Vrije Universiteit Brussel

FACULTY OF SCIENCE AND BIO-ENGINEERING SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Machine Learning for Wireless Sensor Networks

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

Maarten Devillé

Promotor: Prof. Dr. Ann Nowé
Supervisor: Dr. Yann-Aël Le Borgne

Academic year 2010 - 2011

Vrije Universiteit Brussel

FACULTEIT WETENSCHAPPEN EN BIO-INGENIEURSWETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN

Machine Leertechnieken voor Draadloze Sensor Netwerken

Proefschrift ingediend met het oog op het behalen van de graad van
Master in de Ingenieurswetenschappen: Computerwetenschappen

Maarten Devillé

Promoter: Prof. Dr. Ann Nowé
Begeleider: Dr. Yann-Aël Le Borgne

Academiejaar 2010 - 2011

Abstract

A Wireless Sensor Network (WSN) is a network of distributed sensor nodes which are capable of monitoring their environment through sensors. Because of the various types of sensors that are available, WSNs have a wide field of applications, each with their own characteristics and requirements. Although WSN platforms are the subject of constant evolution, most sensor nodes remain reliant on a limited capacity power source. This limitation requires WSN software to be economical with available resources, as applications often require sensor nodes to be located in remote locations, making battery replacement expensive or even impossible.

The wide field of available applications for WSNs means that developing algorithms which can run on sensor nodes and which are suitable for multiple applications becomes ever more challenging. However, one constraint sensor nodes are likely to face in most scenarios is a limited power source. Furthermore, because sensor nodes are mostly located in extreme locations, they often form very irregularly connected network topologies. To ensure a satisfying network performances a routing algorithm has to be able to cope with these irregular topologies, while conserving energy whenever possible in order to maximise the network's lifetime.

Reinforcement Learning (RL) is a machine learning variation that enables an agent to learn which actions it can take when it is executing a task in order to maximise a long-term reward. The principles from RL have already been applied to the routing problem, resulting in an algorithm that delivers good network performance in wired networks by using the communication delay across the network as a metric for network performance. A RL based routing algorithm requires little information about its environment and it is able to adjust its routing behaviour to dynamical conditions during the network's lifetime. Because of this RL based routing algorithms are very suitable for WSNs and recent work has applied some of the existing RL routing principles on WSNs by using the energy of sensor nodes as a metric.

In this thesis we propose two RL based routing algorithms for WSNs. The first algorithm is able to significantly improve the lifetime of a WSN in certain scenarios by propagating a sensor node's energy information throughout the network. The second algorithm combines both metrics for energy and message delay into a single algorithm, resulting in a routing policy able to balance the network performance and network lifetime.

Abstract

Een draadloos sensor network (DSN) bestaat uit een netwerk van gedistribueerde toestellen die in staat zijn hun omgeving waar te nemen aan de hand van sensoren. Gezien de variatie van beschikbare sensoren bestaan er voor deze netwerken een groot aantal toepassingen, elk met hun eigen eigenschappen en vereisten. Alhoewel er in het DSN domein een constante evolutie aan de gang is van verbeteringen in hard- en software platformen blijven de sensor toestellen afhankelijk van een beperkte stroombron. Deze beperking zorgt ervoor dat software ontwikkeld voor DSN platformen zuinig moet omspringen met de beschikbare middelen, aangezien de verschillende toepassingen vaak vereisen dat sensor toestellen worden geplaatst in afgelegen en moeilijk bereikbare plaatsen, wat het vervangen van de stroombron vaak duur of zelfs onmogelijk maakt.

Door de grote variatie aan toepassingen voor DSN platformen wordt het steeds moeilijker om algoritmen te ontwikkelen die zowel geschikt zijn voor uitvoering op sensor toestellen als voor gebruik in verschillende toepassingen. De voornaamste beperking in de meeste toepassingen is echter de beperkte stroombron. Aangezien de sensor toestellen vaak op bijzondere locaties worden geplaatst vormen ze meestal zeer onregelmatig opgebouwde netwerken. Om goede netwerk prestaties te waarborgen moet een routing algoritme in staat zijn om hiermee om te gaan terwijl het, waar mogelijk, energie bespaart om de autonomie van het netwerk te maximaliseren.

Reinforcement Learning (RL) is een machine leertechniek die een agent in staat stelt om te leren welke acties het interessantst zijn om tijdens het uitvoeren van een opdracht een zo hoog mogelijke beloning te verzamelen. De principes van RL zijn reeds toegepast in netwerk routing algoritmes. Dit resulteerde in een algoritme dat goede netwerk prestaties levert door de vertraging te meten die een data pakket kan oplopen tijdens het doorsturen tussen verschillende toestellen. Aangezien RL algoritmen weinig informatie over hun omgeving nodig hebben en ze hun routing beleid kunnen blijven aanpassen aan dynamische omstandigheden zijn ze zeer geschikt voor gebruik in DSN platformen. In recent onderzoek zijn bepaalde RL technieken reeds toegepast op een DSN platform in combinatie met metingen van de energiewaarden als prestatiemaatstaf.

In dit proefschrift stellen we twee nieuwe op RL gebaseerde routing algoritmen voor. Het eerste algoritme is in staat om de levensduur van een DSN significant te verbeteren in bepaalde toepassingen dankzij het delen van energie informatie over het hele netwerk. Het tweede algoritme combineert prestatiemaatstaven voor energie en vertragingen tijdens het verzenden van berichten om zo een routing beleid te kunnen voeren dat een evenwicht levert tussen netwerk prestaties en de levensduur van het netwerk.

Acknowledgements

After a whole year of reading, developing, testing and writing I am very grateful that I was able to work with my supervisor Dr. Yann-Aël Le Borgne. From the first day I got interested in the topic up until the day of the thesis deadline, I always received a great amount of time and attention. Thanks for providing me with TelosB hardware to acquire some real word experience with sensor networks, being able to work with a real WSN platform really motivated me. But also thanks for guiding me through the amounts of literature and especially, thank you for reading and reviewing this document on multiple occasions throughout the year.

I would also like to thank my thesis promotor, Prof. Dr. Ann Nowé, for the guidance during the research and for providing valuable ideas for the future work. Also thank you for taking the time to review this document.

After two years in the computer science master programme, I also would like to thank some of my fellow students. Kevin Van Vaerenbergh who has been a long time friend and with who I partnered up for many assignments. The rush of an approaching deadline was so much more bearable when we were able to joke about it together. Besides having a personal friend I also feel lucky to have great friends in the Artificial Intelligence department. We may be a small group but attending class together was always a pleasure, as well as working together on multiple occasions. Bart, Christophe and Tim are all great guys and I can't imagine having a better atmosphere in a class room than the one we had.

A final thank you goes to Charlotte Van Loo for reviewing the Dutch abstract.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Wireless Sensor Networks

Wireless sensor networks were originally conceived for military applications like battlefield monitoring. But today these networks have many purposes in both industry and consumer applications. Applications like environmental monitoring, industrial process control and machine health monitoring are just a few examples of areas where wireless sensor networks are being put to use [Tiwari et al., 2007]. The growth of applications and the constant evolution of cheaper and more capable hardware (see figure 1.1) make wireless sensor networks an interesting domain with a lot of potential research opportunities.

A wireless sensor network is typically composed of multiple autonomous wireless sensor nodes. These nodes gather data about their environment and work together to forward their data to centralised locations called base stations or sinks. The sensor nodes are equipped with various sensors that allow them to monitor their environment. The type and amount of sensors that are available on the sensor nodes depends on the application. Some of the common examples are sensors for measuring temperature, humidity, movement, visible light, and so forth. [Akyildiz et al., 2002, Römer and Mattern, 2004]. The data that each node gathers from its sensors is passed along through the wireless network to special nodes that are responsible for collecting data. These nodes usually do not perform any measurements themselves and are called base stations or sink nodes. The sink nodes forward the collected data to a different system that they are connected to. This can be a directly connected computer or a device on a second network interface that the sink could be connected to. This usage scenario is often also referred to as data collection. An example overview of a data collection set-up is shown in figure 1.2.

Besides a set of sensors, each node features a micro controller, a wireless antenna and a battery power source. The micro controller integrates a low-power processor core, program memory, RAM memory and an input / output interface for the wireless antenna. Because of their structure these sensor nodes seemingly resemble modern day computers, but it is important to keep in mind that the capabilities of these nodes are of a much smaller magnitude in terms of performance. Processors work at clock cycles of only a few megahertz and the amount of available program and RAM memory is typically less then 100 kilobytes [Langendoen, 2007]. This limited hardware however allows sensor nodes to achieve acceptable lifetimes on regular batteries. We illustrate this with an example of the TelosB sensor node.

This sensor node was introduced in 2005 and is equipped with a 16-bit 8 MHz processor,

	René 1999	Mica-2 2002	Tmote Sky 2005	Imote2 2007
CPU	**ATMEL 8535** 8-bit, 4 MHz 36 μW sleep 60 mW active	**ATmega128L** 8-bit, 8 MHz 36 μW sleep 60 mW active	**TI MSP430** 16-bit, 8 MHz 15 μW sleep 5.4 mW active	**Intel PXA271** 32-bit, 13-416 MHz 390 μW sleep $\geq$ 31 mW active
Memory	512 B RAM 8 KB Flash	4 KB RAM 128 KB Flash	10 KB RAM 48 KB Flash	32 MB RAM 32 MB Flash
Radio	**RFM TR1000** 10 Kbps 2 μW sleep 12 mW receive 36 mW xmit 0.5 ms setup	**CC1000** 76 Kbps 100 μW sleep 36 mW receive 75 mW xmit 2 ms setup	**CC2420** 250 Kbps 60 μW sleep 63 mW receive 57 mW xmit 1 ms setup	

Figure 1.1: Evolution of wireless sensor network platforms over time. Taken from [Langendoen, 2007]. When minimal energy consumption is a priority, the Tmote platform still is the most interesting platform to date.

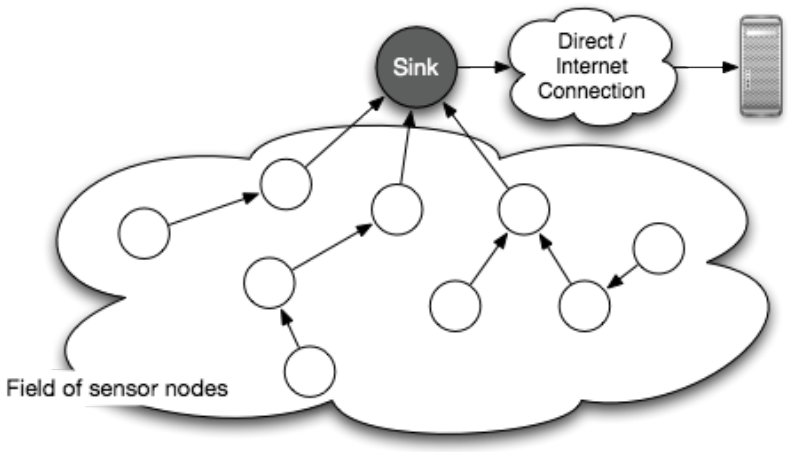

Figure 1.2: Overview of a Wireless Sensor Network topology. Regular nodes shown in white. The sink node is shown in blue and forwards its gathered data to the connected computer.

10KB RAM, 48KB Flash and a 250 Kbps wireless radio. When it is equipped with a single 3000 mAh AA battery this node can achieve up to 95 hours of continuous operation [Nguyen et al., 2011]. In order to further increase the network lifetime most usage scenarios in wireless sensor networks use low duty cycles in which nodes are active only at predefined time intervals. This means that the nodes will spend most of their lifetime in a sleep state. In this sleep state, nodes draw only a minimal amount of power from their batteries. A node will only go to its fully powered active state at certain time intervals to gather sensor readings and to forward these sensor readings through the network. When we take into account that most sensor networks have a duty cycle of less then 5% [Langendoen, 2007] it becomes clear that these nodes can operate for several weeks or even months without requiring maintenance.

1.2 Routing Challenges

Wireless sensor networks raise several issues related to network communication. Because of the limited range of the wireless signal, traffic will mostly have to travel along multiple hops before reaching its destination (a sink node). But in order to increase their lifetime, nodes will spend most of their time in a sleep state, making them unavailable to forward other data. This means that some paths through the network might not be available at all times. The possibilities of a hardware malfunction or a drained battery also increase the odds of a changing network topology. The combination of limited wireless antenna range and uncertainty about a node's availability means that Wireless Sensor Networks tend to have irregularly connected topologies that change dynamically during the network's lifetime.

The multi-hop routing in wireless sensor networks is possible because sensor nodes tend to be densely deployed near an area of interest (also called phenomenon in [Akyildiz et al., 2002]). The exact positions of nodes however are not always determined before deployment. This allows for rapid deployment of networks in extreme terrains, but it also contributes to the irregular nature of network topologies. The combination of limited information before deployment and irregular topologies also means that sensor networks should posses self-organising abilities.

Another issue that needs to be considered is the impact of using the wireless radio on the lifetime of a node. Since using the wireless radio is a big drain on a node's battery life [Langendoen, 2007], using it too often can result in heavily reduced lifetime. So if one node needs to forward considerably more packets than other nodes its lifetime will be far below the average lifetime of other nodes. This can have a serious impact on general network performance, since a lot of data will now have to be re-routed along other paths. A big difference in lifetime between nodes can also be undesirable when multiple nodes in critical locations suffer from this problem. This would require more regular maintenance which can be very costly in some scenarios (eg. sensors located in remote or dangerous locations [Tiwari et al., 2007]).

These issues are recognised by Akyildiz et al. in their survey on wireless sensor networks [Akyildiz et al., 2002]. In their survey they propose a model for a protocol stack for wireless sensor networks. This stack is shown in figure 1.3 and consists of five layers which are from top to bottom; the application layer, the transport layer, the network layer, the data link layer and the physical layer. The problem of routing data, which is the subject of this master thesis, is part of the network layer. Readers with experience in data networks might notice that this stack heavily resembles the OSI model protocol stack [Zimmermann, 1988]. It is

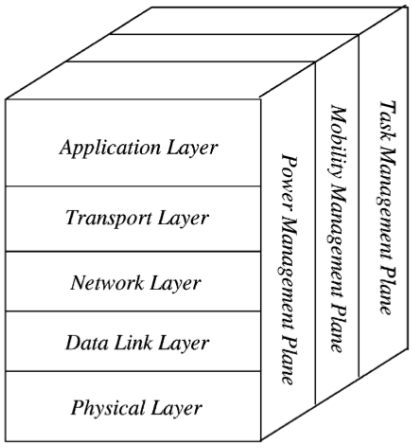

Figure 1.3: A protocol stack for sensor networks, as designed in [Akyildiz et al., 2002].

important however to note that at each layer, protocols need to take into account three domain specific constraints for wireless sensor networks. These constraints concern power management, mobility management and task management.

As general guidelines Akyildiz et al. suggested that protocols at the network layer should take the following principles into account for their design:

- Power efficiency is always an important consideration.
- Sensor networks are often data-centric.
- Data aggregation is only useful when it does not hinder the collaborative behaviour of other nodes.
- An ideal sensor network has attribute-based addressing and location awareness.

The scenario we are looking at is limited purely to the routing of data from sensors nodes to a sink and assumes communication through the network to be one way from nodes to the sink, so we do not need to worry about a data-centric approach [Krishnamachari et al.,]. The fourth item from their guideline is somewhat out of reach for our research, as even in todays modern sensor network hardware location awareness almost always comes at a great energy cost. Furthermore our experiments are focussed on non-mobile sensor networks.

1.2.1 Conflicting optimisation goals

There are multiple ways we can measure a network's performance and energy management. When considering energy management, we could measure the average lifetime of all nodes, the lifetime of the node that depletes its energy in the least amount of time or the amount of time before the final node goes down. Several scenarios can have different requirements, but for the data collection scenario which is the subject in this master thesis, we decide that the most important requirement is to postpone the time when the first node dies for as long as possible, because as soon as nodes start to die, we increase the odds of disconnecting parts of the network [Forster and Murphy, 2008].

As soon as the first node dies, this has a significant impact on the network, since the first node to die is very likely to be important in the routing of data generated by other

nodes. So all the data that was routed by this node will now have to be rerouted along other nodes. Furthermore if there is a large time gap between the moment the first few nodes go offline and the moment the majority of other nodes go offline this will mean that the amount of gathered data will drop at an earlier moment in the network lifetime. If this data is critical then the network will need more maintenance and as we mentioned earlier in this chapter, replacing nodes in a wireless sensor network can be difficult and in some instances it is even impossible.

