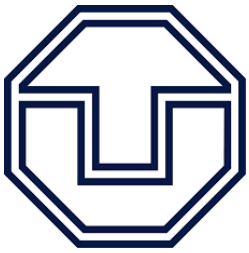


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE (DEI)

DEUTSCHE TELEKOM CHAIR OF COMMUNICATION NETWORKS - TU DRESDEN

MASTER DEGREE IN ICT FOR INTERNET AND MULTIMEDIA

Approximating optimal Broadcast in Wireless Mesh Networks with Machine Learning

---

9 September 2019

ACADEMIC YEAR 2018/2019

Candidate:
Giovanni PERIN

Supervisor:
Prof. Leonardo BADIA

"Experience serves not only to confirm theory, but differs from it without disturbing it, it leads to new truths which theory only has not been able to reach."

Jean-Baptiste Le Rond d'Alembert

UNIVERSITÀ DEGLI STUDI DI PADOVA

Abstract

Dipartimento di Ingegneria dell'Informazione (DEI)

Master Degree in ICT for Internet and Multimedia

Approximating optimal Broadcast in Wireless Mesh Networks with Machine Learning

by Giovanni PERIN

With the advent of the Internet of Things and the forthcoming beginning of the 5G era, the need for efficient wireless broadcast protocols arises, in order to enable a wide range of future use cases. Mesh networks are a very generic type of topology, working among peers and therefore perfectly adapting to the IoT case. Moreover, they are attractive for their reliability from a security point of view and high adaptability. Nevertheless, state-of-the-art protocols use very primitive solutions for broadcast transmissions, like repeating each message three times, which is the case of a commonly used protocol named BATMAN. Hence, a more efficient solution has to be found.

The problems of minimum-delay as well as power-optimal broadcast have been researched and proved to be either NP-hard or NP-complete, respectively. Therefore, Machine Learning shall be used to approximate an efficient solution. Specifically, in this thesis, a reinforcement learning approach based on Multi-Armed Bandits is explored, designing two algorithms, one employing traditional Q-tables and the other one using Bayesian Neural Network (BNNs).

The wireless medium exhibits a broadcast characteristic by itself which is exploited by Opportunistic Routing. Under certain assumptions, a protocol called MORE, that employs Random Linear Network Coding, was proved to be optimal in terms of airtime for a unicast stream. In a first stage, this protocol is used to establish how close the developed solutions come to optimality. Then, the algorithms are applied to the broadcast scenario and analyzed in terms of airtime and latency, inspecting as well the adaptability of the learning based protocols to channel quality variations. The simulations, performed on a simple environment created from scratch with *Python*, show that the BNN based algorithm has a median improvement of $8\% - 12\%$ and $10\% - 22\%$ in terms of airtime concerning MORE multicast and BATMAN respectively, while being $6\% - 17\%$ and $4\% - 31\%$ better in latency when comparing to the same two protocols. Moreover, when testing with the addition of a maximum allowed delay, the constraint is satisfied consistently in over the 97% of the cases.

Acknowledgments

This thesis is the result of a work carried out during an Erasmus period at the Technische Universität Dresden. It was possible thanks to Professors Leonardo Badia, who directed me towards the Communication Networks research group and Frank H. P. Fitzek, who accepted me in his Chair.

I want to thank, then, my Dresdner supervisor and advisor David Nophut, who has always been available with his tips and knowledge about routing protocols, and led me on the right direction, helping to keep the focus on what was relevant. I am also grateful to Dr. Riccardo Bonetto for his precious advices about reinforcement learning and machine learning in general.

Moreover, a special thanks goes to my fellows and office mates, Marek, Peter, Sandra, Alexander, Ullrich-Matthias and Andreas, with whom I had interesting discussions about engineering and differences between Italy and Germany. I want to thank also the International friends I found in Dresden during my stay. With all of them, and with Martina in particular, I had a great time outside the office. Finally, big thanks go to my family, as they always supported me.

Contents

List of Figures

List of Tables

List of Abbreviations

ACR	**A**nt **C**olony **R**outing
AODV	**A**d-hoc **O**n-demand **D**istance **V**ector
ARQ	**A**utomatic **R**epeat re**Q**uest
BATMAN	**B**etter **A**pproach **T**o **M**obile **A**d-hoc **N**etworks
BNN	**B**ayesian **N**eural **N**etwork
BS	**B**ase **S**tation
CAM	**C**ooperative **A**wareness **M**essage
CBF	**C**ontention **B**ased **F**orwarding
CMAB	**C**ontextual **M**ulti-**A**rmed **B**andit
DDPG	**D**eep **D**eterministic **P**olicy **G**radient
DENM	**D**istributed **E**nvironment **N**otification **M**essage
DQN	**D**eep **Q**-**N**etwork
DRL	**D**eep **R**einforcement **L**earning
DSR	**D**ynamic **S**ource **R**outing
EAX	**E**xpected **A**nypath T**X** (nr. of transmissions)
ETSI	**E**uropean **T**elecommunications **S**tandards **I**nstitute
ETX	**E**xpected **TX** (nr. of transmissions)
ExOR	**Ex**tremely **O**pportunistic **R**outing
FEC	**F**orward **E**rror **C**orrection
GeRaF	**Ge**ographic **Ra**ndom **F**o**r**warding
GFC	CBF with **G**reedy **FC** (forwarding)
HARQ	**H**ybrid **A**utomatic **R**epeat re**Q**uest
IETF	**I**nternet **E**ngineering **T**ask **F**orce
IoT	**I**nternet **o**f **T**hings
IP	**I**nternet **P**rotocol address
MAB	**M**ulti-**A**rmed **B**andit
MACMAB	**M**ulti-**A**gent **C**ontextual **M**ulti-**A**rmed **B**andit
MADCMAB	**M**ulti-**A**gent **D**eep **C**ontextual **M**ulti-**A**rmed **B**andit
MADDPG	**M**ulti-**A**gent **D**eep **D**eterministic **P**olicy **G**radient
MADQN	**M**ulti-**A**gent **D**eep **Q**-**N**etwork
MADRL	**M**ulti-**A**gent **D**eep **R**einforcement **L**earning
MAE	**M**ean **A**bsolute **E**rror
MANET	**M**obile **A**d-hoc **NET**work
MDP	**M**arkov **D**ecision **P**rocess
ML	**M**achine **L**earning

MORE	**M**AC-independent **O**pportunistic **R**outing & **E**ncoding
MPR	**M**ulti**P**oint **R**elay
MSE	**M**ean **S**quared **E**rror
NC	**N**etwork **C**oding
NE	**N**ash **E**quilibrium
NN	**N**eural **N**etwork
OGM	**O**ri**G**inator **M**essage
OLSR	**O**ptimized **L**ink **S**tate **R**outing
OR	**O**pportunistic **R**outing
RERR	**R**oute **ERR**or
RL	**R**einforcement **L**earning
RLNC	**R**andom **L**inear **N**etwork **C**oding
RREP	**R**oute **REP**ly
RREQ	**R**oute **REQ**uest
SAC	**S**oft **A**ctor-**C**ritic
SARSA	**S**tate-**A**ction-**R**eward-**S**tate-**A**ction
SDN	**S**oftware **D**edined **N**etwork
SNR	**S**ignal to **N**oise **R**atio
SOAR	**S**imple **O**pportunistic **A**daptive **R**outing
TC	**T**opology **C**ontrol
TTL	**T**ime **T**o **L**ive
UCB	**U**pper **C**onfidence **B**ound
VANET	**V**ehicular **A**d-hoc **NET**work
WMN	**W**ireless **M**esh **N**etwork
WSN	**W**ireless **S**ensor **N**etwork
XOR	e**X**clusive logic **OR**

Chapter 1

Introduction

A Wireless Mesh Network (WMN) is a type of communication network which consist of a set of radio nodes organized in a mesh topology, able to dynamically self-organize. Part or all of these nodes have routing functions, meaning that they actively participate to the decisions about how the packets' delivering is performed in the network. Other nodes not being *routers* are called *clients* and they can be terminals such as laptops, mobile phones and other wireless devices. Mesh routers form the backbone for the clients and often, but not always, have minimal mobility. Advantages of using this kind of topology in the wireless medium include high resilience and ability to rapidly find new routes in case of link failure or topology changes. Further more, it allows multiple sources to transmit simultaneously, hence exploiting spatial reuse, and it is a decentralized architecture without any network controller, improving thus security. Just this last fact is sufficient to understand that mesh networks will be at the centre of the Internet of Things revolution that will have place in the next decade, since IoT requires indeed a distributed and self-organizing network architecture.

Because WMNs can be large, routing algorithms must be efficient. Optimization can concern a number of different objectives, like throughput, energy and end-to-end delay. In this thesis, I develop a framework based on machine learning for the minimization of the airtime cost of the network. Since, for the sake of simplicity, I work in a single rate and flow scenario, this corresponds to the minimization of the number of transmissions. This optimization objective reflects directly on both goodput and energy consumption, improving them. This is because transmitting less means having less overhead that fills the network uselessly and, at the same time, saving energy for future transmissions, which is extremely important when devices have a finite battery life. In a second moment, I also add to the optimization goal hard latency constraints, which are typical of real-time applications like streaming, online gaming, vehicular and healthcare communications.

In the past, routing algorithms operated sending uncoded packets along a computed path considered to be the shortest one according to a certain metric. For example, Dijkstra's and Bellman-Ford's algorithms are well known for finding shortest paths in a graph in terms of hop counts. Therefore, to guarantee a sufficient reliability against lossy links, which in wireless networks are very common, existing

protocols used Automatic Repeat reQuest (ARQ), Forward Error Correction (FEC) or a mixture of the two approaches (HARQ). ARQ consists in asking the repetition of the transmission of a corrupted packet, whereas FEC involves the attempt of reconstructing the original transmitted symbol starting from the corrupted one. From the beginning of the millennium, however, Network Coding (NC) started to be studied and researchers found that it was a suitable method for achieving the maximum theoretical multicast throughput [2]. NC techniques integrate original informative symbols by means of elementary operations like bit-by-bit XOR, creating in this way coded packets, sent by the source and decoded at the destination. Since the wireless medium is a broadcast one by nature and at that time the wireless technology was starting to be good enough, the first protocols combining routing with network coding arose, showing immediately that this solution was worthy.

In the same years, Opportunistic Routing (OR) was conceived, with a protocol named ExOR [13], that sent messages across multiple paths, thus exploiting lucky receptions. With MORE [16], Network Coding was integrated to this kind of approach, allowing for a better spatial reuse and a higher throughput performance. MORE was designed for the minimization of the airtime cost in wireless networks for the unicast case, i.e. the scenario in which a single source sends a message to a single destination. This protocol was also proven to be theoretically optimal in such a case.

MORE was then extended to multicast communications, namely the setting in which the source transmits to more than one destination. When the message has to be delivered to all the nodes of the network, we talk about message broadcasting. This paradigm is used nowadays for the spreading of audio and video contents, for instance for the television or web streaming. With the expected diffusion of the IoT, however, message broadcasting will soon become of extreme importance also for connected applications such as smart cities, smart grids, vehicular networks, domotics, digital health and industrial and medical robotics. Even though broadcast will assume an increasing important role in the near future, a theoretical optimum is not yet available for this type of communications, at least for what concerns the airtime [1].

The objective of this thesis is thus to approximate an optimal solution in terms of airtime cost for message broadcasting in WMNs, where all the nodes are considered to have full routing functions. To reach this goal, I use some of the most recent advances in machine learning, particularly in the field of reinforcement learning, to develop an unsupervised and data-driven framework. Following this direction, I implement contextual Multi-Armed Bandits in two versions: a classic Q-table and a Neural Network based solution. The goal of the bandits is to determine the minimum number of transmissions that each node should perform to broadcast the information in the network. Moreover, the implemented framework supports fully

[1] Indeed, a theoretical optimum generally exists in terms of throughput [6]. However, besides being the objectives different, the distributed Linear Programs proposed are infeasible for large networks due to the high computational complexity.

distributed algorithms, since nodes in the network are treated as independent learners, each with its own bandit in a multi-agent scenario. The only information about the network known by each node is what it can get just by overhearing to its one-hop neighbors transmissions. This is extremely important because many protocols rely on full knowledge of the network's topology, having thus a relevant amount of control packets traffic. This thesis is structured in two steps. In the first one, the learning framework is compared with the theoretically optimal solution represented by MORE in the unicast case, to assess whether the designed algorithms are satisfactory enough. This first phase can be considered as a preliminary for the second one, which is the actual work on broadcast. Here, a comparison of the implemented algorithms with the multicast version of MORE and of another popular protocol named BATMAN [19] is accomplished. To summarize, the original contributions of this work are:

- Implementation of a **reinforcement learning** framework for the minimization of the airtime cost. Previous work employing machine learning focused on single path selection minimizing the end-to-end delay and was done only for the unicast case.
- Use of only **local information**, gathered by each node just by overhearing its neighbors' transmissions. The topology configuration at a larger scale is learned directly from data and this allows to reduce significantly the overhead due to common hello messages.
- Realization of solutions based on **fully distributed** algorithms, which is a strict requirement of WMNs, since they lack of a network controller in their infrastructure.
- Good **scalability** with respect to the mesh size, which is a direct consequence of the fact that nodes only use local information and the optimization is approximated in a distributed way.
- High **adaptability** to the environment and to non-stationary conditions, as routers learn directly from on-field data.

The rest of this thesis is structured as follows. In Chapter 2 an intuitive explanation of the use of network coding is given, together with a motivating example and the definitions of the most important parameters of Random Linear Network Coding (RLNC). The chapter proceeds with a review of the most commonly used routing algorithms, both in the unicast and multicast case, with a particular emphasis on MORE and BATMAN, which constitute the main terms of comparison with the implemented learning algorithms. Finally, previous work found in the literature about the use of reinforcement learning for routing tasks is presented in the last section. Chapter 3 is dedicated to the mathematical definition of the optimization objectives, as well as the presentation of the framework employed in this thesis. Specifically,

some theoretical background on recent findings in the Deep Reinforcement Learning (DRL) field and multi-agent control will be given before a detailed explanation of the learning algorithms designed to reach the defined goals. In Chapter 4 the simple simulation environment created in Python is presented, together with the chosen settings and the experiments performed. Chapter 5 is dedicated to the simulation results and their analysis and, finally, in Chapter 6 a resume of the notable results of this thesis is drawn up, possible future work is suggested and current open issues are highlighted.

Chapter 2

Related Work

2.1 Network coding

Network coding was introduced between the late '90s and the beginning of the millennium to improve network throughput, in particular as for what concerns the multicast scenario. Besides, it offers improvements in wireless resources, security, complexity and resilience. Studies show that using network coding is also beneficial in terms of the minimization of the number of transmissions in broadcast, because coded packets are more likely to be useful for a higher number of nodes and most of the ARQ retransmissions can be avoided [7], [8].

The classical introductory example is the so called "butterfly" network, that is shown in Fig. 2.1 and was firstly introduced in [2]. The source s must deliver to the destinations t and u three packets, namely x, y and z. A traditional routing solution can be the one that follows. At time slot 1, the source sends x and y, routing x to both the destinations and y only to u. At time slot 2, the source emits y and z, successfully delivering z to both the destinations and y only to t. The multicast throughput of this solution is 1.5 packets per channel use, because three packets are delivered in 2 time slots. This throughput is the best achievable by any routing solution. However, if node b, instead of routing one packet and blocking the other, transmits the bit-by-bit XOR of x and y, both destinations receive the packet $x \oplus y$, from which node t can immediately recover y as $x \oplus (x \oplus y) = y$ and, similarly, node u can recover x. In this way, the multicast throughput is increased to 2 packets per channel use, at the cost of performing a coding operation at node b and the correspondent decoding operations at the destinations. In the case of this topology, there is no way to do better: actually, the destinations are connected to the network by two edges, so it is impossible to receive more than two packets per unit time [1].

2.1.1 Random Linear Network Coding (RLNC)

Linear coding is an operation defined over a Galois finite field, usually denoted as $\mathbb{F}_{p^n}$ or $GF(p^n)$, where p is a prime number and $n \geq 1$. Since we want to send bits 0 or 1 over the channel, in network coding we use finite fields of the form $\mathbb{F}_{2^n}$, so that the sum of two polynomials in these fields is simply defined as the bit-by-bit XOR of them, like in the example above. If we also need multiplications by scalar

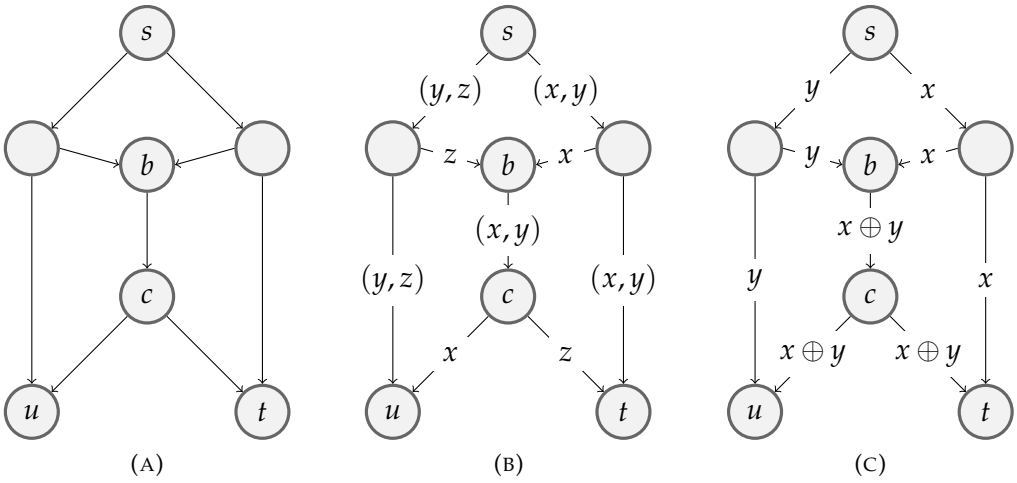


FIGURE 2.1: Butterfly example.

values, it is sufficient to combine XOR operations with left shifts. The number n defines the complexity of the field, that is to say how many different combinations of polynomials can be generated. In our case, if $n = 1$ so that we are in $\mathbb{F}_2$, each packet can be either selected or not selected, with scalar coefficients 0 or 1. As n grows, the number of possible scalar coefficients grows accordingly. In our scenario, sometimes $\mathbb{F}_2$ is chosen for its simplicity both in the coding and decoding operations. To increase reliability, however, often $\mathbb{F}_8$ is preferred, at the cost of a higher decoding computational complexity.

Random Linear Network Coding (RLNC) is a linear coding technique which consists of summing up available packets multiplied by coefficients randomly picked from the finite field selected. Two different approaches can be distinguished and they are generation-based RLNC and sliding window RLNC [4]. In the former approach a set of symbols to be coded is called generation and the generation size defines the number of original symbols inside each disjoint generation. In the latter case, a sliding window with a certain size and stride spans the symbols. Here, the coding is partially overlapping between subsequent windows. In both cases native symbols are picked up and coded with the previously described approach to generate coded symbols, described by *code vectors*. The code vector, of length equal to the number of symbols in a generation or a window, collects the multiplicative coefficients of each symbol. Intermediate nodes and destinations collect the data arriving from the upstream and build a matrix with the code vectors. When this matrix reaches full rank, every symbol of the current generation or window can be decoded. If K is the number of native symbols, we need $N \geq K$ coded symbols to retrieve the original information, because, in general, some of the coded symbols received may be linearly dependent on the others.

Although the butterfly example showed the advantage of using Network Coding in terms of throughput, there are still some open issues:

- High computational complexity due to the gaussian elimination process, used to solve the linear system in the decoding phase;
- Linear dependency among coding blocks, which can reduce significantly the rate of innovative packets received;
- High overhead due to the fact that code vectors must be appended in the header of transmitted packets;
- High latency, because all information will be retrieved when the matrix reaches full rank, which happens at the end of the transmission in generation-based coding.

For what concerns the first two points, a trade-off is obtained by choosing a proper finite field. A Galois field with higher complexity, as already remarked, will require higher efforts in the gaussian elimination process but will also decrease the probability of sending linearly dependent packets, as there are more coefficients combinations available. The overhead is instead kept acceptable for a certain protocol choosing the generation or window size, which determine the code vectors' lengths. The last point is a bit trickier but recent literature proposed PACE, a generation-based coding algorithm which consists of adding m coded packets every n non coded ones, thus increasing the latency performance while keeping the reliability advantage of NC [9]. Sliding window coding, instead, suffers less the latency problem but more the computational complexity. Therefore, Caterpillar was introduced as a sliding-window solution approaching the lower computational complexity of generation RLNC while keeping the typical latency of this kind of approach [10].

2.2 Routing protocols

Traditional routing approaches include Optimized Link State Routing (OLSR) and Ad hoc On-demand Distance Vector (AODV), both standardized by the Internet Engineering Task Force (IETF) in 2003 [5], [11].

OLSR is a proactive routing protocol in which each node of the network keeps a routing table. This table is built basing on a high knowledge of the topology, deriving from the exchanging of Topology Control (TC) packets between routers. Nodes in OLSR also keep a list of neighbors, only considering bi-directional links, estimated via HELLO messages. Control packets are flooded from a node by a subset of relays in its symmetric 1-hop neighborhood. This set is called MultiPoint Relay (MPR) and is chosen so that it covers all the symmetric strict 2-hop nodes. There may be multiple choices of valid MPRs, however, in general, the smaller the set, the less the control traffic. In OLSR each node selects the path towards the destination by means of Dijkstra's shortest path algorithm, using the simple hop-count as metric. However, in a more recent version of the protocol, olsrd, the metric used has been changed to ETX [12], which has been proved to be more performing. An advantage

of OLSR and pro-active protocols in general is that they always maintain a fresh list of routes, thus making path immediately available. The main drawback is instead the huge amount of routing traffic generated for routes that may never be used.

AODV is a reactive protocol, meaning that it does not keep any routing table but rather every time a transmission is requested a new route is established thanks to the route discovery process. When a node needs to transmit, it broadcasts a Route REQuest (RREQ) through the network. Candidate forwarders respond with a Route REPly (RREP) or a Route ERRor (RERR) in the case in which a relay that was previously available is now not reachable any more. RREQs have a Time To Live (TTL) that limits the number of times that they can be retransmitted, thus preventing the network from being overloaded by control packets. Basing on the RREPs received, the source can establish the best route, using the hop-count metric. AODV is mostly used in mobile ad-hoc networks, where routes change frequently and it is thus useless to store them in a table since they will be outdated after a short while. The advantage of reactive protocols is that they do not need to compute routes that possibly will never be used. On the other hand, when a new route is required, the route discovery process may delay transmissions unacceptably.

