. When I wrote about the complexity and difficulty of managing a testbed in chapter 3, I did it not only out of theoretical but out of experience. The testbed I built to verify my simulations required careful placing of each node in the correct physical location to produce the desired topology. This requires being able to place the nodes in locations that are safe, and in which they have access to Wi-Fi in order to receive commands.

In my case, the best location for such deployment was the Computer Science & Technology Department (commonly known as the CL) at University of Cambridge, where I could monitor and access my devices as needed (see figure 6.1). However, even there, the locations for some of the nodes were not ideal, as they had no access to continuous power

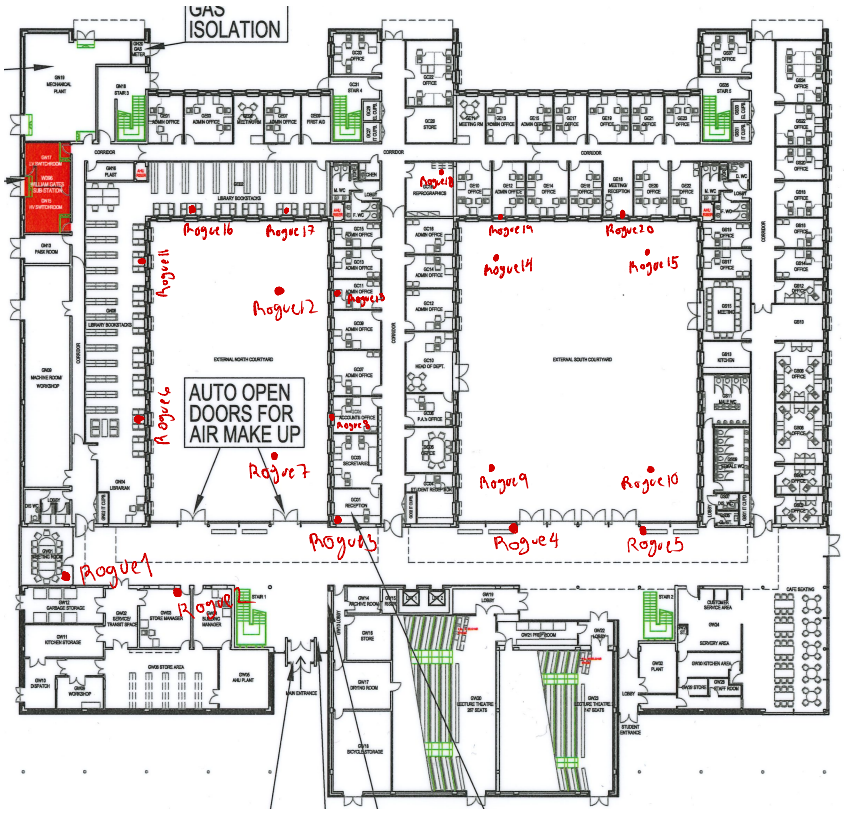


Figure 6.1: Deployment map of the verification experiment.

sources and thus operated on batteries. This makes it so that for every deployment, I need to make sure the nodes are charged and correctly placed. Lastly, all nodes are configured to work with Eduroam wireless network as the control channel, and must be reachable by the control server located in my office at the CL. All of the above translates into a complex and lengthy experimental procedure that can only be performed at the physical location of the testbed. Unfortunately, these factors prevented me from completing these experiments when the COVID-19 pandemic hit the world, as I thus became unable to access the CL.

The preliminary tests I did perform were promising, but confirmed my expectations about the difficulty of setting up the testbed. One such test included loading the RGR-III protocol in the testbed in order to validate its correct use of the custom device drivers for the Lora and Wi-Fi radios, which were successful. A more interesting test was one I performed using the calibration protocol I introduced in chapter 3, which I use to obtain connectivity data. The test in question intended to corroborate that the deployment configuration depicted in figure 6.1 produced the expected network topology shown in figure 6.2. As experienced readers might expect, the produced topology looked nothing like what I anticipated (see figure 6.3).

Not only did a number of nodes failed to report any connectivity data, but the quality of the links was also unexpected. The root-cause analysis investigation included several rounds of spectrum measurement, inspection of the architectural drawings of the building,

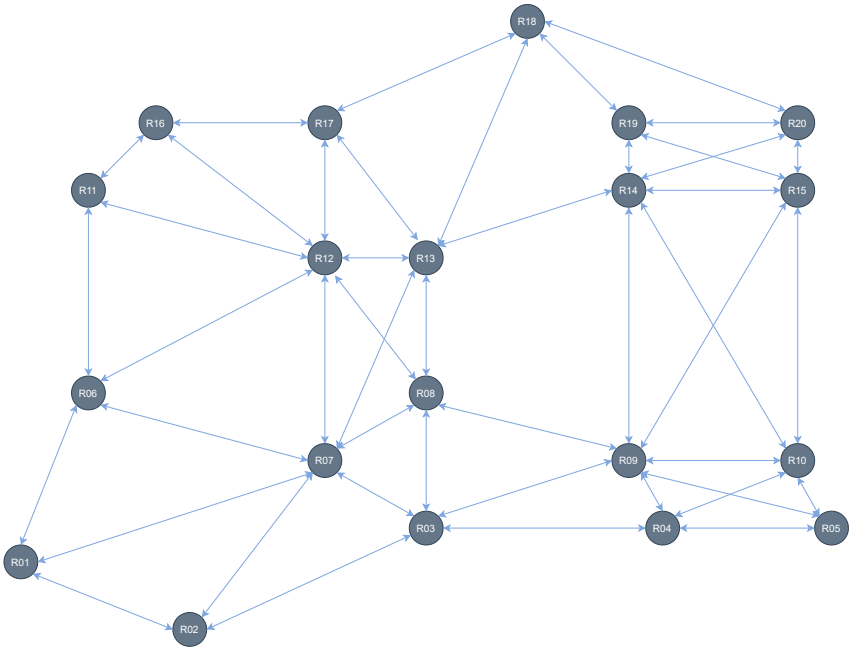


Figure 6.2: Expected topology for the validation deployment.

and conversations with the facility's staff. I concluded that multipath propagation was one of the main culprits of the produced topology, given that all nodes were placed on the ground floor of the building, which I learned was especially shielded and likely very reflective of radio frequency signals. Whether I get the chance to perform these experiments, or someone else takes over, the likely correction for this would be to spread the nodes across the floors of the building.

6.1.4 MeshSim improvements

As I finish my PhD and prepare for the next stage of my career, I intend to introduce a series of updates to the MeshSim platform, its public repository and the documentation. The intention of these changes is to facilitate the contributions of the open source community to the platform, to increase quality, and to add features such that it can be used for further scenarios. When that is done, I suggest the following improvements to the platform to cover more use cases within the MANET community.

6.1.4.1 Extended link layer

The current support MeshSim has for link-layer operations is limited. As mentioned in section 6.1.2, my initial focus on distance-vector protocols did not justify a heavier investment in a rich link layer that could enable protocols to perform operations at this level. The current link layer abstraction is only concerned with providing medium contention, which it does in discrete fashion, sensing whether the medium is busy or not. In order to

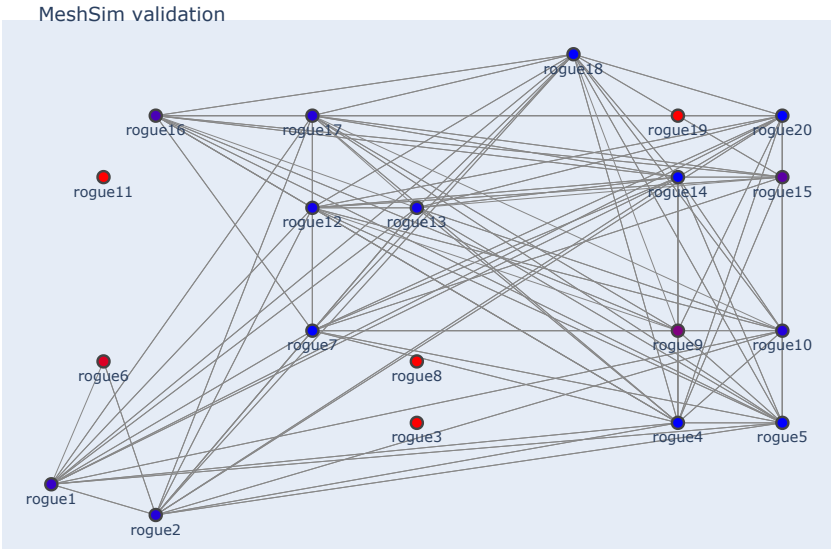


Figure 6.3: Obtained topology for the validation deployment.

support link-state protocols, this feature-set must be expanded.

An interesting challenge to do so, however, is that there is no obvious better design approach to do this. The simplified networking approach of MeshSim has it using a pseudo-IP layer to handle all packets, which could be extended to incorporate these features. Alternatively, a pseudo-link-layer frame could be added to the system, which would wrap the existing `MessageHeader` and contain all the relevant link layer fields. This approach could be extended to add physical layer headers as well if desirable in the future. The problem with this approach, however, is that it is not trivial to implement either layer fully, and careful consideration must be given in case a simplified version is used.