Besides increasing the network lifetime, we still want to guarantee a certain level of performance. When we talk about performance in networks, we usually measure it as the latency for communication between nodes. This is the amount of time that passed between the moment when a message is sent at its source and the moment when a message is received at its destination. So when we evaluate the performance of a network we want to minimise the latency between nodes. In the data collection scenario that we investigate, all messages will always have the same destination, so we want to minimise the latency between all nodes and the sink. This is usually done by exploiting the best paths to the sink that are available within the network.

The problem this causes is that the exploitation of fastest paths increases energy consumption for the nodes along these routes. This can have a negative impact on the general lifetime of the network. So it is very important to establish clear goals before choosing or designing a routing algorithm. Achieving both the best network performance and the best lifetime with a single routing policy is impossible because of the conflicting nature of these requirements.

Even after we establish clear goals for our routing algorithm in terms of desired network lifetime and network performance, there remain multiple ways for achieving these goals. We will further discuss the impact of these conflicting goals on the design of a routing algorithm in section 2.2.

The routing problem is a subject of ongoing research in the field of machine learning. In the following section we will present an overview of protocols and frameworks that can be used to route data in networks. Some of these protocols are developed specifically for wireless sensor network applications and while others are not designed with sensor networks in mind they might still be useful in the field.

1.3 Related Work

This section summarises existing work in the field of both routing algorithms for Wireless Sensor Networks and Reinforcement Learning techniques for routing algorithms. Subsequently, this section is divided into two parts. In the first part we discuss some general routing algorithms as presented in the Survey on Wireless Sensor Networks [Akyildiz et al., 2002]. Next we discuss existing routing algorithms based on Reinforcement Learning techniques.

1.3.1 Routing in WSNs

In their survey on wireless sensor networks, [Akyildiz et al., 2002] presented an overview of protocols that were available or under development for different layers of the sensor networks protocol stack. We will use the remainder of this section to discuss some of the protocols that are available for the network layer of the sensor network protocol stack.

Small minimum energy communication network (SMECN) This approach is proposed by [Li and Halpern, 2001] and works by computing an energy efficient subnetwork for a given communication network. The algorithm works by taking into account energy costs for different communication links when it attempts to construct a minimum energy graph.

Flooding An old technique that has been used in many networks and that is also applicable on WSNs [Sohrabi et al., 2000]. Each node that receives a message broadcasts it to all of its neighbours, increasing the messages hop count. This process is continued until the message expires after reaching the maximal amount of hops. Over the years this techniques has shown many shortcomings. In densely connected networks messages can cause implosions of traffic, because two nodes with a lot of common neighbours may receive each others messages once for every neighbour they broadcasted too, causing for a lot duplicated messages and a lot of wasted resources.

Gossiping This technique is a variation on Flooding that attempts to avoid the implosion of duplicated messages by sending each message only to one randomly selected neighbour instead of broadcasting it to all neighbours [Hedetniemi et al., 1988]. While this approach works to conserve resources, it can also be very slow to deliver messages.

Sensor protocols for information via negotiation (SPIN) In [Heinzelman et al., 1999] the authors propose a collection of adaptive protocols that are designed to overcome the shortcomings of classic algorithms such as flooding. The algorithms are based on two main principles; Sensor nodes can operate more efficiently by sending information about their sensor data instead of the actual sensor data and sensor nodes should monitor changes in their available resources, e.g., energy level, available bandwidth etc.

The SPIN system is based on data-centric routing [Krishnamachari et al.,], this means that sensors periodically broadcast an availability of updated data and wait for a request from a sink before sending any actual data.

Sequential assignment routing (SAR) In [Sohrabi et al., 2000] a set of algorithms are proposed. These algorithms handle the organisation, management and mobility management operations in a sensor networks. Self-organising MAC for sensor networks (SMACS) is a distributed protocol that enables a collection of sensor nodes to discover their neighbours and establish communication schedules without the need for centralised coordination. The eavesdrop and register (EAR) algorithm is designed to support seamless interconnection of the mobile nodes. The EAR algorithm is based on invitation messages and on the registration of stationary nodes by the mobile nodes.

The SAR algorithm creates multiple trees where the root of each tree is an one hop neighbour from the sink. Each tree grows outward from the sink while avoiding nodes with a low performance (i.e., low throughput/high de- lay) and low energy reserves. At the end of this procedure, most nodes belong to multiple trees. This allows a sensor node to choose a tree to relay its information back to the sink. There are two parameters associated with each path, i.e., a tree, back to the sink:

- Energy resources: The energy resources is estimated by the number of packets, which the sensor node can send, if the sensor node has exclusive use of the path.
- Additive QoS metric: A high additive QoS metric means low QoS.

The SAR algorithm selects the path based on the energy resources and additive QoS metric of each path, and the packet's priority level. As a result, each sensor node selects its path to route the data back to the sink. Also, two more algorithms called single winner election and multi-winner election handle the necessary signalling and data transfer tasks in local cooperative information processing.

Directed diffusion (DD) The directed diffusion data dissemination paradigm is proposed in [Intanagonwiwat et al., 2003]. Directed diffusion is a data-centric routing protocol, this means that it requires the sink in a WSN to broadcast an interest for a certain type of data (this is called a query). This interest is propagated through the network to all sensors. All sensors remember the type of information that is queried before forwarding the query further into the network. When the query is propagated, all nodes also update the query with a timestamp and a gradient, nodes receiving a query store this gradient information in a local cache before updating the message. This way the query actually builds the routing tree, so when a sensor node has information that is of interest to the sink, it knows where to send the information. In order to achieve a good routing behaviour the sink needs to refresh and reinforce the interest as soon as it starts to receive data from a source.

1.3.2 Reinforcement Learning in Routing

This section briefly covers some applications of Reinforcement Learning principles in routing algorithms. We focus on the two algorithms that are detailed further in this thesis (Q-routing and FROMS) and two other algorithms that have achieved success in routing algorithms but that we deem less interesting for applications in wireless sensor networks.

Q-routing In "Packet routing in dynamically changing networks" [Boyan and Littman, 1994] the authors showed that reinforcement learning can be used to achieve better network performance when there is a lot of network traffic, especially in networks with irregular topologies. While the main concern for developing the Q-routing algorithm was network performance, the algorithm achieves its results by distributing traffic across the network. This can also be an advantage when we try to improve the average lifetime of a wireless sensor network, since we reduce the likeliness that one node will have to process considerable more packets then other nodes. The work from [Boyan and Littman, 1994] has served as a starting point for a lot of related research and we discuss it in more detail in section 2.4.

FROMS In [Forster and Murphy, 2008] the authors already successfully used Q-routing principles in wireless sensor networks, however they have done this as an extension to their own developed FROMS framework. This is a framework that is developed specifically for a multi-source and multi-sink usage scenario's. The idea of this thesis is to develop a general routing algorithm for wireless sensor networks that can serve in multiple general scenarios. For this reason, we will attempt to apply some of the ideas from [Forster and Murphy, 2008] on a more general routing algorithm and attempt to combine them with other metrics. The most essential information about the authors' approach is detailed in section 3.3.3.

GRE-DD In [Zhiyu and Haoshan, 2007] the authors propose an extension of a Direct Diffusion algorithm [Intanagonwiwat et al., 2003], which considers energy information from

local neighbors when generating gradients to the sink. A drawback of Direct Diffusion algorithms is that they only update gradients during sink floodings, which are only performed at certain intervals to save energy. So even if a node exhausts its energy, its neighbors will not update their gradient until the next flooding.

Swarm Intelligence The term Swarm Intelligence does not belong to a single algorithm, instead it refers to a class of machine learning techniques [Eberhart et al., 2001]. These algorithms are all biologically inspired by the behaviour of insects. The main principle of the algorithm is distribution, a system is composed out of many individual agents, but these agents all possess very limited computational capabilities. The agents are able to communicate with each other through a shared environment, this is inspired on the way ants use pheromone trails. This shared communication allows the agents to form a cooperative behaviour. Swarm intelligence algorithms are very suitable for scenarios where coping with mobility and topology changes is of great importance and they have already been applied in routing algorithms [Kassabalidis et al.,]. However most current algorithms require a significant amount of available energy and since the focus of this thesis is on non mobile networks with very limited energy, current Swarm Intelligence algorithms are not suitable for achieving a maximal network lifetime.

In the data link layer of the protocol stack (Medium Access Control), [Liu and Elhanany, 2006] showed that reinforcement learning techniques can be used to improve a nodes energy efficiency by limiting its access to the antenna while still achieving acceptable network performance. Mihaylov et al. [Mihaylov et al., 2011] showed that reinforcement techniques can also be used to achieve synchronising and desynchronising behaviour between nodes. This results in less traffic collision and less overhearing, lowering the traffic overhead and the amount of packets that need to be transmitted.

1.3.3 Summary

Most of the algorithms that we discussed in this section use energy level as a metric, but when it comes to packet routing most of these algorithms only use a hop distance or link quality as a metric. Indeed a lot of algorithms currently available for Wireless Sensor Networks make their routing decisions based solely on information about the link quality between nodes [Fonseca et al., 2006, Gnawali, 2006, Levis et al., 2004]. While this approach works to guarantee the most reliable packet deliveries, it does this by exploiting the best path to a sink. This approach however is prone to bottlenecks, a problem that is highly prevalent in irregular topologies, which can result in both poor network and lifetime performance. We can try to avoid these problems by modifying algorithms in order to take into account extra metrics besides the link quality, such as delivery time, and attempt to combine these network performance metrics with energy levels. A lot of research on this topic has already been performed in other fields of network communication. In this thesis we will investigate the Q-routing algorithm for packet switched networks by Boyan and Littman [Boyan and Littman, 1994] and a variation from [Forster and Murphy, 2008] that has been developed for WSNs specifically.

1.4 Outline

1.4.1 Contributions

The aim of this master thesis is to apply reinforcement learning techniques to a packet routing algorithm in order to increase the lifetime of a sensor network while maintaining an acceptable performance level in terms of packet latency. Although there has already been some research in the field of network routing for sensor networks, most algorithms still struggle to achieve a good balance between network lifetime and network performance. To address these issues we highlight four distinct contributions in this document.

First we compare existing Reinforcement Learning based routing algorithms. We show that by combining Q-routing with a delay based metric from [Boyan and Littman, 1994] and Q-routing with an energy based metric from [Forster, 2007], it is possible to design an algorithm that achieves a well balanced routing policy. This balanced routing policy is able to extend a networks lifetime beyond the lifetime of a regular Q-routing algorithm while still achieving a high level off network performance, effectively offering a trade-off between the advantages of two different Q-routing algorithms.

Besides combing two metrics into a single algorithm, we also developed a modified version of the algorithm proposed in [Forster, 2007]. In the original algorithm, a node's energy information is never shared beyond its adjacent nodes. We proposed a new version for this algorithm that allows the energy information of any node to be propagated throughout the network. The modifications we made to the original algorithm allow for a different energy aware routing behaviour that we call "lowest energy on path routing". Our experiments show that in certain scenarios this can result in significant improvements of the network's lifetime. We also combined this modified algorithm with the delay based metric from the original Q-routing in [Boyan and Littman, 1994]. This combined algorithm delivers comparable network performance and network lifetime results on most topologies, and in some more complex and irregular network topologies it is able to deliver a significantly better performance than the algorithm that combines the original energy metric with the delay based metric.

Our final experiments incorporate optimisations techniques and basic Quality of Service features into our adapted algorithm. Our experiments show that these optimisations are able to further improve the network lifetime.

Finally we will show that there still exists a research opportunity for dynamic routing algorithms that can deal with QoS in the dynamic environment of WSNs.

1.4.2 Document structure

This thesis consists of eight chapters. Each chapter starts of with a small introduction that describes it's content to the reader. Chapter 2 briefly details general properties of the routing problem and the WSN specific constraint of a limited energy power source. After examining the routing problem we investigate the potential of Reinforcement Learning for WSNs by further detailing some existing work from [Boyan and Littman, 1994].

Chapter 3 discusses the different elements from the Q-routing algorithm detailed in chapter 2. We discuss how different behaviours and metrics can be suitable for different scenarios and how they can impact network performance and lifetime.

Chapter 4 proposes the adjustments that we made to existing algorithms in an attempt to develop an algorithm that is able to achieve even better network lifetimes while being

suitable for a larger range of usage scenarios.

Chapter 5 provides essential information about the simulation environment and about the variables which can be modified. Furthermore it also highlights the different topologies that have been used during experiments together with their characteristics that motivated us to select them for the experiments.

The results from the performed experiments are provided in chapter 6 and we discuss our findings from these experiments in chapter 7.

Finally the most interesting findings and contributions from this master thesis are summarised in chapter 8 together with some possibilities for future research based on the work performed for this master thesis.

1.5 Summary

This chapter introduced wireless sensor networks. Wireless sensor networks are distributed networks composed from autonomous sensor nodes which are able to gather different types of data from their environment, depending on the type and amount of sensors that are available on the hardware. The data that is gathered by sensor nodes is forwarded through the network to a special type of node whose sole purpose is to collect data. This node is called a base station or sink. Wireless sensor networks pose a challenge for routing algorithms because they often form irregular connected topologies, requiring elaborate routing strategies. Furthermore the hardware of a sensor node is powered by a limited capacity power source requiring software to be economical with the available resources. This requires a routing algorithm to economise power while still delivering an acceptable network performance. These requirements seem to conflict with each other, as delivering a high level of network performance is usually achieved by exploiting the best paths in the network, resulting in a fast depletion of the resources along the best paths.

This chapter also introduced some of the related work in routing algorithms and reinforcement learning algorithms. We provided a motivation for the usability of these algorithms and also introduced Q-routing and the FROMS framework. The Q-routing algorithm uses a delay based metric to achieve a high level of network performance in wired networks and has served as the basis of multiple routing protocols. The FROMS framework has been developed specifically for wireless sensor networks and offers many algorithms. However, we focus on the energy aware Q-routing algorithm in FROMS. The algorithm uses a nodes' energy level as a metric in the routing policy. This allows the routing algorithm to distribute traffic based on the amount of energy available at different nodes, extending the network's lifetime. These algorithms will serve as the start of our original work and are discussed in more detail in the following chapters.

Chapter 2

Reinforcement Learning in network routing

2.1 Introduction

This chapters gives some essential information about the routing problem. It provides details on some of the general concepts as well as on some of the more specific properties that are typical for wireless sensor networks. Section 2.2 introduces the routing problem and some concerns about energy aware routing. Section 2.3 introduces general reinforcement learning concepts and Q-learning, one of many reinforcement learning variations. Section 2.4 introduces Q-routing, which applies Q-learning techniques to a network routing algorithm. Section 2.5 evaluates the Q-routing algorithm in the context of wireless sensor networks. Section 2.6 reviews the performance criteria that will be used to evaluate an algorithms performance when dealing with the routing problem.

2.2 Routing problem

2.2.1 Introduction

Most of today's networks are not fully connected networks. This means that not all nodes in the network are connected directly to each other.

Let each node be identified by a unique integer, We define the set of nodes in a network as:

$S = \{1, 2, ..., n\}$, with node 1 representing the sink of the network.

Next lets consider all the nodes that a node i can directly communicate with. These are all the nodes located within the transmission range of node i. We call this set of nodes the neighbourhood of node i and represent it as $\mathcal{N}_i$, $i \in S$. Please note that we assume all links to be directional, so $j \in \mathcal{N}_i \Leftrightarrow i \in \mathcal{N}_j$.

In order to communicate between nodes that are not directly connected, packets will have to 'hop' along multiple nodes that are located on a path between these nodes, hence the name 'multi-hop networks'. A path P from node i to the sink is a sequence of pairwise adjacent nodes that starts at node i and ends at node 1, the sink of the network.

$P = \{p_0, p_1, ..., p_n\}, p_0 = \text{node } i,\ p_n = \text{node 1 and } p_x \in \mathcal{N}_{x-1}$

In order to find paths between non-adjacent nodes, there exist a number of routing algorithms. These routing algorithms need some sorts of metric to compare different paths so they are able to generate good paths. When generating a path, an algorithm can compare the links between a node i nodes and its adjacent nodes and assign a certain cost for using a link to communicate between two adjacent nodes. We define a function that represents the cost of communication between two adjacent nodes i and j at a certain moment in time t. Adding a variable for time allows for the cost of a link to vary over time.

$c(i, j, t), j \in \mathcal{N}_i$

The introduction of this cost function allows us to define a similar cost function for an entire path:

$$c(P, t) = \sum_{p_i \in P} c(p_i, p_{i+1}, t) \tag{2.1}$$

Now that we have established formal ways of representing a network and different paths within a network, lets take a closer look at how algorithms can use information to generate paths within a network.