While in commonly used protocols multicast communications are often seen as a set of unicast ones, where NC can increase the efficiency (Section 2.1), broadcasting is tackled with a slightly different approach. Actually, a common and surely reliable way to broadcast messages is by flooding, which consists in transmitting the upcoming packet through all the possible routes except the one from which it just arrived. The drawback in this simple algorithm that makes it usually infeasible in this basic version is that the wireless medium gets overloaded rapidly. A trick used to prevent this problem is to choose a probability value p defining a Bernoulli random variable. If this random variable assumes value 1, the packet is transmitted, otherwise it is discarded. However, this procedure is certainly suboptimal at least in terms of latency when using full RLNC, as keeping information for a second moment does not guarantee any advantage.

2.2.1 Opportunistic routing

Opportunistic or anypath routing is the paradigm proposed to improve the throughput of mesh networks by exploiting lucky receptions. In traditional routing, like OLSR and AODV protocols, nodes receiving packets discard them if they are not the next-hop in the pre-computed path between the source and the destination. However, because of the broadcast nature of the wireless medium and the fact that connections are often unreliable, it is extremely inefficient to discard packets at nodes that are closer to the destination. In opportunistic routing every node closer to the destination in some metric sense is a potential forwarder and a specific path is never pre-computed, although a list of possible forwarders may in fact be.

The first protocol that was introduced in this sense was ExOR [13] in 2005. It selects nodes basing on an immediate version of the Expected Transmission Count

(ETX) metric, considering only the forward delivery probabilities. Let i and j be two connected nodes and p_{ij} the delivery probability from i to j. This simplified version of the ETX can be written as:

$$ETX_{ij} = \frac{1}{p_{ij}} \tag{2.1}$$

Multi-hops ETXs are computed simply summing up ETXs of each link in the path. In ExOR a node is considered closer to the destination if it has an ETX to the destination smaller than the one of the current transmitter. However, in large networks, the number of possible forwarders rapidly explodes. This is why ExOR performs pruning selecting in the forwarders list only those nodes that perform at least 10% of the transmissions thanks to a batch simulation. The knowledge of the state of the network, that is to say the estimation of delivery probabilities, is retrieved with periodic link-state flooding of per node measurements. In ExOR nodes have priorities based on their ETX distance to the destination and only one node is allowed to transmit at a time. Thus, higher priority nodes transmit first and lower priority nodes do not transmit if their radio overhears a transmission of the same packet done by a higher priority relay.

Simple Opportunistic Adaptive Routing (SOAR) is a variant of ExOR in which forwarders are constrained to be near the shortest path to improve coordination and communications between relays [14]. Performances are very similar to the ones found for ExOR.

MORE

ExOR and SOAR impose a strict schedule on routers' access to the medium and, although it shows an opportunistic gain, it prevents spatial reuse and thus it may underutilize the channel. To cope with this limit, MAC-independent Opportunistic Routing & Encoding (MORE) was proposed in 2007 [16], [15].

MORE is the first opportunistic protocol which integrates network coding, having the beneficial effects described in Section 2.1. In MORE, the source sends to the destination N transmission batches, each split in K packets, called *native packets*, where K is the generation size of the coding. When the MAC allows the source to transmit, it generates a new coded packet and broadcasts it to its neighbors. In the header of each packet there are the code vector, the batch ID, the source and destination addresses and the forwarders list computed by the source. Each relay, instead, when receiving a packet, performs the following operations:

1. Checks whether it is in the forwarding list. If not, it stops here.
2. Checks the code vector to see if the packet is *innovative*, i.e. if it contains new information. If not, it stops here as well.
3. Creates a new recoded packet which contains also the information of the most recent arrived informative packet and puts it into a buffer.

4. Increments its *credit counter* by its *transmission credit*.

5. Waits for the MAC to allow transmissions and broadcasts the most recent packet in the buffer if its counter is positive.

6. Decrements the *credit counter* by a unit.

As soon as the coding buffer of the destination reaches full rank, the destination sends a batch ACK to the source to stop transmissions. All the relays that overhear the ACK message stop forwarding as well.

The optimization objective of MORE is the minimization of the airtime cost, which produces a fair utilization of spectral resources. To reach this objective MORE computes the optimal forwarding rate for each relay belonging to the forwarders list. The first thing MORE does, is that it orders forwarders basing on their ETX to the destination. It defines the forwarding rate z_i as the expected number of transmissions that forwarder i must do to route one packet from the source, s, to the destination, d. The number of packets that node j receives from the upstream is $\sum_{i>j} z_i(1-\epsilon ij)$, where ϵij is the loss probability of the link between i and j. Every received packet should be forwarded only if no node in the downstream has already overheard it and this happens with probability $\prod_{k<j} \epsilon_{ik}$. The number of packets that j must forward is therefore

$$L_j = \sum_{i>j} \left(z_i(1-\epsilon_{ij}) \prod_{k<j} \epsilon_{ik} \right) \tag{2.2}$$

if j is a relay. If instead we are considering the source we have $L_s = 1$ because it must forward every packet of the generation.

To compute the forwarding rate, MORE considers that node j should transmit each packet until a node closer to the destination does not receive it. Therefore, the number of transmissions that j should perform follows a geometric random variable with parameter $p = \left(1 - \prod_{k<j} \epsilon_{jk}\right)$ and the forwarding rate is thus:

$$z_j = \frac{L_j}{\left(1 - \prod_{k<j} \epsilon_{jk}\right)} \tag{2.3}$$

The last step is the computation of the *transmission credit*, which corresponds to the number of transmissions that j should perform to for every packet it receives from a node in the upstream. For each packet sent by the source, node j receives $\sum_{i>j}(1-\epsilon_{ij})z_i$. So, the *transmission credit* of node j is:

$$TX\_credit_j = \frac{z_j}{\sum_{i>j}(1-\epsilon_{ij})z_i} \tag{2.4}$$

Pruning is performed in MORE like in ExOR, with the exception that the former does not need to simulate a batch to evaluate which nodes should be excluded from the forwarders list. Actually, it is sufficient for MORE to exclude those nodes for

which $z_i < 0.1 \sum_{j \in N} z_j$. Since the list of forwarders is kept in the header, this prevents the size of the overhead to increase too much when networks are large.

The author proves in [16] that MORE is the optimal solution to the problem of the minimization of the overall network-wide cost of the flow, under the constraint that full information is retrieved at the destination. This is true only for an infinite generation size and it is suboptimal for small ones. However, MORE represents the best approximation of an optimal solution for big enough generation sizes. It works well, for example, in the practical scenarios where it is typically between 64 and 256.

There exists also a straightforward extension of the unicast version of MORE to the multicast scenario. The only changes that must be done are:

- The source waits for the ACKs of all the destination before proceeding to the following batch;
- The list of forwarders is the union of the lists for each destination;
- Transmission credits are computed with Eq. 2.4 considering the maximum of the z_i for each unicast flow;
- Both the forwarders list and the transmission credits are recomputed dynamically when a destination is acknowledged, considering only those destinations that are not acknowledged yet.

In terms of throughput performance, MORE reaches 22% better throughput than ExOR and 95% better than traditional routing in the unicast case and this gain is mainly due to MORE's spatial reuse. Throughput gain is instead up to $35 - 200\%$ with respect to ExOR and $3x$ to traditional routing in the multicast case.

Finally, an improved version of MORE uses the metric Expected Anypath Transmissions (EAX or EOTX), that allows the choice of a better list of forwarders [17], [18]. First of all, the ETX based set of forwarders of the source is computed as in the older version. Then, let $C_i^{s,d}$ be the candidate with highest priority (with 1 being the highest) with delivery probability p_i, the EAX can be computed recursively as:

$$EAX(s,d) = \frac{1 + \sum_i EAX(C_i^{s,d}, d) p_i \prod_{j=1}^{i-1}(1 - p_j)}{1 - \prod_i (1 - p_i)} \tag{2.5}$$

A practical forwarders selection is the following. Firstly, a set of forwarders is determined with the classic ETX metric, including in the list neighbors that have a lower ETX to the destination. Then, the node with the best priority is selected to initialize a new set and, finally, the rest of the forwarders are selected incrementally. A new node among the ones that are candidates, joins the actual forwarders only if it reduces the EAX of a factor of at least ϕ.

2.2.2 BATMAN

Better Approach to Mobile Ad-hoc Networking (BATMAN) is a proactive distance vector routing protocol developed in 2008 and proposed as an IETF standard [19]. It

was born as a response to the shortcomings of OLSR, such as the unnecessary flushing up of routing tables or the keeping of outdated routes that may cause routing loops. Its routing objective is the maximization of the probability of delivering a message.

The only thing BATMAN cares about is to remember which is the best next-hop given the source and the destination. It is thus a distributed protocol, requiring only local information and does not need to broadcast the information about topology changes. This is why it is particularly suitable for Mobile Ad-hoc Networks (MANETs) and also for Wireless Mesh Networks (WMNs), where, due to the nature of the wireless medium, the link quality often changes quite rapidly. In BATMAN, each routing node generates periodically Originator Messageses (OGMs) with a sequence number and broadcasts them to its neighbors, setting a TTL and attaching its own IP as originator address. Neighbors that receives the message change the sending address to their own, increase the sequence number and re-broadcast the OGM if the TTL is not expired. The originator does a bidirectional link check when receiving back its own OGM using the sequence number, to verify if the link can be used in both directions. Thus, links are just compared in terms of number of originator messages received within the current sliding window. From a practical viewpoint, a BATMAN route is chosen as [20]:

1. Let m be the message from s to d on the graph G. Remove all the links $(s, i)\ \forall i \notin K$, where K is the set of the neighbors of s;
2. Associate to each link the weight w_{si}, the weight being the number of originator messages received from the destination through i during the current window;
3. Send m to the neighbor with highest link weight;
4. Repeat steps 1-3 until $i = d$.

This protocol, unlike OLSR and AODV, takes into account the possibility that links have different qualities in the two directions.

Comparisons between OLSR and BATMAN show how the latter is able to exploit asymmetrical links, having an improvement in terms of throughput of about 17%, even though OLSR keeps a 1% better packet loss probability. It is also important to highlight that BATMAN is a lightweight protocol, having a 10 fold reduction in CPU load with respect to OLSR in a testbed of 49 nodes and the routing overhead is also much lower [20]. However, there are particular cases where still OLSR has a better throughput, basing on the size of the sliding window [19]. Performance comparison against AODV has also been performed [22]. Authors show how AODV has generally a slightly lower delay because the path selected are shorter, whereas throughput results are very close. BATMAN shows better performances in cases where link connections are poor or in presence of bottlenecks and is in general more

reliable for multi-hops. However, it also causes routes to change at a much higher rate, because AODV performs the route discovery process to find a new route and keeps it unvaried until it is not broken. Finally, AODV is faster on average in route recovery in case of failure, even though it has a larger variance.

There exists also a broadcast extension of BATMAN using the simple concept of classic flooding. This approach consists just in repeating those frames that were received with a new sequence number, unseen until the current time step. Each new frame is transmitted three times with a timeout of $5ms$ to increase the probability of reception at another node in the network, avoiding at the same time the problem of broadcasting storms. A couple of other rules are added to prevent the relay from retransmitting the frame, reducing thus the overhead. An intermediate node, actually, does not rebroadcast the received frame if one of the following cases occur:

- the interface has no neighbor among the ones that require the message, so sending the frame into the void would be useless;
- the interface has only one neighbor among the ones that require the message and it coincides with the previous sender.

2.2.3 Geographical routing

One of the first routing protocols entirely working with geographical information is called Geographic Random Forwarding (*GeRaF*) and was proposed in 2003 [23]. This protocol, conceived particularly for WSNs, relies on the assumption that nodes can determine their location and that the position of the final destination and of the transmitter are known and attached on the message. In this way, an intermediate node which overhears the transmission can easily determine its priority in the set of the forwarders, and decide accordingly if it has to forward the message or not. The main advantage of this idea is that, unlike traditional routing, the source does not need to precompute any end-to-end route to the destination but packets can rather be sent on the fly.

This simple unicast routing solution employing geographical information, represents an inspiration also for applications in several broadcast scenarios. For example, in vehicular networks (VANETs), having destinations in a target area is extremely useful. This can be because nodes, here represented by vehicles, are prone to frequent and quick topology changes and events on the street may have to be notified basing on a geographical position in which nodes are at a certain time frame. Computing fixed routes in such a scenario would be extremely inefficient because these routes would become outdated too fastly.

GeoNetworking is the response to this problem by the European Telecommunications Standards Institute (ETSI) and it is explained in the EN 302 636 series. It supports both the transport of packets to an individual node (GeoUnicast) or any node inside a certain area (GeoBroadcast/GeoAnycast), as well as the broadcast in a

n-hops neighborhood. In this protocol, two types of transport packets are used: Cooperative Awareness Messages (CAMs) for periodic single-hop broadcast and Distributed Environment Notification Messages (DENMs), which are the ones actually used for the delivery of event-driven messages in a geo-area. Further more, there exist three main forwarding schemes:

- **Simple GeoBroadcast** (S) restricts the retransmissions inside the geo-area. Every node inside the area retransmits the packet immediately after the reception. A duplicate detection is performed checking the source identifier and the sequence number. Packets are also endowed of a maximum hop count and a lifetime.

- **Contention-based forwarding** (CBF) sets timers for the receiving neighbors, with duration which is inversely proportional to the distance from the source, i.e. the further the node from the source the shorter the timer. The packet is either rebroadcasted when the timer expires or discarded if it was received a second time in the meanwhile.

- **CBF with greedy forwarding** (GFC) adds a selection of the next-hop forwarder for immediate transmission performed by the current sender, to avoid the delay caused by the timer in CBF. In this scheme, the sender transmits packets to the unicast MAC address of a fictitious destination which correspond to its best geographically located neighbor. There exist also two advanced flavours of GFC. The first one uses a retransmission threshold (GFC-RT) so that a packet is only retransmitted if the number of reception after the first one is above a threshold. CBF has the threshold set to 1, since it discards the packet after the second reception. The second advanced version utilizes a sectorial contention area (GFC-SECT) to restrict the number of candidate forwarders.

A deep performance evaluation of the different algorithms of GeoNetworking in terms of traffic overhead, coverage ratio and end-to-end delay is carried out in [24]. Authors show that S only performs acceptably in low vehicle density conditions and it is soon outperformed by the other algorithms. The CBF algorithm suffers from high overhead in high vehicle density conditions because packets must wait in the queue until the MAC let nodes transmit. In the meanwhile, yet, other timer expires and more nodes than needed transmit the packet. This behavior directly translates into poor performances in terms of coverage ratio for high vehicle densities. GFC, instead, shows difficulties in the latency performance, because the virtual target often lies in the gray zone of the wireless medium and more retransmissions may be required. In light of this, authors suggest to set a minimum link quality requirement for the virtual destination and to enable multipath transmissions instead of persisting using the lossy link.

2.3 Routing with reinforcement learning

In 1994 the work of Boyan and Littman [25] was revolutionary as it proposed *Q-routing*, an algorithm able to find the best path in a generic network, minimizing the expected delivery time. It estimates the delivery time as a Q-value and updates it basing on the rule:

$$\Delta Q_x(d, y) = \eta(q + s + t - Q_x(d, y)) \tag{2.6}$$

where $q + s + t$ is the new estimate, including also the time spent in the queue, $Q_x(d, y)$ is the old Q-value and η is the learning rate, set to 0.5 to account more for the recent behavior of the network. Authors compare this solution with Bellman-Ford's shortest path algorithm and show that Q-routing has much better performances in high load conditions, as this method is able to find rapidly alternative ways avoiding possible bottlenecks. The problem of poor performances in low load conditions is addressed and improved in [26], whereas Kumar and Miikkulainen further improve the exploration-exploitation policy, obtaining better results [27], [28], [29]. A gradient ascent algorithm is finally proposed in [30], with exploration policy based on a softmax rule with varying temperature. Although performances with respect to the shortest path algorithm, that was the state-of-the-art in the '90s, were much better, these algorithms suffered problems of scalability in large networks. This is because the authors proposed solutions based on Q-tables as at that time neural networks for reinforcement learning were not yet effective.

More recently, *d-AdaptOR* has been proposed [31] as a reinforcement learning framework for opportunistic routing. It guarantees low complexity and overhead and a distributed asynchronous implementation. This proposal uses Q-learning to minimize the expected transmissions per packet considering also a generic cost c_i for every node i in the network. The authors prove the convergence of their policy comparing the results of the trained framework with the ones of a genie-aided policy having full knowledge on the network. They also compare the performance of their opportunistic routing scheme with the ones of ExOR (Sec. 2.2.1) for random networks with 36 nodes. Results show how their method performs better in terms of number of transmissions per packet.

In literature we also find reinforcement learning to maximize path shortness and stability in MANETs [33]. In this work the author performs a comparison with Dynamic Source Routing (DSR) and ant-colony based routing (ACR), finding better results as for both the end-to-end delay and the packet delivery probability. Here the Q-table is also compared with a deep Q-network (DQN) solution, which, however still performs poorly.

Only very recently neural networks started to work properly, and they were particularly applied to the domain of Software Defined Networks (SDN). The first work found in the literature uses an off-policy actor-critic deep deterministic policy gradient method (DDPG) [34]. Here the authors makes use of the Traffic Matrix (TM,

being the bandwidth request between each source-destination pair) as input to guess the weights of the links between nodes. The goal of their framework is the minimization of the end-to-end delay. They use a scale-free network with 14 nodes to test their policy, comparing it with thousands of randomly generated different valid policies, showing that the learned policy always places itself in the first quartile of the random policies, varying the traffic intensity. Following basically the same settings and the same framework, *DROM* is proposed in [35]. The comparison performed is also the same, as similar are the results. In this paper the author also inspects the throughput maximization with respect to two theoretically optimal solutions (SDN-LB and QAR). When the traffic load exceeds the 30%, DROM performs slighlty better than the other algorithms if trained for 10^5 steps. Although these two solutions proved good results, they deal with SDNs, which have a centralized controller. This network element does not exist in WMNs and for applying reinforcement learning to this scenario, a more decentralized algorithm is needed.

In [36], authors propose a fully-distributed learning following the settings of the Q-routing cited above and with the goal of delay minimization also here. Each node is treated as an independent learner. They use a DQN with replay buffer to stabilize the training and ϵ-greedy exploration policy, sampling a training batch for each decision epoch. They compare two settings, one in which the input to the neural network is the same as in the original Q-routing, the other in which the information is expanded, including the action history and the future destinations of the next m packets. Results show that the neural network with non expanded input does not provide any improvement on tabular Q-routing, performing even worse. Providing additional information is instead the trick that makes the neural network effective, as it shows gains of about 30% in the average time delay with respect to classic Q-routing.

Chapter 3

Theory

3.1 Optimization objective

The primary goal of this thesis is to develop a suitable distributed algorithm for routing in mesh networks. As shown in literature, the minimization of the airtime cost is beneficial both in terms of energy saving and throughput, because this type of optimization objective promotes a low traffic congestion in the network. In this work only single-rate communications are considered and all the transmissions are regarded as equally expensive. The problem of airtime optimization can be thus defined as the minimization of the total number of transmissions performed in the network to deliver a message from source to destination, expressed as:

$$\min \sum_{i=1}^{N} TX_i \tag{3.1}$$

where N is the total number of nodes taking part in the forwarding from source to destination, and TX_i indicates the number of transmissions executed by node i.

The first phase of my thesis regards a comparison between MORE (Section 2.2.1) and the implemented learning algorithms for unicast communications. Therefore, the same optimization must be performed to compare the protocols on the same field. As MORE does not consider any constraint on other costs but the one that full information must be retrieved at the destination, in this part of my work no further objectives are added. However, for broadcast applications very often we face latency constraints, because they may be quasi-real time ones, such as video and audio streaming or controllers for robotics. In these cases, information goes rapidly out of scope and it is therefore valuable to make sure that all the nodes that need it have it in useful time. We can thus define a constraint on the end-to-end delay, considering a maximum delivery time τ that the network must satisfy for all the destinations. In mathematical terms, if we call d_i the delay associated to the message delivery to node i, we have:

$$d_i < \tau \qquad \forall\, i \in [N] \tag{3.2}$$

The constraint (3.2) can be further simplified pondering the fact that if the maximum delay satisfies it, then automatically all the others do. It becomes thus:

$$\max(d_i) < \tau \tag{3.3}$$

So, in the second part of this thesis, the minimization objective pursued is the one of equation (3.1) subject to the constraint (3.3). Performances comparison with MORE and BATMAN are also evaluated in terms of how much the protocols are able to satisfy the imposed time constraint.

3.2 Reinforcement Learning

Reinforcement Learning (RL) is a tool for learning how to map situations into actions maximizing a numerical reward. The learner, called agent, must discover from raw data which actions yields the highest reward, just by trying them.[1] The intuitive explanation of reinforcement learning is provided in Fig. 3.1. The agent, finding itself at time t in a certain state s_t interacts with the environment performing an action a_t and, as a result, will get a new state s_{t+1} and a reward r_{t+1}, which is the numerical value of the action taken previously. Formally, reinforcement learning is a way to solve Markov Decision Processes (MDPs), discrete time stochastic control processes defined by the tuple of elements $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where:

- $\mathcal{S}$ is a set of states;
- $\mathcal{A}$ is a set of actions;
- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is a state-transition probabilities function;
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is a reward function;
- $\gamma \in [0, 1]$ is a discount factor.

As we want to learn the process, we are supposed not to know the function p, which describes the dynamics of the process itself. In this case we talk about model free reinforcement learning and it is the one of interest in this thesis.

The ultimate goal of the agent is to maximize its expected accumulated future reward, defined by the discounted value function:

$$V(s_t) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \, r_{t+1}\right] \tag{3.4}$$

where γ specifies how much the agent cares about the future. For $\gamma < 1$, distant rewards in time are less important and this may be a condition that should be satisfied in order to guarantee the convergence of the series. In practice, however, these processes often have a finite horizon, so the summation does not go to infinity and

[1] Theory about reinforcement learning throughout this chapter is based on [37].