6.1.4.2 Deployment integration

Currently, the experimentation method for simulated scenarios is streamlined for most scenarios. Calling the `master_cli` is simple, and the interface is robust enough to support scripting around (as I did for the experiments in chapter 4). For device deployments, one can choose to manually deploy the `worker_cli` client into each device and configure them, or rely on the `Cluster` tool. In an abstract level, both tasks are quite similar, but are currently disassociated with each other and use different tools, and specification files.

However, the Cluster tool could be extended to include simulations. It could register the simulation servers similarly to how it does for the testbed nodes, but using a different pool. Running a simulation would follow the same process as that of a device deployment, but using the appropriate server and tools.

6.1.4.3 Automated verification

One of the benefits of working with MeshSim is that all events of the system are logged as JSON-formatted lines in the corresponding files. This feature greatly simplifies the data analysis done after each experiment, letting me capture different behaviours of the system by aggregating relevant metrics. This granularity of the data could also be used to automate to some degree the verification step of the experimentation cycle. For instance, in chapter 3 I show how I manually processed such timings to calibrate how long should a simulated radio broadcast operation should take. This process could be automated after each experiment for several low-level operations of the system, and then compared against their testbed counterparts. This would provide researchers a *fidelity level* after each experiment, alerting them when any particular operation is outside of the expected parameters. This feature could be made easier if the previous one (deployment integration) has been implemented, as any given testbed experiment could be used as a point of reference for a simulation.

6.1.4.4 Distributed-simulation support

The most significant blocker for new users wanting to use MeshSim for their own research is having a suitable server to run the simulations on. As I demonstrated in chapter 3, one of the main issues a real-time simulator like MeshSim can have, is that any of its entities may be slowed down if it does not have enough processing time. By analysing the run-queue of the server on which I run my experiments, I can guarantee that is not the case. However, not all users will have access to such a computer. Thus, modifying MeshSim to spread the processes corresponding to the simulated nodes across multiple machines would be a huge boon to its adoption. This feature, alongside the two previous ones, are highly related to each other.

6.1.4.5 Augmented mobility

In my experiments, I used the Random Waypoint mobility model, arguably the most widely used model in the MANET literature, so that my work could be comparable to that. However, that model is well known to be completely unrealistic in its patterns. In order to better represent more simulation scenarios, it is necessary to update the simulator to support a wider range of mobility models, such as Group or Manhattan. Further, while it would be very desirable to support those well-known models, it would also be very interesting if the system could support the use of mobility traces from any real-life deployment to generate a mobility distribution based on it, and apply it to a simulation.

6.1.4.6 Simulation replay

The current data model of MeshSim has the state of all nodes recorded at any given time in the experiment's database, and all events that occur in the system are logged in the structured JSON format. As mentioned earlier, the formatted log entries facilitate data analysis, but they also make debugging easier. Since MeshSim is a real-world simulator, it is impossible to pause the execution of any process by attaching a debugger to it without affecting the results. Thus, debugging of the system involves a careful analysis of the detailed logging the system provides.

This logging is so thorough that in fact, in conjunction with the state of all entities from the database, it should be possible to *replay* an entire simulation with it. It is unclear what such replay would imply, whether just an animation of the events of the simulation, or a full execution that can now be paused. This would likely be very useful for the development and debugging process of any future protocols.

Bibliography

[1] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958, ISSN: 0033-569X, 1552-4485. DOI: 10.1090/qam/102435. [Online]. Available: https://www.ams.org/qam/1958-16-01/S0033-569X-1958-0102435-2/ (visited on 01/29/2021) (cit. on pp. 21, 157).

[2] N. Abramson, "THE ALOHA SYSTEM: Another alternative for computer communications," in *Proceedings of the November 17-19, 1970, Fall Joint Computer Conference*, ser. AFIPS '70 (Fall), New York, NY, USA: Association for Computing Machinery, Nov. 17, 1970, pp. 281–285, ISBN: 978-1-4503-7904-5. DOI: 10.1145/1478462.1478502. [Online]. Available: https://doi.org/10.1145/1478462.1478502 (visited on 03/01/2021) (cit. on p. 31).

[3] J. Burchfiel, R. Tomlinson, and M. Beeler, "Functions and structure of a packet radio station," in *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, ser. AFIPS '75, Anaheim, California: Association for Computing Machinery, May 19, 1975, pp. 245–251, ISBN: 978-1-4503-7919-9. DOI: 10.1145/1499949.1499989. [Online]. Available: https://doi.org/10.1145/1499949.1499989 (visited on 05/22/2020) (cit. on p. 32).

[4] R. E. Kahn, "The organization of computer resources into a packet radio network," *NTC Proc*, p. 10, Dec. 1975 (cit. on pp. 31, 135).

[5] R. Kahn, S. Gronemeyer, J. Burchfiel, and R. Kunzelman, "Advances in packet radio technology," *Proceedings of the IEEE*, vol. 66, no. 11, pp. 1468–1496, Nov. 1978, ISSN: 1558-2256. DOI: 10.1109/PROC.1978.11151 (cit. on pp. 31, 32, 87).

[6] J. F. Shoch and L. Stewart, "Interconnecting local networks via the Packet Radio Network," in *Proceedings of the Sixth Symposium on Data Communications*, ser. SIGCOMM '79, New York, NY, USA: Association for Computing Machinery, Nov. 27, 1979, pp. 153–158, ISBN: 978-1-4503-7399-9. DOI: 10.1145/800092.802993. [Online]. Available: https://doi.org/10.1145/800092.802993 (visited on 09/15/2020) (cit. on p. 32).

[7] J. McQuillan, I. Richer, and E. Rosen, "The New Routing Algorithm for the ARPANET," *IEEE Transactions on Communications*, vol. 28, no. 5, pp. 711–719, May 1980, ISSN: 1558-0857. DOI: 10.1109/TCOM.1980.1094721 (cit. on p. 20).

[8] M. Schwartz and T. Stern, "Routing Techniques Used in Computer Communication Networks," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 539–552, Apr. 1980, ISSN: 1558-0857. DOI: 10.1109/TCOM.1980.1094690 (cit. on p. 51).

[9] N. Gower and J. Jubin, "Congestion Control using Pacing in a Packet Radio Network," in *MILCOM 1982 - IEEE Military Communications Conference - Progress in Spread Spectrum Communications*, vol. 1, Oct. 1982, pp. 23.1-1-23.1–6. DOI: 10.1109/MILCOM.1982.4805945 (cit. on p. 32).

[10] N. Shacham, E. Craighill, and A. Poggio, "Speech Transport in Packet-Radio Networks with Mobile Nodes," *IEEE Journal on Selected Areas in Communications*, vol. 1, no. 6, pp. 1084–1097, Dec. 1983, ISSN: 1558-0008. DOI: 10.1109/JSAC.1983.1146017 (cit. on p. 135).

[11] J. Jubin and J. Tornow, "The DARPA packet radio network protocols," *Proceedings of the IEEE*, vol. 75, no. 1, pp. 21–32, Jan. 1987, ISSN: 1558-2256. DOI: 10.1109/PROC.1987.13702 (cit. on pp. 19, 32).

[12] W. W. Royce, "Managing the development of large software systems: Concepts and techniques," in *Proceedings of the 9th International Conference on Software Engineering*, ser. ICSE '87, Washington, DC, USA: IEEE Computer Society Press, Mar. 1, 1987, pp. 328–338, ISBN: 978-0-89791-216-7 (cit. on p. 24).

[13] C. L. Hedrick, "Routing Information Protocol," IETF, RFC1058, Jun. 1988. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1058.html (visited on 09/30/2020) (cit. on p. 36).

[14] N. Gower, "Packet Radio Integrated System Module (PRISM) - Design Document," ROCKWELL INTERNATIONAL RICHARDSON TX COLLINS DEFENSE COMMUNICATIONS, Aug. 1, 1989. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA214723 (visited on 09/16/2020) (cit. on p. 34).

[15] D. Young and P. Bausbacher, "PC-NETSIM: A PC Based Network Simulation," ROCKWELL INTERNATIONAL, Richardson, Texas, SRNTN64, 1989, p. 14 (cit. on p. 34).

[16] D. Beyer, "Accomplishments of the DARPA SURAN Program," in *IEEE Conference on Military Communications*, Sep. 1990, 855–862 vol.2. DOI: 10.1109/MILCOM.1990.117536 (cit. on pp. 24, 33, 87, 95).

[17] S. Keshav, "A control-theoretic approach to flow control," *ACM SIGCOMM Computer Communication Review*, vol. 21, no. 4, pp. 3–15, Aug. 1, 1991, ISSN: 0146-4833. DOI: 10.1145/115994.115995. [Online]. Available: https://doi.org/10.1145/115994.115995 (visited on 01/31/2021) (cit. on p. 160).