2.2.2 Types of routing algorithms

The design of routing algorithms has a long history in the literature and there exist a number of classic routing algorithms for transmitting data packets across multi-hop networks [Tanenbaum, 2003]. The majority of these algorithms can be divided into two distinct classes, distance-vector routing protocols and link-state protocols. While these two classes mainly rely on the same metrics to achieve a good behaviour, they vary in the way that they propagate their information between different nodes in the network.

A typical characteristic of distance-vector protocols is that routers periodically need to inform their direct neighbours about changes in the topology whereas link-state protocols require routers to share changes in the topology with all nodes in the network. This means that routers using link-state protocols have full knowledge over the paths within a network. When using a distance-vector routing protocol, routers do not need to have total knowledge of paths to destinations in the network. Instead each router just knows which neighbours are on a path to which destination and how far they are located from each destination. The path that a packet will follow once it is beyond the direct neighbour is unknown for a router.

While link-state protocols have some advantages and are successfully being deployed for IPv4 and IPv6 protocols, they are more demanding in terms of computational complexity and generate a bigger message overhead[1]. This makes them less desirable candidates for use in wireless sensor networks. Remember that in the type of wireless sensor networks that we are looking at, all nodes are able to forward data packets from other nodes, and thus all nodes act as routers.

While both types of algorithms have different ways of sharing and storing data about the network, they both rely on variations of the Bellman-Ford algorithm to compute the shortest paths between nodes. This brings us to an important part of network routing, namely a way of determining the cost of a certain path in the network. We have already introduced the

notion of a cost function in this chapter and in the next section we will look at different metrics that can be used as a cost function when routing data in wireless sensor networks.

2.2.3 Cost function

Traditional algorithms measure the distance between nodes by counting the number of hops on a path between nodes. In wired networks with reliable connections this can be a good measure to find fast connections to destinations. The cost function for a link can then be described as the following constant:

$$\forall i, j \in S \,|\, j \in \mathcal{N}_i : c(i, j, t) = 1 \tag{2.2}$$

When using this cost function, the cost of a complete path can be calculated as in equation 2.1.

The problem with this metric however is that it gives no indication of the current state of a connection. So in order to avoid slow downs under high traffic, a number of other metrics are available. It is possible for algorithms to measure the available bandwidth on a link, as well as the amount of time a node needs to process a packet. These metrics give a more accurate representation of the networks state and help algorithms to cope with increasing amounts of traffic. Bandwidth and delay metrics are used in the IGRP protocol, which is developed by Cisco and used in modern internet routing equipment [Rutgers, 1991].

As discussed in section 1.2.1, when routing data in a wireless sensor network, we need to take into account that there exist two conflicting goals. On the one hand we want to maximise the lifetime of the network but on the other hand we still want to guarantee an acceptable network performance. The remainder of this section is divided into two parts that discuss metrics that can be used by a cost function to optimise either the network performance (latency) or the network lifetime (energy).

Network performance

Besides factors like bandwidth and the amount of time that is needed to switch a message at a hop, which can be measured by a delay based Q-routing algorithm, wireless sensor networks have another property that can greatly influence the network performance. This factor is called link reliability. Link reliability in wireless networks tends to be much less consistent than in regular wired networks. While wired communication is not totally free from noise and interference, these factors tend to have only a limited impact on performance. Radio communication however is much more sensitive to these factors and they can have a significant impact on general network performance. This is an even bigger problem in wireless sensor networks, since these networks do not have predetermined links between nodes. Instead deployed nodes will form communication links with all their neighbouring nodes, including those nodes situated at the edge of their communication range. This can result in very unreliable (also called 'lossy') connections between certain nodes, which can in turn result in poor network performance.

In order to provide a metric that accounts for a wireless link's reliability, authors in [Morris et al., 2004] introduced the expected transmission count (ETX) metric in 2004. The

[1] As detailed by Cisco, developer of multiple routing protocols. `http://docwiki.cisco.com/wiki/Routing_Basics#Link-State_Versus_Distance_Vector`

authors noticed that using an unreliable, lossy link can have the same impact on network performance as adding an extra link to a path. This is because the sender may need to resend the same packet multiple times. This takes more time and can also cause a delay for other messages in the network. So besides just trying to use the shorter routes, routing algorithms for wireless sensor networks should also try to minimise the use of lossy links. In an attempt to accomplish this, the ETX metric no longer measures the distance in a number of hops. Instead it measures the number of transmissions that are needed to deliver a packet. It is calculated by dividing the total number of transmission attempts, including retransmissions, by the number of transmitted packets. This means that for a perfectly reliable link, the ETX value will be 1.The ETX for a route is the sum of the ETX values for all links on the route. So the ETX metric for a 3-hop route with perfect link quality will be 3, while the ETX metric for a 2-hop route containing two links with 50% reliability will be 4 (ETX value of 2 for each link).

The cost function for a link when using ETX values would be:

$$c(i, j, t) = \frac{1}{d_f * d_r} \tag{2.3}$$

- d_f is the probability that a message from node i arrives at node j
- d_r the probability that an acknowledgement sent by node j arrives at node i
- d_f and d_r are obtained through samples gathered over a period of time w. So at time t, d_f and d_r are based on samples from between $t - w$ and t.
- The cost of a path can be calculated according to equation 2.1

Due to its simplicity, the ETX metric has been adopted by multiple sensor network platforms as the default metric for different routing algorithms,e.g., TinyOS platform for low power wireless devices [Fonseca et al., 2006, Gnawali, 2006].

Network lifetime

Throughout this thesis we will measure the network lifetime as the amount of time that passes until the first node in the network's topology dies. In order to maximise the network's lifetime we need a metric to represent the concept of lifetime. One piece of information that is available and that is explicitly related to the lifetime of the network is the amount of energy that is available on each node. This amount will change over time as the node consumes energy.

$e(i, t) = x, i \in S, x$: amount of energy still available.

Similarly we can define the total amount of energy available on a path P as:

$e(P, t) = \sum e(i, t), \forall i \in P$

This however, leads to a situation where longer paths will be more likely to have better energy values, making the use of this function as a metric limited.

Besides the amount of energy available, we can also couple the cost function $c(i, j, t)$ for a link between two nodes to an energy cost. In the example in figure 2.1, the amount of energy that a node needs to supply to its radio in order to communicate over a certain link is shown on the link as η. So for this example the energy cost of using a link between nodes i and j at time t would be: $c(i, j, t) = \eta$.

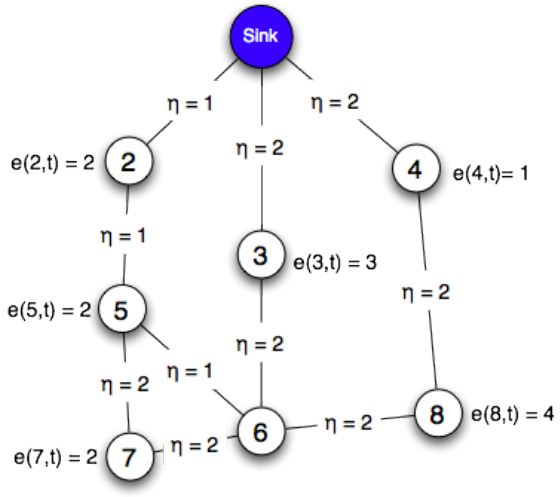

Figure 2.1: Example topology for different types of energy efficient routes. Based on an example from [Akyildiz et al., 2002].

Now that we have established some possible metrics for energy in a network, we need to find a way to find the best route given these metrics. But as we will illustrate through the following example, there are multiple possible approaches that can be used in an attempt to improve a networks lifetime.

The topology in figure 2.1 shows the energy cost of using different links and the amount of power available on different nodes. Assume we want to route data from node 6 to the sink. If we examine the topology we find that their are four possible paths that can be considered for this purpose:

P_1: 6 - 5 - 2 - Sink	$e(P_1) = 4$	$c(P_1, t) = 3$
P_2: 6 - 7 - 5 - 2 - Sink	$e(P_2) = 6$	$c(P_2, t) = 6$
P_3: 6 - 3 - Sink	$e(P_3) = 3$	$c(P_3, t) = 4$
P_4: 6 - 8 - 4 - Sink	$e(P_4) = 5$	$c(P_4, t) = 6$

Now based on the two metrics that we just introduced, i.e., energy available on a node and the energy cost for using a link, their are four distinct methods for selecting one of these routes:

Minimum hop route: The route that needs the least amount of hops is chosen. In our example this is P_3. This method will select the same routes as the minimum energy route method for scenarios where the energy cost of a link is unknown or where all links have the same fixed energy cost.

Maximum available power: This method prefers the route that has the highest amount of total power available. The maximum available power for a route is calculated by summing the PA values for all nodes on the route. When we apply this rule to the example we would select P_2. However, because P_2 is an extension of P_1, it is not an energy efficient route. So instead we select P_4 which has the second largest amount of maximum available power and does not extend any other routes.

Minimum energy route: The minimum energy route is the one that consumes the least energy. This route can be found by summating all alpha values on each route and selecting the route with the lowest summated value. In our example P_1 has the lowest summated alpha value.

Lowest energy on path: This method prefers the route along which the lowest available power for any node is higher then the lowest available power for any node on any other route. In our example this would be P_3. The main advantage of this method is that it decreases the probability of depleting one nodes energy early in the networks lifetime because it is located on a path that otherwise has a high amount of available power.

The lowest energy on path algorithm serves as an inspiration for one of our own algorithms, which is presented in section 4.3 of this document.

2.3 Reinforcement Learning

Reinforcement learning is a machine learning variation that allows an agent to learn which actions to take when executing a certain task in an environment while attempting to maximise a long-term reward received for completing the task. In this definition an agent is an autonomous entity which observes and acts upon an environment. The agent learns which actions to take by a process of trial and error. This process of trial and error requires the agent to explore his available actions. Whenever an agent tries an action, it will receive a reinforcement signal, or reward, that indicates how good or bad the selected action was. In the long run an agent will try to maximise selecting actions that resulted in high reinforcement signals while attempting to avoid actions that led to low reinforcement signals. This will result in the agents behaviour converging to the maximisation of the reinforcement function, also called the optimal policy. We can formalise this as follows:.

An agent is able to observe a state s from its environment. In each state the agent will select an action a from a set of available actions according to its policy π. When an agent selects an action a from a state s given a policy π we can define the reward or action-value for the state-action pair (s, a) as:

$$Q^{\pi}(s, a) = E[R \mid s, a, \pi] \tag{2.4}$$

E is the estimation of the total reward, with R being the reward after selecting an action. The estimation is initialised at random, afterwards it is updated with received rewards.

In reinforcement learning an agent decides which actions to take based on its policy. This policy, , π, maps a set of observed environment states to a set of actions. This policy is often initialised in a naive or random way, but the agent updates its policy every time it receives a reinforcement signal. The agent will update its policy in such a way as to maximise the reinforcement signals that it will receive in the future. If taking an action after a certain state is observed results in a positive state change and a positive reward then the probability for choosing that action in that state will be reinforced in the policy. If the results of taking an action are less positive the probability for choosing that action in that state will be lowered. An overview of the reinforcement learning model is available in figure 2.2.

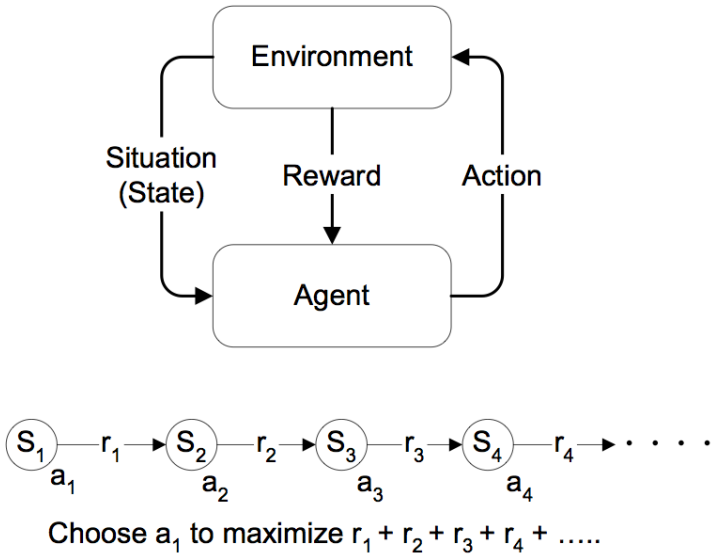

Figure 2.2: Overview of the reinforcement learning process, taken from [Fakir, 2004]. An agent in state S_1 will try to choose his action a_1 as to maximise the total reward $\sum_1^n(r_n)$.

The reinforcement learning technique has some properties that distinguish it from traditional supervised learning algorithms [Kaelbling et al., 1996]. The most important difference is that reinforcement learning algorithms do not need predetermined sets of input and output information. This is possible because reinforcement learning algorithms do not need to be told wether a selected action was good or bad. Instead the algorithm receives a reward and information on its state. Therefore the agent needs to gather useful experience about different environment states, actions and resulting rewards and transitions to different states in order to achieve an optimal behaviour. The main advantage of this approach is that it allows for online learning: the agent can learn to execute a task in an environment while it is already in that environment.

2.3.1 Q-learning

We briefly elaborate on Q-learning, which is one of several reinforcement learning techniques, because it is used as inspiration for the routing algorithm that is covered in the next section.

Q-learning works by learning an action-value function, Q, that represents the expected reward (or utility) of taking a certain action from a certain state, provided the policy is followed in subsequent actions [Sutton and Barto, 1998]. As described in the previous section on Reinforcement Learning, Q-learning works with an agent, a set of states S and a set of actions per state A. By performing an action $a \in A$ the agent can move between states. Each action is coupled with an immediate reward r. The simple form of Q-learning updates its action-value function as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2.5}$$

The action-value function for $Q(s_t, a_t)$ is updated with information the agent receives after selection action a_t, i.e., a direct reward r and an estimation of the maximal return (collection of future rewards) that can be achieved,$\max_a Q(s_{t+1}, a)$, provided the optimal policy is followed.

The importance of the future rewards can be influenced by a discount factor γ. A discount factor of 0 will cause an agent to act very opportunistic by only considering immediate rewards while a discount factor approaching 1 will result in an agent trying to maximise the long term reward.

The importance of newly acquired information can be modified by the learning rate α. Setting the learning rate to 0 will result in the agent discarding any new information and not learning anything, while a learning factor of 1 will result in the agent considering only new information and discarding older collected information.

The learned action-value function Q approximates the optimal solution Q^* as long as all state-action pair values continue to be updated. (i.e., the policy needs a sufficient exploring behaviour)

Cost minimisation The Q-learning algorithm provided in this section attempts to maximise a reward through an action-value function. However, for some scenarios it might be desirable to instead minimise an expected cost. This is equally possible in Q-learning by changing the maximum operator in formula 2.5 to a minimum operator.

2.4 Q-routing

Q-routing is an adaptive algorithm for packet routing in networks originally proposed in [Boyan and Littman, 1994]. The algorithm to achieves a balance between classical shortest path routing and load balancing to avoid network congestion along popular routes in the network. This is achieved by trying different routing policies and gathering statistics to compare which routing decisions result in better delivery times. An interesting characteristic of the algorithm is that each node only uses local communication with its direct neighbours to evaluate its actions. Because the algorithm is adaptive and online it is able to cope with varying network conditions like dynamic topologies and dynamic network loads. In this section we will formally describe the Q-routing algorithm as was originally proposed by Boyan and Littman.

The Q-routing algorithm uses an action-value function to minimise the delivery time of messages to their destination. Let $Q_i(d, j)$ be the time that node i estimates it will take for a packet to arrive at destination node d when it forwards the packet to its neighbouring node j. This estimate includes any delay time that the packet may experience at node i. Every node will store a local table with estimates for each pair of neighbours and destinations. When a routing decision needs to be made the default Q-routing policy will select the neighbour with the lowest Q-value for the destination of a message.