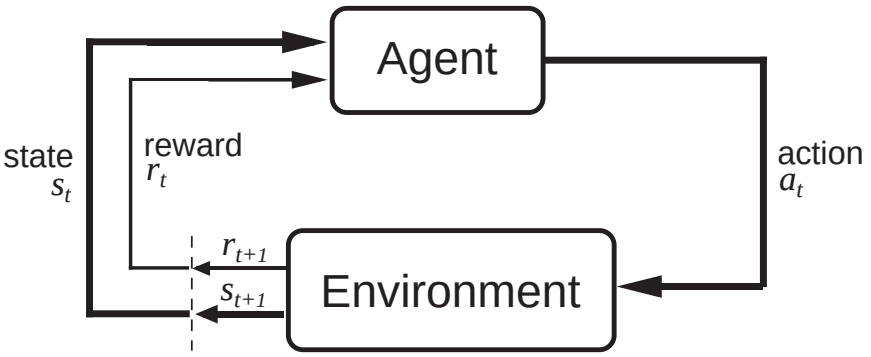


FIGURE 3.1: The agent–environment interaction in a Markov Decision Process.

what happens between an initial and a final state is called episode. The equation above can be expanded taking out the first term of the summation to derive the fundamental Bellman recursive equation:

$$V(s_t) = \mathbb{E}\left[r_t + \gamma V(s_{t+1})\right] = \mathbb{E}[r_t] + \gamma V(s_{t+1}) \tag{3.5}$$

The way the agent maximizes the function expressed in (3.4) is by learning a control policy function to choose which actions will result in greater reward basing on which is the current observed state and the previous experience. If deterministic, this policy is defined as:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a) \tag{3.6}$$

where Q is the action value function, stating which is the expected value of the action a when the observed state is s. However, often the policy is stochastic and its function becomes the distribution $\pi(a \mid s)$ defining the probabilities of taking each action given the state s. Intuitively, the meaning of the state value function $V^\pi(s)$ is the expected return of following policy π when the state is s, whereas $Q^\pi(a, s)$ gives the value of taking action a in state s. The relation between $Q(\cdot)$ and $V(\cdot)$ is:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, Q^\pi(a, s) \tag{3.7}$$

and it is valid also in case of deterministic policy (3.6), as we can see it as a degenerate distribution probability which outputs probability 1 for the best action and 0 otherwise. In other words, an indicator function selecting the action which yields the maximum value. In this case the relation is thus reduced to:

$$V^\pi(s) = \max_{a \in \mathcal{A}} Q^\pi(a, s) \tag{3.8}$$

Two common ways to learn the state value function are Monte Carlo (MC) methods and Temporal Difference (TD) learning. Both of them try to reach the actual value of $V(\cdot)$ with progressive updating of their estimate based on experience. Monte

Carlo methods wait to see the return G_t of the actions taken in the whole episode and use that value as a target, moving towards convergence with rate α and following the updating rule:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[G_t - V(s_t) \right] \tag{3.9}$$

The main drawback of this method is that it needs to wait the end of the episode, that is the moment when the agent reaches a final state, evaluating in a single shot the whole history of actions taken. TD learning, on the other side, exploits feedback after each action performed by the agent, as the reward r_{t+1} is immediately available after a_t is taken. The simplest update of TD is based on a rule derived from Eq. (3.5):

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{3.10}$$

As can be seen the target here is represented by $r_{t+1} + \gamma V(S_{t+1})$, which is a better estimate of $V(\cdot)$ than the one before. A step of size α is taken in the direction of this new estimate also in this case. Experiments show that, in general, the possibility of updating the estimate after each action induces a faster convergence to the actual value, which, however, is guaranteed for both the methods.

However, learning the state value function is not sufficient for the agent to find the optimal policy, as it is only learning which is the value of each state following a certain given policy π. As said at the beginning of this chapter, we want to learn how to map an observation coming from the environment into an action and in this way learn which actions lead to the highest rewards. Therefore, it is straightforward at this point that we need to learn the action value function $Q(s, a)$ and then follow the optimal policy defined in (3.6).

3.2.1 Tabular Reinforcement Learning

TD learning can be used to learn as well the action value function. The first approach to achieve this goal is to create a storage structure having the form of a matrix, called Q-table. It keeps on the entry $q_{i,j}$ the estimated value of taking action a_j while in state s_i. Naturally, the MDP modeled must be finite, namely $\mathcal{A}$ and $\mathcal{S}$ must have a finite cardinality in order to use the Q-table, because we do not have infinite storage capabilities. Two methods are commonly used: *SARSA* and *Q-learning*, belonging respectively to the on-policy and off-policy class. On-policy methods base the updating rule on the policy that is currently on learning and yielded past actions, whereas off-policy ones do not care about the current policy but rather select actions that will yield the highest value. As for the state value function learning, also the Q-value is updated basing on Bellman recursive equation:

$$Q(s_t, a_t) = \mathbb{E}[r_t] + \gamma \max_a Q(s_{t+1}, a) \tag{3.11}$$

State-Action-Reward-State-Action (SARSA) update is based therefore on the rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{3.12}$$

where $Q(s_{t+1}, a_{t+1})$ is computed with the learned policy π and defined to be 0 if s_{t+1} is a final state, i.e. the episode has ended. This rule employs all the elements of the tuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, giving rise to the name SARSA. Q-learning, instead, uses the rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{3.13}$$

and it is considered off-policy, because the learned policy is not used to compute the value of the next state-action pair but rather the maximum over all the possible actions in the observed new state is taken. An algorithm which is very similar to Q-learning is *Expected SARSA*, whose update rule is equal to the one expressed in (3.13), with the exception that $\max_a Q(s_{t+1}, a)$ is replaced with the expectation

$$\sum_a \pi(a \mid s_{t+1}) \, Q(s_{t+1}, a) \tag{3.14}$$

resulting often in faster convergence.

There is still a problem to be faced in these algorithms: if the action taken in a certain state is always selected with the rule defined in (3.6), the same action will always be chosen until its Q-value does not decrease below the one of another action. This is inefficient because it is highly dependent on initializations and also it would not adapt easily if the environment was non-stationary. A common solution to this issue is to choose $\epsilon \in [0, 1]$ and play a random action with probability ϵ, otherwise follow the greedy rule (3.6). This algorithm, called precisely ϵ-greedy, ensures that all actions are taken, because if we do not observe what happens when taking action a_j in state s_i, we will never know if it is a good action or not. It is thus important to find a good balance between action *exploration* and policy *exploitation*.

However, although tabular methods are proved to converge to the optimum and often in a reasonable time, they suffer from an important drawback. Actually, even though they are perfectly suitable when the number of possible states and actions is low, we should remember that, if there are M states and N actions, the number of Q-values to store is $O(N \cdot M)$. Intuitively, it is convenient to have the best possible description of the environment to decide which is the best action to perform. This means that it is desirable to feed the algorithm with as much information as possible, thus enlarging the dimension of the state space. Moreover, and most importantly, often relevant control tasks are in the continuous domain, so that actions are defined with real numbers. These considerations make Neural Networks attractive solutions for reinforcement learning as they are universal functions approximators.

3.2.2 Deep Reinforcement Learning

Deep Reinforcement Learning is the attempt to combine the possibilities of Neural Networks with the ones of reinforcement learning. The advantage, as said before, is that the chance of having a wider description of the state and also accounting for a higher number of actions, if allowed, comes at a minor cost. This is because it would be sufficient to change the number of neurons at the input and output layers, leaving the deeper layers unchanged. The first working algorithm after some promising attempts produced Deep Q-Network (*DQN*) in 2013 [38], a neural network able to play Atari games. The network is trained minimizing, at each iteration i, the loss

$$\mathcal{L}_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s_t, a_t \mid \theta_i))^2\right] \tag{3.15}$$

where the target is $y_i = \mathbb{E}[r_{t+1}] + \gamma \max_a Q(s_{t+1}, a \mid \theta_{i-1})$ and θ_i are the network weights at iteration i. In this version, a single Q-Network is used both to evaluate the current Q-value and the target. The fundamental novelties introduced with works on DQN are:

- The use of a replay buffer with a fixed capacity to store the most recent tuples $(s_t, a_t, r_{t+1}, s_{t+1})$. The training of the network is executed offline sampling n of these tuples, n being the minibatch size. This shrewdness serves to reduce sample correlation, since the training of a NN suffers if following samples are correlated and this usually happens in an MDP.
- The introduction of two Q-Networks, one for the policy and the other for the target. The target network is periodically updated with the policy network weights, every k iterations. This addition improves the velocity and the stability of training and solves the problem of optimistic estimation of the Q-values computed by the standard version of DQN [39].

With DQN an estimate of the value of taking action a in state s can be computed. However, still it is not possible to solve continuous control tasks. The only way to do so using this framework would be to discretize the continuous space. Yet, this could be inefficient in many cases. This is why Deep Deterministic Policy Gradient (*DDPG*) was proposed in 2016, dealing with the problem of continuous control [40]. In DDPG a double actor-critic approach is used, inspired from the version of double DQN. The *actor* is here a parameterized function $\mu(s \mid \theta^\mu)$ which deterministically maps states to specific actions and is used as policy. The *critic* $Q(s, a)$ is instead the estimation of the Q-value, learned with the standard Bellman equation as in DQN. The actor parameters are updated following the direction suggested by the critic:

$$\nabla_{\theta^\mu} J = \nabla_a Q(s, a \mid \theta^Q) \nabla_{\theta^\mu} \mu(s \mid \theta^\mu) \tag{3.16}$$

and approximated taking the average over a minibatch. There are moreover two versions of the actor and two of the critic, each couple is used to compute either

the target or the current policy. Unlike in DQN, where updates of the target are accomplished abruptly and occasionally, here a constant and low process with $\tau \ll 1$ is proposed every iteration:

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \tag{3.17}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{3.18}$$

The last relevant introduction of this work is the exploration policy: instead of using the classic ϵ-greedy method, suitable for discrete action spaces, an approach based on the addition of a certain noise process $\mathcal{N}$ is proposed:

$$\mu'(s_t) = \mu(s_t \mid \theta_t^{\mu}) + \mathcal{N} \tag{3.19}$$

The most recent advance in DRL, is represented probably by the Soft Actor-Critic (*SAC*) algorithm, dated 2018, which modifies the learning objective of the agent [41], [42]. Actually, while before it was said that the goal of the agent was to maximize its expected reward, given by the discounted value function in (3.4), now a term accounting for the entropy is added, obtaining the policy:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_t \gamma^t \, \mathbb{E}\left[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot \mid s_t))\right] \tag{3.20}$$

where α is a temperature term regulating the importance of the entropy in the maximization task, controlling thus the stochasticity of the optimal policy. The advantage of introducing the entropy term in the maximization objective is on the exploration behavior: the policy is incentivated to explore more widely, while discarding unpromising avenues. Moreover, in this way multiple near-optimal modes can be captured, and to this a similar probability is assigned. The process of adjusting the temperature parameter is also performed during learning, providing an automated way to tune the exploration policy. Results on typical RL tasks show that learning benefits from this novelty, as other state-of-the-art frameworks are outperformed.

Although single agent DRL is starting to work with quite satisfying results, the same thing cannot be said about the Multi-Agent case (MADRL). The first investigated algorithm was *MADQN*, in a scenario where the two involved agents played the popular game Pong [43]. Here, cooperative and non-cooperative tasks are explored but the dynamics of the environment are extremely easy, with just two agents, four actions available per agent and an extremely simple reward function. An approach become more popular is the one proposed with *MADDPG* [44]. As the standard multi agent version of DDPG performed poorly, here the critic network is augmented with the information coming from other agents' actors. Therefore, the critic represents a centralized action-value function, making the algorithm not suitable for scenarios where it is not possible to have an omniscient critic. Moreover, these algorithms in general suffer when the number of involved agents increase. This happens

because each agent is interacting with the environment while other agents may not know it, thus adding noise to their observations. Because in mesh networks there is no network controller and for the above mentioned reasons, these algorithms would work poorly for the task of this thesis. It is therefore necessary to find an alternative solution, able to reach a fair result in a completely distributed way and at the same time efficiently handle many agents, as WMNs can easily have many nodes with routing functions.

3.2.3 Multi-Armed Bandits

A simpler form of reinforcement learning is represented by the Multi-Armed Bandits (MABs), so called in analogy with the popular slot machine. Each arm, namely action, available to the bandit is like a lever of the slot machine and the goal of the bandit is to learn which arm will yield the highest expected reward. In the most basic version, the bandit finds itself always in the same state and has to choose which of the k available arms it should play. The episode lasts one single step, meaning that in this case the target is just the reward:

$$q^*(a) = \mathbb{E}[r_t \mid a_t = a] \tag{3.21}$$

This expression is a particular case of the Bellman recursive equation (3.11), where $Q(s_{t+1}, \cdot) = 0$ because there is no next state.

Contextual Multi-Armed Bandits (CMABs) are a more interesting tool, being the extension of MABs when there are many possible states but the horizon is still limited to one step. In this case there is a Q-value for each state-action pair, exactly like explained in Section 3.2.1. The tabular solution of the contextual bandit problem relies therefore on the update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r - Q(s, a)] \tag{3.22}$$

where α can be selected as $\frac{1}{n}$, n being the number of times that a was selected while in state s, thus performing an online average. Often, however, tasks are non-stationary and we may want to assign higher value to more recent experience, thus choosing a fixed rate α.

3.3 Algorithmic details

CMABs are an attractive solution for the unicast routing problem, which constitutes the first task of this thesis. This is because the network finds itself in a certain state, or context, described by the source ID, the destination one and possibly other information about the topology. It is reasonable to think that the network topology, if non-stationary, changes with a negligible speed if compared to the time needed for a single end-to-end communication. Moreover, once a full communication is ended,

it is also reasonable to consider the following one as uncorrelated. In this way, an episode starts with the beginning of the transmissions from the source and ends with the acknowledgment message of the destination. Nodes participating to the forwarding of the information represent agents involved in the interaction with the environment and are allowed to choose a single action per batch transmission. In analogy with MORE (Section 2.2.1), routers can choose their own *TX_credit*, i.e. the number of packets that they forward for each innovative packet received from the upstream. In this thesis, a CMAB version of Q-learning is implemented as tabular solution and a Deep CMAB version of DQN as a neural networks based solution. Naturally, as said before, this comes at the cost of discretizing the continuous space and it would be interesting to investigate a solution that can handle the continuous control of the forwarding rate as future work. The same framework is then applied to broadcast communications, considering that nodes can choose a forwarding rate at the beginning of each batch transmission and must keep it unvaried until the last acknowledgment is broadcasted. Some simple local rules for flow control are added to improve the behavior of the network in the optimization objective sense.

3.3.1 Exploration vs exploitation

The most basic exploration-exploitation policy for MABs is the already explained ϵ-greedy. However, it often performs poorly and, in this thesis, a different approach called *Upper Confidence Bound* (UCB) is preferred. Unlike ϵ-greedy, UCB does not explore actions completely randomly but rather tries actions that are close to be optimal or particularly uncertain, almost ignoring the ones that are clearly not worthy. A way to look for actions that have a higher potential is by following the policy defined by the equation

$$\pi(s_t) = \underset{a}{\operatorname{argmax}} \left[Q(s_t, a) + c \sqrt{\frac{\ln t}{N_t(s_t, a)}} \right] = a_t \tag{3.23}$$

Here $c > 0$ controls the trade-off between exploration and exploitation. The square-root term is a measure of the uncertainty or variance of a's value. Since $\ln t$ is at the numerator, the uncertainty grows less than linearly as time goes by. It is also true that the logarithm makes the exploration policy try all the actions, because it is unbounded, but together less frequently when t increases. At the denominator, $N_t(s_t, a)$ indicates the number of times that a was taken while in state s_t. Therefore, when an action is performed frequently, the uncertainty keeps low, if instead that action is not taken for a certain amount of time, the uncertainty grows.

This method perfectly applies to the tabular solution when the state space is not too large. The training algorithm is schematically presented in Alg. 1. At first, Q-values are initialized to zero for every agent and state-action pair, then the simulation of the environment begins. If the communication is in unicast mode, the context of the bandit contains the couple (source, destination), whereas in the broadcast case

Algorithm 1: Multi Agent Contextual MAB (MACMAB)

Input : set of states $\mathcal{S}$; set of actions $\mathcal{A}$; reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$; communication environment $\mathcal{E}$; transmission mode, either U (unicast) or B (broadcast); learning rate α.

```
begin
    foreach agent i do
        foreach (s, a) do Q_i(s, a) = 0;
    end
    foreach communication do
        src ← source ID;
        dst ← destination(s) ID(s);
        N ← set of the routers involved in this communication;
        if mode = U then context ← (src, dst);
        else context ← src;
        foreach batch do
            foreach agent i ∈ N do
                s_i ← context;
                sample a_i ← UCB(s_i) transmission credit;
            end
            perform the batch transmission in E;
            foreach agent i ∈ N do
                R_i ← r(s_i, a_i);
                Q_i(s_i, a_i) ← Q_i(s_i, a_i) + α [R_i − Q_i(s_i, a_i)];
            end
        end
    end
end
```

only the source ID is required. This means that in the first case we have to store $O(KN^2)$ values, where K is the number of arms of the bandit and N the number of nodes in the network, since there are $N(N-1)$ possible source-destination pairs. In the second case the storage space required is much less as it is just $O(KN)$. Each communication from source to destination(s) has a certain number of subtransmissions, named batches according to MORE (Section 2.2.1). For each of these batch transmissions every router samples a *TX_credit* from its action space given the observed state and following the UCB exploration-exploitation policy (3.23). The batch transmission is then simulated and, as a final result, each node gets a reward (Section 3.3.2) coming from the final ACK of the destination and can update its Q-value with equation (3.22).

However, when using the neural network, it would be desirable to expand the available information and the UCB policy is not a suitable method when the number of possible states is too large. Exploration policies like ϵ-greedy and UCB are called *distribution-free*, because they do not need any prior about the probability distribution over the action values. Methods that require such an initial information, are called instead *Bayesian*, and they update the known distribution at each iteration.

Hence, a possible solution is to select actions basing on their posterior probability of being the best one and this algorithm is known as *Thompson sampling*.

A neural network can be seen as a function approximator parameterized by its weights θ. If the network outputs the required distribution probability discussed before, the initial prior is simply given by the random weights initialization and the network is progressively trained to learn the actual distribution. Therefore, Thompson sampling can be used with profit when dealing with neural networks. However, problems arise when considering that the output of a NN is deterministic and specified by the trainable parameters θ. Thus, the exploration behavior would not be sufficient and we need to add uncertainty on estimates. Luckily, there exists an easy way to make the output stochastic and it is to use dropout: it is proved, actually, that by using the standard Bernoulli dropout we can build a Bayesian Neural Network (BNN) [47].

Dropout is defined by the Bernoulli random variable with parameter p in the following way. When performing a forward pass, for every neuron unit in each layer but the last one, a Bernoulli binary variable with parameter p is sampled. If the resulting value is one, the neuron is kept, otherwise it is dropped, i.e. turned off. The fact that the activity of neurons is regulated by a random variable generates uncertainty, giving birth to the desired BNN and to an effective way of performing Thompson sampling, since now the network's output is stochastic.

To these properties, it would be desirable to add another feature: the ability to automatically tune the trade-off between exploration and exploitation, because, as the model earns experience, its initial uncertainty should collapse. With *Concrete Dropout* [48], the parameter p becomes a trainable one as network's weights are and it is also shown that this method gives better performances, calibrated uncertainties and faster convergence. Concrete dropout can be seen as a continuous relaxation of Bernoulli dropout, described by a distribution $q_\theta(\boldsymbol{\omega})$, where $\boldsymbol{\omega} = \{W\}_{l=1}^{L}$ is the set of random weight matrices with L layers and θ are the variational parameters. The dropout probability is optimized using a gradient method following the objective

$$\hat{\mathcal{L}}(\theta) = -\frac{1}{M}\sum_{i\in\mathcal{S}} \log\left(p(\mathbf{y}_i) \mid \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)\right) + \frac{1}{N}\mathbf{KL}(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})) \tag{3.24}$$

where θ are the parameters to be optimized, N is the number of observed data points, $\mathcal{S}$ a random set of M data points, $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)$ the BNN's output on input $\mathbf{x}_i$ and $p(\mathbf{y}_i) \mid \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)$ is the model's likelihood. The Kullback-Leibler divergence is a term accounting for entropy regularization and ensuring that the approximate posterior $q_\theta(\boldsymbol{\omega})$ does not deviate too far from the prior distribution $p(\boldsymbol{\omega})$. The KL divergence depends linearly, with negative slope, on the entropy of the Bernoulli random variable defined by

$$\mathcal{H}(p) := -p\log p - (1-p)\log(1-p) \tag{3.25}$$

So, this term is regularizing dropout, since it depends on the probability p alone.

Minimizing the KL divergence is thus equivalent to maximizing the entropy $\mathcal{H}(p)$, which means that p is pushed towards 0.5 when N is small, decreasing towards 0 as N increases, as can be deduced from equation (3.24).

The last step of this method is the moving from a discrete random variable to a continuous one, now that we are able to tune the parameter p. This is done with the concrete distribution relaxation $\bar{z}$ of the Bernoulli random variable:

$$\bar{z} = sigmoid\left(\frac{1}{t}\left(\log p - \log(1-p) + \log u - \log(1-u)\right)\right) \tag{3.26}$$

with $u \sim \mathcal{U}(0,1)$ and t being the temperature parameter of the sigmoid. This version is differentiable with respect to p and therefore can be optimized accordingly to the objective (3.24).

The training procedure for the neural network based bandits is resumed in Alg. 2. At the beginning, all the BNNs are randomly initialized and empty memory buffers are created for the storage of replay data. As in the case of the tabular solution, for each batch transmission inside a whole message communication, a *TX_credit* is chosen for every router involved in the forwarding operation, this time thanks to the neural networks. The state in this case is increased with the available transmission credits of the neighbors and with the known estimates of the delivery probabilities to nodes with which a link is active. The environment is then simulated to get a new reward observation together with the batch ACK but, now, differently from before, the Q function update is not performed immediately but rather the information about the state-action pair and the reward obtained is pushed into the memory buffer. Since training a neural network is an expensive operation, this is done at the end of the batch transmission, averaging the gradient of m random samples taken from the memory. Moreover, the rate of training decreases exponentially, as we need to train less often when the experience is already sufficient. Because the rate decreases, the amount of data observed in the meanwhile is larger, of size N, therefore $\lfloor N/m \rfloor$ epochs of m samples are required. In this way, also the dropout parameter is progressively brought to zero accordingly to the optimization objective (3.24).