[18] C. Alaettinoğlu, K. Dussa-Zieger, I. Matta, A. U. Shankar, and Ó. Gudmundsson, "Introducing MaRS, a routing testbed," *ACM SIGCOMM Computer Communication Review*, vol. 22, no. 1, pp. 95–96, Jan. 1992, ISSN: 0146-4833. DOI: 10.1145/141790.141798. [Online]. Available: https://dl.acm.org/doi/10.1145/141790.141798 (visited on 12/18/2020) (cit. on pp. 94, 101).

[19] W. Diepstraten, G. Ennis, and P. Belanger, "Distributed Foundation Wireless Medium Access Control," *IEEE Standards*, IEEE Document P802.11-93/190, Nov. 1993. [Online]. Available: http://www.ieee802.org/11/Documents/DocumentArchives/1993_docs/1193190_scan.pdf (visited on 09/24/2020) (cit. on p. 35).

[20] G. S. Malkin, "RIP Version 2 Carrying Additional Information," Internet Engineering Task Force, Request for Comments RFC 1388, Jan. 1993, 7 pp. DOI: 10.17487/RFC1388. [Online]. Available: https://datatracker.ietf.org/doc/rfc1388 (visited on 09/15/2021) (cit. on p. 20).

[21] I. Society. "Internet Engineering Task Force," IETF About. (1993), [Online]. Available: /about/ (visited on 10/02/2020) (cit. on p. 36).

[22] R. L. Bagrodia and Wen-Toh Liao, "Maisie: A language for the design of efficient discrete-event simulations," *IEEE Transactions on Software Engineering*, vol. 20, no. 4, pp. 225–238, Apr. 1994, ISSN: 1939-3520. DOI: 10.1109/32.277572 (cit. on p. 94).

[23] Mathworks. "Simulink - Simulation and Model-Based Design." (1994), [Online]. Available: https://uk.mathworks.com/products/simulink.html (visited on 02/19/2021) (cit. on p. 93).

[24] C. E. Perkins and P. Bhagwat, "Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 4, pp. 234–244, Oct. 1, 1994, ISSN: 0146-4833. DOI: 10.1145/190809.190336. [Online]. Available: https://doi.org/10.1145/190809.190336 (visited on 05/22/2020) (cit. on pp. 20, 36).

[25] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *Wireless Networks*, vol. 1, no. 1, pp. 61–81, Mar. 1995, ISSN: 1022-0038, 1572-8196. DOI: 10.1007/BF01196259. [Online]. Available: http://link.springer.com/10.1007/BF01196259 (visited on 10/08/2020) (cit. on p. 63).

[26] S. McCanne, S. Floyd, and K. Fall. "LBNL Network Simulator, ns version 1," Berkeley Lab. (1995), [Online]. Available: https://ee.lbl.gov/ns/ (visited on 12/18/2020) (cit. on p. 94).

[27] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile Computing*, ser. The Kluwer International Series in Engineering and Computer Science, T. Imielinski and H. F. Korth, Eds., Boston, MA: Springer US, 1996, pp. 153–181, ISBN: 978-0-585-29603-6. DOI: 10.1007/978-0-585-29603-6_5. [Online]. Available: https://doi.org/10.1007/978-0-585-29603-6_5 (visited on 01/16/2022) (cit. on pp. 65, 81).

[28] S. Murthy and J. J. Garcia-Luna-Aceves, "An efficient routing protocol for wireless networks," *Mobile Networks and Applications*, vol. 1, no. 2, pp. 183–197, Jun. 1996, ISSN: 1383-469X, 1572-8153. DOI: 10.1007/BF01193336. [Online]. Available: http://link.springer.com/10.1007/BF01193336 (visited on 09/14/2020) (cit. on p. 52).

[29] R. W. Bohannon, "Comfortable and maximum walking speed of adults aged 20—79 years: Reference values and determinants," *Age and Ageing*, vol. 26, no. 1, pp. 15–19, 1997, ISSN: 0002-0729, 1468-2834. DOI: 10.1093/ageing/26.1.15. [Online]. Available: https://academic.oup.com/ageing/article-lookup/doi/10.1093/ageing/26.1.15 (visited on 01/22/2019) (cit. on p. 146).

[30] C. Chiang, H. Wu, W. Liu, and M. Gerla, "Routing in clustered multihop, mobile wireless networks with fading channel," *In proceedings of IEEE SICON*, vol. 97, no. 1997, pp. 197–211, Apr. 1997 (cit. on pp. 55, 141).

[31] Z. Haas, "A new routing protocol for the reconfigurable wireless networks," in *Proceedings of ICUPC 97 - 6th International Conference on Universal Personal Communications*, vol. 2, Oct. 1997, 562–566 vol.2. DOI: 10.1109/ICUPC.1997.627227 (cit. on p. 72).

[32] D. S. Hochba, "Approximation Algorithms for NP-Hard Problems," *ACM SIGACT News*, vol. 28, no. 2, pp. 40–52, Jun. 1, 1997, ISSN: 0163-5700. DOI: 10.1145/261342.571216. [Online]. Available: https://doi.org/10.1145/261342.571216 (visited on 10/15/2021) (cit. on p. 79).

[33] IETF. "Mobile Ad-hoc Networks (manet) -." (1997), [Online]. Available: https://datatracker.ietf.org/wg/manet/about/ (visited on 10/02/2020) (cit. on p. 36).

[34] V. Park and M. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proceedings of INFOCOM '97*, vol. 3, Apr. 1997, 1405–1413 vol.3. DOI: 10.1109/INFCOM.1997.631180 (cit. on p. 66).

[35] C.-K. Toh, "Associativity-Based Routing for Ad Hoc Mobile Networks," *Wireless Personal Communications*, vol. 4, no. 2, pp. 103–139, Mar. 1, 1997, ISSN: 1572-834X. DOI: 10.1023/A:1008812928561. [Online]. Available: https://doi.org/10.1023/A:1008812928561 (visited on 10/13/2020) (cit. on p. 67).

[36] R. Bagrodia, R. Meyer, M. Takai, *et al.*, "Parsec: A parallel simulation environment for complex systems," *Computer*, vol. 31, no. 10, pp. 77–85, Oct. 1998, ISSN: 1558-0814. DOI: 10.1109/2.722293 (cit. on p. 94).

[37] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward, "A distance routing effect algorithm for mobility (DREAM)," in *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, ser. MobiCom '98, New York, NY, USA: Association for Computing Machinery, Oct. 25, 1998, pp. 76–84, ISBN: 978-1-58113-035-5. DOI: 10.1145/288235.288254. [Online]. Available: https://doi.org/10.1145/288235.288254 (visited on 10/05/2020) (cit. on pp. 59, 139).

[38] Y.-B. Ko and N. H. Vaidya, "Location-aided routing (LAR) in mobile ad hoc networks," in *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, ser. MobiCom '98, New York, NY, USA: Association for Computing Machinery, Oct. 25, 1998, pp. 66–75, ISBN: 978-1-58113-035-5. DOI: 10.1145/288235.288252. [Online]. Available: https://doi.org/10.1145/288235.288252 (visited on 10/05/2020) (cit. on p. 68).

[39] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla, "GloMoSim: A Scalable Network Simulation Environment," University of California Los Angeles, Technical Report 990027, May 1999, p. 12 (cit. on p. 94).

[40] J. Garcia-Luna-Aceves and M. Spohn, "Source-tree routing in wireless networks," in *Proceedings. Seventh International Conference on Network Protocols*, Oct. 1999, pp. 273–282. DOI: 10.1109/ICNP.1999.801950 (cit. on p. 53).

[41] M. Joa-Ng and I-Tai Lu, "A peer-to-peer zone-based two-level link state routing for mobile ad hoc networks," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 8, pp. 1415–1425, Aug. 1999, ISSN: 07338716. DOI: 10.1109/49.779923. [Online]. Available: http://ieeexplore.ieee.org/document/779923/ (visited on 09/25/2019) (cit. on p. 73).

[42] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," in *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, ser. MobiCom '99, New York, NY, USA: Association for Computing Machinery, Aug. 1, 1999, pp. 151–162, ISBN: 978-1-58113-142-0. DOI: 10.1145/313451.313525. [Online].

Available: https://doi.org/10.1145/313451.313525 (visited on 10/05/2020) (cit. on pp. 41, 130).

[43] C. Perkins and E. Royer, "Ad-hoc on-demand distance vector routing," in *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, USA: IEEE, 1999, pp. 90–100, ISBN: 978-0-7695-0025-6. DOI: 10.1109/MCSA.1999.749281. [Online]. Available: http://ieeexplore.ieee.org/document/749281/ (visited on 08/27/2019) (cit. on pp. 20, 70, 143).

[44] G. Phipps, "Comparing observed bug and productivity rates for Java and C++," *Software: Practice and Experience*, vol. 29, no. 4, pp. 345–358, 1999, ISSN: 1097-024X. DOI: 10.1002/(SICI)1097-024X(19990410)29:4<345::AID-SPE238>3.0.CO;2-C. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-024X%2819990410%2929%3A4%3C345%3A%3AAID-SPE238%3E3.0.CO%3B2-C (visited on 12/27/2021) (cit. on p. 96).