When a node i sends a packet with destination d to node j, node j will immediately respond to node i with its own estimate τ for the remaining travel time to destination d. The formula for obtaining a value for τ is shown in formula 2.6.

$$\tau = \min_{k \in \mathcal{N}_j} Q_j(d, k) \tag{2.6}$$

Node i will take this variable τ into account when updating its local table of estimates. When a node updates its local table of estimates, it also take into account some other

factors. These include the time that the sending node needed to process the message (the time between receiving a message and forwarding it), represented below as q as well as the time the message spent traveling to the next node (indicating link quality), represented as s. The rule for updating the local estimates takes all off the above variables into account. It then compares the new estimate to the old estimate and applies a learning factor α to the difference. This results in a delta value that will be used to update the local estimate for a node's neighbour.

$$\text{Update rule:}$$
$$Q_i^{t+1}(d,j) = Q_i^t(d,j) + \Delta Q_i(d,j)$$
$$\text{with}$$

$$\Delta Q_i^t(d,j) = \alpha(q + s + \tau - Q_i^t(d,j)) \tag{2.7}$$

Readers familiar with Q-learning principles might notice that Q-routing uses the measured delay at a node and the travel time between nodes (q and s in formula 2.7) as an immediate cost and the estimated remaining travel time from the next node (formula 2.6) as the estimated future cost. The algorithm does not use a discount for the future cost but it does apply a learning rate α to the acquired information.

In the early stages a node's estimates for travel time contain errors, because the ideal routing paths in the network are unknown. But each time a node forwards a message to another node it is able to update its estimates for that node / destination pair. As Boyan and Littman have shown, these estimates will converge over time and after an initial learning phase a stable routing tree will emerge.

The original Q-routing algorithm received criticism because its greedy action selection made it impossible for the algorithm to recover from a load-balancing routing policy to a shortest path routing policy when the network traffic load decreases. Whenever the network load increases and delays start to occur, the increase of Q-values will lead to exploration for a load-balancing routing policy. But when the network load starts to decrease again, the Q-values for the current policy are also likely to improve, causing the algorithm to stick with its load-balancing routing policy.

Multiple solutions for this problem have been proposed in the literature and we will address some of these solutions in section 3.2 about exploration policies.

2.5 Q-routing in sensor networks

Although the Q-routing algorithm was not developed for wireless sensor networks specifically, the algorithm's specific characteristics make it suitable for usage in wireless sensor networks. As emphasised in the introduction, wireless sensor networks tend to generate very irregular and dynamically changing network topologies. The Q-routing algorithm was developed to operate in dynamically changing network topologies and its traffic balancing behaviour can help to distribute traffic load in irregular topologies. So the algorithm's routing properties are well suited for wireless sensor networks. Furthermore the Q-routing algorithm requires only local communication between neighbouring nodes to achieve a good learning behaviour. As said in the introduction, radio communication is very energy consuming in wireless sensor networks. So it is important that any routing algorithm can achieve good results while only generating minimal traffic overhead.

In this thesis we focus on a usage scenario where all messages need to be delivered to a single sink, as this is the most commonly used usage scenario to date. This allows us to slightly simplify the Q-routing algorithm, as all messages have the same destination, by reducing the number of Q values that need to be stored locally at each node. The regular Q-routing algorithm requires that each node stores an estimate for every pair of neighbours and destinations. In wireless sensor networks, nodes only need to store a single Q estimate per neighbour, since the sink is the only destination. So instead of storing $Q_x(d, y)$, where Q is the estimated travel time from x to d via y, it suffices to store only $Q_x(y)$. Where Q is the lowest estimated travel time to the sink from node x if we forward a packet to neighbouring node y. This allows us to reduce the number of Q-values stored at a node x from $N(x) * n$ to $N(x)$ where $N(x)$ is the number of neighbours for node x and n is the total number of sensor nodes in the network. As an example we give an overview of a small topology in figures 2.3, 2.4 and 2.5. The example shows the evolution from Q-values at initialisation and how nodes explore alternative paths until the policy converges to a stable routing tree.

2.6 Performance Criteria

When comparing different algorithms and their variations we need to consider how we can measure network performance. Multiple aspects of the network can be compared to evaluate the performance of different algorithms:

- The average latency for a message. Measured for each message as the time between first transmission and arrival at a sink. The latency for a node i at time t is reported as $l(i, t)$. The average, minimum and maximum latency for the network at time t are reported as $L_{avg}(t)$, $L_{min}(t)$ and $L_{max}(t)$. The average latency time for messages is a good indication of general network performance. A good average travel time should ensure that network performance is acceptable. The maximal travel time can also be compared to make sure that all results fall within a certain threshold of the average.

- The average amount of messages a node has to forward, to identify the load at individual nodes. The amount of messages a node has to forward can help to indicate how good the network load is currently spread across the topology.

- The average lifetime of a network. Measured as the amount of time steps in the simulation before the first node dies. Node battery levels and battery consumption were simulated during experiments by coupling an energy cost to sending and receiving messages. The moment in time when the first node runs out of energy is measured as the network's lifetime. Besides the lifetime average, minimum and maximal energy levels for the network on time t cam also be reported as $E_{avg}(t)$, $E_{min}(t)$ and $E_{max}(t)$.

2.7 Summary

This chapter introduced the routing problem. Most of today's networks are not fully connected networks, there is no direct connection between each pair of nodes in the network. A routing algorithm is needed to form paths between nodes that are not directly connected. Many types of routing algorithms exist and we introduced distance-vector protocols. These

protocols can route traffic on a network by letting all nodes share certain information with their direct neighbours. Because information is only shared between adjacent nodes, a node is not aware of the route that a path follows beyond its direct neighbours. Each node only knows which destinations can be reached through which neighbours, together with an estimation of the time or distance that will be needed to reach a destination. The estimation of time to reach a destination or the distance that needs to be travelled to a destination is obtained by using a cost function. This cost function grades the quality of different direct links between adjacent nodes. By summating the cost function for all links on a path an algorithm can calculate the cost of a path, enabling it to compare between paths. Different metrics can be used as a cost function. We introduced the ETX metric, which represents the quality of a wireless link by estimating the amount of transmission attempts that are needed to obtain a successful transmission. We also introduced different methods of energy aware routing in networks. The algorithm that will be further explored in this thesis is called lowest energy on path routing. This algorithm works by comparing the energy levels of all nodes on a path and using the lowest energy level on a path as a metric. This allows the algorithm to identify nodes that are low on energy and to extend their lifetime by avoiding them in the routing process.

This chapter also introduced reinforcement learning, a machine learning variation that allows an agent to learn which actions to take when executing a task in an environment while attempting to maximise a long term reward. The agent learns which actions to take by a process of trial and error, which requires the agent to explore the available actions. Whenever an agent tries an action it will receive a reinforcement signal that indicates how interesting the action was. In the long run an agent will try to maximise selecting actions that will result in a high reinforcement signal.

Q-learning is one of many reinforcement learning variations. It works by learning an action-value function that represents the expected reward of taking a certain action from a certain state, provided the policy is followed when selecting subsequent actions. A Q-learning algorithm has two customisable parameters, the learning rate α and the discount factor γ. The learning rate determines how important newly acquired information is, a value of 0 discards new information, resulting in the agent not learning anything, while a value of 1 results in an agent only considering new information. The discount factor determines how important estimated future rewards are. A discount factor of 0 results in a very opportunistic behaviour considering only immediate rewards while a value approach 1 results in an agent attempting to maximise the long term reward.

The principles of Q-learning have been applied to routing algorithms. In Q-routing, the time that a message spends travelling between nodes and the time that a message spends being queued at a node are used as immediate rewards. The estimated travel time until a messages destination is used as the long-term reward. The Q-routing algorithm is able to form a delay based load balancing policy that delivers high network performance in scenarios with a high traffic load.

In this thesis we will consider the average latency, the network load at individual nodes and the network lifetime as the main criteria to compare and evaluate different routing algorithms.

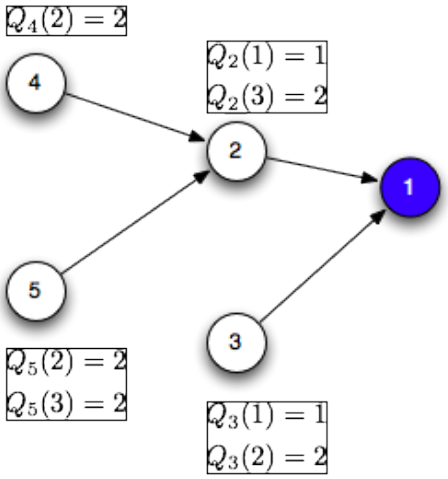

Figure 2.3: Example of Q-values after initialisation by a shortest path algorithm. If all nodes generate a message at the same time and we assume a throughput of 1 message at each node, node 2 will receive two messages and will only be able to immediately forward 1 message. The other message will be slightly delayed.

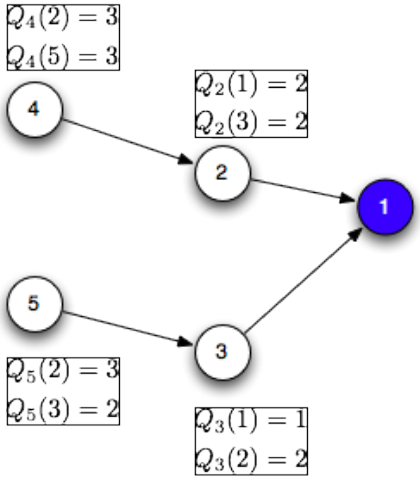

Figure 2.4: After node 2 received two messages, one message will have experienced some delay. This results in node 2 increasing its Q-value. Next time node 2 receives a message it returns the increased Q-value, resulting in nodes 4 and 5 increasing their Q-values. This causes node 5 to explore an alternative route.

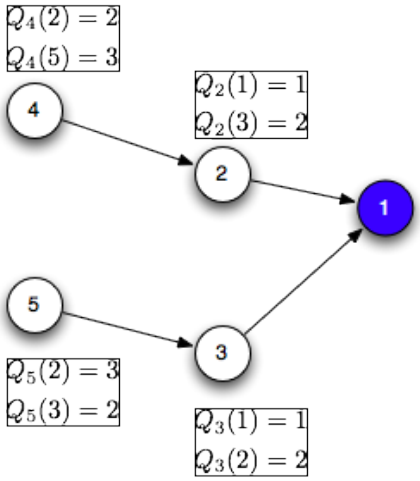

Figure 2.5: The current routing tree spreads the load optimally and remains stable, the policy has converged.

Chapter 3

Q-routing variations

3.1 Introduction

This chapter is divided into three major sections. Each of these sections discusses a specific part of a Q-routing algorithm in more detail. Section 3.2 emphasises the importance of an exploration strategy in a Q-routing algorithm and it provides an overview of some of the more commonly used exploration strategies. Section 3.3 provides an overview of the different metrics that a Q-routing algorithm can take into account when learning its action-value function and how they will influence the algorithms routing behaviour. Section 3.4 provides an overview of different approaches that can be used to determine the initial Q-values before the first run of the algorithm and how this initialisation process can influence the routing algorithm's policy.

3.2 Exploration Strategies

Exploration strategies are a very important aspect of reinforcement learning algorithms. Since reinforcement learning is based on exploration, the reward that is received for choosing an action does not tell if that action is in fact the best possible action. Instead it merely gives an indication of how good or bad one action is in comparison to other actions. This means that in order to learn which actions are best, the system will need to try all of the available actions at least once before they can be compared. So the algorithm has to explore the environment.

Choosing a good exploration strategy is of great importance. An insufficient amount of exploration might lead to scenarios where the optimal solution cannot be found. This can be the case when the early sampling of a suboptimal action results in a better actual reward then the initially estimated expected reward for the unexplored optimal action. On the other hand, a system that performs too much exploration can suffer a lot of overhead from exploring sub-optimal actions. This can often lead to an unsatisfying performance level.

Because of their stochastic nature most problems require a policy of ongoing exploration. A policy that seems favourable at one moment in time might be sub-optimal a few moments later. A typical example for this is a problem that plagued the initial Q-routing algorithm. While this algorithm was able to shift from a shortest path routing policy to a load balancing policy under an increasing traffic load, a lack of exploration caused for the algorithm to stick

to a load balancing routing policy even as the network load started to decrease and when a shortest path routing policy would be able to deliver better performance. Scenarios like this example require exploration in order to maintain a good routing policy.

We will use the remainder of this section to discuss some of the different exploration strategies that are available for Q-learning algorithms.

3.2.1 Greedy

The Q-routing algorithm as implemented by Boyan and Littman uses greedy exploration as default exploration strategy. When using a greedy exploration strategy, an agent will always choose the action that currently has the best estimated reward. In the Q-routing systems these estimated rewards are represented as Q-values, so the parent for a node i can be formally selected in the following way:

$$parent(i) = \underset{j}{\operatorname{argmin}}\, Q_i^t(j) \,|\, j \in \mathcal{N}_i$$

A downside of this approach is that it has a high probability of getting 'stuck' in suboptimal actions if it receives a high reward for a suboptimal action in the early learning phase. Since the greedy selection process will keep choosing the suboptimal action.

The problem of getting stuck with suboptimal actions can be partly avoided by using highly optimistic values as initial estimates for all available actions. Using highly optimistic values will result in all actions being considered multiple times, since the first actual rewards will always be less good then the initially estimated rewards. This approach can greatly improve the greedy exploration strategy's performance and the risk of getting stuck with suboptimal actions can be made arbitrary small. Combining a greedy exploration technique with optimistic initial values has been done in several routing algorithms and will be discussed in more detail in section 3.4 about initialisation techniques.

3.2.2 Epsilon greedy

The ϵ-greedy exploration technique has been used a lot in Q-learning algorithms. It's behaviour is to make greedy decisions by default, but with a probability of p, choose a random action instead. One variation on this approach initialises the algorithm with a high probability p and decreases it over time. This is done to encourage early exploration [Kaelbling et al., 1996].

$$parent(i) = \begin{cases} \underset{j}{\operatorname{argmin}}\, Q_i^t(j) \,|\, j \in \mathcal{N}_i & \text{with prob } 1-\epsilon \\ \text{random neighbour} & \text{with prob } \epsilon \end{cases}$$

A problem that is recognised in this approach is that when exploring a sub-optimal action, the algorithm considers all actions equal. So it will not distinguish between actions that are only slightly less interesting then the optimal action and actions that show no potential to improve upon the best known action. A slight adaption to this strategy is called the Boltzmann exploration [Kaelbling et al., 1996]. This approach will take the expected reward for actions into account and apply it to a Boltzmann distribution, resulting in a higher selection probability for potentially more interesting neighbours.

3.2.3 Full echo

In a Full Echo exploration strategy, a node asks its neighbours for an update on their estimates before choosing an action. Every time a node needs to choose an action, it sends a request for an update value t to all of its immediate neighbours. It then uses these values to update the $Q_x(y)$ value for each neighbour y. This approach is able to detect shortcuts and inefficient policies and it is able to propagate this information through the network quickly. This allows the global network to adjust its policy in a limited amount of time. It also allows the Q-routing algorithm to switch back to shorter paths at times when the amount of traffic on the network is decreasing and there no longer is a need to distribute the traffic.

A downside of this approach is that it can generate a lot of overhead traffic, influencing general network performance. Boyan and Littman suggested to only use this technique on hardware that supports multiple communication channels, to avoid congesting the channels for regular traffic. Taking into account the limited capabilities of sensor node hardware and the cost of radio communication, the full echo exploration strategy seems like a less interesting candidate to deploy on wireless sensor networks. Since the impact on the lifetime of the network might not be worth the limited gain in performance in situations where the Q-routing algorithm needs to switch from a distributing strategy to a shortest path strategy.

3.2.4 Enhanced Q-routing

In [Nowé et al., 1998] the author proposes a modified version of the original Q-routing algorithm. Enhanced Q-routing is able to converge faster as well as to adapt its routing behaviour under both increasing and decreasing traffic loads while still using a greedy exploration policy.

The idea behind enhanced Q-routing is that the direct reinforcement signals in Q-routing (delay at the transmitting node and transmission time to the neighbouring node) are independent of a message's destination. So whenever a node x sends a message to destination d along a neighbour n the node x will use the feedback received from neighbour n to update the routing information for all destinations.

$$\Delta Q^{(k+1)}(x, n, d) = \alpha_k[r^{(k)}(x, n, d) + \min_{a'} Q^{(k)}(n, a', d) - Q^{(k)}(x, n, d)]$$

3.2.5 Inherent exploration

As shown in [Forster and Murphy, 2008] the need for an active exploration policy can be overcome by using specific metrics in the learning policy. In [Forster and Murphy, 2008] the authors use the energy levels of neighbouring nodes as a metric. Because energy levels can only decrease in their scenarios, using them as a metric automatically enforces a certain level of exploration. We provide more details about this implementation in the next section (3.3) about different metrics that can be used in Q-routing algorithms.

3.3 Influence of the Q-metric

The default formula for the Q-routing algorithm (formula 2.7 in section 2.4) takes different metrics into account that are used as a reward.