3.3.2 Reward shaping

There are two common ways to model multi agent systems: *joint* and *independent learners*, depending on the information available at each agent. In the case of joint learners, the space $\mathcal{A}$ of actions is the joint space $\{a_i, a_{-i}\}$, where a_i denotes the action performed by player i and a_{-i} is a compact way to denote other players' actions a_j, $j \neq i$. This is a powerful method in terms of performances but it is quite infeasible because the cardinality of the joint space grows exponentially with the number of agents. Furthermore, it would require every agent to have full knowledge on what other players are doing and also that agents act synchronously. These conditions are not met in mesh networks. Actually, as it has already been remarked, WMNs

Algorithm 2: Multi Agent Deep Contextual MAB (MADCMAB)

Input : set of states $\mathcal{S}$; set of actions $\mathcal{A}$; reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$; communication environment $\mathcal{E}$; minibatch size m; number of initial steps before training P; exponential retraining coefficient K; maximum retraining interval M.

begin
- **foreach** *agent* i **do**
  - **foreach** (s, a) **do**
    - randomly initialize BNN_i;
    - initialize empty replay memory buffer mem_i;
  - **end**
- **end**
- $k \leftarrow 1$;
- $L \leftarrow 1$;
- $N \leftarrow m$;
- **foreach** *communication* **do**
  - src $\leftarrow$ source ID;
  - dst $\leftarrow$ destination(s) ID(s);
  - $\mathcal{N} \leftarrow$ set of the routers involved in this communication;
  - **foreach** *batch* **do**
    - **foreach** *agent* $i \in \mathcal{N}$ **do**
      - prob $\leftarrow$ delivery probabilities to i's neighbors;
      - act $\leftarrow$ available transmission credits of i's neighbors;
      - $s_i \leftarrow$ (src,dst,prob,act);
      - sample $a_i \leftarrow BNN_i(s_i)$
    - **end**
    - perform the batch transmission in $\mathcal{E}$;
    - $d \leftarrow$ maximum ACK delay;
    - **foreach** *agent* $i \in \mathcal{N}$ **do**
      - $R_i \leftarrow r(s_i, a_i)$;
      - push in mem_i tuple (s_i, a_i, R_i)
    - **end**
  - **end**
  - **if** $k = P$ **then**
    - $L \leftarrow L * K$;
    - dist $\leftarrow \min(\lfloor L \rceil, M)$;
    - $P \leftarrow P +$ dist;
    - $N \leftarrow \min(\text{dist} \cdot m, M)$;
    - **foreach** *agent* i **do**
      - **for** $l = 1$ **to** $\lfloor N/m \rfloor$ **do**
        - sample m random points from mem_i;
        - train BNN_i with that minibatch;
      - **end**
    - **end**
  - **end**
  - $k \leftarrow k + 1$
- **end**

end

need algorithms that use only local and not global information. On the other hand, independent learners work in the space of their own action and nothing more, which is surely an easier solution but often leading to poorer performances.

It is convenient now to introduce the concept of Pareto dominance of a joint strategy, namely an action in the joint actions space. An action a_i is said to be Pareto dominated by a'_i if:

$$\forall\, i \quad q_i^t(s_i, a'_i) \geq q_i^t(s_i, a_i) \tag{3.27}$$

$$\exists\, i \mid \; q_i^t(s_i, a'_i) > q_i^t(s_i, a_i) \tag{3.28}$$

where $q_i^t(s_i, a_i)$ indicates the value of action a_i while in state s_i at time t for player i. If an action a_i is not dominated by any other action a'_i, it is named Pareto efficient or optimal and it belongs to the so called Pareto frontier, the set of these type of actions.

Recall that the network's objective is the minimization of the overall airtime cost, defined for the sake of simplicity as just the number of transmissions (3.1). Pareto optimal solution are efficient in a global welfare sense, since they rely on the joint action space, therefore an ideal solution to the optimization problem of this thesis should belong to the Pareto frontier. However, as said before, working in the joint space is not a feasible solution and, while in this case agents are pushed towards a Pareto optimal solution, in the independent learners scenario they naturally tend to a Nash Equilibrium (NE), defined by:

$$a_i* = \underset{a}{\operatorname{argmax}}\, q_i(s_i, a_i, a_{-i}) \tag{3.29}$$

where it is expressed also the dependency of the action value on other players' actions. In this way, agents want to maximize their own profit and act in a selfish way, ending up generally in an equilibrium point which guarantees a worse global welfare with respect to a Pareto optimal solution. Which equilibrium point is dependent on the initial conditions and on the dynamics of the specific environment simulation [49].

A way to improve agents' coordination and cooperation and to promote the achievement of a better Nash Equilibrium even if the context is the one of independent learners, is to exploit communications among agents. In the case of this thesis, some useful details at least at local level can be gained by routers just by overhearing neighbors' transmissions. The required information can be thus piggybacked in the header of data packets, being careful not to increase relevantly traffic overhead with this trick. In this way, nodes can be aware at least of what other routers are doing, if not at a global scale, at least at a local one. Another practical way for increasing the probability of reaching a good NE, is to properly project the reward. The design of a reward function is indeed a delicate matter, because sometimes it may happen that the agents learn unexpected behaviors and, moreover, it also affects the rate of convergence. The first solution that may come to mind, is that each router could

track the number of transmissions that it fulfilled and then try to minimize it with its replay experience. This, however, performs poorly, because agents tend to find an egoistic solution rather than pushing towards the one desired. Luckily, we can exploit again the wireless medium and, if we assume that the final ACK is broadcasted from the destination to all the routers involved, the total number of transmission executed for that batch can be easily retrieved. In this way, each node will have the same reward, depending exactly on the global optimization objective (3.1), even if most of the topology information is unknown.

Since the reward, in the traditional form of reinforcement learning, must be maximized, it can be expressed as:

$$r_i(s_i, a_i) = \begin{cases} -\sum_j TX_j(s_j, a_j) & \text{if } \max_j(d_j) < \tau \\ -M & \text{otherwise} \end{cases} \tag{3.30}$$

where the penalty M must be chosen large enough so that agents learn that they should avoid it. In the unicast scenario, where for fair comparison with MORE a maximum delay is not considered, it is sufficient to set $\tau = \infty$. This very easy reward function is directly applied for the tabular case. For the neural network, instead, it turned out to be more convenient to map the output in the range $[0, 1]$, so that the cross-entropy loss could be used instead of the MSE or MAE criteria, since it is empirically proved that it guarantees more training stability. The procedure chosen is a bit convoluted because, from the moment that nodes do not have full knowledge about the topology, it is not possible to compute theoretical bounds on the number of transmissions required for delivering the message. Therefore, an alternative solution must be found if a maximum and a minimum are not available. Two slightly different methods are tested, one considering respectively as minimum and maximum reward the maximum and minimum number of transmissions required for that state until the current moment. The other one considers only the values that are currently in the replay buffer: in principle this should also help to track a possible non-stationarity of the environment but it may cause excessive exploration behavior. Once that this linear mapping is performed, the following non-linear function $f(x)$ is applied to the input in $[0, 1]$ for better reward shaping:

$$f(x) = \begin{cases} w(1 - x^\gamma) + (1 - w) & \text{if } \max_j(d_j) < \tau \\ 0 & \text{otherwise} \end{cases} \tag{3.31}$$

where γ defines the decay rate of the reward and can be chosen smaller than 1 so that the reward tends to the close to optimal solutions with a higher than linear rate. The weight term $w \in [0, 1]$ is added to account for the discontinuity generating the penalty for not satisfying the latency constraint. In the case of unicast transmission, w can be set to 1 so that there is no penalty. In Fig. 3.2 an example of this function can be seen when the constraint (3.3) is satisfied.

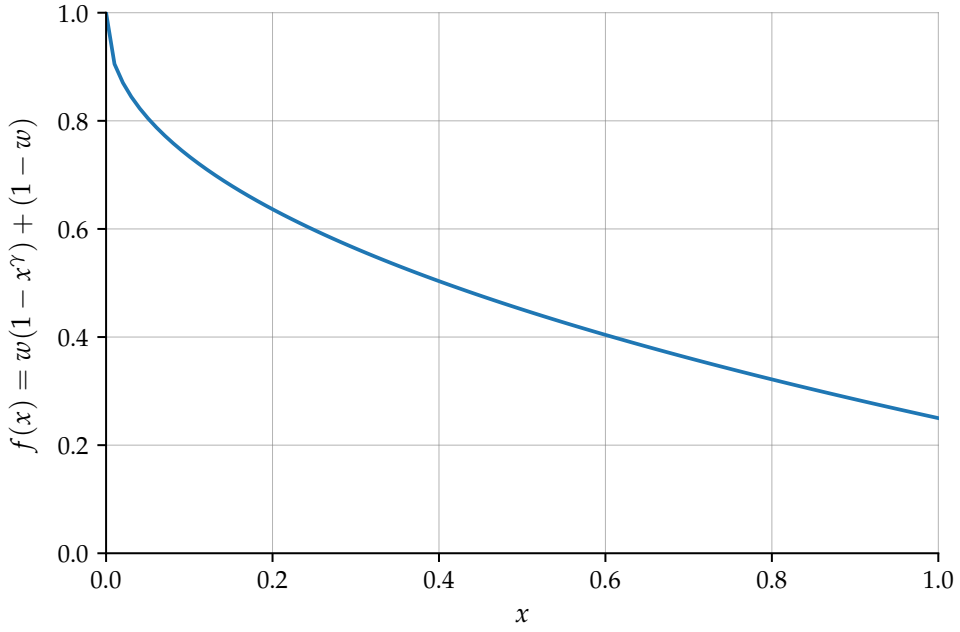


FIGURE 3.2: Mapping function for the reward in the BNN settings. Parameters are chosen as $\gamma = 0.45$ and $w = 0.75$.

Chapter 4

Simulation

The simulation environment is entirely created from scratch, without the help of any existent network simulator. The programming language employed is *Python 3*, for better integration with existing Machine Learning libraries.

4.1 The mesh model

Mesh networks are modeled as graphs $G = (V, E)$, where V is the set of vertices and E the one of edges and created as Random Geometric Graphs (RGGs) with the package *NetworkX*[1]. The size of the mesh is dependent on the experiment performed. It is kept fixed to 20 nodes for the unicast simualtions whereas it is progressively increased for broadcasting. Real numbers in $[0.1, 0.99]$ are assigned as weights to the edges and determined with a law which is inversely proportional to the square of the distance between nodes. This choice is made to simulate in some way the decrease in the power of the received signal accordingly to the path loss formula, assuming that antenna gains and transmitting power are unitary and the wavelength is fixed. Weights on edges represent therefore delivery probabilities on links.

4.2 Transmission rules

The algorithm used for simulating communications is the one explained in Section 2.2.1 for MORE: nodes having routing functions for the current batch transmission increase their credit counter by their own transmission credit when receiving an innovative packet from the upstream and they decrease it by a unit when transmitting a recoded packet. The coding and recoding operations are performed using the NC library *Kodo*[2], that provides a Python-C++ interface for easily handling these operations on bytearrays of data. The Galois field over which the coding is performed is $\mathbb{F}_8$, with varying generation size, depending on the specific experiment executed, and a symbol size of 8 bytes. This approach is kept both for the unicast and the broadcast case, because it is shown that network coding helps to improve the spatial reuse and at the same time keeping new information to be sent in a second moment

[1] `https://networkx.github.io/`

[2] `http://docs.steinwurf.com/kodo.html`

can only cause an increase in the end-to-end delay experienced. To make communications more realistic, the capture effect is considered. Nodes that are too close to each other cannot transmit during the same time frame, otherwise interference would be too high and receiving nodes could not distinguish different data streams. Therefore, a Medium Access Control (MAC) functionality is simulated so that if a certain node is transmitting at time t, other nodes that find themselves at $d < 0.5$ distance units are prevented from transmitting in that frame. Besides the two learning algorithms described in Section 3.3, other protocols are implemented for comparison. Specifically, they are MORE in both unicast and multicast flavors and BATMAN (Section 2.2.2) only for broadcasting. The broadcast version of *batman-adv* does not provide directly an algorithm that employs network coding, yet it is said that it can be used improving performances. So, the same approach as the other protocols is applied for better comparison, considering that the transmission credit for every node in BATMAN is fixed to 3. Moreover, further simple local criteria are added in all the protocols to prevent certainly suboptimal transmissions and they are the ones used also in MORE and BATMAN that are resumed here:

- When a batch ACK is received, nodes stop transmitting data belonging to that batch;
- If the packet is received at j from node i and i is the only active neighbor, j does not transmit;
- If the originator is the only active neighbor of node j, it does not transmit;
- Nodes stop transmitting when all the neighbors have received the message in the broadcast scenario.

In a recent thesis work, it is shown how shifting the source is beneficial for reducing the number of transmissions [50]. However, this improvement is not considered in this thesis for the unicast scenario, where it is implemented a standard and omniscient version of MORE which knows exactly the delivery probabilities without the need for pinging the channel every 10s to estimate its quality. The source shifting behavior is automatic, instead, in broadcast, because of the previous fourth rule. It is important to notice that nodes do not transmit always until all the neighbors can retrieve the full message. They may end their credit in advance and switch to sleep mode to save energy and transmissions if the network's topology allows for it.

The action space $\mathcal{A}$ for the bandits is composed by a discrete grid of linearly spaced values belonging to the range $[l, h]$, where $l = 0.05$ in the unicast case and $l = 0$ for broadcasting. This is desired because in the former case the set of optimal forwarders is precomputed using the EAX metric (2.5), with a threshold of $\phi = 0.2$ and preventing a forwarder from transmitting may cause combinations that break the constraint requiring that full information is retrieved at the destination. Instead, in the broadcast scenario, the task of the bandits is also to find which nodes can avoid transmissions altogether, thus saving energy and airtime cost. The maximum

transmission credit that node i can choose is instead set to be $h = \max_j(1/p_{i,j})$, which is the average number of transmissions required to deliver a packet to the neighbor j with the worst link quality with i.

The timeout threshold is dependent on the size of the mesh as, if we want to broadcast a message, obviously, the larger is the network the more time will be required to successfully complete the transmissions. It is interesting to note that setting a timeout threshold forces the algorithm to learn also which solutions are feasible. This is because if transmissions end before all the destinations can correctly retrieve the message, the timeout will be reached giving as reward the penalty term.

4.3 Neural networks design

The Neural Networks are built with the package *PyTorch*[3], which allows for a wide range of opportunities to personalize the models and the class implementing Concrete Dropout is directly taken from the licensed repository of the author Y. Gal[4]. The architecture of the network is a fully-connected one with 5 layers, input and output included. In the first inner layer there are 4 sets of 32 perceptron units, each set is specializing for the specific input, which can be the source ID, the destination ID, the neighbors' delivery probabilities and the neighbors' chosen transmission credits. Inputs are encoded with the one-hot encoding technique, so neurons in the input layer are $4n$, n being the total number of nodes in the mesh. The other two inner layers are composed by 128 units and the output has finally 50 neurons since the number of possible actions is indeed 50. This means that the total number of trainable parameters, excluding the dropout probability and considering biases, is $132n + 39552$, depending linearly on the size of the mesh network. With networks of 20 nodes, which are used for the unicast problem, they are 42192, which is a very low number if compared with the state-of-the-art solutions in deep learning. The network is represented in Fig. 4.1, where each input represents the encoding in n neurons of the 4 pieces of information that form the context of the bandit.

For the training procedure, the *Adam* optimizer is used, with learning rate 0.01 and weight decay is not applied because regularization is not desired in reinforcement learning, as the overfitting problem does not exist. The replay buffers have a capacity of 8000 samples and the batch size of the transmissions is kept fixed to 32, even though generally it can be variable. The maximum capacity of the memories affects the maximum retraining interval, that is set to $\lfloor 8000/m \rfloor = 125$ full communications, as the minibatch size is $m = 64$. This choice is made not to waste too much experience, even though the minibatches are randomly sampled from the buffers. An alternative solution when reaching the maximum retraining interval would be to train on all the available points shuffling them to cope with the correlation problem. As explained in Section 3.3, the rate of retraining is exponentially

[3] `https://pytorch.org/`

[4] `https://github.com/yaringal/ConcreteDropout`

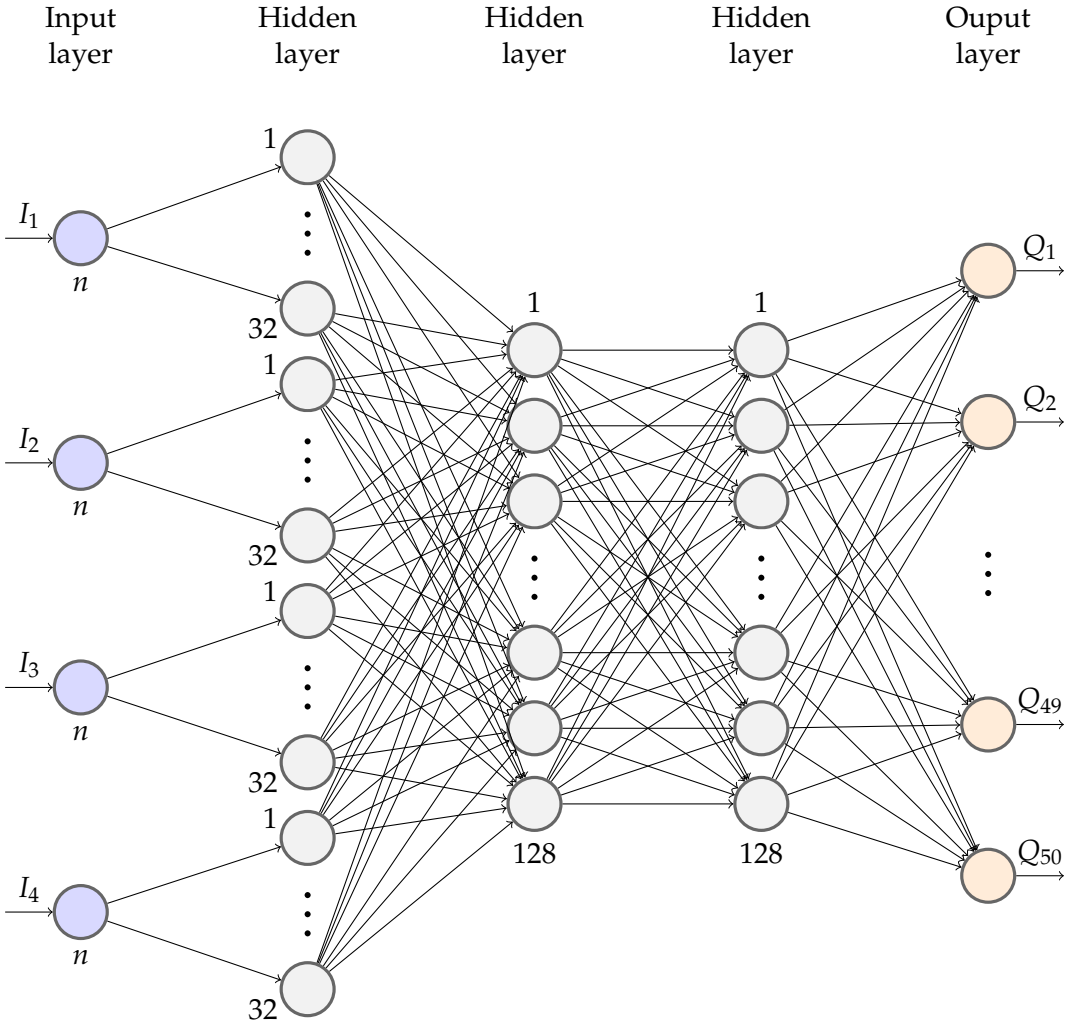


FIGURE 4.1: Architecture of the Neural Network employed.

increasing until the maximum interval is reached. The retraining distance is multiplied by a coefficient $K = 1.05$ every time the optimization is performed and such distance is then rounded, replacing the old one. Choosing such a K and rounding the result means that at the beginning there are 9 retrainings performed after each full communication with points from m random batch transmissions. Therefore, just one optimization step (3.24) is performed with $M = m$ and $N = m$, with an initially high entropy on the dropout parameter p. As time goes by, the distance d progressively increases and there will be d optimization steps again with $M = m$ but with $N = d \cdot m$, relaxing the amount of uncertainty since we have observed more points. Finally, the weight and the exponent on the reward mapping function for the BNNs are set respectively to $w = 0.5$ and $\gamma = 0.45$.

4.4 Tabular bandit parameters

For what concerns the tabular bandit, the exploration-exploitation coefficient is set to $c = 2$ for the unicast case and $c = 10$ for broadcasting. There is no particular reason

for such a difference but the fact that results where empirically better. Actually, in the machine learning field, some parameters must be tuned with a grid search and there is no magic rule to select them. The learning rate α of equation (3.22) is chosen as $1/N_t$, hence the bandits compute the real arithmetic average of the Q-value. In a non-stationary environment, as already discussed, it would be more efficient to choose a fixed learning rate, e.g. $\alpha = 0.1$.

The penalty term M of equation (3.30) is dependent on the diameter of the mesh and it increases accordingly. Also here, there is no special rule to select it, but the one that it must be sufficiently big so that the bandits learn to avoid receiving $-M$ as reward.

Chapter 5

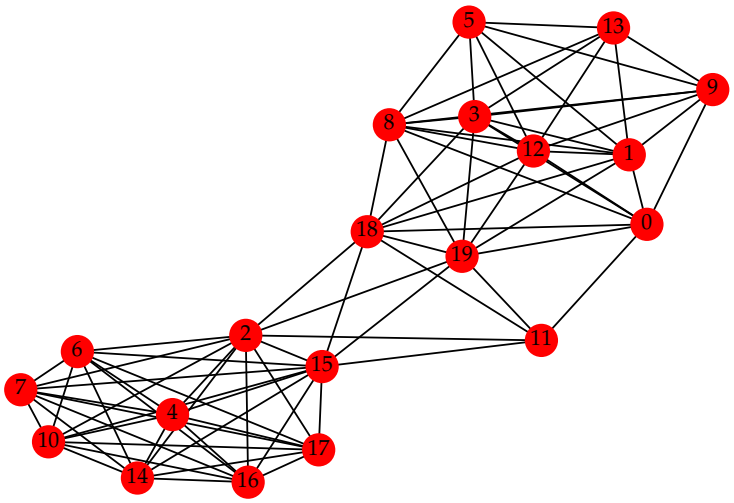
Results and Analysis

In this chapter, results will be shown with the help of plots and tables and further discussed. The first evaluation metric used is obviously the airtime cost, measured as the total number of transmissions carried out by the mesh, as it is the objective of the optimization. However, a special consideration is also given to latency, measured in terms of timesteps required to deliver the message. Latency is certainly dependent on the size and specific topology of the network, but it can be used for comparison between different protocols applied on the same mesh. Moreover, as stated in Section 3.1, latency is also part of the minimization problem as a constraint for broadcasting.