[45] P. Sinha, R. Sivakumar, and V. Bharghavan, "CEDAR: A core-extraction distributed ad hoc routing algorithm," in *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future Is Now (Cat. No.99CH36320)*, vol. 1, Mar. 1999, 202–209 vol.1. DOI: 10.1109/INFCOM.1999.749269 (cit. on p. 74).

[46] P. Gupta and P. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, Mar. 2000, ISSN: 00189448. DOI: 10.1109/18.825799. [Online]. Available: http://ieeexplore.ieee.org/document/825799/ (visited on 05/01/2020) (cit. on pp. 42, 130).

[47] J. Heidemann, D. Estrin, R. Govindan, and A. Goel. "The Network Simulator - ns-2." (2000), [Online]. Available: https://www.isi.edu/nsnam/ns/ (visited on 12/19/2020) (cit. on p. 94).

[48] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '00, New York, NY, USA: Association for Computing Machinery, Aug. 1, 2000, pp. 243–254, ISBN: 978-1-58113-197-0. DOI: 10.1145/345910.345953. [Online]. Available: https://doi.org/10.1145/345910.345953 (visited on 10/07/2020) (cit. on p. 57).

[49] S. Lee and M. Gerla, "AODV-BR: Backup routing in ad hoc networks," in *2000 IEEE Wireless Communications and Networking Conference. Conference Record (Cat. No.00TH8540)*, vol. 3, Sep. 2000, 1311–1316 vol.3. DOI: 10.1109/WCNC.2000.904822 (cit. on p. 82).

[50] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '00, New York, NY, USA: Association for Computing Machinery, Aug. 1, 2000, pp. 120–130, ISBN: 978-1-58113-197-0. DOI: 10.1145/345910.345931. [Online]. Available: https://doi.org/10.1145/345910.345931 (visited on 10/19/2020) (cit. on p. 58).

[51] G. Pei, M. Gerla, and T.-W. Chen, "Fisheye state routing: A routing scheme for ad hoc wireless networks," in *2000 IEEE International Conference on Communications. ICC 2000. Global Convergence Through Communications. Conference Record*, vol. 1, Jun. 2000, 70–74 vol.1. DOI: 10.1109/ICC.2000.853066 (cit. on pp. 56, 141).

[52] J. Place, D. Kerr, and D. Schaefer, "Joint Tactical Radio System," in *MILCOM 2000 Proceedings. 21st Century Military Communications. Architectures and Technologies for Information Superiority (Cat. No.00CH37155)*, vol. 1, Oct. 2000, 209–213 vol.1. DOI: 10.1109/MILCOM.2000.904941 (cit. on p. 34).

[53] J. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile, "IEEE 802.15.4: A developing standard for low-power low-cost wireless personal area networks," *IEEE Network*, vol. 15, no. 5, pp. 12–19, Sep. 2001, ISSN: 1558-156X. DOI: 10.1109/65.953229 (cit. on p. 39).

[54] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century.*, Dec. 2001, pp. 62–68. DOI: 10.1109/INMIC.2001.995315 (cit. on pp. 20, 60).

[55] S. Lee and M. Gerla, "Split multipath routing with maximally disjoint paths in ad hoc networks," in *ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No.01CH37240)*, vol. 10, Jun. 2001, 3201–3205 vol.10. DOI: 10.1109/ICC.2001.937262 (cit. on p. 83).

[56] H. Lim and C. Kim, "Flooding in wireless ad hoc networks," *Computer Communications*, p. 11, 2001 (cit. on p. 41).

[57] M. Marina and S. Das, "On-demand multipath distance vector routing in ad hoc networks," in *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, Nov. 2001, pp. 14–23. DOI: 10.1109/ICNP.2001.992756 (cit. on p. 82).

[58] A. Nasipuri, R. Castañeda, and S. R. Das, "Performance of Multipath Routing for On-Demand Protocols in Mobile Ad Hoc Networks," *Mobile Networks and Applications*, vol. 6, no. 4, pp. 339–349, Aug. 1, 2001, ISSN: 1572-8153. DOI: 10.

1023/A:1011426611520. [Online]. Available: https://doi.org/10.1023/A:1011426611520 (visited on 02/02/2021) (cit. on p. 83).

[59] G. Zaruba, S. Basagni, and I. Chlamtac, "Bluetrees-scatternet formation to enable Bluetooth-based ad hoc networks," in *ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No.01CH37240)*, vol. 1, Jun. 2001, 273–277 vol.1. DOI: 10.1109/ICC.2001.936316 (cit. on p. 39).

[60] Z. Haas, J. Halpern, and L. Li, "Gossip-based ad hoc routing," in *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, Jun. 2002, 1707–1716 vol.3. DOI: 10.1109/INFCOM.2002.1019424 (cit. on pp. 41, 130).

[61] R. Ramanathan and J. Redi, "A brief overview of ad hoc networks: Challenges and directions," *IEEE Communications Magazine*, vol. 40, no. 5, pp. 20–22, May 2002, ISSN: 0163-6804. DOI: 10.1109/MCOM.2002.1006968. [Online]. Available: http://ieeexplore.ieee.org/document/1006968/ (visited on 09/07/2021) (cit. on p. 37).

[62] C. A. Santivanez, B. McDonald, I. Stavrakakis, and R. Ramanathan, "On the scalability of ad hoc routing protocols," in *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, Jun. 2002, 1688–1697 vol.3. DOI: 10.1109/INFCOM.2002.1019422 (cit. on pp. 42, 130).

[63] F. Bai, N. Sadagopan, and A. Helmy, "IMPORTANT: A framework to systematically analyze the Impact of Mobility on Performance of Routing Protocols for Adhoc Networks," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 2, Mar. 2003, 825–835 vol.2. DOI: 10.1109/INFCOM.2003.1208920 (cit. on p. 44).

[64] C. Bettstetter, G. Resta, and P. Santi, "The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 2, no. 3, pp. 257–269, Jul. 1, 2003, ISSN: 1536-1233. DOI: 10.1109/TMC.2003.1233531. [Online]. Available: https://doi.org/10.1109/TMC.2003.1233531 (visited on 02/24/2021) (cit. on p. 146).

[65] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," IETF, RFC3626, Oct. 2003. [Online]. Available: https://www.rfc-editor.org/info/rfc3626 (visited on 06/14/2017) (cit. on p. 61).

[66] D. S. J. De Couto, D. Aguayo, B. A. Chambers, and R. Morris, "Performance of multihop wireless networks: Shortest path is not enough," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 83–88, Jan. 1, 2003, ISSN:

0146-4833. DOI: 10.1145/774763.774776. [Online]. Available: https://doi.org/10.1145/774763.774776 (visited on 10/13/2020) (cit. on p. 158).

[67] S. Gwalani, E. Belding-Royer, and C. Perkins, "AODV-PA: AODV with path accumulation," in *IEEE International Conference on Communications, 2003. ICC '03.*, vol. 1, May 2003, 527–531 vol.1. DOI: 10.1109/ICC.2003.1204232 (cit. on p. 71).

[68] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher, "Range-free localization schemes for large scale sensor networks," in *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '03, New York, NY, USA: Association for Computing Machinery, Sep. 14, 2003, pp. 81–95, ISBN: 978-1-58113-753-8. DOI: 10.1145/938985.938995. [Online]. Available: https://doi.org/10.1145/938985.938995 (visited on 10/24/2021) (cit. on p. 81).

[69] J. Schiller, "Wireless Transmission," in *Mobile Communications*, 2nd edition, Addison-Wesley, Aug. 7, 2003, pp. 25–68, ISBN: 978-81-317-2426-2 (cit. on pp. 36, 91).

[70] X.-Y. Li, P.-J. Wan, Y. Wang, and C.-W. Yi, "Fault tolerant deployment and topology control in wireless networks," in *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, ser. MobiHoc '03, New York, NY, USA: Association for Computing Machinery, Jun. 1, 2003, pp. 117–128, ISBN: 978-1-58113-684-5. DOI: 10.1145/778415.778431. [Online]. Available: https://doi.org/10.1145/778415.778431 (visited on 08/25/2021) (cit. on p. 79).

[71] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC Editor, RFC3561, Jul. 2003, RFC3561. DOI: 10.17487/rfc3561. [Online]. Available: https://www.rfc-editor.org/info/rfc3561 (visited on 11/11/2020) (cit. on pp. 71, 144, 153, 166).

[72] J. Schiller, "Medium Access Control," in *Mobile Communications*, 2nd edition, Addison-Wesley, Aug. 7, 2003, pp. 69–92, ISBN: 978-81-317-2426-2 (cit. on p. 36).