Update rule:
$$Q_i^{t+1}(j) = Q_i^t(j) + \Delta Q_i^t(j)$$
with

$$\Delta Q_i^t(j) = \alpha(q + s + \tau - Q_i^t(j)) \tag{3.1}$$

Formula 2.7 adjusted for single sink sensor networks.

The Q-routing algorithm as proposed by Boyan and Littman takes into account the amount of overhead at a local node (time a packet has to wait between being received and being forwarded), shown as q in the formula and the time a packet has to travel until it is received by the neighbour (indicating the link quality and distance between nodes), shown as s in the formula. In this section will discuss the influence of these metrics together with an extra metric for energy levels, as was first proposed in [Forster and Murphy, 2008].

3.3.1 Distance / Link quality

In the Q-routing formula 3.1, the variable s represents the time a message spends travelling between nodes. The ETX metric that we introduced in 2.2.3 and that represents the estimated amount of transmissions needed for a successful transmission is a good candidate to be used as a distance metric. When using ETX, the value s for a reliable connection would be at least 1.0 [Morris et al., 2004].

The distance between nodes has been the most commonly used metric in packet routing for many years and we discussed some variations of this metric in section 2.2.3 about the distance function in routing policies. Using the distance as a metric seems like an intuitive choice, in order to get packets to their destination as fast as possible we try to send them along the shortest route. While this approach has worked to guarantee a fast delivery from a single packet from one point to another in the network, it quickly showed some shortcomings in real-life applications. This is due to the fact that while the distance metric gives an indication about the length of a path, and in some cases about its quality (ETX, see 2.2.3), it gives no information about the current state of the network. Different links between nodes might have different capacities and even different nodes might have different capabilities when they have to forward a lot of packets. For this reason, this metric works best when combined with other metrics that take into account the actual situation on the network.

When using the number of hops as distance metric, the value will be constant; $s = 1$, when using link quality as a metric the ETX value could be used, $s = ETX$.

3.3.2 Queue load

In the Q-routing formula 3.1, the variable q represents the difference in time between the moment a node has received a message and the moment a node is able to forward that message to the next node on the path to the sink.

In order to detect the existence of bottlenecks, the Q-routing algorithm measures the amount of time a message spends at each node. Each time a node sends a message it will update its local table of rewards (Q-values). When doing this the node will take into account how much time it needed to forward that message. So when the node reaches a point where it has to forward more messages than it can handle, this value starts to rise,

increasing the Q-values. Remember that the Q-values represent the estimated time until a message will be delivered, so the Q-routing algorithm aims at minimising these values. So when a node's Q-values begin to rise, the next time it receives a message, it will respond with an increased estimate for the remaining travel time. As this estimate starts gets higher, it encourages nodes to explore other neighbouring nodes as their relay node to the sink. This allows nodes to detect when one of their neighbours is becoming too popular, even when it is well connected (good link quality)

3.3.3 Energy level

In [Forster and Murphy, 2008] the authors propose an energy aware adaption for their routing framework FROMS. This framework applies Q-learning techniques for routing data in WSNs. The goal of the algorithm is to route data efficiently from one source to multiple mobile sinks. Like most Q-learning algorithms it is fully distributed and can be applied to multiple WSN scenarios, but the specific nature of the one source multi-sink scenario requires the algorithms to be adaptive and therefore computational complexity for calculating efficient routes becomes significantly larger [Egiriva-Forster and Murphy, 2007].

Their approach is based on using a neighbouring node's energy level as a metric, an idea first proposed in the context of WSNs in [Schurgers and Srivastava, 2001].

Parent energy feedback: The approach in FROMS works by appending a node's energy level, $A = e(i, t)$ to the feedback message τ (formula 2.6) that it sends after receiving a message. When a node receives feedback it will use the received energy level to modify the neighbours Q-value in its routing table.

Let the Q-value that is stored in the routing table at node i for a neighbour j at time t be $Q_i^t(j)$. This value represents how interesting a neighbour is as a parent and can be obtained by a number of algorithms (e.g., Shortest path algorithm). Besides this value each node now also stores a separate energy weighted value of the Q-value, $e(Q_i^t(j))$, this energy weighted value is the value that is used when the algorithm needs to select a node's parent. Whenever a node receives feedback about the energy level of a neighbouring node j containing its energy level, $E_i^t(j)$, the node will updated its energy weighted Q-value $e(Q_i(j))$ for that neighbour in the following way:

$$e(Q_i^t(j)) = Q_i^t(j) * w(E_i^t(j)) \tag{3.2}$$

The w in formula 3.2 represents an energy weighing function. This energy weighing function generates an energy multiplier for the regular Q-value based on a nodes energy level. A lower energy energy level will result in the generation of higher multiplier in order to discourage the selection of nodes with lower power levels as a parent.

The challenge here is to find a balance between both latency and energy consumption. For this purpose three different functions were considered. Two linear functions (function 3.3 and 3.4) and an exponential function (function 3.5). These functions are graphed in figure 3.1. The problem with linear functions w_1 and w_2 is that they make no difference between high and low energy values, due to this they will always influence the routing policy in the same way throughout the network lifetime. Using exponential function w_3 results in more aggressive energy management as different nodes start to approach the end of their life cycle, while it has a smaller influence in the early phase when most nodes have high energy levels.

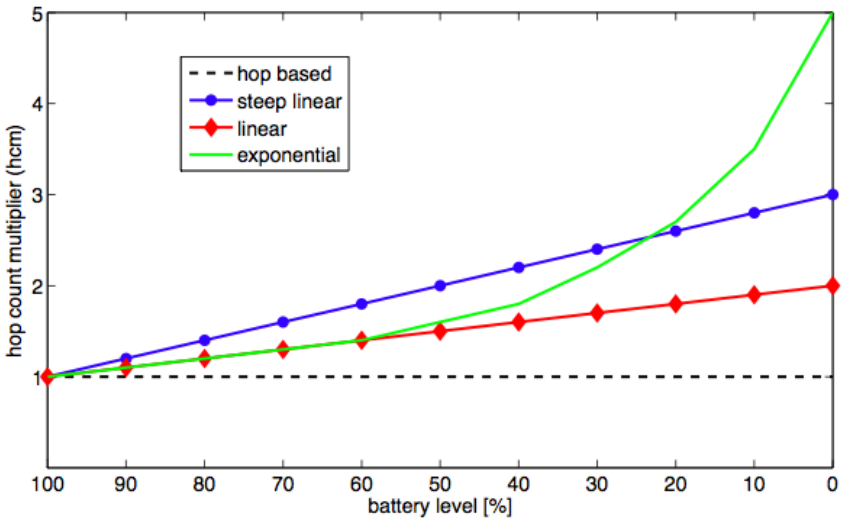

Figure 3.1: Graphed out overview of different energy weighing functions . Taken from [Forster and Murphy, 2008]

Let the energy levels of individual nodes range between 0 and 1, for all nodes $max\_energy = 1$.

$$w_1(available\_energy) = (max\_energy * 2) - available\_energy \tag{3.3}$$

$$w_2(available\_energy) = (max\_energy * 3) - available\_energy \tag{3.4}$$

$$w_3(available\_energy) = 5^{(max\_energy - available\_energy)} \tag{3.5}$$

FROMS uses a multi-sink scenario and each node calculates the cost of complete routes that service all sinks. Specifics of how the FROMS algorithm calculates Q-values are available in [Egiriva-Forster and Murphy, 2007, Forster and Murphy, 2008], instead we focus on how an energy level provided through a feedback message influences the stored Q-values.

3.4 Initialisation techniques

Before a reinforcement learning system can start operating, we need to provide it with initial estimates for all of the available actions. Each node needs to populate its routing tables with initial estimates at time $t = 0$ for $Q_i^0(j) \mid j \in \mathcal{N}_i$.

There are different approaches to how these initial values can be determined, which can have a significant influence on the performance or lifetime of a network. It is also important to realise that the combination of certain exploration strategies, as discussed in section 3.2, and certain initialisation techniques can have a significant impact on the algorithms performance. In this section we will discuss the initialisation techniques that were used during the performed experiments.

3.4.1 Constant value

A common approach that is used to choose the initial Q-values is to give all values on all nodes the same fixed initial value, i.e., $Q_i^0(j) = cte \,\forall\, c(i,j)$

While this seems like a simple and straightforward approach it is important to realise that the choice of this value can have a significant impact on the Q-routing algorithm performance, depending on the kind of exploration strategy that is used. Suppose that our initially estimated values are significantly worse then the actual reward that is received after selecting an action. When an algorithm receives the reward for its first chosen action, this will be a better value then all the other estimated rewards. A greedy algorithm will now no longer consider actions with a low estimated reward and will instead keep reselecting the same action for which it received a better actual reward. This action might actually be a really bad action, but because the initial estimates were pessimistic when compared to the real world rewards, the system will no longer consider other options.

In figure 3.2 we show an example of a small topology, for this example we only use the link quality between nodes to calculate the Q-values (we omit metrics for delays at nodes etc.). Node 5 was initialised with a constant reward of 4. The real costs for different links are shown on the figure, but a node only receives this information when it tries a link. Since all neighbours are initialised at the same value, node 5 will randomly select a neighbour. Suppose it selects node 3. Node 5 will receive information about the actual cost of the path through its reward. The updated Q-values are shown in figure 3.3. Under a greedy exploration policy, node 5 will never try node 2 again, even while in reality this path is more reliable. This can be offset by choosing arbitrary low values as initial estimates. These initial estimates will seem optimistic when compared to real world rewards. This will result in an initial phase of exploration, since the actually received reward for all actions will be worse then the estimated reward for these actions. This will force a certain amount of exploration in the early stages of the networks lifetime.

Boyan and Littmann used a greedy exploration strategy (ref. section 3.2.1) combined with low, optimistic initial estimates for their design of the Q-routing algorithm. This approach works very well when the algorithm has to cope with increasing network loads. Increasing traffic will cause delays in the delivery and this will result in actual rewards on the shortest path getting worse and encourage more exploration. This approach however is not free of problems. While it's behaviour is interesting for situations where the network traffic load is increasing, the greedy exploration technique will result in a permanent routing tree once the network load starts to drop. This will most likely be a sub-optimal solution when compared to regular shortest path algorithms, and when there is only a small amount

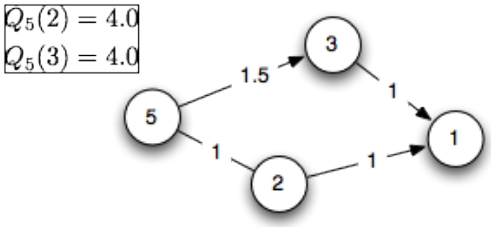

Figure 3.2: When multiple neighbours have the same estimate, node 5 randomly selects one as a parent. Here node 3 is selected.

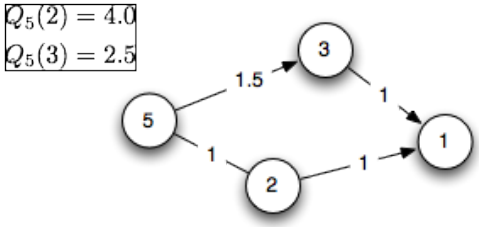

Figure 3.3: Node 5 will greedily reselect node 3 every time, even while in reality node 2 is a better option.

of traffic on the network the advantage of distributing the traffic load across the network might not be worth the extra distance a packet needs to travel.

3.4.2 Shortest Path

Even if reinforcement learning algorithms do not require any prior knowledge about their environment (as discussed in section 2.3), we might be able to speed up the learning process by choosing initial estimates for the Q-values based on information that we do have. The learning process is a very costly phase for a network routing algorithm, as randomised traffic is a waste of resources and might cause extra delays on the network. If we are able to speed up the learning process we might not only achieve better results in less time but also waste less resources while exploring uninteresting routes. In a lot of reinforcement learning scenarios there is limited or no prior information available [Kaelbling et al., 1996], but for the routing problem, we could initialise our estimated rewards based on the values generated by a shortest path algorithm.

$$Q_i^0(j) = \text{shortest path from i to sink through j}$$

This way we might be able to avoid exploring routes that obviously have no potential and that will only lead to extra overhead traffic and wasted resources. Shortest path algorithms have been the subject of a lot of research in the past and many algorithms exist to solve this problem. One of the more interesting and commonly used algorithms for this problem is the Bellman-Ford algorithm [Akyildiz et al., 2002].

While this technique might be able to improve performance during early runs of the algorithm, it is important to realise that it adds a certain bias to the initialisation. So depending on the metrics that are used by the Q-routing algorithm combined with its exploration behaviour, it might be possible that some solutions become harder to discover. This is because paths that seem less interesting in terms of distance could be more interesting according to some other metrics used by the Q-routing algorithm. So when using this initialisation technique, it is important to look at the influence on the different performance criteria discussed in section 2.6.

3.4.3 Randomised

In many domains reinforcement learning algorithms often initialise their estimated rewards with a randomised algorithm [Kaelbling et al., 1996]. This approach works well to guaran-

tee a degree of exploration, but as we already discussed, randomised traffic in a wireless sensor network comes at a great cost, because resources like energy are very limited. Furthermore, the environment for agents solving the routing problem often is not as complex when compared with environments that for example agents in robots have to deal with. Experiments have shown that a Q-routing algorithm using different dynamic metrics will be able to find good solutions on a greedy exploration strategy combined with low constant estimated rewards [Forster and Murphy, 2008].

3.5 Summary

Section 3.2 introduced different exploration strategies for Q-learning algorithms. The most commonly used approach is a greedy selection of the action with the best estimation. This approach however can have some shortcomings, as it can get stuck in sub-optimal solutions. Recent work has proposed an enhanced routing algorithm that uses the received immediate reward to update multiple actions instead of only the chosen actions [Nowé et al., 1998]. This is possible because in Q-routing, the immediate reward is only coupled to the adjacent neighbour and not to the destination of a message. So the immediate reward can be used to update all Q-values for that neighbour instead of only the Q-value for the specific destination. Furthermore, studies in [Forster and Murphy, 2008] show that the need for a non-greedy exploration strategy can be overcome by using energy metrics in a Q-routing algorithm.

Section 3.3 introduced different metrics that can influence the routing policy of a Q-routing algorithm. The amount of time a message spends at a node between being received and being forwarded can be used as a delay metric for a load balancing policy, which can improve network performance [Boyan and Littman, 1994]. The energy level that is available on a node can be used as a metric for an energy balancing policy, which can in turn improve network lifetime [Forster and Murphy, 2008].

Section 3.4 introduced different initialisation techniques. Q-learning algorithms are able to converge to an optimal policy without requiring knowledge about their environment. However, when solving the routing problem we can easily acquire some information about the environment to speed up the learning process. This can be done by using a shortest path algorithm to initialise the estimated Q-values.

Chapter 4

Q-routing adaptions

4.1 Introduction

This chapter highlights the original work that is provided in this master thesis. In section 4.2 we motivate the decision to base our work on existing Q-routing algorithms and explain which design decisions in the existing work conflicted with our own requirements, resulting in the need of an adapted Q-routing algorithm. We then detail our contributions, which can be divided into three distinct parts.

Section 4.3 proposes a modified, energy based Q-routing algorithm which propagates energy information in a different way throughout the network and which better suits our single sink usage scenario. Section 4.4 proposes an algorithm which combines energy metrics with traffic delay metrics. This is done in order to obtain a balance of both good network performance and good network lifetime. Section 4.5 proposes an additional optimisation that can be added to the proposed Q-routing algorithms to further extend the network lifetime.

4.2 Motivation

Experiments with the original Q-routing algorithm that we discussed in the sections 1.3 and 2.4 have shown the algorithm to be effective for balancing network traffic and improving network latency in high traffic scenarios. Follow up research has resulted in multiple modified versions of the original Q-routing algorithm which can overcome some shortcomings of the original Q-routing algorithm [Choi and yan Yeung, 1996, Sun et al., 2002]. However these optimisations are not always suitable for applications in Wireless Sensor Networks.

The interesting properties of the Q-routing algorithm together with the extensive amount of related research led us to the decision of extending the Q-routing algorithm with energy aware metrics. When researching this opportunity we noticed similar work which combines Q-learning principles with energy metrics in [Forster and Murphy, 2008]. The results from [Forster and Murphy, 2008] clearly highlight the benefits of using energy metrics to improve the lifetime of a Wireless Sensor Network. However there are a couple of design decisions in the algorithm that we wanted to explore for possible modification:

- The algorithm does not keep track of any other metrics besides energy metrics. The routing aspect is handled by a basic shortest path algorithm.

- The algorithm does not propagate a node's energy level beyond its direct neighbours.
- The algorithm is tailored to a very specific multi-source multi-sink routing scenario. While this is an interesting scenario a lot of WSNs still rely on a multi-source single sink scenario, which is the scenario we focus on in this master thesis.