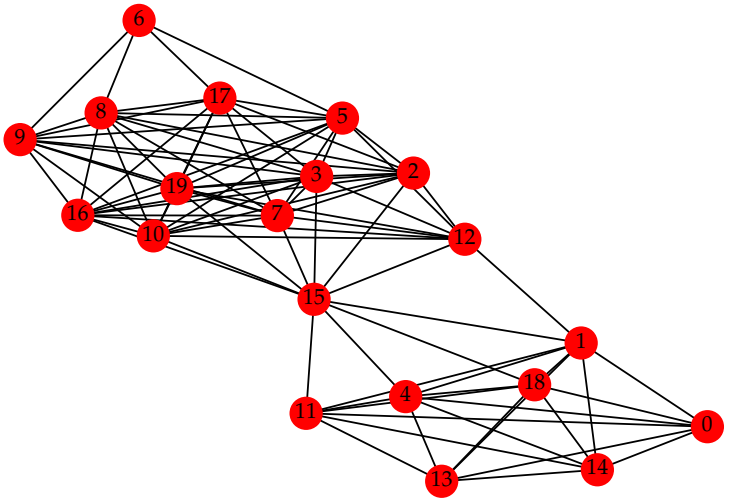
5.1 Unicast results

The three mesh networks employed for the simulations in the unicast case are depicted in Fig. 5.1 and they are generated as explained in Section 4.1. These random geometric graphs have 20 nodes each and, as can be seen, are highly connected. However, only the links between very close nodes have a high quality in terms of delivery probability. The majority of the links, actually, belong to the so called *gray area*, a wide zone where receptions happen more rarely but can be well exploited in opportunistic routing with network coding. Twelve random pairs source-destination are chosen for each graph, with distance ranging uniformly from 1-hop to the maximum number of hops of the meshes. The simulation is then performed on the subgraph defined by the set of forwarders delivering the message from source to destination. This means that a total number of 36 different topologies are tried for every generation size $G \in \{32, 64, 128, 160\}$.

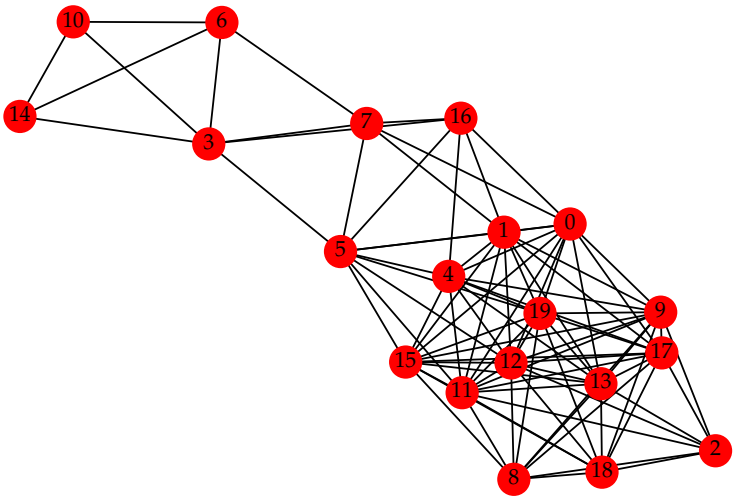
The first 1500 transmissions are considered as training period, then, the exploration behavior of the tabular bandit is switched off, while the neural network tunes it on its own. The last 500 are hence considered for evaluation. The following figures and discussion refer to graph 1 and the few differences with other meshes will be commented afterwards. In Fig. 5.2 the result relative to the airtime cost is shown, in logarithmic scale. In the background the actual value of the airtime cost of each batch transmission is shown, whereas a thicker line obtained with a moving average



(A) Graph 1.



(B) Graph 2.



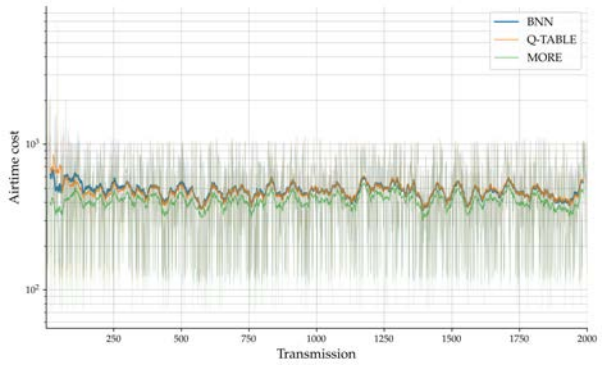
(C) Graph 3.

FIGURE 5.1: Random geometric graphs employed for the unicast simulations.

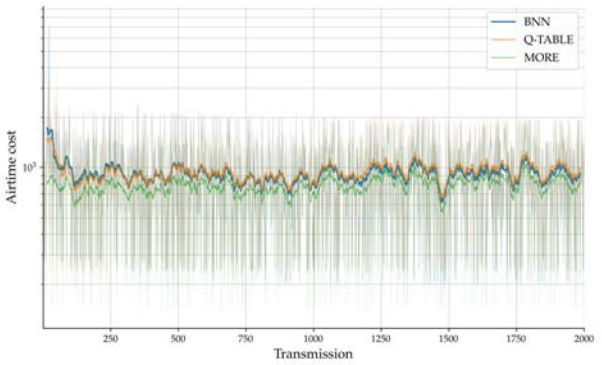
filter of length 32 samples (one full transmission) is plotted on top to better appreciate the behavior of the algorithms. As can be seen, the learning response of the two algorithms is consistent even though the generation size increases. Both the algorithms requires about 100 transmissions at the beginning to find the way towards a good optimum. After that, only some refinement is needed, although some kind of oscillations can occur from time to time. This may happen because the task is multi agent and the action exploration of only one node may perturb significantly other routers. It is evident the case of Fig. 5.2c around transmission 700, where the neural network based algorithm worsens considerably its performance, except then recovering a good one. In the evaluation range, it can be seen how the BNN based bandits perform slightly better than the tabular algorithm, confirming that increasing the information available as input is beneficial. However, both the methods perform around $12\% - 30\%$ worse than MORE on average. This is an expected result, as we are approximating the set of real numbers with which MORE computes the optimal transmission credits with a discrete grid of 50 evenly spaced possible values. Of course, even though optimality could be reached in this action space, the possibility of enlarging it by learning real numbers in a continuous set would certainly decrease this distance. A possible solution would be the use of infinitely many multi-armed bandits and a proper formulation should be found. In Fig. 5.4 boxplots of the airtime cost in the evaluation period are illustrated, in linear scale. Here we can see how the variance of the solutions learned by the two automated algorithms is generally slightly larger than the one belonging to MORE. The general shape of random variables describing the airtime is however consistent among protocols, as transmissions are performed on the same source-destination pairs. The tabular algorithm is moreover on average very close to the neural networks based one only for generation size 32 (Fig. 5.4a), then the gap increases for $G = 64$ and, afterwards, it keeps almost constant if we do not consider the obvious little variabilility due to the specific learning history of the simulation.

Besides the airtime performance, which, however, is the addressed optimization objective, latency is also evaluated and results are shown for the same settings in Fig. 5.3. At the beginning, when credits are randomly chosen, latency can be very high because choosing a too low credit on an important path may cause significant bottlenecks. Also here it is evident that about 100 transmissions are required before finding a good way to the optimum, that yields approximately the same end-to-end delay of MORE. Indeed, it can be assessed that being suboptimal in terms of airtime, i.e. performing more transmissions than required to deliver the message, can be advantageous for the delay. The two learning algorithms, particularly the tabular one, experience an e2e delay of up to 5% better than MORE's one.

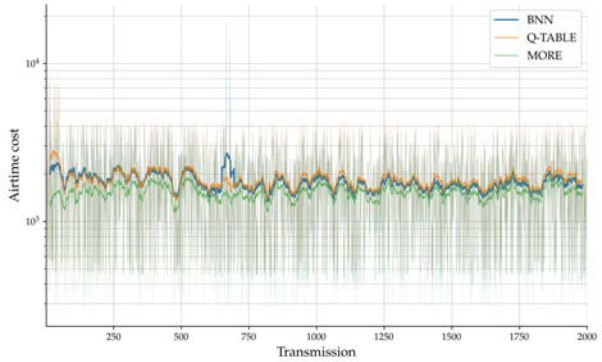
It is important to recall that results are averaged and, also for MORE, they change over time because source-destination pairs are randomly picked each time and airtime and latency highly depend on the number of hops selected. Moreover, even



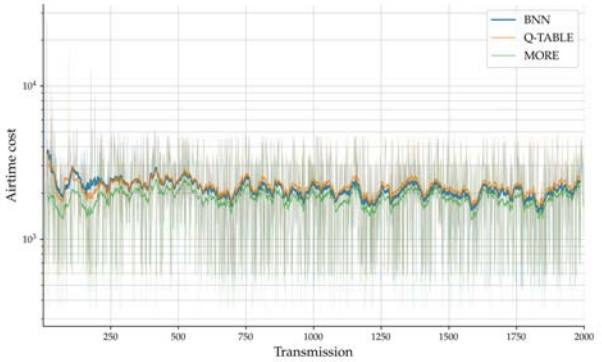
(A) Generation size 32.



(B) Generation size 64.

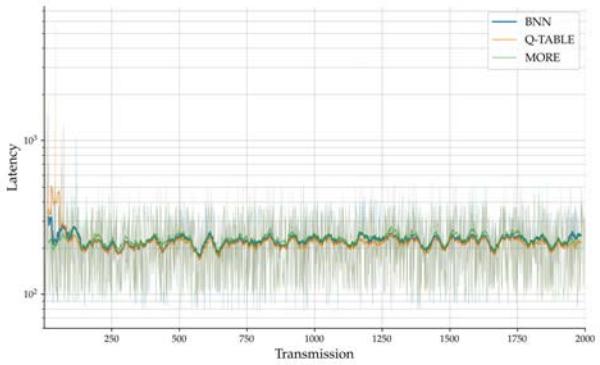


(C) Generation size 128.

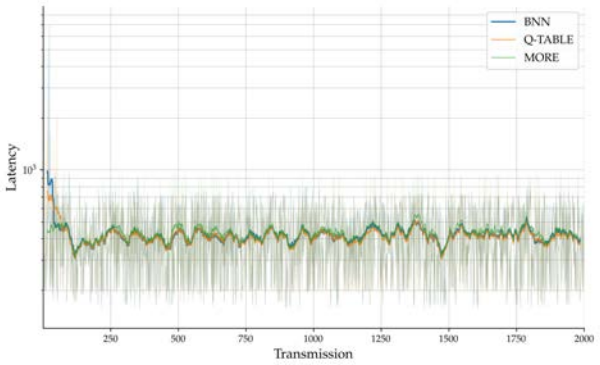


(D) Generation size 160.

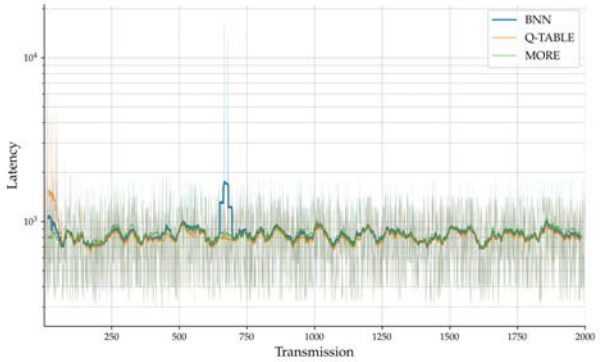
FIGURE 5.2: Airtime cost as a function of the transmission index. MORE, depicted in green, represents the optimal policy to be approached by the two learning algorithms.



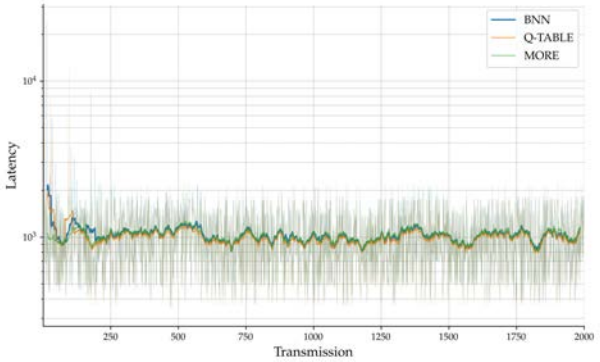
(A) Generation size 32.



(B) Generation size 64.

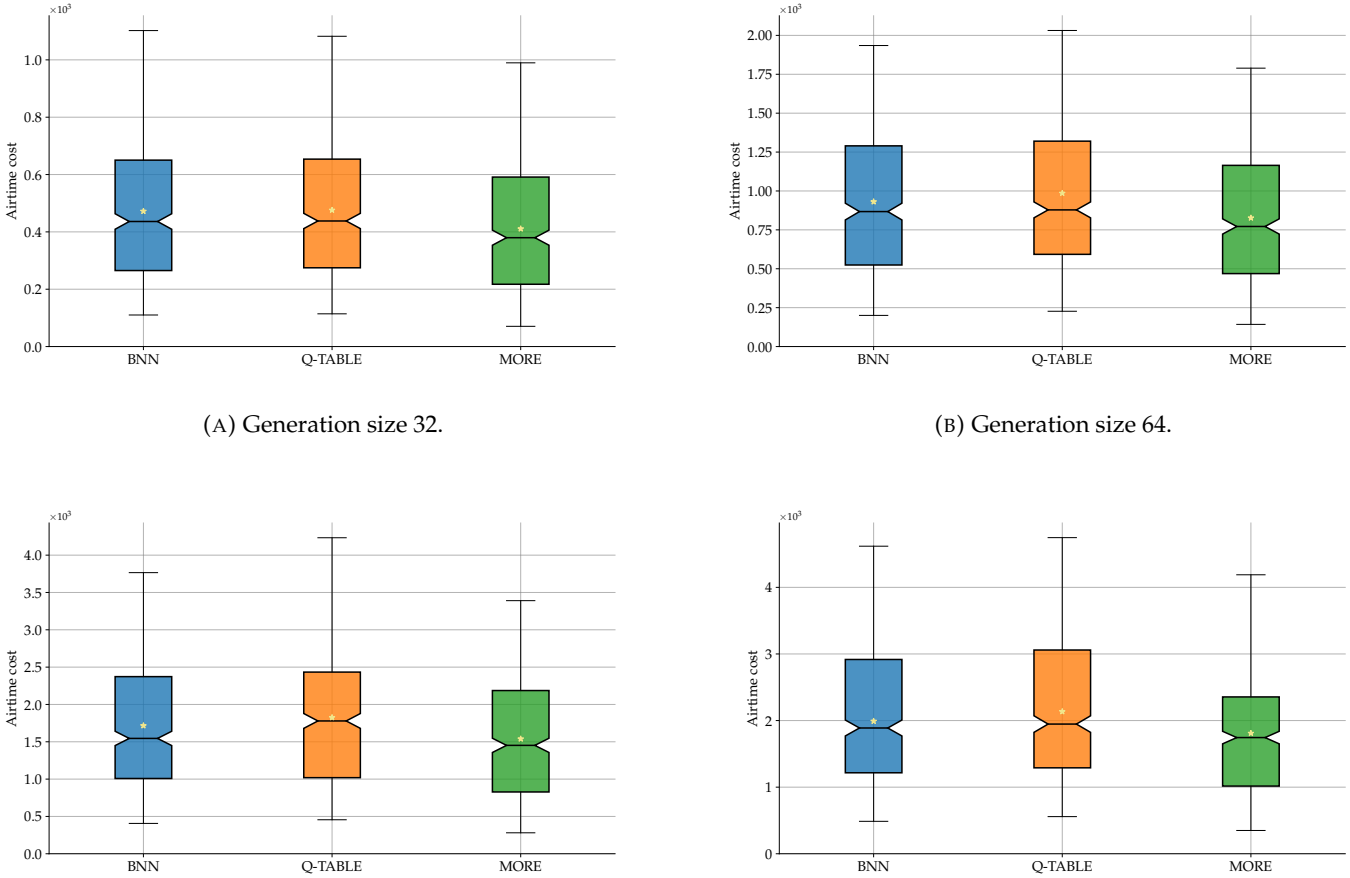


(C) Generation size 128.



(D) Generation size 160.

FIGURE 5.3: End-to-end delay (latency) as a function of the transmission index. Here, it can be seen how being not optimal in terms of airtime cost can be beneficial as for the latency.



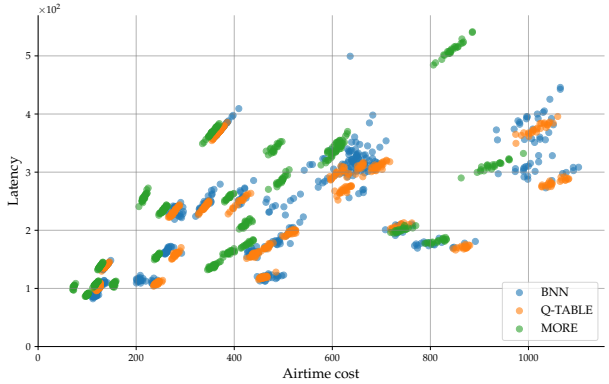
(A) Generation size 32.

(B) Generation size 64.

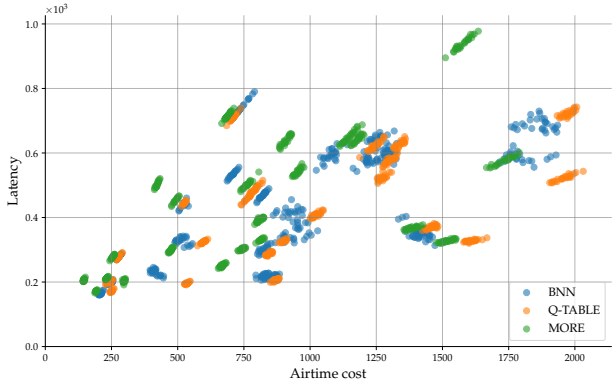
(C) Generation size 128.

(D) Generation size 160.

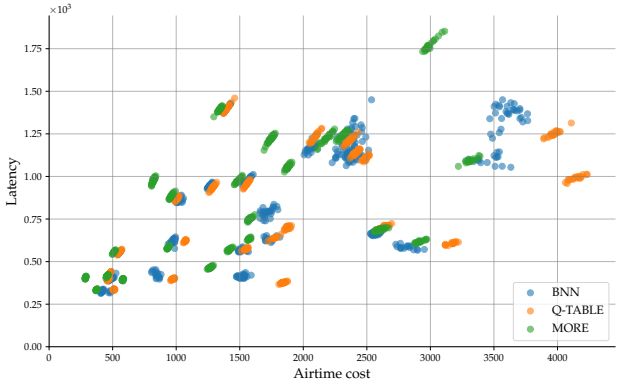
FIGURE 5.4: Boxplots of the airtime costs of the three compared algorithms. Results are evaluated on the last 500 full transmissions.



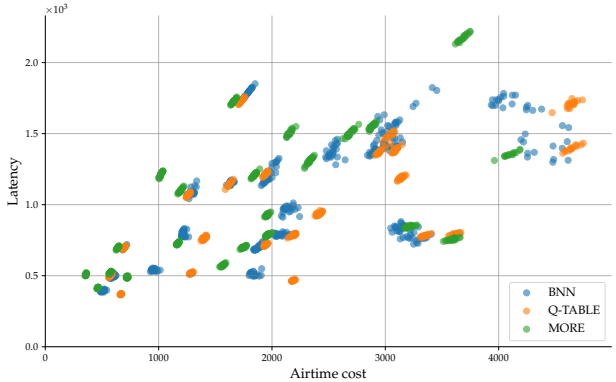
(A) Generation size 32.



(B) Generation size 64.

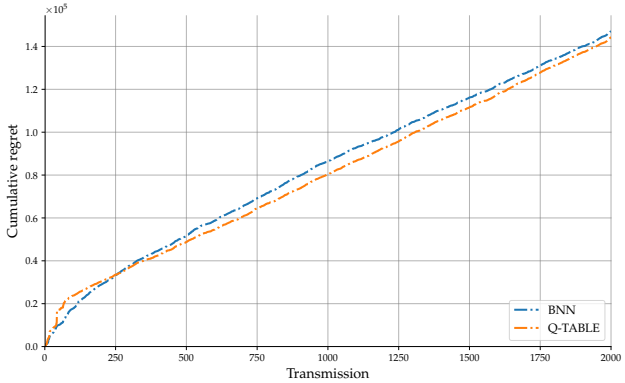


(C) Generation size 128.

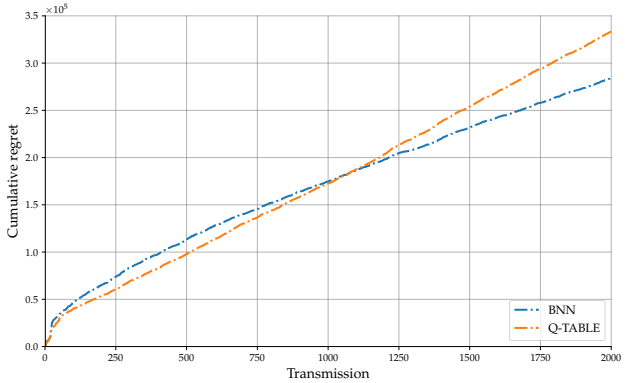


(D) Generation size 160.

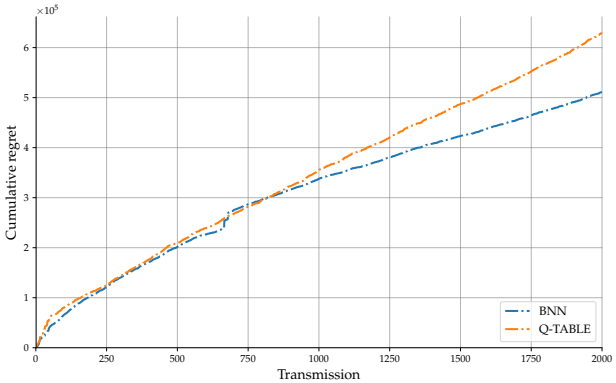
FIGURE 5.5: Scatter plots of airtime cost and latency.



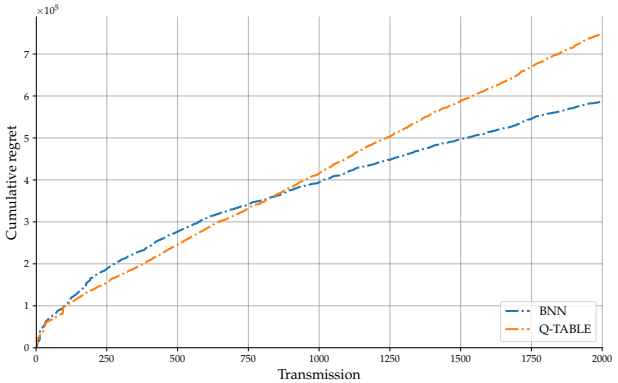
(A) Generation size 32.