[73] A. Adya, P. Bahl, J. Padhye, A. Wolman, and Lidong Zhou, "A multi-radio unification protocol for IEEE 802.11 wireless networks," in *First International Conference on Broadband Networks*, San Jose, CA, USA: IEEE Comput. Soc, 2004, pp. 344–354, ISBN: 978-0-7695-2221-0. DOI: 10.1109/BROADNETS.2004.8. [Online]. Available: http://ieeexplore.ieee.org/document/1363823/ (visited on 02/07/2020) (cit. on pp. 141, 159).

[74] P. Bahl, A. Adya, J. Padhye, and A. Walman, "Reconsidering wireless systems with multiple radios," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 5, p. 39, Oct. 1, 2004, ISSN: 01464833. DOI: 10.1145/1039111.1039122. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1039111.1039122 (visited on 09/25/2019) (cit. on p. 141).

[75] P. Basu and J. Redi, "Movement control algorithms for realization of fault-tolerant ad hoc robot networks," *IEEE Network*, vol. 18, no. 4, pp. 36–44, Jul. 2004, ISSN: 1558-156X. DOI: 10.1109/MNET.2004.1316760 (cit. on p. 79).

[76] G. Di Caro, F. Ducatelle, and L. M. Gambardella, "AntHocNet: An Ant-Based Hybrid Routing Algorithm for Mobile Ad Hoc Networks," in *Parallel Problem Solving from Nature - PPSN VIII*, X. Yao, E. K. Burke, J. A. Lozano, *et al.*, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2004, pp. 461–470, ISBN: 978-3-540-30217-9. DOI: 10.1007/978-3-540-30217-9_47 (cit. on p. 75).

[77] R. Draves, J. Padhye, and B. Zill, "Comparison of routing metrics for static multi-hop wireless networks," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 133–144, Aug. 30, 2004, ISSN: 0146-4833. DOI: 10.1145/1030194.1015483. [Online]. Available: https://doi.org/10.1145/1030194.1015483 (visited on 01/26/2021) (cit. on pp. 160, 163).

[78] A. Kumar, D. Manjunath, and J. Kuri, "Shortest Path Routing of Elastic Aggregates," in *Communication Networking*, Elsevier, 2004, pp. 677–711, ISBN: 978-0-12-428751-8. DOI: 10.1016/B978-012428751-8/50014-8. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780124287518500148 (visited on 03/03/2021) (cit. on p. 73).

[79] R. Draves, J. Padhye, and B. Zil, "Routing in Multi-radio, Multi-hop Wireless Mesh Networks," in *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '04, New York, NY, USA: ACM, 2004, pp. 114–128, ISBN: 978-1-58113-868-9. DOI: 10.1145/1023720.1023732. [Online]. Available: http://doi.acm.org/10.1145/1023720.1023732 (visited on 12/08/2017) (cit. on pp. 141, 161, 163).

[80] Y. Xue and K. Nahrstedt, "Providing Fault-Tolerant Ad hoc Routing Service in Adversarial Environments," *Wireless Personal Communications*, vol. 29, no. 3, pp. 367–388, Jun. 1, 2004, ISSN: 1572-834X. DOI: 10.1023/B:WIRE.0000047071.75971.cd. [Online]. Available: https://doi.org/10.1023/B:WIRE.0000047071.75971.cd (visited on 08/26/2021) (cit. on pp. 80, 81).

[81] R. Bruno, M. Conti, and E. Gregori, "Mesh networks: Commodity multihop ad hoc networks," *IEEE Communications Magazine*, vol. 43, no. 3, pp. 123–131, Mar. 2005, ISSN: 1558-1896. DOI: 10.1109/MCOM.2005.1404606 (cit. on p. 37).

[82] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-throughput Path Metric for Multi-hop Wireless Routing," *Wirel. Netw.*, vol. 11, no. 4, pp. 419–434, Jul. 2005, ISSN: 1022-0038. DOI: 10.1007/s11276-005-1766-z. [Online]. Available: http://dx.doi.org/10.1007/s11276-005-1766-z (visited on 02/25/2019) (cit. on p. 160).

[83] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET Simulation Studies: The Incredibles," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 4, pp. 50–61, Oct. 2005, ISSN: 1559-1662. DOI: 10.1145/1096166.1096174. [Online]. Available: http://doi.acm.org/10.1145/1096166.1096174 (visited on 04/19/2017) (cit. on pp. 23, 24, 26, 87, 95).

[84] T. Spyropoulos, K. Psounis, and C. 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*, ser. WDTN '05, New York, NY, USA: Association for Computing Machinery, Aug. 22, 2005, pp. 252–259, ISBN: 978-1-59593-026-2. DOI: 10.1145/1080139.1080143. [Online]. Available: https://doi.org/10.1145/1080139.1080143 (visited on 09/02/2021) (cit. on p. 44).

[85] Telcos. "NetSim-Network Simulator & Emulator — Home," Telcos. (2005), [Online]. Available: https://www.tetcos.com/ (visited on 10/07/2020) (cit. on pp. 94, 95).

[86] T. Henderson, G. Riley, S. Floyd, and S. Roy. "NS-3 Emulation Overview," nsnam. (2006), [Online]. Available: https://www.nsnam.org/docs/models/html/emulation-overview.html (visited on 12/27/2021) (cit. on p. 95).

[87] B. Hull, V. Bychkovsky, Y. Zhang, *et al.*, "CarTel: A distributed mobile sensor computing system," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '06, New York, NY, USA: Association for Computing Machinery, Oct. 31, 2006, pp. 125–138, ISBN: 978-1-59593-343-0. DOI: 10.1145/1182807.1182821. [Online]. Available: https://doi.org/10.1145/1182807.1182821 (visited on 09/02/2021) (cit. on p. 44).

[88] N. Li and J. Hou, "Localized fault-tolerant topology control in wireless ad hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 4, pp. 307–320, Apr. 2006, ISSN: 1558-2183. DOI: 10.1109/TPDS.2006.51 (cit. on p. 80).

[89] B. Oommen and S. Misra, "A Fault-Tolerant Routing Algorithm for Mobile Ad Hoc Networks Using a Stochastic Learning-Based Weak Estimation Procedure," in *2006 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, Jun. 2006, pp. 31–37. DOI: 10.1109/WIMOB.2006.1696374 (cit. on p. 81).

[90] A. A. Pirzada, M. Portmann, and J. Indulska, "Evaluation of multi-radio extensions to AODV for wireless mesh networks," in *Proceedings of the 4th ACM International Workshop on Mobility Management and Wireless Access*, ser. MobiWac '06, New York, NY, USA: Association for Computing Machinery, Oct. 2, 2006, pp. 45–51, ISBN: 978-1-59593-488-8. DOI: 10.1145/1164783.1164791. [Online]. Available: https://doi.org/10.1145/1164783.1164791 (visited on 09/14/2021) (cit. on p. 20).

[91] P. A. Spagnolo and T. R. Henderson, "Comparison of Proposed Ospf Manet Extensions," in *MILCOM 2006 - 2006 IEEE Military Communications Conference*, Oct. 2006, pp. 1–7. DOI: 10.1109/MILCOM.2006.302376 (cit. on p. 20).

[92] T. Henderson, G. Riley, S. Floyd, and S. Roy. "What is ns-3?" ns-3. (2006), [Online]. Available: https://www.nsnam.org/about/ (visited on 12/27/2021) (cit. on pp. 94, 95).

[93] S. Medidi and J. Wang, "A Fault Resilient Routing Protocol for Mobile Ad-Hoc Networks," in *Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2007)*, Oct. 2007, pp. 41–41. DOI: 10.1109/WIMOB.2007.4390835 (cit. on p. 81).

[94] T. Wark, P. Corke, P. Sikka, *et al.*, "Transforming Agriculture through Pervasive Wireless Sensor Networks," *IEEE Pervasive Computing*, vol. 6, no. 2, pp. 50–57, Apr. 2007, ISSN: 1558-2590. DOI: 10.1109/MPRV.2007.47 (cit. on p. 38).

[95] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "CORE: A real-time network emulator," in *MILCOM 2008 - 2008 IEEE Military Communications Conference*, Nov. 2008, pp. 1–7. DOI: 10.1109/MILCOM.2008.4753614 (cit. on p. 95).

[96] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco, "Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 5, pp. 748–760, Jun. 2008, ISSN: 1558-0008. DOI: 10.1109/JSAC.2008.080602 (cit. on p. 38).

[97] D. Ferguson, A. Lindem, and J. Moy, "OSPF for IPv6," Internet Engineering Task Force, Request for Comments RFC 5340, Jul. 2008, 94 pp. DOI: 10.17487/RFC5340.

[Online]. Available: https://datatracker.ietf.org/doc/rfc5340 (visited on 09/15/2021) (cit. on p. 20).

[98] H. Hartenstein and L. P. Laberteaux, "A tutorial survey on vehicular ad hoc networks," *IEEE Communications Magazine*, vol. 46, no. 6, pp. 164–171, Jun. 2008, ISSN: 1558-1896. DOI: 10.1109/MCOM.2008.4539481 (cit. on p. 38).