This motivated us to develop a routing algorithm that combines both traffic delay metrics as originally proposed in [Boyan and Littman, 1994] together with a modified version of the energy metrics that were originally proposed in [Forster and Murphy, 2008].

4.3 Lowest energy on path algorithm

We propose a Q-routing algorithm with energy metrics based on the algorithm proposed in [Forster and Murphy, 2008], but modified in such a way that the algorithm propagates energy feedback throughout the network. Furthermore the algorithm is also designed for a multi-source single-sink scenario instead of a multi-sink scenario.

The extensions we made to the original energy aware algorithm are focussed on the manner nodes provide feedback of their energy levels. As detailed in section 3.3.3, the approach in FROMS works by appending the energy level of a parent node to the feedback it returns. As a reminder we provide a quick overview of the original algorithm.

In the energy aware routing algorithm, all nodes store a pair of values for each neighbour, as opposed to a single Q-value in delay based routing. The pair of values consists from a value for the travel distance through the neighbour (in [Forster and Murphy, 2008] this is the shortest path) and a value for the neighbour's energy level. When a node j receives a message from node i it will send a feedback message θ back to node i. This feedback will contain the energy level of node j at the time it received the message.

Energy feedback in [Forster and Murphy, 2008]:

$$\theta = e(j, t)$$

Node i will store this energy level for node j as $E_i^t(j)$. Whenever a routing decision needs to be made, the algorithm will use a weighing function to determine the influence of the energy level on the stored routing value. Let $Q_i^t(j)$ be the routing metric that is stored at node i for routing a message through node j. The energy weighed version of this routing metric is $e(Q_i^t(j)$, with w being a weighing function that determines the influence of $E_i^t(j)$, the stored energy level for node j at node i.

$$e(Q_i^t(j)) = Q_i^t(j) * w(E_i^t(j)) \text{ (formula 3.2 from section 3.3.3)}$$

This algorithm creates an energy aware routing policy by sharing a node's energy level with its immediate neighbours. However, we argue that by not sharing energy information beyond immediate neighbours, a routing policy might form where nodes adjacent to low energy nodes are not be able to route traffic around the weak node without generating a considerable overhead. Consider the example on the left in figure 4.1. The node marked with an X suffers from a low energy level. By sharing its energy level neighbouring nodes are less likely to route their messages through the node, as is indicated by the red exclamation marks. However, the top left and top right nodes are not aware of this situation. When these nodes route their traffic through the middle node, the middle node will either have to

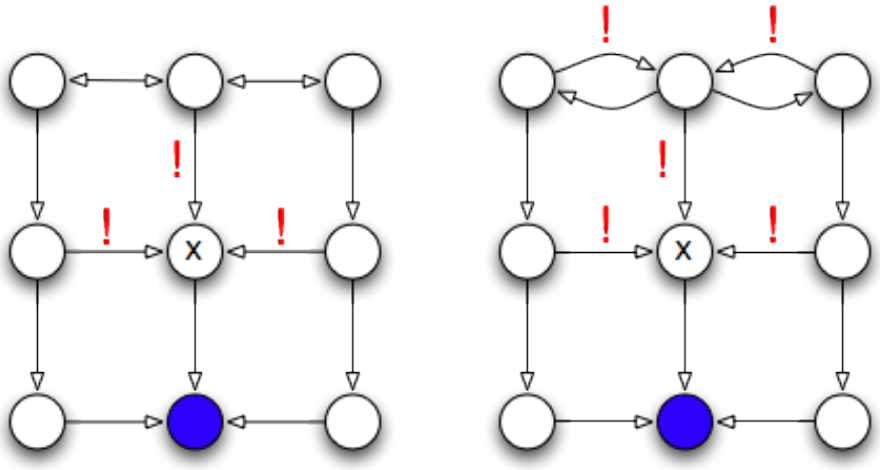

Figure 4.1: In this example, the node marked with an X has a critical energy level.The exclamation marks show which links are influenced by the introduction of an energy metric. The topology on the left shows the influence of the parent energy algorithm while the example on the right shows the influence of the lowest energy on path algorithm.

route it through the low energy node or back through one of the top nodes, resulting in a significant overhead.

To overcome this behaviour, we proposed a modified algorithm which is inspired by the routing behaviour that is referred to as "Maximum minimum energy available node route" in [Akyildiz et al., 2002] and that is referred to as **"lowest energy on path routing"** in section 2.2.3. The aim of this approach is to identify the weakest node in terms of available energy on each path and use the energy levels of these nodes as the metric for the entire path that they are located on.

Because Q-learning algorithms generate routing trees dynamically we cannot determine these weakest nodes precisely in real time. To overcome this, all nodes keep an estimate of the lowest energy level on the path that is most likely to be chosen by their parent. Because of the dynamic nature of the routing trees nodes need to update this estimate constantly. This allows the routing algorithm to find alternative routes around nodes which are approaching the end of their life cycle. We implemented this as follows.

Just like in the original algorithms, all nodes store a pair of values. One for the routing metric and one for the energy metric. However the algorithm differs in the way energy values are shared across the network. Instead of storing the energy levels of their neighbours, nodes store an estimated energy level for each of their neighbours, $\xi_i(j) \,|\, j \in \mathcal{N}_i$. This estimation reflects the lowest energy level on the path to the sink that will be followed by that neighbour. Estimates are initialised as $\xi_i^0(j) = 100\%$. During the network lifetime, energy estimates are updated as follows.

Whenever a node j receives a message from node i, node j provides a feedback message θ to node i. In order to obtain a value for the feedback θ, node j determines its current paren, based on the energy weighted Q-value from formula 3.2. The node then compares the estimated energy level of the path along this parent node $\xi_j(p) \,|\, p \in \mathcal{N}_j$ to its own energy level $e(j, t)$. Node j will then return a feedback message θ containing the lowest of the two energy levels, as such the minimum energy level on the current path is propagated towards the source of the path.

$$\theta = \begin{cases} e(j,t) & \text{if } e(j,t) \leq \xi_j(n) \mid n \in \mathcal{N}_j \\ \xi_j(n) & \text{if } e(j,t) > \xi_j(n) \mid n \in \mathcal{N}_j \end{cases}$$

When node i receives the feedback, it will update its energy estimate $\xi_i(j) \mid j \in \mathcal{N}_i$ with the new value.

The routing algorithm still uses formula 3.2 to obtain energy weighted Q-values and uses an exponential function as weighing function w.

When we look at our example in figure 4.1, we see that this modified algorithm is able to propagate energy information further into the network, enabling the algorithm to better route around problem areas. As we will show in our results, this modification can result in significant improvements of the network lifetime in certain scenarios.

Another advantage of separating Q-values for energy and latency is that it allows the routing algorithm some extra possibilities for Quality of Service routing. An algorithm could decide to route certain types of messages along the fastest path while sending other messages along more energy aware paths.

4.4 Combining energy and latency metrics

In [Forster and Murphy, 2008] the authors propose a Q-learning algorithm that takes the energy levels of nodes into account as a metric. However their experiments are limited to algorithms using only one extra routing metric, being the hop distance. While this results in an algorithm that is able to maximise the network lifetime, the algorithm is not able to maintain network performance in heterogeneous networks with dynamic network loads. To cope with this we implemented an algorithm that combines the delay based properties proposed in [Boyan and Littman, 1994] with the energy aware properties from our modified energy feedback algorithm. We believe that this implementation is necessary in order to obtain an algorithm that is able to deliver both good network lifetime as well as network performance in heterogeneous networks.

In our combined algorithm, nodes give feedback about both measured traffic delay, τ and energy levels on the path, θ.

$$\tau = \min_{n \in \mathcal{N}_j} Q_j(n)$$

$$\theta = \begin{cases} e(j,t) & \text{if } e(j,t) \leq \xi_j(n) \mid n \in \mathcal{N}_j \\ \xi_j(n) & \text{if } e(j,t) > \xi_j(n) \mid n \in \mathcal{N}_j \end{cases}$$

Each node stores two values per neighbour. A regular Q-value indicating the estimated travel time of the path along a neighbour and an energy value ξ indicating the estimated lowest energy level on the path along a neighbour. The algorithm makes its routing decisions based on the energy weighted Q-value which is obtained as follows:

$$e(Q_i^t(j)) = Q_i^t(j) * w_3(\xi_i(j)) \text{ (formula 3.2 from section 3.3.3)}$$

The combined algorithm always uses an exponential weighing function (w_3 formula 3.5)).

4.5 Delta updates

One of the key concepts of Reinforcement Learning algorithms is that they continuously update their policy according to the feedback they receive. However, if the policy is able to converge, the sending of feedback messages can become redundant, because the feedback signal that is returned remains about the same and no new information is propagated. This allows us to further optimise the network lifetime by limiting the amount of feedback messages. This can be done at the cost of a little extra storage.

Each node stores two types of informations per neighbour:

- the routing feedback τ last returned (Formula 2.6)
- the energy feedback θ last returned

In a regular scenario, all nodes return a feedback message whenever they receive a regular message. By adding this optimisation, whenever a node needs to send feedback to one of its neighbours, it compares the current feedback to the previously returned value.

$$\Delta_\tau = \frac{|\tau_{old} - \tau_{new}|}{\tau_{new}} \tag{4.1}$$

$$\Delta_\theta = \frac{|\theta_{old} - \theta_{new}|}{\theta_{new}} \tag{4.2}$$

If either of the delta values is higher than a threshold x the node will send feedback as usual, if both delta values are lower than the threshold, the node will not communicate any feedback, saving both the sending and receiving node some energy. When a node received a message, it will only respond with an updated τ and energy level if either Δ_τ or Δ_θ is larger then a threshold x.

Chapter 5

Experimental Setup

5.1 Introduction

This chapter provides essential information about the environment that was used to perform the different experiments. The first section describes the design and the possibilities of the discrete simulator that was used to evaluate the different routing algorithms. The second section details which parameters were varied over the experiments and the third and final section in this chapters elaborates on the different network topologies that have been used, providing a small description and motivation for each of the used topologies.

5.2 Simulator

All experiments were performed in a self-developed discrete simulation environment for Wireless Sensor Networks. This environment has been developed and tested in the R software environment [1]. The simulator was designed to test different routing algorithms and supports multiple protocols for initializing the routing tables as well as multiple routing algorithms. In order to validate the simulator we tested multiple well known algorithms and verified that their behaviour was in line with expectations, both on smaller (13 nodes) and larger (42 nodes) topologies.

In the simulator topologies are specified by a list of x and y coordinates that position nodes on a two dimensional grid. Every node has a property that determines wether the node will act as a sink. Sink nodes will only receive data and will not generate or forward data. The simulator is capable of handling scenarios with multiple sink nodes.

All nodes have an identical fixed range in which they can communicate with neighbours. Because we use a two dimensional model we can calculate the communication ranges as the Euclidean distance between nodes. An example of a topology and the maximal communication ranges is shown in figure 5.2. The system does not simulate noise and assumes a perfect link quality between all nodes that are within the communication range of each other. Communication between neighbouring nodes takes one discrete time step. This means that a message can travel one hop per time step.

While the simulator is limited, some design decisions were made in an attempt to make the simulator more representative of real word scenarios. First, the system simulates an energy level per node. A node's energy level will deplete at a constant rate over time.

[1] http://www.r-project.org/

Besides this continuous depletion extra energy is consumed as more messages are sent and received. This is done by coupling fixed costs to the actions of transmitting and receiving of a message. Furthermore, the order in which the nodes are handled by the simulator as well as the order in which messages are delivered to their destination, are randomised during each simulation time step. This should avoid possible biases that might exist due to the order of nodes in our simulator.

We provide an overview of the different algorithms that were implemented into the simulator.

Initialisation techniques: Each node's routing table can contain information for multiple possible paths to the sink along different neighbours. It suffices to store one Q-value per neighbour since all data in the network has the same destination, so each value is an indication of how capable this neighbour is for forwarding data to the sink . Due to the nature of Q-routing algorithms values could either be initialised with randomised or constant values and a Q-routing algorithm with a sufficient exploring behaviour would still be able to find good routes. But to improve initial performance all routing tables are initialised by a shortest path algorithm that is based on a Dijkstra algorithm that starts at the sink and spreads out along the neighbours, relaxing edges between different neighbours.

Routing algorithms: When a node has to forward a data packet, it can select a neighbouring node based on the values stored in its routing tables. The default behaviour for any routing algorithm is to greedily select the neighbour with the best value from the routing table. While this was a source of criticism for the original Q-routing algorithm, it was shown in [Forster and Murphy, 2008] that using energy levels of neighbouring nodes as a metric ensured a good exploration behaviour and eliminated the need of a non greedy exploration strategy. The Q-routing algorithms have specific rules for calculating cost functions and updating the routing tables at runtime and these are specified in their descriptions in section 1.3.

In the initial experiments we considered shortest path routing as the routing policy. In this case the routing tables were initialized once at runtime. Let us define $\mathcal{N}_i$ as the set of neighbors for node i and $l_{i,j}$, $j \in \mathcal{N}_i$ as a link between both nodes function. Because we use a single sink scenario we define the hop cost function $c()$ for a link $l_{i,j}$ as:

$$\forall i, j \in S \,|\, j \in \mathcal{N}_i : c(l_{i,j}) = 1 \tag{5.1}$$

Subsequently we can define a function for the cost of an entire path P to the sink:

$$c(P) = \sum_{l_{i,j} \in P} c(l_{i,j}) \tag{5.2}$$

Feedback: When a node receives a message it always provides feedback to the sender of the message. Which information is returned as feedback and how this feedback is handled depends on which algorithm is being used. There are three possible kinds of feedback that can be returned

- Delivery delay based feedback for delay based Q-routing
- Parent energy feedback for energy based routing
- Lowest energy on path feedback for energy based routing

The delay based feedback can be combined with any of the two types of energy feedback, or either one of the energy based feedback types can be used without any delay based feedback. In our experiments we will differentiate our results based on the type of feedback that was used.

Energy: To simulate energy levels each node has its own energy capacity. This can be fixed or randomised at start and will vary over time. There is a fixed energy cost for a node to be awake, which is necessary in order to send or receive data. Furthermore, sending and receiving messages results in an extra energy consumption to simulate the cost of actively using a wireless antenna.

5.3 Variables

When performing experiments there are a number of predetermined factors that can influence the experimental results.

Message generation rate There are two ways that a network can generate messages. The first approach is to have all sensor nodes generate a data packet at the same time on a fixed time interval (e.g., every 10 time steps). This results in 'waves' of data packets that need to be routed to the sink.

Another approach that is used in some experiments is to assign a certain probability p with which a node will generate a message every time step. For example a probability of $p = 0.10$ means that each node has a 10% probability of generating a message during each time step of the simulation.

Initialisation algorithm As we discussed in earlier sections there exist multiple ways for nodes to initialise the Q-values for their neighbours. $(Q_i^0(j) \mid j \in \mathcal{N}_i)$. We will specify which method was used for each experiment.

Feedback algorithm For each experiment we will specify wether we use delay based feedback, parent energy feedback, lowest energy on path feedback or a combination of delay and energy feedback.

Learning rate One of the main parameters of any Q-learning algorithm is the learning rate. It determines how responsive the algorithm is to changes in the received rewards. A learning rate of 0 results in an agent discarding new information and not learning anything, while a learning rate of 1 results in an agent only considering the most recent information. Whenever we run experiments with delay based feedback we will also specify the use learning rate α.

5.4 Topologies

During experiments we used several distinct topologies to test the different algorithms. In our simulations all nodes have an identical fixed communication distance range for nodes

they consider neighbours. This range is shown as an example on the small homogeneous topology in figure 5.2

Whenever we refer to a topology as homogeneous we mean that all non-sink nodes have identical properties. This means they have an identical energy capacity, identical energy costs, identical probabilities of generating a message and identical capacities for sending and receiving messages. Sink nodes always have an infinite amount of energy and can receive and process an unlimited amount of messages every time step.

The first topology is a small homogeneous topology and is shown in figure 5.1. It was chosen for two main reasons. The first reason is that this topology has a visible bottleneck at node 5, so it allows us to test how algorithms deal with this bottleneck. Another important reason is that because of the topology's small size, it is possible to easily overview the network layout and the network performance performance in a visual way.