(B) Generation size 64.



(C) Generation size 128.



(D) Generation size 160.

FIGURE 5.6: Cumulative regret of the two learning algorithms versus the performances of MORE.

when considering the same path, there will always be minor oscillations due to particular realizations about message deliveries.

Scatter plots of evaluation points, namely after transmission number 1500, in the airtime-latency plane are presented in Fig. 5.5. From this plot, ideally we would like to find points lying on the bottom left corner, i.e. for which both the airtime and the latency are low. Many clusters can be observed in these plots and they correspond to different realizations of the same sampled pair source-destination. Also, as we go far from the bottom left corner, we find paths of increasing length, as they require both more time and more transmissions. It is not surprising that the area where points lie gets wider as hops increase, because the more routers are involved in the task, the more possible solutions will exist. We can see that, as expected from previous results, blue and orange clusters are often moved to the right and slightly down with respect to their correspondent green cluster. This is again because the performance of the two proposed methods are worse in terms of airtime, along the x-axis and equal or slightly better as for the latency, along the y-axis.

In Fig. 5.6, finally, the cumulative regret of the two versions of the bandits with respect to MORE is shown. It is simply computed as the cumulative sum of the difference between the number of transmissions performed with the bandits and the ones that MORE executes. As can be seen, a complete saturation is not reached and will never be, because of the already mentioned problem of the approximation of the action space with a discrete set of transmission credits. However, as the generation size increases, the BNN bandit accumulates a regret which is significantly less than the one of the Q-table. Also, from the plot, it seems that, although diverging in both cases, the Q-table's regret increases linearly whereas the neural networks' one only logarithmically, which is way better in an asymptotic sense.

A comparison between the airtime performance obtained in the different meshes can be inferred looking at Tab. 5.1. While the general behavior of the approaches based on the bandits versus MORE does not change when varying the mesh and the generation size, there are notable results to highlight. To begin, in general there exists, for both of the two automated protocols, a gap between between the mean and median values, the latter measure being significantly lower. Moreover, we can observe how this gap is smaller for graph 2, and this is confirmed also by the smaller standard deviation measured. This gap is instead quite big for graph 3, and particularly in the case of the BNN algorithm. It is useful to look at Fig. 5.7 to understand to what is due this difference. For graph two, we find that the distribution of the airtime costs in the evaluation period has a mean value which is almost coincident with the median value, and from the scatter plot we do not observe significant distances between clusters of different protocols. On the other hand, for graph 3 we observe that the mean value is far from the median value also for MORE, and this is due to some statistical outliers, represented by points due to two pair source-destination with a high hop-count value. Moreover, for these two settings, the airtime performances of the two proposed algorithms are significantly worse, while the latency

	mean	median	std
graph 1	+18.56%	+13.04%	0.169
graph 2	+15.96%	+14.75%	0.085
graph 3	+25.83%	+14.35%	0.266

(A) BNN - G = 32

	mean	median	std
graph 1	+19.48%	+14.68%	0.172
graph 2	+21.27%	+17.78%	0.131
graph 3	+26.88%	+15.47%	0.263

(B) Q-table - G = 32

	mean	median	std
graph 1	+17.42%	+11.51%	0.188
graph 2	+19.28%	+17.51%	0.121
graph 3	+25.33%	+9.97%	0.282

(C) BNN - G = 64

	mean	median	std
graph 1	+24.53%	+16.28%	0.218
graph 2	+22.74%	+18.66%	0.158
graph 3	+29.09%	+19.16%	0.275

(D) Q-table - G = 64

	mean	median	std
graph 1	+15.40%	+8.23%	0.179
graph 2	+12.33%	+11.23%	0.072
graph 3	+23.17%	+8.36%	0.276

(E) BNN - G = 128

	mean	median	std
graph 1	+21.90%	+12.52%	0.207
graph 2	+26.55%	+21.87%	0.166
graph 3	+30.95%	+22.43%	0.287

(F) Q-table - G = 128

	mean	median	std
graph 1	+14.07%	+8.35%	0.173
graph 2	+17.33%	+16.84%	0.092
graph 3	+21.39%	+8.52%	0.250

(G) BNN - G = 160

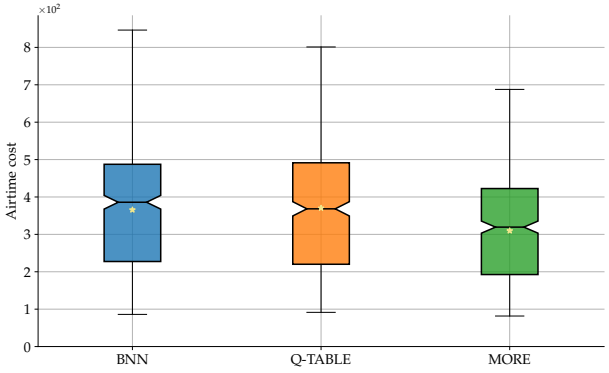
	mean	median	std
graph 1	+23.08%	+14.17%	0.220
graph 2	+18.86%	+16.75%	0.108
graph 3	+28.46%	+23.19%	0.273

(H) Q-table - G = 160

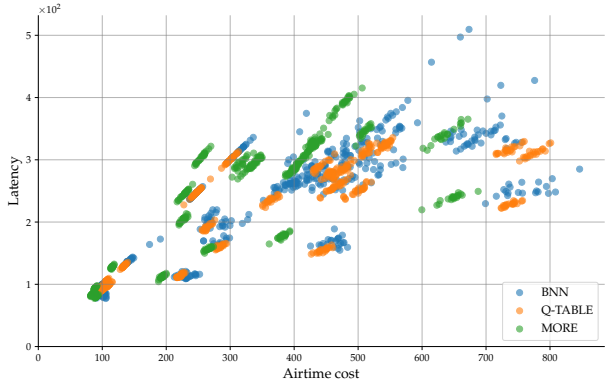
TABLE 5.1: Performance comparison in terms of airtime of the two proposed algorithms considering MORE as a baseline. Results are shown for all the three random geometric graphs of Fig. 5.1.

experienced is almost the same. Probably, these two clusters simply needed more training epochs, as they are the ones where the highest number of agents are involved, and this means more noise. These outliers affect results on the mean value of the airtime for graph 3, as can be deduced from Tab. 5.1.

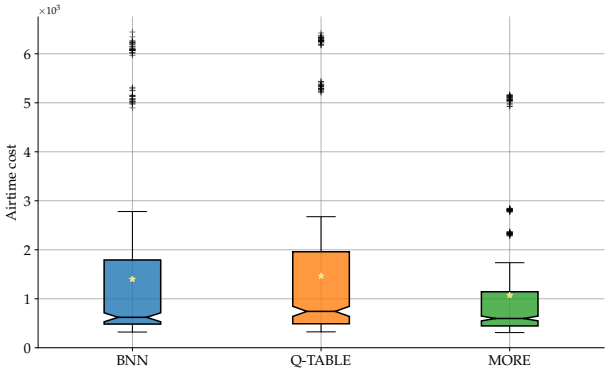
A last interesting point to notice is that MORE is optimal for an infinite generation size in an average sense, which means that it should approach actual optimality as long as the generation size increases. Therefore, a wider gap between MORE and the learning algorithms would be expected increasing G. Surprisingly, this is not true: as we can get from Fig. 5.8, where the average distance from MORE is represented as a function of the generation size, the Q-table is generally oscillating without a clear increasing behavior. The behavior of the BNN is even more peculiar: in two cases out of three, i.e. in meshes 1 and 3, it shows a clear linear decrease in the average distance from MORE as G grows. Since the length of symbols is fixed and the generation size is larger, more data is transmitted in this case. Therefore, because transmissions last longer and require a higher airtime accordingly, the difference in absolute terms between two different solutions will be enlarged by the same factor as well. This may be helpful for the learning procedure and speed of the bandits.



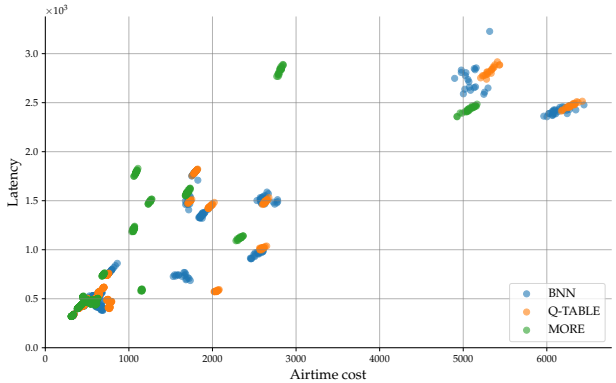
(A) Box plot - graph 2.



(B) Scatter plot - graph 2.

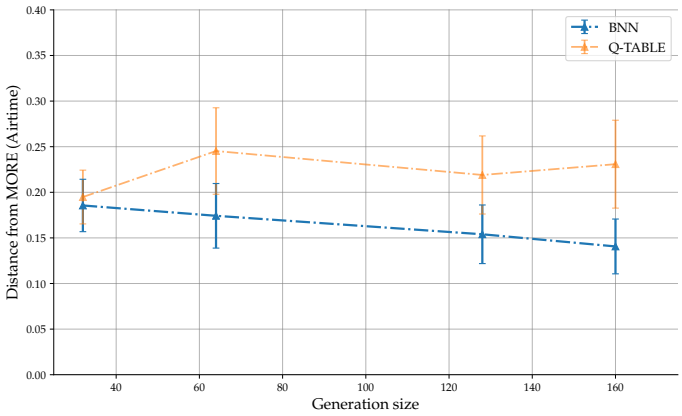


(C) Box plot - graph 3.

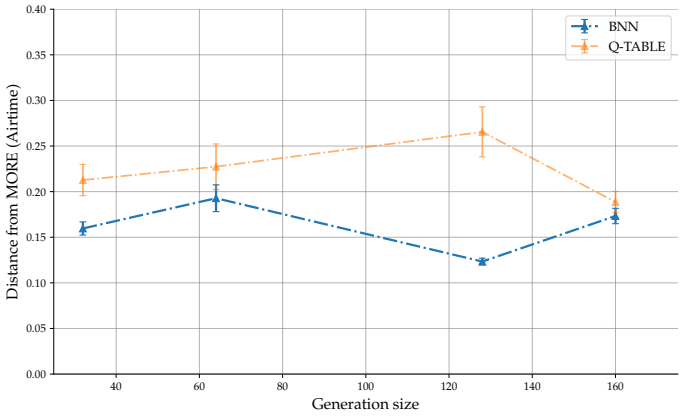


(D) Scatter plot - graph 3.

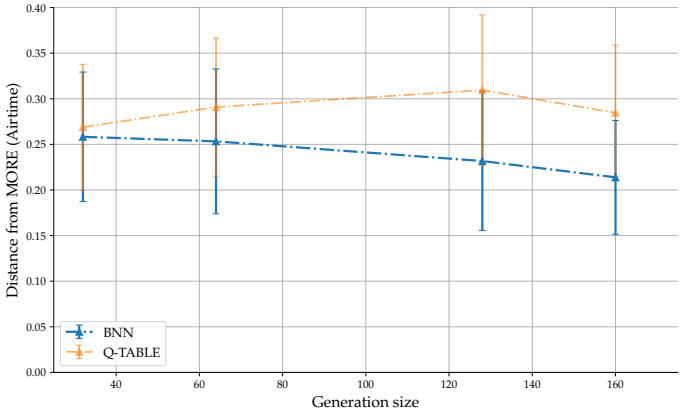
FIGURE 5.7: Comparison of the performance of graphs 2 and 3 with generation size 160.



(A) Graph 1.



(B) Graph 2.



(C) Graph 3.

FIGURE 5.8: Average airtime cost performance of the two algorithms as the generation size increases.

How much the differentiation is significant certainly depend on the specific topology and on the path. Actually, the same result is not valid for graph 2, but the argument is still valid because, as it was highlighted before, points in the airtime-latency plane are very close and performances are already quite good even with a low generation size (Tab. 5.1).

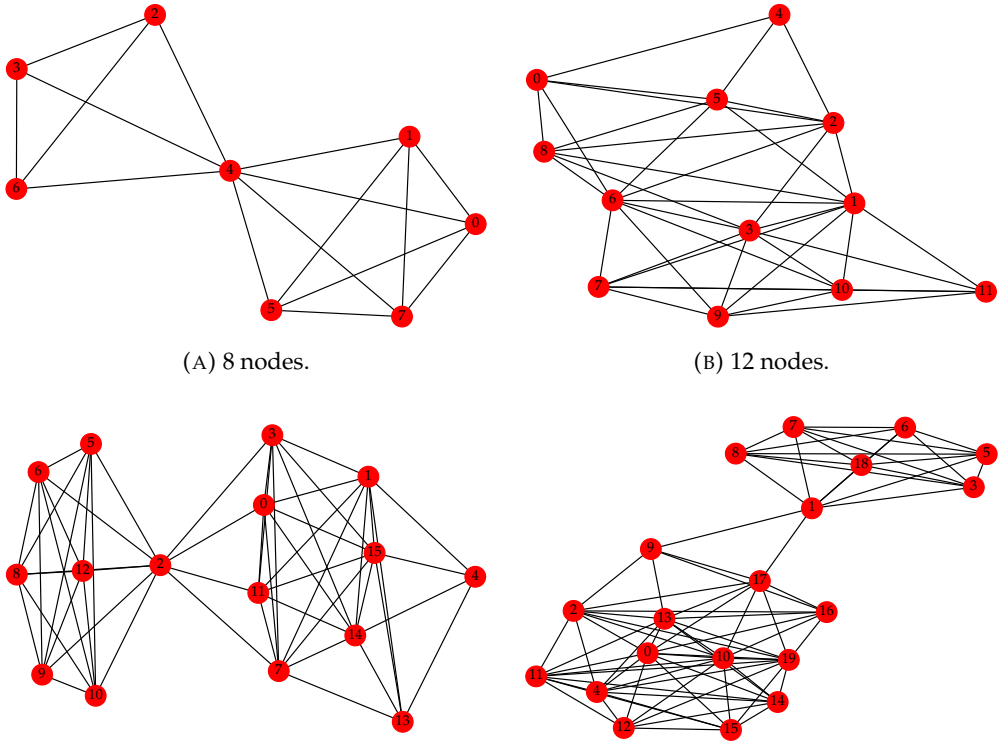
5.2 Broadcast results

Simulations about message broadcasting in mesh networks are performed using the same methods of the unicast case. There is a minor difference with respect to what was declared in Alg. 2: at each iterations every neural network is trained with 5 random minibatches, just for diminishing the duration of the simulations and the learning period. Here, the task is more complex as, while before the set of forwarders was predetermined thanks to the EAX metric (2.5), now this same measure has no more meaning, because it is applied only to streams with single destination. Therefore, the learning algorithms should assimilate from experience which routers can be switched off, choosing a transmission credit equal to 0 and saving thus energy and transmissions. Moreover, any source can be the message originator, which also makes the task difficult. In many practical cases, originators can be a small subset of the nodes of the mesh, which, however, may also be much larger than 20 nodes.

In this section, studies on the behavior of airtime and latency are evaluated as the size of the mesh increases, up to 20 nodes and keeping a fixed generation size $G = 64$. In Fig. 5.9 the RGGs used as meshes are shown. They have a generic shape, also with the presence of bottlenecks between two clusters of highly connected components. The mesh with 12 nodes is also used to evaluate the adaptability of the algorithm to channel quality variations, i.e. in case of link failure, if the channel gets worse, and link addition, if the channel gets better.

5.2.1 Airtime and latency

The optimization objective for broadcast communications described in Section 3.1 includes the minimization of the airtime subject to a latency constraint, modeled as a maximum timeout threshold τ before which all the nodes must be able to retrieve the full message. Therefore, airtime and latency should be evaluated together, with timeout thresholds set to the number of time steps reported in Tab. 5.2. From Fig. 5.10 to Fig. 5.13, the plots of airtime and latency are shown, for the increasing size meshes, and with a moving average filter of length 320 samples, corresponding to 10 full transmissions. As a general result, it is evident that the two learning algorithms are able to avoid the timeout after a first adjustment period, which is required also for avoiding infeasible solutions for which the information flow is smaller than the 100%, which is of course mandatory. On the other hand, MORE multicast and BATMAN often do not satisfy the delay constraint, hitting therefore the top of the latency



(A) 8 nodes.

(B) 12 nodes.

(C) 16 nodes.

(D) 20 nodes.

FIGURE 5.9: Random geometric graphs employed for the broadcast simulations.

plots. More in the detail, for the smallest mesh with 8 nodes, it can be observed from the plot that MORE multicast seems to have a better average airtime cost than the BNN based algorithm. However, this is true only because the majority of the transmissions was stopped after reaching the timeout: in this case, MORE only satisfies the constraint in the 17.40% of the cases[1]. BATMAN's behavior is instead quite random in that, regardless of the specific topology of the mesh, it always forces routers to transmit three new packets for every innovative packet received. Results show that BATMAN is the worse protocol in terms of airtime in three cases out of four and the delay constraint is satisfied consistently only for the mesh with 12 nodes. Moreover, although the tabular based algorithm performs better than the two routing protocols used for the comparison with regard to the time constraint, a clear gain is not observed in airtime. Unlike the deep learning solution, actually, it only learns to avoid the penalty reducing the airtime just marginally. The reason for this can be found in the different reward functions: the developed mapping function (3.31) in $[0, 1]$ for the reward seems to work better than the commonly used large penalty when breaking constraints. More detailed numerical results are summarized in Tab. 5.2.

[1]In fact, MORE is implemented exactly as described theoretically. However, in order to make it work practically, some feedback with possible retransmissions should be implemented to increase reliability and resilience against dead ends. This, of course at a cost of some airtime and overhead.

Nodes	Timeout τ	BNN	Q-table	MORE multicast	BATMAN
8	1250	98.52%	96.68%	17.40%	52.19%
12	1500	99.94%	99.85%	49.94%	97.08%
16	2000	97.17%	72.10%	64.39%	45.18%
20	4500	96.91%	95.34%	85.65%	38.23%

TABLE 5.2: Percentage of transmissions for which the maximum delay constraint (3.3) is satisfied by the different protocols.

Nodes	MORE multicast	BATMAN
8	8.4%	22%
12	9.5%	18%
16	10%	14%
20	12%	10%

(A) BNN.

Nodes	MORE multicast	BATMAN
8	-7.6%	8.4%
12	1.5%	11%
16	1.4%	6.1%
20	1.7%	$\sim$

(B) Q-table.

TABLE 5.3: Improvement in the median airtime cost of the two learning algorithms with respect to MORE multicast and BATMAN.

From these time plots, it can be assessed that the Neural Network based algorithm has the best performance, as it keeps the latency under the timeout threshold consistently above the 97% in every mesh while having always the smallest airtime cost. Also, the Q-table learns well to avoid the timeout in at least three meshes, yet, airtime results are not as good as the BNN's ones. As it is shown in Tab. 5.3, with the BNN there is an improvement in airtime of $8\% - 12\%$ and $10\% - 22\%$ when comparing with MORE and BATMAN respectively. The Q-table, instead performs really close to MORE multicast and up to 10% better than BATMAN. In Tab. 5.4, similar results for the latency are reported: the Neural Network has a gain of up to 17% and 31%, whereas the Q-table here looses some points, performing worse until the 16% and 18% respectively for MORE multicast and BATMAN. As can be seen graphically in Fig. 5.14, the gain with respect to BATMAN decreases as the size of the mesh increases. It would be interesting, with further studies, to assess whether this behavior saturates or not. Also, it is notable the sudden drop in the latency gain for the mesh with 20 nodes with the tabular bandits. It may be due to a particular topology, having a heavy bottleneck (see Fig. 5.9) or maybe just we reached the maximum number of agents that can be handled efficiently by a set of tabular bandits.

Nodes	MORE multicast	BATMAN
8	17%	31%
12	14%	25%
16	13%	17%
20	5.6%	3.6%

(A) BNN.

Nodes	MORE multicast	BATMAN
8	-2.7%	15%
12	$\sim$	12%
16	4%	8%
20	-16%	-18%

(B) Q-table.

TABLE 5.4: Improvement in the median end-to-end delay (latency) of the two learning algorithms with respect to MORE multicast and BATMAN.

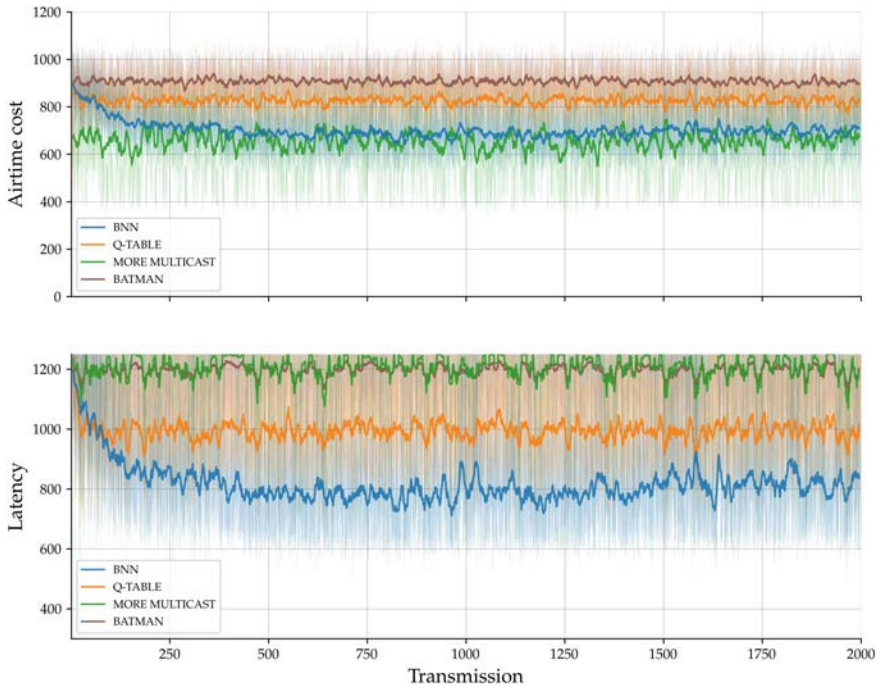


FIGURE 5.10: Airtime cost and latency, mesh with 8 nodes.