[99] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: Social-based forwarding in delay tolerant networks," in *Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '08, New York, NY, USA: Association for Computing Machinery, May 26, 2008, pp. 241–250, ISBN: 978-1-60558-073-9. DOI: 10.1145/1374618.1374652. [Online]. Available: https://doi.org/10.1145/1374618.1374652 (visited on 10/12/2020) (cit. on p. 38).

[100] D. Jiang and L. Delgrossi, "IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments," in *VTC Spring 2008 - IEEE Vehicular Technology Conference*, May 2008, pp. 2036–2040. DOI: 10.1109/VETECS.2008.458 (cit. on p. 39).

[101] D. Johnson, N. S. Ntlatlapa, and C. Aichele, "Simple pragmatic approach to mesh routing using BATMAN," in *2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries*, Pretoria, South Africa: CSIR, Oct. 2008, ISBN: 978-84-612-5570-2. [Online]. Available: https://researchspace.csir.co.za/dspace/handle/10204/3035 (visited on 10/09/2020) (cit. on pp. 20, 61).

[102] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ser. Simutools '08, Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Mar. 3, 2008, pp. 1–10, ISBN: 978-963-9799-20-2 (cit. on p. 94).

[103] J. Zhao and G. Cao, "VADD: Vehicle-Assisted Data Delivery in Vehicular Ad Hoc Networks," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 3, pp. 1910–1922, May 2008, ISSN: 1939-9359. DOI: 10.1109/TVT.2007.901869 (cit. on p. 143).

[104] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Single-source shortest paths," in *Introduction to Algorithms*, 3rd edition, Cambridge, Mass: MIT Press, Jan. 1, 2009, pp. 643–683, ISBN: 978-0-262-03384-8 (cit. on pp. 51, 55).

[105] N. Ivanic, B. Rivera, and B. Adamson, "Mobile Ad Hoc Network emulation environment," in *MILCOM 2009 - 2009 IEEE Military Communications Conference*, Oct. 2009, pp. 1–6. DOI: 10.1109/MILCOM.2009.5379781 (cit. on p. 95).

[106] M. Musolesi and C. Mascolo, "Mobility Models for Systems Evaluation," in *Middleware for Network Eccentric and Mobile Applications*, B. Garbinato, H. Miranda, and L. Rodrigues, Eds., Berlin, Heidelberg: Springer, 2009, pp. 43–62, ISBN: 978-3-540-89707-1. DOI: 10.1007/978-3-540-89707-1_3. [Online]. Available: https://doi.org/10.1007/978-3-540-89707-1_3 (visited on 10/04/2021) (cit. on p. 146).

[107] J. Nzouonta, N. Rajgure, G. Wang, and C. Borcea, "VANET Routing on City Roads Using Real-Time Vehicular Traffic Information," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 7, pp. 3609–3626, Sep. 2009, ISSN: 1939-9359. DOI: 10.1109/TVT.2009.2014455 (cit. on p. 38).

[108] T. Zahn, G. O'Shea, and A. Rowstron, "Feasibility of content dissemination between devices in moving vehicles," in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies - CoNEXT '09*, Rome, Italy: ACM Press, 2009, p. 97, ISBN: 978-1-60558-636-6. DOI: 10.1145/1658939.1658951. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1658939.1658951 (visited on 09/27/2021) (cit. on p. 38).

[109] G. R. Hiertz, D. Denteneer, S. Max, *et al.*, "IEEE 802.11s: The WLAN Mesh Standard," *IEEE Wireless Communications*, vol. 17, no. 1, pp. 104–111, Feb. 2010, ISSN: 1558-0687. DOI: 10.1109/MWC.2010.5416357 (cit. on p. 37).

[110] N. R. Labs. "CORE/EMANE," core. (2010), [Online]. Available: http://coreemu.github.io/core/emane.html (visited on 02/18/2021) (cit. on p. 95).

[111] Naval Research Labs. "Extendable Mobile Ad-hoc Network Emulator (EMANE) — Networks and Communication Systems Branch." (2010), [Online]. Available: https://www.nrl.navy.mil/itd/ncs/products/emane (visited on 09/22/2020) (cit. on p. 95).

[112] C. Shallahamer. "The OS CPU Run Queue; Not What It Appears," OraPub. (Jul. 7, 2010), [Online]. Available: https://blog.orapub.com/20100707/the-os-cpu-run-queue-not-what-it-appears.html (visited on 01/11/2021) (cit. on p. 127).

[113] A. Jabbar, H. Narra, and J. P. Sterbenz, "An approach to quantifying resilience in mobile ad hoc networks," in *2011 8th International Workshop on the Design of Reliable Communication Networks (DRCN)*, Oct. 2011, pp. 140–147. DOI: 10.1109/DRCN.2011.6076896 (cit. on p. 47).

[114] J. Pojda, A. Wolff, M. Sbeiti, and C. Wietfeld, "Performance analysis of mesh routing protocols for UAV swarming applications," in *2011 8th International Symposium on Wireless Communication Systems*, Nov. 2011, pp. 317–321. DOI: 10.1109/ISWCS.2011.6125375 (cit. on p. 20).

[115] H. Saleet, R. Langar, K. Naik, R. Boutaba, A. Nayak, and N. Goel, "Intersection-Based Geographical Routing Protocol for VANETs: A Proposal and Analysis," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 9, pp. 4560–4574, Nov. 2011, ISSN: 1939-9359. DOI: 10.1109/TVT.2011.2173510 (cit. on p. 143).

[116] F. Knorr, D. Baselt, M. Schreckenberg, and M. Mauve, "Reducing Traffic Jams via VANETs," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 8, pp. 3490–3498, Oct. 2012, ISSN: 1939-9359. DOI: 10.1109/TVT.2012.2209690 (cit. on p. 38).

[117] H. Maowad and E. Shaaban, "Efficient routing protocol for Vehicular Ad hoc networks," in *Proceedings of 2012 9th IEEE International Conference on Networking, Sensing and Control*, Apr. 2012, pp. 209–215. DOI: 10.1109/ICNSC.2012.6204918 (cit. on p. 20).

[118] S. Misra, P. V. Krishna, A. Bhiwal, A. S. Chawla, B. E. Wolfinger, and C. Lee, "A learning automata-based fault-tolerant routing algorithm for mobile ad hoc networks," *The Journal of Supercomputing*, vol. 62, no. 1, pp. 4–23, Oct. 1, 2012, ISSN: 1573-0484. DOI: 10.1007/s11227-011-0639-8. [Online]. Available: https://doi.org/10.1007/s11227-011-0639-8 (visited on 08/20/2021) (cit. on p. 81).

[119] A. Pyattaev, K. Johnsson, S. Andreev, and Y. Koucheryavy, "3GPP LTE traffic offloading onto WiFi Direct," in *2013 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, Shanghai: IEEE, Apr. 2013, pp. 135–140, ISBN: 978-1-4799-0108-1. DOI: 10.1109/WCNCW.2013.6533328. [Online]. Available: http://ieeexplore.ieee.org/document/6533328/ (visited on 10/06/2020) (cit. on p. 38).

[120] P. K. Sahu, E. H.-K. Wu, J. Sahoo, and M. Gerla, "BAHG: Back-Bone-Assisted Hop Greedy Routing for VANET's City Environments," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 1, pp. 199–213, Mar. 2013, ISSN: 1558-0016. DOI: 10.1109/TITS.2012.2212189 (cit. on p. 143).

[121] M. M. Alani, "OSI Model," in *Guide to OSI and TCP/IP Models*, ser. SpringerBriefs in Computer Science, M. M. Alani, Ed., Cham: Springer International Publishing, 2014, pp. 5–17, ISBN: 978-3-319-05152-9. DOI: 10.1007/978-3-319-05152-9_2. [Online]. Available: https://doi.org/10.1007/978-3-319-05152-9_2 (visited on 02/17/2021) (cit. on p. 36).

[122] A. Bland, "FireChat – the messaging app that's powering the Hong Kong protests," *The GuardianWorld news*, Sep. 29, 2014, ISSN: 0261-3077. [Online]. Available: https://www.theguardian.com/world/2014/sep/29/firechat-messaging-app-powering-hong-kong-protests (visited on 09/24/2021) (cit. on p. 38).

[123] T. Clausen, C. Dearlove, P. Jacquet, and U. Herberg, "The Optimized Link State Routing Protocol Version 2," RFC Editor, RFC7181, Apr. 2014, RFC7181. DOI: 10.17487/rfc7181. [Online]. Available: https://www.rfc-editor.org/info/rfc7181 (visited on 10/07/2020) (cit. on pp. 61, 157).

[124] J. Liu, Y. Kawamoto, H. Nishiyama, N. Kato, and N. Kadowaki, "Device-to-device communications achieve efficient load balancing in LTE-advanced networks," *IEEE Wireless Communications*, vol. 21, no. 2, pp. 57–65, Apr. 2014, ISSN: 1558-0687. DOI: 10.1109/MWC.2014.6812292 (cit. on p. 38).