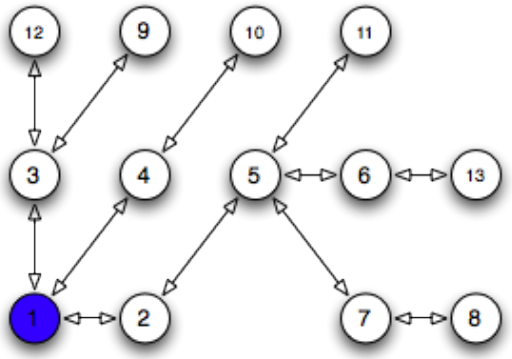

Figure 5.1: Small homogeneous topology used in experiments, shown as initialised by Dijkstra algorithm

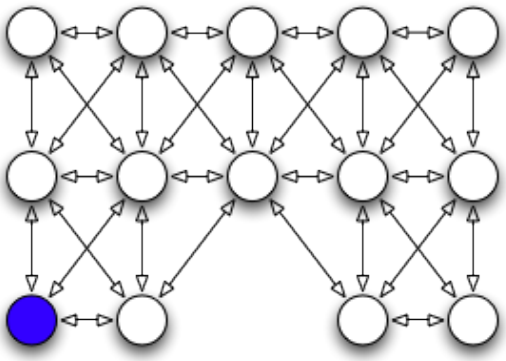

Figure 5.2: Small homogeneous topology. All nodes have the same fixed range for communication, connections shown between all possible neighbours.

While the first topology is very interesting to investigate algorithm behaviours and compare different algorithms, its limited size might mean it is not a good representation of algorithm behaviour on larger sensor netwerks. For this reason we also used a second, larger homogeneous topology which is shown in figure 5.3 and consists out of 36 nodes. This topology allows us to see how the different algorithms scale in scenarios where there is a lot more data passed around in a larger network. Because the size of this second topology is much bigger it should be able to represent real world scenarios in a more realistic way. Furthermore this topology has also been used by other researchers like Forster and Boyan in their experiments.

The third topology that was used during experiments is shown in figure 5.4. This is a large, heterogeneous topology. We created a group of nodes around the sink that are three times more likely to generate a message for the sink. This means they will consume their resources faster than other nodes around the sink, leading to opportunities for energy aware routing algorithms to route around this area.

Besides these topologies, we also tested some irregular topologies in different situations. These irregular topologies are shown in figure 5.5.

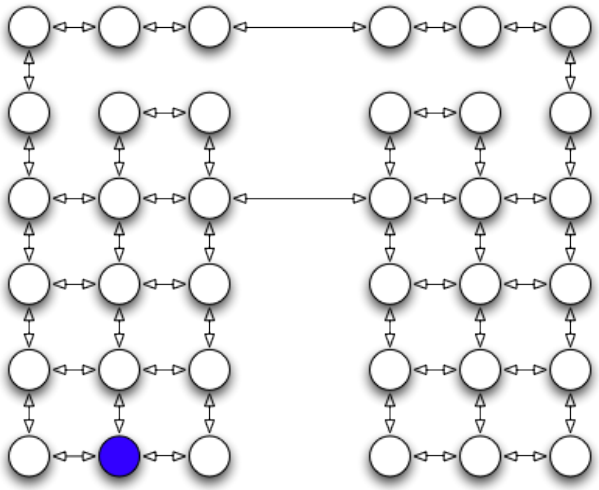

Figure 5.3: Large homogeneous topology with possible neighbour connections.

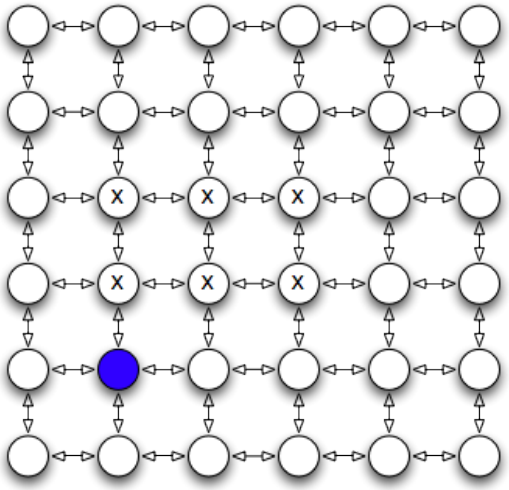

Figure 5.4: Large heterogeneous topology. Nodes marked with an X are three more likely to generate messages than regular nodes.

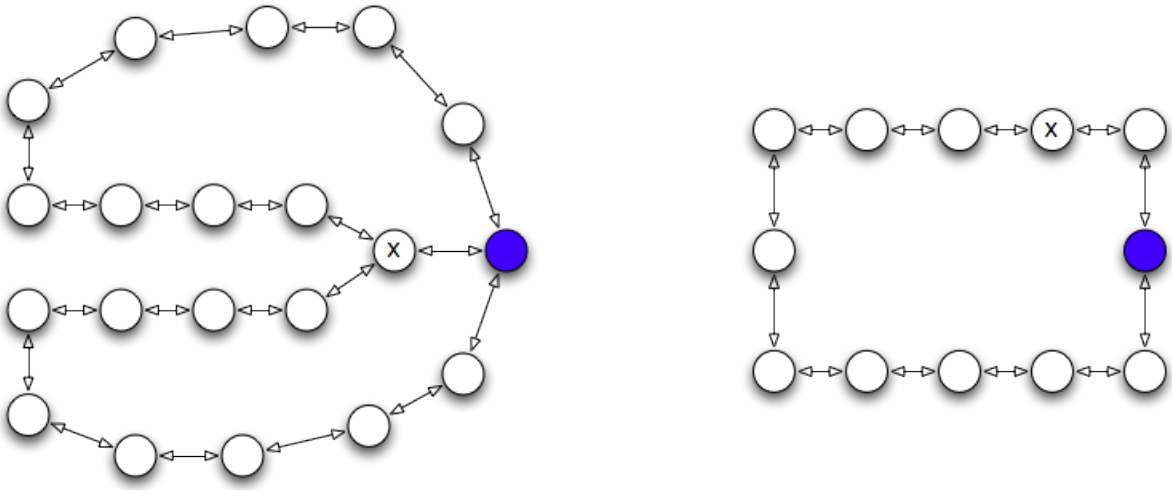

Figure 5.5: Irregular topologies.

Chapter 6

Experiment Results

6.1 Introduction

This chapter provides an overview of the experimental results that were obtained over the course of this master thesis. All the reported numbers were acquired by averaging the results over 30 independent runs. The chapter is divided in five sections. The first section provides results about the network and energy performance of the default Q-routing algorithm. The second section provides network performance and lifetime results for the parent energy and the lowest energy on path algorithms on homogeneous topologies. In the third section we evaluate the lifetime performance of energy aware algorithms on heterogeneous and irregular topologies. In the fourth section of this chapter we evaluate the influence of our added optimisations. The fifth and final chapter gives a short summary of the results provided in this chapter.

6.2 Delay based Q-routing

This section reports on results obtained with the regular Q-routing algorithm. First we evaluate the network performance of the algorithm on different topologies and in different situations. Next we evaluate the impact of different initialisation techniques on both network latency and network lifetime. The final part of this section gives an overview of the influence of different learning rates on the algorithms performance.

6.2.1 Uniform initialisation

As a starting point for our experiments we tested an implementation of the greedy, delay based Q-routing algorithm as described by Boyan and Littman [Boyan and Littman, 1994]. The Q-routing tables at each node were initialised with an optimistic constant value: $Q_i^0(j) = 1 \,|\, j \in \mathcal{N}_i$ Because we use hop distance as the distance metric, this initial estimate will be to optimistic for all nodes that are not adjacent to the sink. This algorithm was then tested on the small homogeneous topology (ref. figure 5.1) under different traffic loads. To evaluate the network performance we compared the average delivery times for messages to reach the sink.

Experimental setup:

- Initialisation: $Q_i^0(j) = 1 \,|\, j \in \mathcal{N}_i$
- Learning rate: $\alpha = 0.5$
- Message generation: Average traffic load: Simultaneous every 10 time steps
- Message generation: High traffic load: Simultaneous every 5 time steps
- Topology: Small homogeneous topology

Average traffic load Our experiments showed results in line with those obtained by Boyan and Littman. In simulations with an average amount of traffic load on the network, the Q-routing algorithm was able to converge to the same performance level as a shortest path algorithm. In it's early life cycle the Q-routing algorithm cannot match the average delivery times from a shortest path algorithm, but over time it was able to achieve delivery times similar to those achieved by a regular shortest path algorithm. The average delivery times for the network, $L_{avg}(t)$, over its lifetime are shown in figure 6.1. During it's early lifetime the Q-routing algorithm requires more time to deliver messages as it tries out the different neighbours for each node as possible destinations. Over time the feedback provided by all these neighbours helps the algorithm to learn the interesting routes to the sink.

In this experiment, delivery times for messages started to improve significantly when at least two messages from every node were delivered to the sink. Under average traffic levels this happens after about 20 time steps. Once the sink received about 11 messages from each node in the system the average delivery time started to converge to a value that is about the same as the delivery times for a shortest path routing algorithm. In an average traffic simulation this happens after about 75 time steps.

High traffic load When we performed the same experiment with a higher traffic load, the differences between a regular shortest path algorithm and the Q-routing algorithm became more apparent. This is also in line with results obtained by Boyan and Littman, as they designed the algorithm specifically for situations where there is a lot of traffic on the network. The graph in figure 6.2 shows that as the simulator keeps running for a longer time, the average delivery times for the shortest path routing algorithm start to worsen. This is because the algorithm is not able to properly balance the traffic across the network resulting in slow downs at the bottleneck nodes in the topology. The Q-routing algorithm on the other hand is able to keep improving it's performance while it explores different actions. After a period of exploration the algorithm is able to find a policy that can better spread the traffic over the network, resulting in less slow downs at the bottleneck nodes in the topology.

The latency performance off both the shortest path algorithm and the Q-routing algorithm under increasing traffic on the small homogeneous topology are shown in figure 6.3.

6.2.2 Shortest path initialisation

As discussed in section 3.4.2, when using a constant value to initialise the estimated rewards, there is a lot of random traffic during the early lifetime of the network. This results in both poor network performance (high latency) and wasted resources (sending messages back and forth along dead end routes). It might be interesting to initialise the estimated rewards for

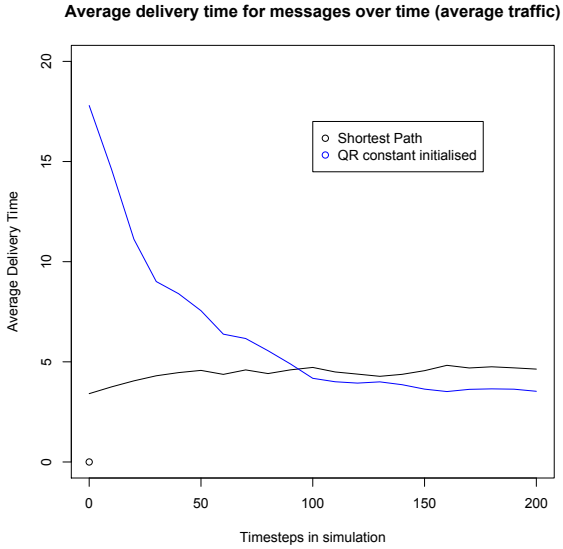

Figure 6.1: Average delivery times for messages during network lifetime under average traffic load. After an initial phase of exploration the Q-routing algorithm converges to a stable policy that achieves about the same performance as a regular shortest path routing algorithm.

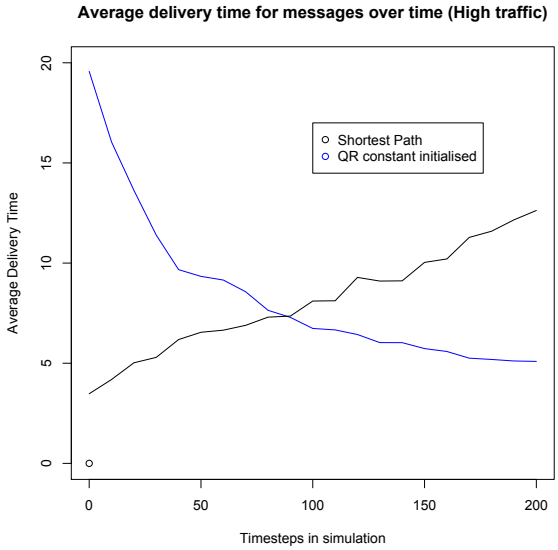

Figure 6.2: Average delivery times for messages during network lifetime under high traffic load. Delivery times $L_{avg}(t)$ for the shortest path algorithm increase over time t as the algorithm suffers from a bottleneck on the shortest path. After an exploration phase the Q-routing algorithm converges to a policy that better distributes the network traffic and does not suffer from a bottleneck. This results in significantly better latency.

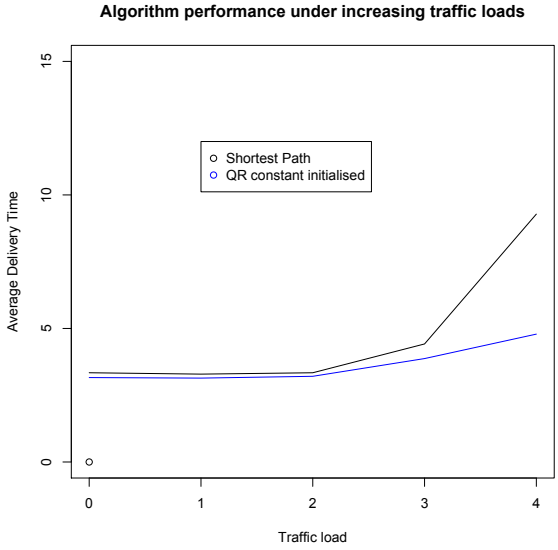

Figure 6.3: Evolution of network performance on the small homogeneous topology under increasing traffic load. As the amount of traffic increases, the shortest path routing algorithm's performance starts to suffer from the bottleneck on the shortest path in the topology. The Q-routing algorithm is able to achieve better performance by distributing the traffic along multiple routes.

different actions in a smarter way that utilises available information. For the next experiment we adapted the regular Q-routing algorithm by running a Dijkstra shortest path algorithm and using these values as initial values for the estimated rewards of the different actions. The adapted algorithm runs a Dijkstra algorithm after the network is brought online. The distance will be calculated as the number of hops that need to be made in order to reach the sink. Each hop has the same fixed cost. No other metrics will be used at initialisation.

We evaluate this adapted version and compare it with the regular Q-routing algorithm on two levels. First we evaluate the influence on general network performance by comparing network latency. Second we evaluate the amount of energy consumed by both algorithm variations during the network lifetime.

Experimental setup:

- Initialisation: $Q_i^0(j) =$ Shortest path (Number of hops, Dijkstra algorithm)
- Learning rate: $\alpha = 0.5$
- Message generation: Average traffic load: Simultanious every 10 time steps
- Message generation: High traffic load: Simultanious every 5 time steps
- Topology: Small homogeneous topology

Network latency The average delivery times for both variations in an average traffic simulation are shown in figure 6.4. Our results show that using the Dijkstra algorithm to initialise the estimated rewards allows the Q-routing algorithm to approach the same performance as a regular shortest path algorithm during the early lifetime of the network.

The Dijkstra initialised Q-routing algorithm will perform slightly worse then the regular shortest path algorithm as it explores some alternative routes. The cost of exploration however is very limited as the algorithm will never choose neighbours that would lead to drastically worse delivery times.

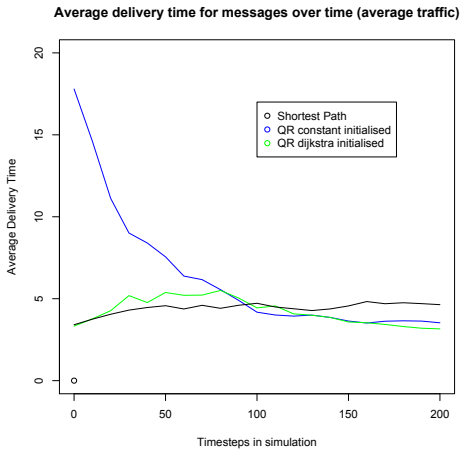

Figure 6.4: Dijkstra initialisation helps the Q-routing algorithm to perform significantly better in the early life time of the network. During this early phase the Dijkstra Q-routing algorithm still performs slightly worse then regular shortest path routing because of the exploration performed by the Q-routing algorithm.