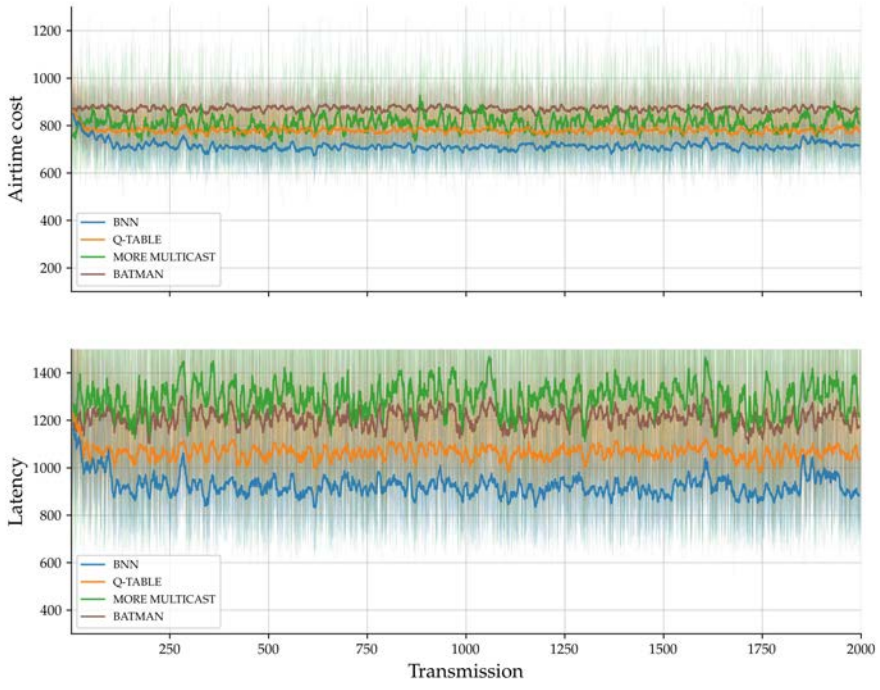


FIGURE 5.11: Airtime cost and latency, mesh with 12 nodes.

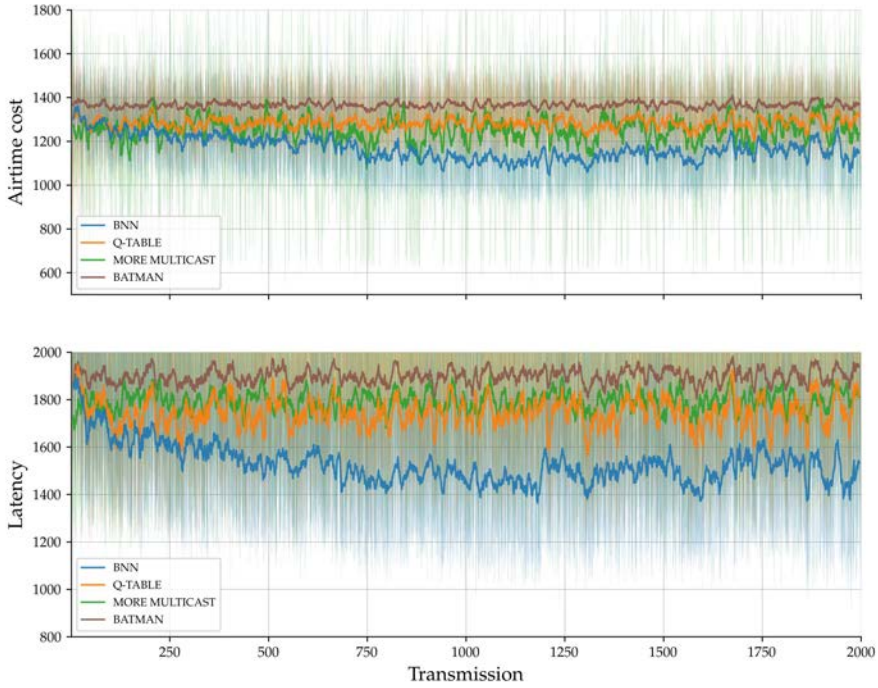


FIGURE 5.12: Airtime cost and latency, mesh with 16 nodes.

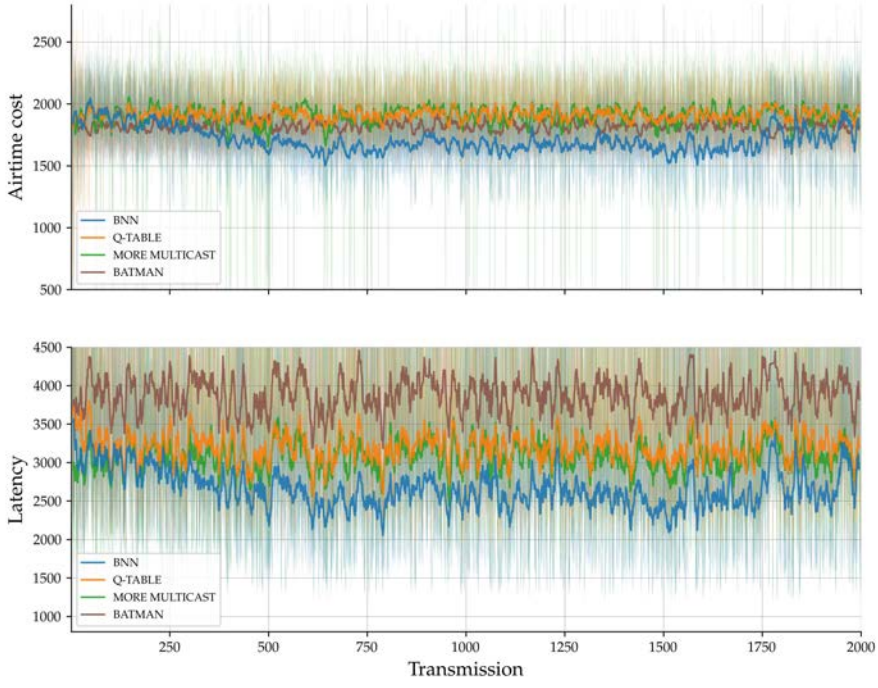
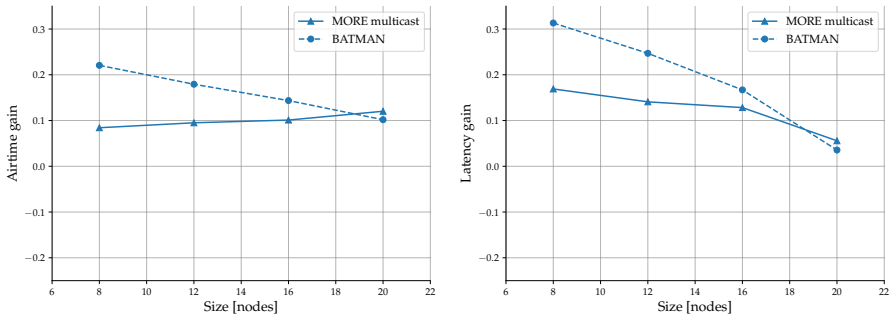
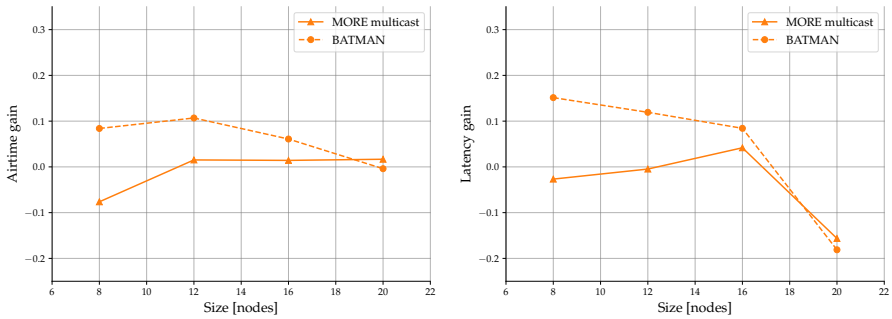


FIGURE 5.13: Airtime cost and latency, mesh with 20 nodes.



(A) BNN performance

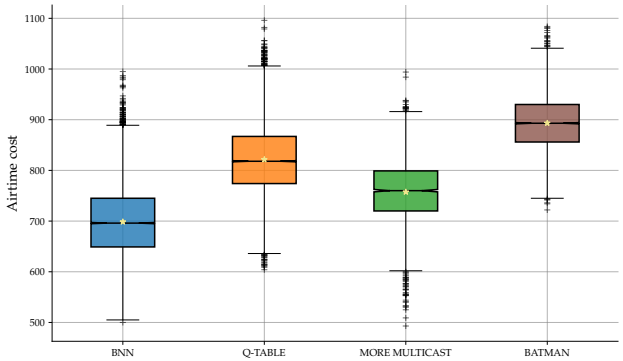


(B) Q-table performance

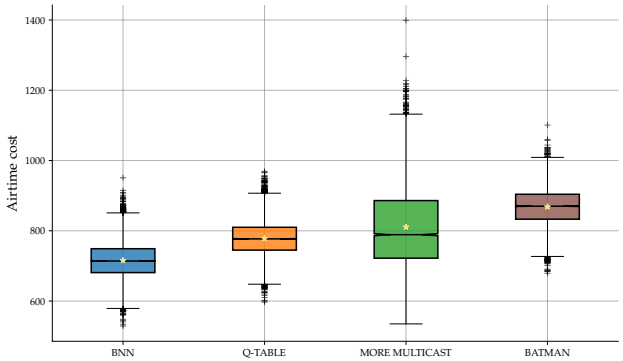
FIGURE 5.14: Median performance of the learning algorithms as the size of the mesh increases.

From the box plots of Fig. 5.15 a general evaluation of the airtime statistics can be derived. Only points that satisfies the latency constraint are considered here and boxes confirm that the BNN algorithm has both mean and median values below all the other algorithms. In some cases the variance is slightly larger than BATMAN, but we shall consider that the number of points with which the boxes are built is much higher for the BNN, which considers also transmissions that BATMAN is not able to manage.

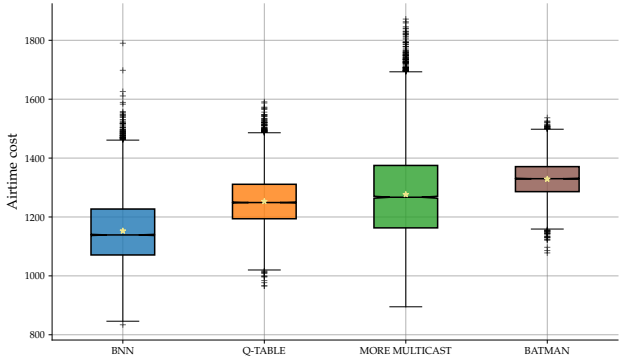
Finally, and being probably visually the most effective plots, from Fig. 5.16 to Fig. 5.19 heatmaps in the airtime-latency are shown. Like for the unicast case, our wish is to find hot spots in the bottom left corner, where both the latency and the airtime are low. Further more, in these plots, not only the position of the spot is relevant, but also its color: the more dark it is, the better, as the concentration of points in that area gets higher. Also, the darkest value in each heatmap is considered to be the maximum value among all the evaluated protocols. Therefore, lighter heatmpas also means that the number of valid points are less, as it is the case of MORE multicast for the 8 nodes mesh. In every plot, it can be seen how hot spots of the BNN are constantly on the bottom left corner, whereas for the Q-table they are more centred. BATMAN lies instead on the upper right part and it approaches the centre progressively with the increasing mesh size. MORE multicast, finally, keeps close to the bottom left corner but its hot spots are often weak and the distribution of points is much more spread.



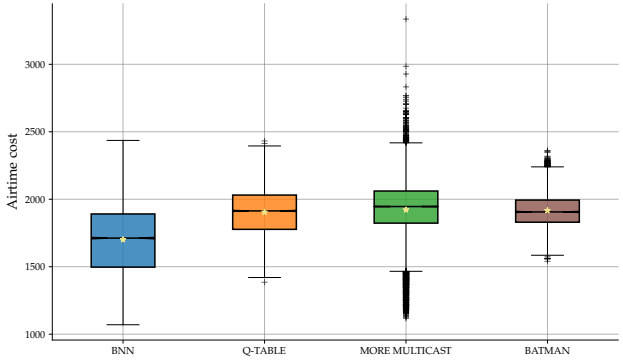
(A) Mesh with 8 nodes.



(B) Mesh with 12 nodes.



(C) Mesh with 16 nodes.



(D) Mesh with 20 nodes.

FIGURE 5.15: Box plots of the airtime cost for the different protocols, considering only points satisfying the delay constraint (3.3).

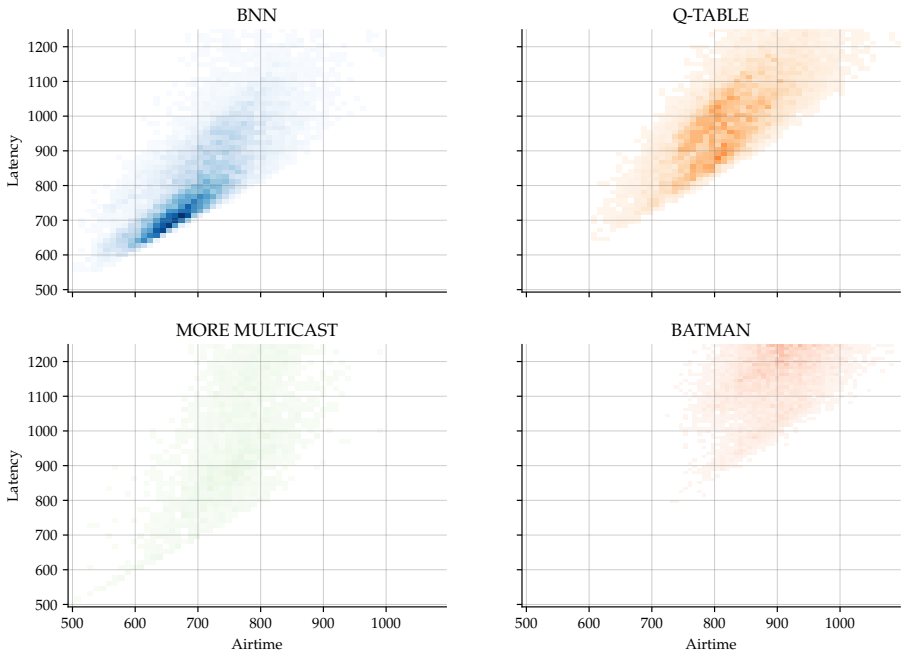


FIGURE 5.16: Heatmap in the airtime-latency plane for the four evaluated protocols, mesh with 8 nodes.

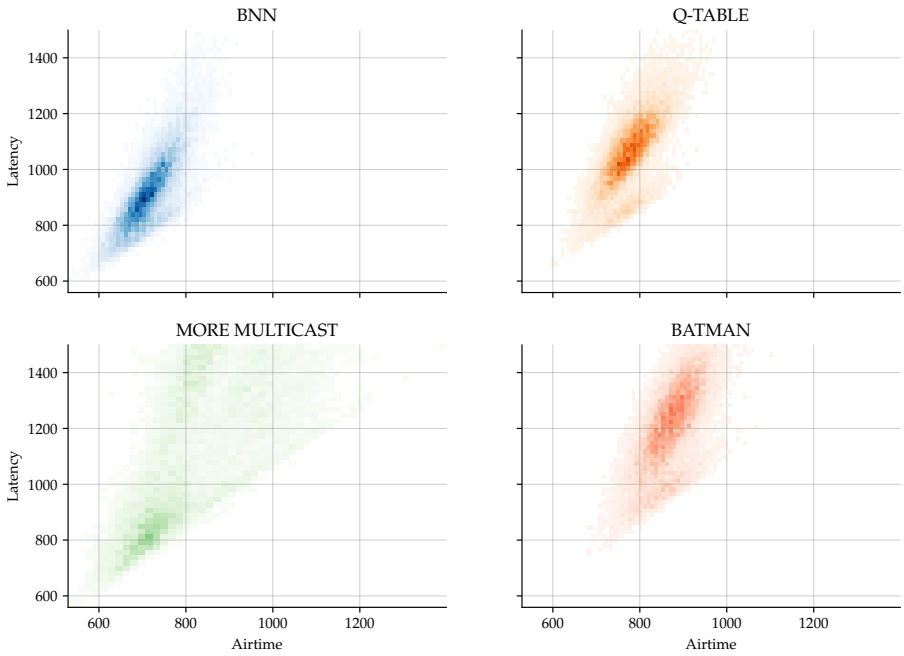


FIGURE 5.17: Heatmap in the airtime-latency plane for the four evaluated protocols, mesh with 12 nodes.

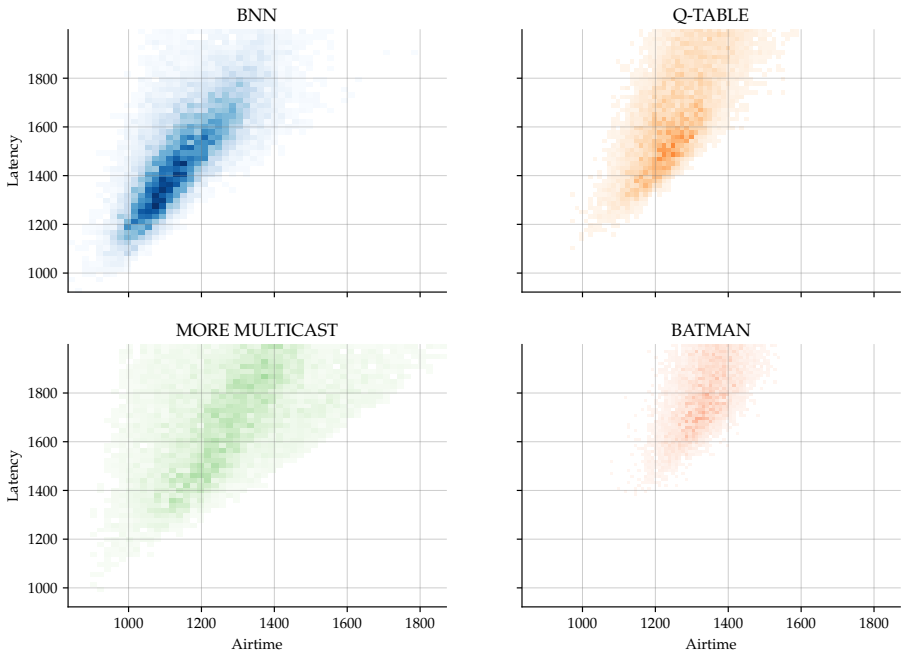


FIGURE 5.18: Heatmap in the airtime-latency plane for the four evaluated protocols, mesh with 16 nodes.

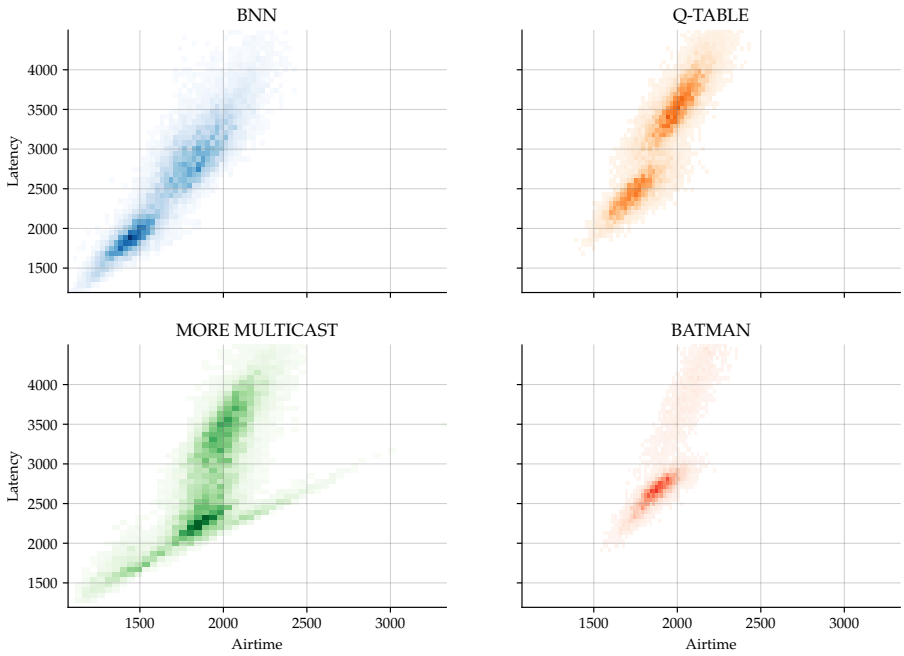


FIGURE 5.19: Heatmap in the airtime-latency plane for the four evaluated protocols, mesh with 20 nodes.

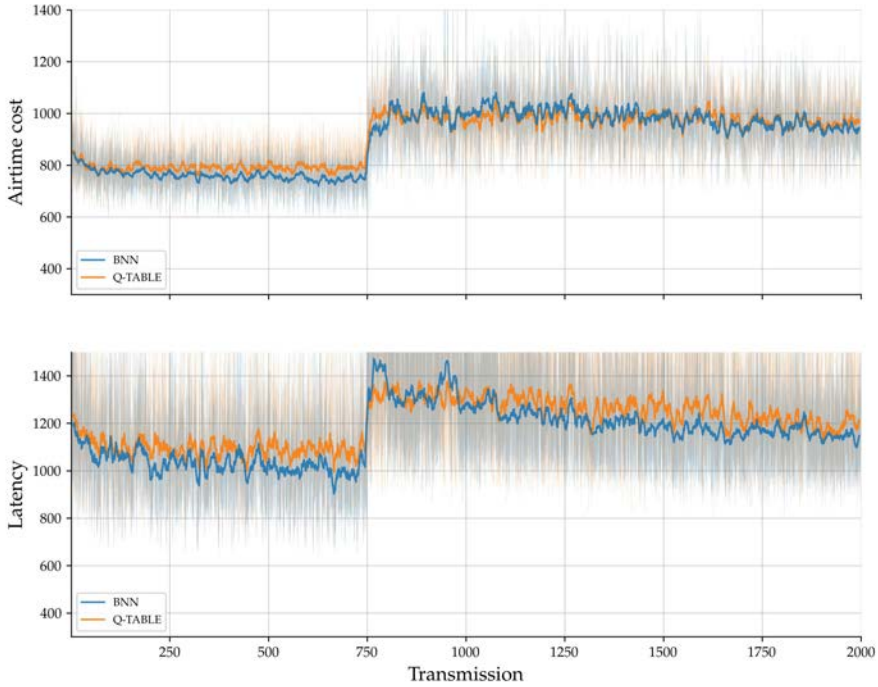
5.2.2 Adaptability to channel variations

Besides the study on the performance reached on a stationary environment, it is interesting to inspect also the adaptability of the algorithms when the wireless medium changes. In this section, results relative to two simple experiments are presented, employing the mesh with 12 nodes (Fig. 5.9b). In the first one, five random links are removed, so that the quality of the channel gets worse; in the second one, three new random links are added, improving thus the medium.