[125] G. Tyson. "Mesh networks and Firechat make 'switching off the internet' that much harder," The Conversation. (Oct. 6, 2014), [Online]. Available: http://theconversation.com/mesh-networks-and-firechat-make-switching-off-the-internet-that-much-harder-32588 (visited on 09/24/2021) (cit. on p. 38).

[126] D. S. Vasiliev, D. S. Meitis, and A. Abilov, "Simulation-Based Comparison of AODV, OLSR and HWMP Protocols for Flying Ad Hoc Networks," in *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, S. Balandin, S. Andreev, and Y. Koucheryavy, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2014, pp. 245–252, ISBN: 978-3-319-10353-2. DOI: 10.1007/978-3-319-10353-2_21 (cit. on p. 143).

[127] M. foundation. "Rust Programming Language," Rust - A language empowering everyone to build reliable and efficient software. (May 15, 2015), [Online]. Available: https://www.rust-lang.org/ (visited on 12/21/2020) (cit. on pp. 23, 97).

[128] T. Ojha, S. Misra, and N. S. Raghuwanshi, "Wireless sensor networks for agriculture: The state-of-the-art in practice and future challenges," *Computers and Electronics in Agriculture*, vol. 118, pp. 66–84, Oct. 1, 2015, ISSN: 0168-1699. DOI: 10.1016/j.compag.2015.08.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169915002379 (visited on 09/27/2021) (cit. on p. 38).

[129] J. Petajajarvi, K. Mikhaylov, A. Roivainen, T. Hanninen, and M. Pettissalo, "On the coverage of LPWANs: Range evaluation and channel attenuation model for LoRa technology," in *2015 14th International Conference on ITS Telecommunications (ITST)*, Dec. 2015, pp. 55–59. DOI: 10.1109/ITST.2015.7377400 (cit. on p. 142).

[130] Y. Yan, N. H. Tran, and F. S. Bao, "Gossiping along the Path: A Direction-Biased Routing Scheme for Wireless Ad Hoc Networks," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2015, pp. 1–6. DOI: 10.1109/GLOCOM.2015.7417867 (cit. on p. 41).

[131] N. Hossein Motlagh, T. Taleb, and O. Arouk, "Low-Altitude Unmanned Aerial Vehicles-Based Internet of Things Services: Comprehensive Survey and Future Perspectives," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 899–922, Dec. 2016, ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2612119 (cit. on pp. 19, 38).

[132] International Telecommunication Union, "Article 2 - Nomenclature," in *ITU Radio Regulations (4 Vol. Set), 2016 Edition*, 2016th ed., Geneva, Switzerland: International Telecommunication Union, Nov. 1, 2016, pp. 27–28, ISBN: 978-92-61-19997-5 (cit. on p. 31).

[133] ——, *ITU Radio Regulations (4 Vol. Set), 2016 Edition*, 2016th ed. Geneva, Switzerland: International Telecommunication Union, Nov. 1, 2016, 442 pp., ISBN: 978-92-61-19997-5 (cit. on pp. 35, 165).

[134] S. Milani and A. Memo, "Impact of drone swarm formations in 3D scene reconstruction," in *2016 IEEE International Conference on Image Processing (ICIP)*, Sep. 2016, pp. 2598–2602. DOI: 10.1109/ICIP.2016.7532829 (cit. on p. 19).

[135] T. Prevot, J. Rios, P. Kopardekar, J. E. Robinson III, M. Johnson, and J. Jung, "UAS Traffic Management (UTM) Concept of Operations to Safely Enable Low Altitude Flight Operations," in *16th AIAA Aviation Technology, Integration, and Operations Conference*, Washington, D.C.: American Institute of Aeronautics and Astronautics, Jun. 13, 2016, ISBN: 978-1-62410-440-4. DOI: 10.2514/6.2016-3292. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2016-3292 (visited on 09/10/2021) (cit. on pp. 19, 38).

[136] F. project. "Firechat - open source chat built on Firebase," Firechat. (Aug. 31, 2016), [Online]. Available: https://firechat.firebaseapp.com/ (visited on 11/21/2020) (cit. on p. 38).

[137] I. T. Union, "Article 5 - Frequency Allocations," in *ITU Radio Regulations (4 Vol. Set), 2016 Edition*, 2016th ed., Geneva, Switzerland: International Telecommunication Union, Nov. 1, 2016, pp. 37–185, ISBN: 978-92-61-19997-5 (cit. on p. 91).

[138] I. Voitenko, *Iuriivoitenko/simpleMANET*, 2016. [Online]. Available: https://github.com/iuriivoitenko/simpleMANET (visited on 02/19/2021) (cit. on p. 92).

[139] A. Balasubramanian, M. S. Baranowski, A. Burtsev, A. Panda, Z. Rakamarić, and L. Ryzhyk, "System Programming in Rust: Beyond Safety," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, ser. HotOS '17, New York, NY, USA: Association for Computing Machinery, May 7, 2017, pp. 156–161, ISBN: 978-1-4503-5068-6. DOI: 10.1145/3102980.3103006. [Online]. Available:

https://doi.org/10.1145/3102980.3103006 (visited on 11/22/2021) (cit. on p. 97).

[140] M. Caballero and J. Crowcroft, "Demo: MeshSim: A Wireless Ad-Hoc Network Development Platform," in *Proceedings of the 12th Workshop on Challenged Networks*, ser. CHANTS '17, New York, NY, USA: ACM, 2017, pp. 25–27, ISBN: 978-1-4503-5144-7. DOI: 10.1145/3124087.3124089. [Online]. Available: http://doi.acm.org/10.1145/3124087.3124089 (cit. on pp. 94, 95).

[141] C. L. Popa, G. Carutasu, C. E. Cotet, N. L. Carutasu, and T. Dobrescu, "Smart City Platform Development for an Automated Waste Collection System," *Sustainability*, vol. 9, no. 11, p. 2064, 11 Nov. 2017. DOI: 10.3390/su9112064. [Online]. Available: https://www.mdpi.com/2071-1050/9/11/2064 (visited on 09/27/2021) (cit. on p. 38).

[142] C. Rey-Moreno. "Supporting the Creation and Scalability of Affordable Access Solutions: Understanding Community Networks in Africa," Internet Society. (May 23, 2017), [Online]. Available: https://www.internetsociety.org/resources/doc/2017/supporting-the-creation-and-scalability-of-affordable-access-solutions-understanding-community-networks-in-africa/ (visited on 09/10/2021) (cit. on p. 20).

[143] L. Vangelista, "Frequency Shift Chirp Modulation: The LoRa Modulation," *IEEE Signal Processing Letters*, vol. 24, no. 12, pp. 1818–1821, Dec. 2017, ISSN: 1558-2361. DOI: 10.1109/LSP.2017.2762960 (cit. on p. 142).

[144] S. Gallagher. "The Army's costly quest for the perfect radio continues," Ars Technica. (Aug. 3, 2018), [Online]. Available: https://arstechnica.com/information-technology/2018/03/the-armys-costly-quest-for-the-perfect-radio-continues/ (visited on 09/22/2020) (cit. on pp. 35, 87).

[145] D. Medhi and K. Ramasamy, "Routing Algorithms: Shortest Path, Widest Path, and Spanning Tree," in *Network Routing*, Elsevier, 2018, pp. 30–63, ISBN: 978-0-12-800737-2. DOI: 10.1016/B978-0-12-800737-2.00003-X. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B978012800737200003X (visited on 09/15/2021) (cit. on p. 21).

[146] A. Nayyar, "Flying Adhoc Network (FANETs): Simulation Based Performance Comparison of Routing Protocols: AODV, DSDV, DSR, OLSR, AOMDV and HWMP," in *2018 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, Aug. 2018, pp. 1–9. DOI: 10.1109/ICABCD.2018.8465130 (cit. on p. 143).

[147] S. Aoki and R. Rajkumar, "V2V-based Synchronous Intersection Protocols for Mixed Traffic of Human-Driven and Self-Driving Vehicles," in *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug. 2019, pp. 1–11. DOI: 10.1109/RTCSA.2019.8864572 (cit. on pp. 20, 38).

[148] T. R. P. Foundation. "Raspberry Pi 4 Model B," Raspberry Pi. (Jun. 24, 2019), [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ (visited on 01/18/2021) (cit. on p. 114).

[149] H. Kang, H. Kim, and Y. M. Kwon, "RECEN:Resilient MANET Based Centralized Multi Robot System Using Mobile Agent System," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec. 2019, pp. 1952–1958. DOI: 10.1109/SSCI44817.2019.9002704 (cit. on p. 19).

[150] C. Perkins, J. Dowdell, S. Ratliff, L. Steenbrink, and V. Pritchard, "Ad Hoc On-demand Distance Vector Version 2 (AODVv2) Routing," IETF Secretariat, RFC draft-perkins-manet-aodvv2-03, Feb. 28, 2019. [Online]. Available: https://tools.ietf.org/html/draft-perkins-manet-aodvv2-03#section-16.2 (visited on 10/07/2020) (cit. on pp. 71, 144, 157).