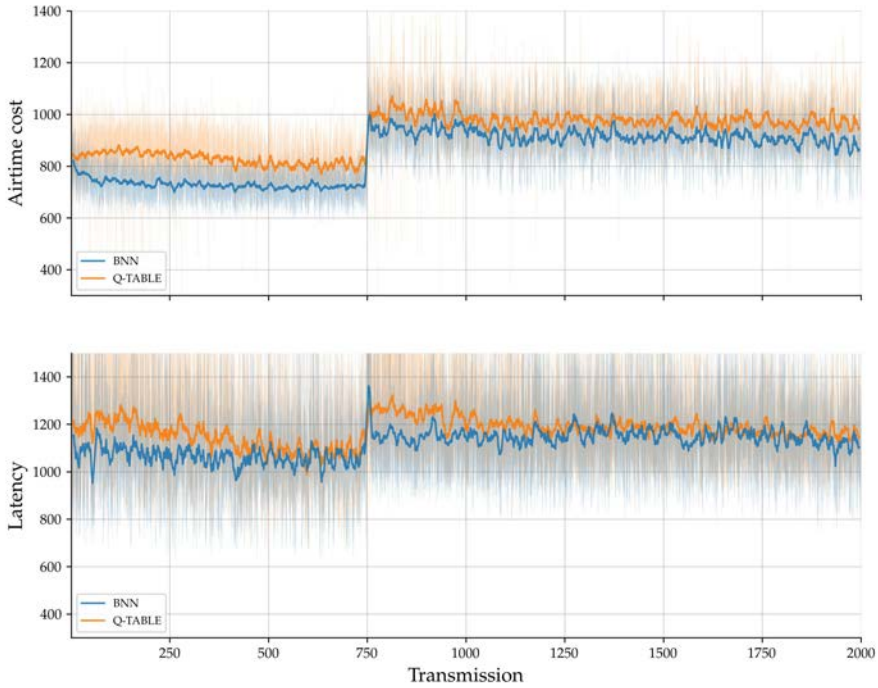
As for what concerns the link failure experiment, if we just let the algorithms run in their basic form as explained in Section 3.3, the performance is poor in terms of convergence time, as can be seen in Fig. 5.20a. Actually, even though the two algorithms start again to look for a new approximately optimal solution after the breaking point, which is around transmission 750, the period in which the latency constraint cannot be satisfied lasts at least 500 full transmissions. Moreover, during the evaluation period, which is again after transmissions 1500, the reached airtime is of 952 and 965 transmission, for BNN and Q-table respectively. As this is not satisfying at all, because in a wireless medium the rate of non-stationarity can be higher than the time needed to recover a good perfomance, the algorithm should be slightly changed to allow for more exploration when needed.

We can easily detect if the exploration should be increased because we can measure the perfomance to assess whether it is worsening too much. In this case, for the Neural Networks, we can reset the buffer so that old values that are no more updated for the current state of the channel are deleted. The minimum and maximum value for the reward mapping in $[0, 1]$ stored until that moment should be deleted as well. Further more, the dropout entropy can be maximized again restoring to the original value N, the size of the observed points, as well as the rate of training, which is re-initialized if needed. The Q-tables, instead, only need to change the learning rate α to a fixed value, here 0.1. In this way an exponentially weighted average over the past history is computed and recent samples are considered more than older ones. Results are shown in Fig. 5.20b and it can be seen how for the BNN based solution the latency constraint is satisfied again after only about 20 full transmissions. The tabular solutions requires more time, about 300 transmissions. So, the convergence speed is greatly improved for both the methods and the percentage of satisfied time-out constraints increases as well, particularly 6.5% for the tabular solution. Also, the BNN based algorithm finds a better optimum as for the airtime cost, performing 5% better than before. Results are summarized in Tab. 5.5.

In Fig. 5.21 results relative to the link addition experiment are shown. Here, we can observe how, for the BNN algorithm, results in the evaluation window are unchanged. However, the speed of convergence is about five times faster when the exploration reset is active. In this case, however, no gain in speed is observed for the tabular bandits and also we find a decrease in performance as for the latency constraint. On the other hand, the airtime cost is slightly improved, of about 1.3%. Results are finally summarized in Tab. 5.6.

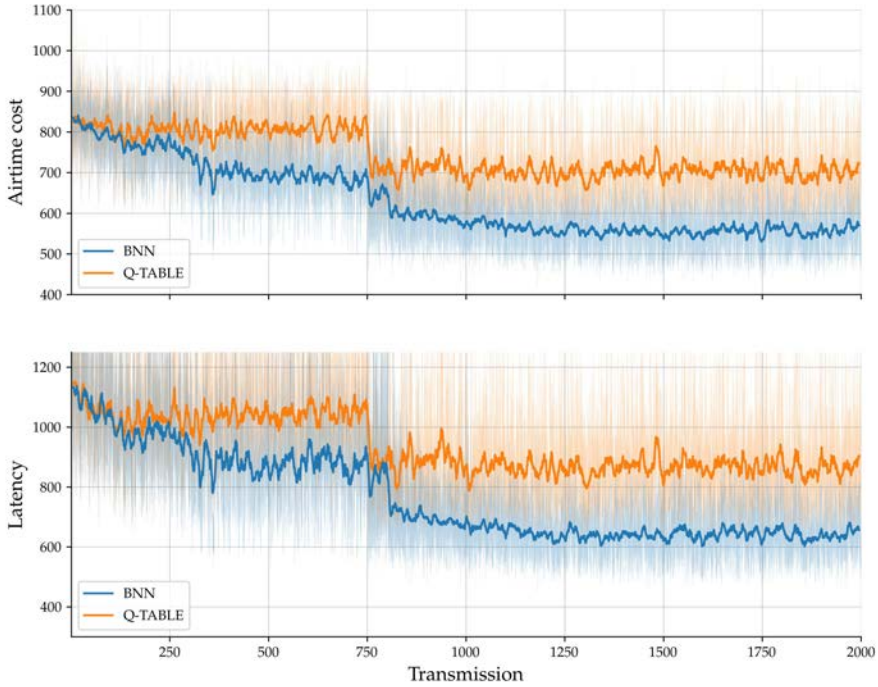


(A) Without exploration reset.

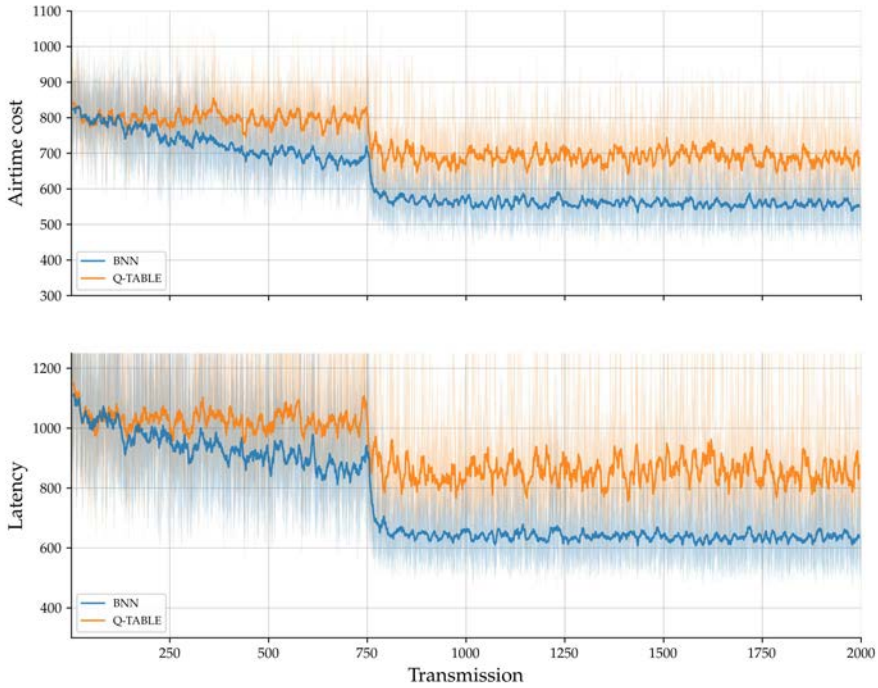


(B) With exploration reset.

FIGURE 5.20: Learning behavior with link failure, mesh with 12 nodes.



(A) Without exploration reset.



(B) With exploration reset.

FIGURE 5.21: Learning behavior with link addition, mesh with 12 nodes.

	BNN	**Q-table**
Timeout	97.74%	92.05%
Transmissions	952	965

(A) No reset.

	BNN	**Q-table**
Timeout	98.78%	98.63%
Transmissions	905	969

(B) With reset.

TABLE 5.5: Fraction of latency constraint respected and median number of transmissions in the link failure experiment.

	BNN	**Q-table**
Timeout	100%	99.42%
Transmissions	552	693

(A) No reset.

	BNN	**Q-table**
Timeout	100%	96.77%
Transmissions	553	684

(B) With reset.

TABLE 5.6: Fraction of latency constraint respected and median number of transmissions in the link addition experiment.

The BNN algorithm shows a good level of adaptability in both the experiments when the exploration policy is renewed. However, we should keep in mind that it means that the neural networks are trained at a higher rate every time a reset is needed. Therefore, it is important to understand whether significant changes in the channel, like relevant SNR variations or moving nodes that produce a change in the mesh topology, comes at a rate that makes infeasible this solution from a practical point of view. IoT devices, for instance, often have a limited computational power and battery life and training for a long time a neural network is energy consuming. For this kind of application, therefore, the channel should not vary with a too high rate. On the contrary, when dealing with routers installed in powerful Base Stations (BSs) or when the power grid is available, energy does not respresent a problem any more. A possible trade-off could be the definition of a threshold of maximum performance decay, considering also the duration of this degradation. The latest reward values are considered and, when the quality threshold is overcome for more than the choosen maximum period, the exploration policy is reset. Depending on our power possibilities we can thus be more or less strict with these thresholds, conditioning the rate of learning in non-stationary conditions.

Chapter 6

Conclusion

In this thesis, two novel algorithms for routing are proposed, basing on the paradigm of Opportunistic Routing. Machine learning is introduced in this domain, more specifically the tool of reinforcement learning, and the objective is to find an optimal forwarding rate for each router for the minimization of the airtime cost with delay constraints. The use of Neural Networks as function approximators reveals that increasing the dimensionality of the state space is beneficial for the task, as the deep learning based solution works generally better than the traditional Q-table.

As this work is applied to Wireless Mesh Networks (WMNs), a fully distributed algorithm is highly desirable, because nodes act as peers and not in the classic masters-slaves configuration, nor a network controller is present, which is the case of Software Defined Networks (SDNs). Therefore, every router of the mesh is considered as one agent in a multi-agent with independent learners scenario, requiring only local information. This last point is also advantageous, because nodes do not need to know the global topology of the mesh, which may be large, requiring thus too much information to be shared by far routers. It is well known, however, that independent learners often do not lead to good Nash Equilibria, hence, in this thesis, convergence is improved exploiting the already existing communications among neighbors, piggybacking the additional useful information. Moreover, through a proper reward shaping, the convergence is pushed towards a Pareto optimal solution, improving global welfare. At the same time, this design keeps a very lightweight protocol in terms of overhead, since it only requires to estimate the delivery probabilities in the 1-hop range and communicate the transmission credits played by the neighbors.

In a first stage, the learning frameworks are compared, in a unicast scenario, with MORE, a protocol which is proved to be optimal in terms of airtime when the generation size is infinite. Results sow that airtime performance is about $15\% - 25\%$ worse when reaching convergence. Nevertheless, the algorithms developed in this thesis only employ local information, and they learn the actual topology of the mesh, whereas MORE relies on a complete knowledge of the topology and of its changes.

In the second part of the thesis, the broadcast behavior of the algorithms is inspected. Here, a theoretical optimum does not exist and protocols employed are adaptations from unicast or suboptimal heuristics. In this case, the two learning algorithms outperform both MORE multicast and BATMAN when it comes to the

delay constraint. Moreover, the deep learning bandits show significant improvements also in the airtime and latency, as clearly visible from the heatmaps in Section 5.2.

The BNN based routing protocol seems to be highly promising as the algorithm is fully distributed, scalable as far as experiments were performed and with a good capability of adaptation to channel and topology variations. Nevertheless, this thesis only represent a starting point for future developments. Specifically, these modifications may be tried to improve performances:

- Switch to a **continuous control** framework, since discretizing the action space of the bandits is certainly a practical solution, but something is certainly lost with respect to the optimal solution that could possibly be reached with a continuous action space.
- Extend the bandits to **full reinforcement learning** in broadcast, because in this case we may see an advantage in adopting MORE multicast's behavior of dynamically changing the transmission credits every time the ACK of one of the destination is received. At the current state, actually the bandit adopts a static solution for the whole batch, regardless of the fact that optimal credits may change.
- **Tune** progressively the **latency constraint**, as neural networks require data at the beginning to find a way towards a local optimum. With a random initialization, actually, it is very likely that combinations tried are not good enough to satisfy a tight contraint and rewards are therefore always very low, resulting in a non learning framework. If, instead, the delay costraint is made tighter after the networks have learned which are fair enough combinations, convergence would be faster and, possibly, a shorter delay could at the end be reached.

Moreover, the simulation environment was extremely simplified and rather unrealistic. The following step would be the use of a more advanced network simulator, including more different topologies, also less connected, and the on-field testing. However, following the principle that the developed tool learns from data, it should adapt as well to a real scenario and better than other heuristic based protocols. It would be interesting to evaluate also the performance in a multi-rate scenario, possibly using as input feature the maximum receiving rate of neighbors. Another case in which reinforcement learning should be effective, finally, is in the presence of multi-flow communications, when the network is overloaded by traffic data. Actually, Q-routing has already proved to be effective in finding alternative paths under these conditions, improving hence throughput by avoiding congestions.

Bibliography

[1] M. Médard and A. Sprintson, *Network Coding: Fundamentals and Applications*. Elsevier, 2012.

[2] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network information flow", *IEEE Transactions on information theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[3] C. Fragouli, E. Soljanin, *et al.*, "Network coding fundamentals", *Foundations and Trends® in Networking*, vol. 2, no. 1, pp. 1–133, 2007.

[4] J. Heide, S. Shi, K. Fouli, M. Medard, and V. Chook, "Random Linear Network Coding (RLNC)-Based Symbol Representation", Tech. Rep., Jul. 2019. [Online]. Available: `https://datatracker.ietf.org/doc/html/draft-heide-nwcrg-rlnc-02`.

[5] T. Clausen and P. Jacquet, "Optimized link state routing protocol (olsr)", Tech. Rep., 2003. [Online]. Available: `https://www.ietf.org/rfc/rfc3626.txt`.

[6] Z. Li, B. Li, and L. C. Lau, "On achieving maximum multicast throughput in undirected networks", *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2467–2485, 2006.

[7] C. Fragouli, J. Widmer, and J.-Y. Le Boudec, "Efficient broadcasting using network coding", *IEEE/ACM Transactions on Networking (TON)*, vol. 16, no. 2, pp. 450–463, 2008.

[8] D. Nguyen, T. Tran, T. Nguyen, and B. Bose, "Wireless broadcast using network coding", *IEEE Transactions on Vehicular technology*, vol. 58, no. 2, pp. 914–925, 2008.

[9] S. Pandi, F. Gabriel, J. A. Cabrera, S. Wunderlich, M. Reisslein, and F. H. Fitzek, "Pace: Redundancy engineering in rlnc for low-latency communication", *IEEE Access*, vol. 5, pp. 20 477–20 493, 2017.

[10] S. Wunderlich, F. Gabriel, S. Pandi, F. H. Fitzek, and M. Reisslein, "Caterpillar rlnc (crlnc): A practical finite sliding window rlnc approach", *IEEE Access*, vol. 5, pp. 20 183–20 197, 2017.

[11] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing", Tech. Rep., 2003. [Online]. Available: `https://tools.ietf.org/rfc/rfc3561.txt`.

[12] D. S. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing", *Wireless networks*, vol. 11, no. 4, pp. 419–434, 2005.

[13] S. Biswas and R. Morris, "Exor: Opportunistic multi-hop routing for wireless networks", *ACM SIGCOMM computer communication review*, vol. 35, no. 4, pp. 133–144, 2005.

[14] E. Rozner, J. Seshadri, Y. Mebta, and L. Qiu, "Simple opportunistic routing protocol for wireless mesh networks", in *2006 2nd IEEE Workshop on Wireless Mesh Networks*, IEEE, 2006, pp. 48–54.

[15] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing", *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 169–180, Aug. 2007, ISSN: 0146-4833. DOI: 10.1145/1282427.1282400. [Online]. Available: http://doi.acm.org/10.1145/1282427.1282400.

[16] ——, *Trading structure for randomness in wireless opportunistic routing*, 4. ACM, 2007, vol. 37.

[17] Z. Zhong, J. Wang, and S. Nelakuditi, "Opportunistic any-path forwarding in multi-hop wireless mesh networks", PhD thesis, University of South Carolina, 2006.

[18] Z. Zhong, J. Wang, S. Nelakuditi, and G.-h. Lu, "Mobicom poster abstract: On selection of candidates for opportunistic anypath forwarding", 2006.

[19] A. Neumann, C. Aichele, M. Lindner, and S. Wunderlich, "Better approach to mobile ad-hoc networking (batman)", Tech. Rep., 2008, pp. 1–24. [Online]. Available: https://tools.ietf.org/id/draft-openmesh-b-a-t-m-a-n-00.txt.

[20] D. Johnson, N. Ntlatlapa, and C. Aichele, "Simple pragmatic approach to mesh routing using batman", 2008.

[21] L. Barolli, M. Ikeda, G. De Marco, A. Durresi, and F. Xhafa, "Performance analysis of olsr and batman protocols considering link quality parameter", in *2009 International Conference on Advanced Information Networking and Applications*, IEEE, 2009, pp. 307–314.

[22] D. Seither, A. König, and M. Hollick, "Routing performance of wireless mesh networks: A practical evaluation of batman advanced", in *2011 IEEE 36th Conference on Local Computer Networks*, IEEE, 2011, pp. 897–904.

[23] M. Zorzi and R. R. Rao, "Geographic random forwarding (geraf) for ad hoc and sensor networks: Energy and latency performance", *IEEE transactions on Mobile Computing*, vol. 2, no. 4, pp. 349–365, 2003.

[24] S. Kuhlmorgen, I. Llatser, A. Festag, and G. Fettweis, "Performance evaluation of etsi geonetworking for vehicular ad hoc networks", in *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, IEEE, 2015, pp. 1–6.

[25] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach", in *Advances in neural information processing systems*, 1994, pp. 671–678.

[26] S. P. Choi and D.-Y. Yeung, "Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control", in *Advances in Neural Information Processing Systems*, 1996, pp. 945–951.

[27] S. Kumar and R. Miikkulainen, "Dual reinforcement q-routing: An on-line adaptive routing algorithm", in *Proceedings of the artificial neural networks in engineering Conference*, 1997, pp. 231–238.

[28] ——, "Confidence-based q-routing: An on-line adaptive network routing algorithm", in *Proceedings of Artificial Neural Networks in Engineering*, 1998.

[29] ——, "Confidence based dual reinforcement q-routing: An adaptive online network routing algorithm", in *IJCAI*, Citeseer, vol. 99, 1999, pp. 758–763.

[30] L. Peshkin and V. Savova, "Reinforcement learning for adaptive routing", in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, IEEE, vol. 2, 2002, pp. 1825–1830.

[31] A. A. Bhorkar, M. Naghshvar, T. Javidi, and B. D. Rao, "Adaptive opportunistic routing for wireless ad hoc networks", *IEEE/ACM Transactions On Networking*, vol. 20, no. 1, pp. 243–256, 2011.

[32] K. Tang, C. Li, H. Xiong, J. Zou, and P. Frossard, "Reinforcement learning-based opportunistic routing for live video streaming over multi-hop wireless networks", in *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, IEEE, 2017, pp. 1–6.

[33] A. Ghaffari, "Real-time routing algorithm for mobile ad hoc networks using reinforcement learning and heuristic algorithms", *Wireless Networks*, vol. 23, no. 3, pp. 703–714, 2017.

[34] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization", *arXiv preprint*, 2017. [Online]. Available: `https://arxiv.org/pdf/1709.07080.pdf`.

[35] C. Yu, J. Lan, Z. Guo, and Y. Hu, "Drom: Optimizing the routing in software-defined networks with deep reinforcement learning", *IEEE Access*, vol. 6, pp. 64 533–64 539, 2018.

[36] X. You, X. Li, Y. Xu, H. Feng, and J. Zhao, "Toward packet routing with fully-distributed multi-agent deep reinforcement learning", *arXiv preprint*, 2019. [Online]. Available: `https://arxiv.org/pdf/1905.03494.pdf`.

[37] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: `http://incompleteideas.net/book/the-book-2nd.html`.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning", *arXiv preprint arXiv:1312.5602*, 2013.

[39] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning", in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[40] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning", *arXiv preprint arXiv:1509.02971*, 2015.

[41] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor", *arXiv preprint arXiv:1801.01290*, 2018.

[42] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications", *arXiv preprint arXiv:1812.05905*, 2018.

[43] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning", *PloS one*, vol. 12, no. 4, e0172395, 2017.

[44] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments", in *Advances in Neural Information Processing Systems*, 2017, pp. 6379–6390.

[45] C. Riquelme, G. Tucker, and J. Snoek, "Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling", *arXiv preprint arXiv:1802.09127*, 2018.

[46] M. Collier and H. U. Llorens, "Deep contextual multi-armed bandits", 2018. [Online]. Available: `http://arxiv.org/abs/1807.09809`.

[47] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning", in *international conference on machine learning*, 2016, pp. 1050–1059.

[48] Y. Gal, J. Hron, and A. Kendall, "Concrete dropout", in *Advances in Neural Information Processing Systems*, 2017, pp. 3581–3590.

[49] K. Verbeeck, A. Nowé, T. Lenaerts, and J. Parent, "Learning to reach the pareto optimal nash equilibrium as a team", in *Australian Joint Conference on Artificial Intelligence*, Springer, 2002, pp. 407–418.

[50] A. I. Grohmann, "Optimizing guaranteed latency in dynamic wireless mesh networks", Diploma Thesis, Technische Universität Dresden, Mar. 2019.