[151] M. Rogers and E. Saitta. "Briar - Secure messaging, anywhere," Briar. (Nov. 6, 2019), [Online]. Available: https://briarproject.org/index.html (visited on 10/06/2020) (cit. on p. 38).

[152] D. Seifert. "Google Nest Wifi review: The smarter mesh router," The Verge. (Nov. 4, 2019), [Online]. Available: https://www.theverge.com/2019/11/4/20946586/google-nest-wifi-review-mesh-router-price-specs-features (visited on 09/24/2021) (cit. on p. 37).

[153] I. T. Union, "RECOMMENDATION ITU-R P.1238-10 - Propagation data and prediction methods for the planning of indoor radiocommunication systems and radio local area networks in the frequency range 300 MHz to 450 GHz*," International Telecommunication Union, Geneva, Switzerland, P.1238-10, Aug. 2019, p. 28 (cit. on p. 25).

[154] C. Welch. "Amazon is buying mesh router company Eero," The Verge. (Feb. 11, 2019), [Online]. Available: https://www.theverge.com/2019/2/11/18220960/amazon-eero-acquisition-announced (visited on 09/24/2021) (cit. on p. 37).

[155] A. BERTOLAUD, I. CALABRESE, J. CATALANO, *et al.*, "LoRaWAN Link Layer Specification v1.0.4," LoRa Alliance, Fremont, CA, USA, TS001-1.0.4, Oct. 2020, p. 90. [Online]. Available: https://lora-alliance.org/resource_hub/lorawan-104-specification-package/ (visited on 09/27/2021) (cit. on p. 39).

[156] Bridgefy. "Bridgefy - Make Your App Work Without Internet," Bridgefy. (2020), [Online]. Available: https://bridgefy.me/ (visited on 11/21/2020) (cit. on p. 38).

[157] W. Chen, J. Liu, H. Guo, and N. Kato, "Toward Robust and Intelligent Drone Swarm: Challenges and Future Directions," *IEEE Network*, vol. 34, no. 4, pp. 278–283, Jul. 2020, ISSN: 1558-156X. DOI: 10.1109/MNET.001.1900521 (cit. on p. 19).

[158] N. Corbyn, *Dash83/cluster*, Dec. 17, 2020. [Online]. Available: https://github.com/Dash83/cluster (visited on 01/18/2021) (cit. on p. 115).

[159] P. Creative. "MPU5," Persistent Systems : Wave Relay, Mobile Ad-Hoc Networking Solution MANET, Wireless Secure Scalable Communication. (Sep. 22, 2020), [Online]. Available: https://www.persistentsystems.com/mpu5/ (visited on 09/22/2020) (cit. on p. 35).

[160] Dragino. "Raspberry Pi HAT featuring GPS and LoRa technology." (Aug. 1, 2020), [Online]. Available: https://www.dragino.com/products/lora/item/106-lora-gps-hat.html (visited on 01/18/2021) (cit. on p. 114).

[161] I. W. S. W. Group, "IEEE Approved Draft Standard for Information Technology – Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks – Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE P802.11-REVmd/D5.0, September 2020*, pp. 1–4668, Dec. 2020 (cit. on pp. 35, 36, 164).

[162] T. P. G. D. Group. "PostgreSQL: About," What is PostgreSQL? (2020), [Online]. Available: https://www.postgresql.org/about/ (visited on 12/01/2020) (cit. on p. 104).

[163] J. C. Kabugo, S.-L. Jämsä-Jounela, R. Schiemann, and C. Binder, "Industry 4.0 based process data analytics platform: A waste-to-energy plant case study," *International Journal of Electrical Power & Energy Systems*, vol. 115, p. 105 508, Feb. 1, 2020, ISSN: 0142-0615. DOI: 10.1016/j.ijepes.2019.105508. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061518336731 (visited on 09/27/2021) (cit. on p. 38).

[164] S. Mushero. "Introducing runqstat — New Linux Run Queue & Load Average Tool," Medium. (Oct. 15, 2020), [Online]. Available: https://medium.com/@steve.mushero/introducing-runqstat-new-linux-run-queue-load-average-tool-b6c196acde94 (visited on 01/11/2021) (cit. on p. 127).

[165] PricewaterhouseCoopers. "The impact of drones on the UK economy," PwC. (Oct. 24, 2020), [Online]. Available: https://www.pwc.co.uk/issues/intelligent-digital/the-impact-of-drones-on-the-uk-economy.html (visited on 10/24/2020) (cit. on p. 38).

[166] C. Systems. "Cisco Meraki — Mesh routing," Cisco Meraki Technologies. (2020), [Online]. Available: https://meraki.cisco.com/technologies/mesh-routing (visited on 09/24/2021) (cit. on p. 37).

[167] C. K. Toh, J. A. Sanguesa, J. C. Cano, and F. J. Martinez, "Advances in smart roads for future smart cities," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 476, no. 2233, p. 20 190 439, Jan. 29, 2020. DOI: 10.1098/rspa.2019.0439. [Online]. Available: https://royalsocietypublishing.org/doi/full/10.1098/rspa.2019.0439 (visited on 09/14/2021) (cit. on pp. 20, 38).

[168] M. Verdi, A. Sami, J. Akhondali, F. Khomh, G. Uddin, and A. Karami Motlagh, "An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020, ISSN: 1939-3520. DOI: 10.1109/TSE.2020.3023664 (cit. on p. 96).

[169] L. Achom, C. Rey-Moreno, S. Song, V. Mabika, and S. Luca de Tena. "Essential Tools and Resources for Community Networks," Internet Society. (Jul. 28, 2021), [Online]. Available: https://www.internetsociety.org/events/summit-community-networks-africa/2021-22/event-1/ (visited on 09/10/2021) (cit. on p. 20).

[170] A. Corp. "Amazon Sidewalk." (2021), [Online]. Available: https://www.amazon.com/Amazon-Sidewalk/b?ie=UTF8&node=21328123011 (visited on 06/27/2021) (cit. on p. 19).

[171] A. Dusia, R. Ramanathan, W. Ramanathan, C. Servaes, and A. Sethi, "ECHO: Efficient Zero-Control-Packet Broadcasting for Mobile Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021, ISSN: 1558-0660. DOI: 10.1109/TMC.2021.3055819 (cit. on p. 41).

[172] M. Emre, R. Schroeder, K. Dewey, and B. Hardekopf, "Translating C to safer Rust," *Proceedings of the ACM on Programming Languages*, vol. 5, 121:1–121:29, OOPSLA Oct. 15, 2021. DOI: 10.1145/3485498. [Online]. Available: https://doi.org/10.1145/3485498 (visited on 11/24/2021) (cit. on p. 97).

[173] S. C. Foundation. "Standard C++." (2021), [Online]. Available: https://isocpp.org/ (visited on 12/27/2021) (cit. on p. 96).

[174] N. Imtiaz and L. Williams, "Memory Error Detection in Security Testing," Apr. 9, 2021. arXiv: 2104.04385 [cs]. [Online]. Available: http://arxiv.org/abs/2104.04385 (visited on 12/27/2021) (cit. on p. 96).

[175] C. I. King, *ColinIanKing/powerstat*, Feb. 19, 2021. [Online]. Available: https://github.com/ColinIanKing/powerstat (visited on 02/25/2021) (cit. on p. 171).

[176] Mathworks. "Choose a Solver - MATLAB & Simulink." (2021), [Online]. Available: https://uk.mathworks.com/help/simulink/ug/choose-a-solver.html (visited on 02/19/2021) (cit. on pp. 92, 94).

[177] T. Preston-Werner and P. Gedam. "TOML specification v1.0.0." (Jan. 11, 2021), [Online]. Available: https://toml.io/en/v1.0.0 (visited on 02/17/2021) (cit. on p. 107).

[178] O. RT. "RT-LAB - Powering Real-Time Simulation," OPAL-RT. (2021), [Online]. Available: https://www.opal-rt.com/software-rt-lab-2/ (visited on 02/19/2021) (cit. on p. 93).

[179] SEMTECH. "Analog and Mixed-Signal Semiconductors — Semtech." (2021), [Online]. Available: https://www.semtech.com (visited on 11/27/2020) (cit. on p. 142).

[180] M. Warning, *Mesh Network Lab*, Nov. 25, 2021. [Online]. Available: https://github.com/mwarning/meshnet-lab (visited on 11/28/2021) (cit. on p. 95).

[181] H. Xu, Z. Chen, M. Sun, Y. Zhou, and M. R. Lyu, "Memory-Safety Challenge Considered Solved? An In-Depth Study with All Rust CVEs," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 1, 3:1–3:25, Sep. 28, 2021, ISSN: 1049-331X. DOI: 10.1145/3466642. [Online]. Available: https://doi.org/10.1145/3466642 (visited on 11/24/2021) (cit. on p. 97).

[182] I. T. Union. "What is meant by ISM applications and how are the related frequencies used?" International Telecommunication Union. (), [Online]. Available: https://www.itu.int/net/ITU-R/terrestrial/faq/index.html#g013 (visited on 06/07/2017) (cit. on p. 39).