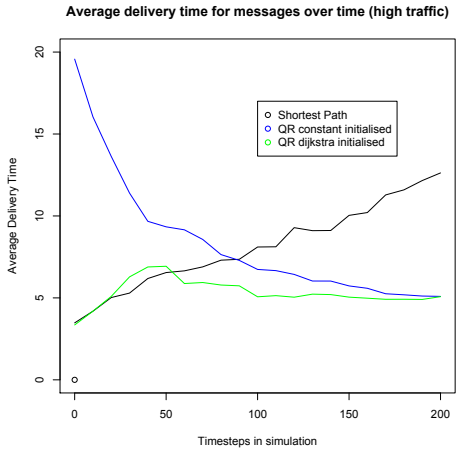

Figure 6.5: Observations for high traffic load simulations are in line with those for average traffic load. The Dijkstra initialised variation is able to make a big difference with the constant initialised algorithm in the time before convergence, this is because the overhead of randomised traffic causes extra delays when there is a lot of traffic.

Network lifetime When we measure the energy consumptions of the initialisation algorithms we note that the after the Dijkstra initialisation is finished, the average energy level in the network, $E_{avg}(t)$, is about 3% lower than before the initialisation procedure. This is to be expected because running the Dijkstra algorithm requires some radio communication and coordination between nodes while a constant initialisation technique requires no extra communication. However, after a few time steps in the simulation the Dijkstra initialised algorithm is able to make up for this difference, as it suffers significantly less overhead from random exploration. When we initialise the estimated rewards at constant values, it is inevitable for some messages to be sent along inefficient routes. This is because the algorithm has no knowledge at all about the topology and needs to acquire this information through exploration.

Our experiments show that this exploration not only results in worse delivery times, but also in an extra energy consumption of about 8%. This means that once both variations of the algorithm have converged, the Dijkstra initialised variation will tend to have consumed about 5% less energy in the topology. The average energy level E_{avg} in the network over time for both algorithms is graphed out in figure 6.6 for the small homogeneous topology and in figure 6.7 for the large homogeneous topology. An important finding of our experiments on the large topology is that on a less densely connected topology the Dijkstra algorithm consumes less energy per node during initialisation. This is due to the fact that the Dijkstra

algorithm requires communication between a node and all of its neighbours, so less densely connected networks consume less energy during start up.

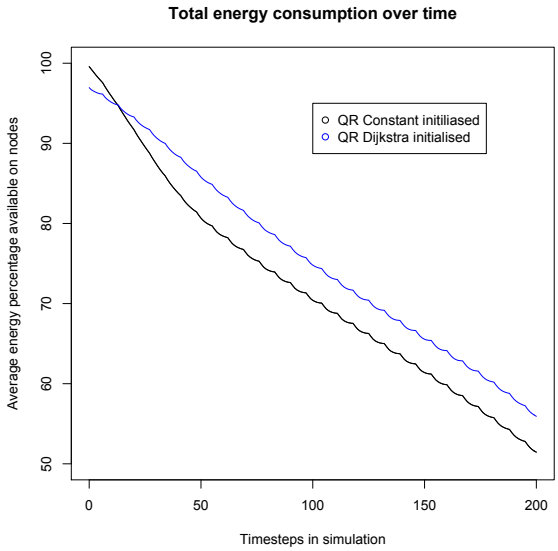

Figure 6.6: Average amount of available energy $E_{avg}(t)$ in the small topology graphed over time t. Dijkstra initialisation consumes slightly more power at start up, but the better exploration behaviour makes up for this difference and when both algorithms converge the Dijkstra initialised variation will have consumed about 5% less energy.

6.2.3 Influence of the learning rate

One of the main parameters of any Q-learning algorithm is the learning rate. It determines how responsive the algorithm is to changes in the received rewards. A learning rate of 0 results in an agent discarding new information and not learning anything, while a learning rate of 1 results in an agent only considering the most recent information.

In [Boyan and Littman, 1994] the authors suggest that a learning rate of $\alpha = 0.5$ delivers the best results. We evaluated this with own experiments and our results were similar. Table 6.1 shows the results for network performance and lifetime with different learning rates.

Experimental setup:

- Initialisation: $Q_i^0(j)$ = Shortest path (Number of hops, Dijkstra algorithm)
- Learning rate: α varied from 0.4 to 1, reported in table 6.1.
- Message generation: probability of $p = 0.10$
- Topology: Small homogeneous topology

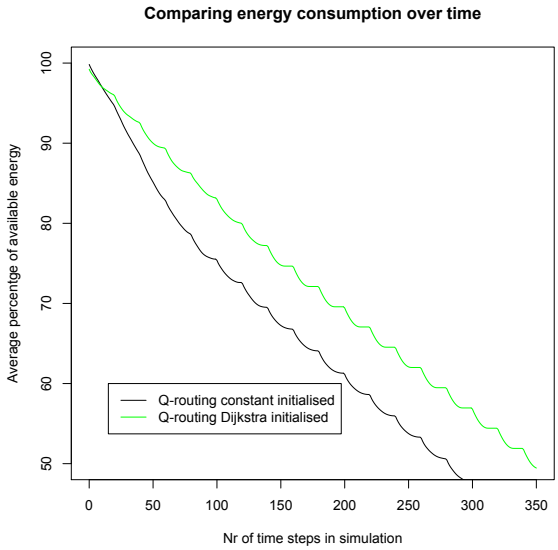

Figure 6.7: Average amount of available energy $E_{avg}(t)$ in the large topology graphed over time t. The difference between both variations is even bigger here and after the early exploration phase the Dijkstra algorithm will have consumed about 9% less energy. Dijkstra algorithm also consumes slightly less energy because this topology is less connected. (DS algorithm involves messages between all neighbours, so a less dense connected topology requires less communication to initialise)

Learning rate α	Average lifetime	Average latency	Total # messages
0.4	240 ($\pm$ 7)	3.33 ($\pm$ 0.15)	287 ($\pm$ 9)
0.5	245 ($\pm$ 12)	3.37 ($\pm$ 0.14)	293 ($\pm$ 10)
0.6	243 ($\pm$ 11)	0.00 ($\pm$ 0.00)	000 ($\pm$ 00)
0.7	240 ($\pm$ 9)	0.00 ($\pm$ 0.00)	000 ($\pm$ 00)
0.8	238 ($\pm$ 11)	0.00 ($\pm$ 0.00)	000 ($\pm$ 00)
0.9	235 ($\pm$ 11)	3.57 ($\pm$ 0.17)	275 ($\pm$ 10)
1.0	224 ($\pm$ 10)	3.87 ($\pm$ 0.19)	269 ($\pm$ 11)

Table 6.1: Effect of learning rate in delay based routing on small homogeneous topology. Reported values are: Network lifetime (time of death of first node), average latency and total amount of messages received at the sink during the network lifetime. 95% confidence intervals reported between brackets.

6.3 Energy based Q-routing

In this section we provide the network performance and lifetime results of algorithms that combine an energy metric with delay based Q-routing in homogeneous topologies. We provide results for the influence of the parent energy algorithm in the first subsection while the results for the lowest energy on path algorithm are available in the second subsection.

6.3.1 Parent energy

The first experiment with energy metrics was performed by adding energy feedback as proposed in [Forster and Murphy, 2008] to the implemented Q-routing algorithm. We tested this adapted algorithm on the small homogeneous topology and our findings confirmed those from [Forster and Murphy, 2008], the usage of energy aware metrics is able to significantly improve the network lifetime. We performed different experiments with the three different weighing functions and the results were as expected, with the exponential function resulting in the biggest lifetime improvement. The lifetime results for all weighing functions are shown in table 6.2, using the exponential weighing function resulted in a 21% increase of the network lifetime.

Experimental setup:

- Initialisation: $Q_i^0(j) =$ Shortest path (Number of hops, Dijkstra algorithm)
- Learning rate: $\alpha = 0.5$
- Message generation: Average traffic load: Simultanious every 10 time steps
- Topology: Small homogeneous topology

While a significant improvement of the network lifetime is a positive result, it is important to keep track of the influence of this new routing behaviour on the network performance in terms of latency. Our experiments have shown that the impact of adding an energy aware metric to a Q-routing algorithm still resulted in an acceptable network latency. During the early lifetime of the network the linear weighing functions have only a very limited influence on the performance, which is to be expected as there average energy level in the network is still pretty high. However, as time progresses and energy levels drop we notice that the different weighing functions result in a higher network latency, as they try to route messages along the paths with the most energy, instead of the paths with the best network performance. As is to be expected, the exponential weighing function performs the most aggressive energy management and as a result has the biggest impact on network latency. However even with this aggressive energy energy management the routing algorithm is able to deliver a very acceptable network performance, leading to only limited increases in latency. A graph comparing the latency for the different energy weighing functions over the network lifetime is available in figure 6.8.

Weighing function	Average lifetime
Regular QR (no energy)	228 (± 7)
QR + Linear energy function	253 (± 6)
QR + Steep linear energy func.	265 (± 6)
QR + Exponential energy func.	275 (± 5)

Table 6.2: Network lifetime for parent energy algorithm with different weighing functions in the small homogeneous topology.Confidence interval reported between brackets. Using energy aware routing can result into 21% better lifetime

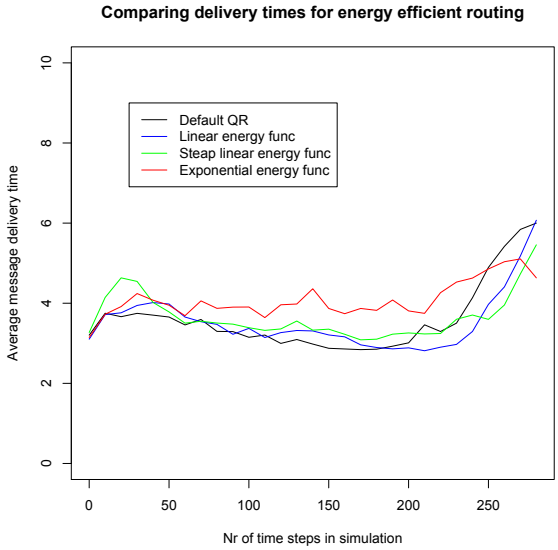

Figure 6.8: Delivery times for algorithms with different parent energy weighing function in the small homogeneous topology. Choosing a more energy efficient routing policy has only limited impact on delivery times. The most energy aware algorithm only performance slightly worse when compared to the regular Q-routing algorithm, with the average delivery times being increases by about two time steps.

Experimental setup:

- Initialisation: $Q_i^0(j) =$ Shortest path (Number of hops, Dijkstra algorithm)
- Learning rate: $\alpha = 0.5$
- Message generation: Average traffic load: Simultanious every 20 time steps
- Topology: Large homogeneous topology

We ran a similar experiment on our larger topology of 37 nodes. The results for earliest node dead are shown in table 6.3. The difference in lifetime between the different algorithms is somewhat smaller for this topology. This can partially be explained by the fact the topology consists from an unavoidable bottleneck. Only two links connect the left and right side of the topology, so the nodes near these links will always have to process a lot of traffic. Due to a lack of any other possible routes no algorithm can perform any actions to offload these nodes. Another important observation is that the delivery times suffered more significantly here, so choosing the exponential weighing function to extend the lifetime will result in notable higher

Weighing function	Average lifetime
Regular QR (no energy)	298 (± 6)
QR + Linear energy function	327 (± 5)
QR + Steep linear energy func.	333 (± 4)
QR + Exponential energy func.	342 (± 4)

Table 6.3: Network lifetime in large homogeneous topology with parent energy feedback. Most energy aware algorithm can extend lifetime up to 15%.

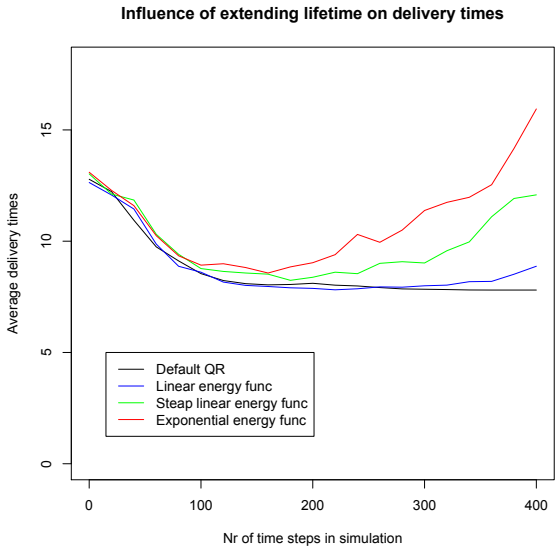

Figure 6.9: Delivery times for parent energy feedback algorithms on the large homogeneous topology. The energy aware algorithms deliver about the same performance, but as time progresses and energy levels drop, more aggressive power management starts to show its influence on network performance.

delivery times. Depending on the scenario, it might be favourable to choose a steep linear function instead of an exponential function. Using this function for an energy aware routing algorithm still results in a notable improvement in lifetime while latency suffers considerable less when compared to using an exponential function. An overview of the delivery times is available in the graph in figure 6.9.

6.3.2 Lowest energy on path

We test our algorithm with modified energy feedback in the same setup as the original energy feedback algorithm. On the small topology, we tested our modified energy feedback algorithms using the three different weighing functions. Results show that our modified version performs slightly worse than the original parent feedback algorithm. The summarised results are shown in table 6.4. When compared to table 6.2 it is visible that the parent energy feedback algorithm achieves a slightly better network lifetime. The delivery times for the modified energy algorithm are shown in figure 6.10 and are comparable to those of the parent energy algorithms.

Experimental setup:

- Initialisation: $Q_i^0(j) =$ Shortest path (Number of hops, Dijkstra algorithm)
- Learning rate: $\alpha = 0.5$
- Message generation: Average traffic load: Simultanious every 20 time steps
- Topology: Small homogeneous topology

Weighing function	Average lifetime
Regular QR (no energy)	228 ($\pm$ 7)
QR + Linear energy function	255 ($\pm$ 5)
QR + Steep linear energy func.	257 ($\pm$ 4)
QR + Exponential energy func.	264 ($\pm$ 3)

Table 6.4: Network lifetime for lowest energy on path feedback on the small homogeneous topology. The exponential energy function can extend the lifetime with about 15%.

The lifetime results for the large topology are provided in table 6.5, the results confirm the findings from the small topology. The lowest energy on path algorithm delivers a slightly shorter lifetime than the parent energy algorithm. Results for network performance are available in figure 6.11 and are in line with the findings from the small topology. Both parent energy and lowest energy on path algorithms deliver about the same network performance.

Experimental setup:

- Initialisation: $Q_i^0(j) =$ Shortest path (Number of hops, Dijkstra algorithm)
- Learning rate: $\alpha = 0.5$
- Message generation: Average traffic load: Simultanious every 20 time steps
- Topology: Large homogeneous topology

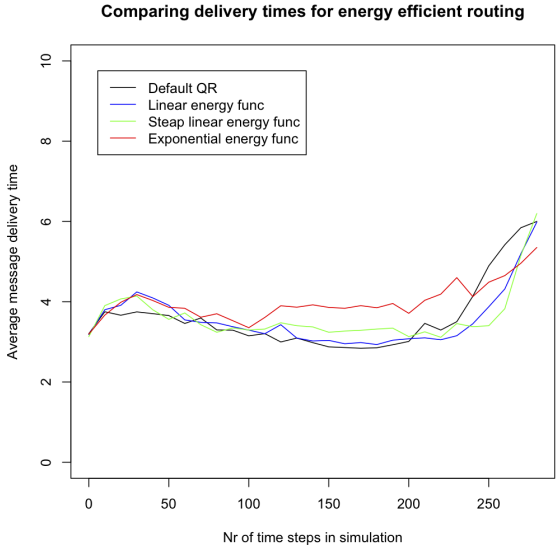

Figure 6.10: Delivery times for lowest energy on path feedback algorithms in the small topology. Energy aware algorithms using lowest energy on path deliver about the same network performance as energy aware algorithms using parent energy.

Weighing function	Average lifetime
Regular QR (no energy)	298 ($\pm$ 6)
QR + Linear energy function	304 ($\pm$ 4)
QR + Steep linear energy func.	305 ($\pm$ 3)
QR + Exponential energy func.	322 ($\pm$ 3)

Table 6.5: Network lifetime for lowest energy on path algorithm on the large homogeneous topology. The algorithm achieves only a limited performance improvement.

6.4 Heterogenous & irregular topologies

Al results in the previous sections report on experiments that were performed on topologies that were completely homogeneous. By homogeneous we mean that all nodes had identical processing capacities, battery capacities, energy consumption and so forth.

In this section we will focus on topologies where some nodes may have different specifications. Depending on the experiment we modified the energy capacity or message generation rate of some nodes. Results in this section also focus on comparing the two energy aware algorithms head to head, making it easy to compare the performance differences between both algorithms in similar conditions.

Large heterogeneous topology 1 For this experiment we modified the large homogeneous topology. The node marked with an X in figure 6.12 has been modified with a limited energy capacity that is about 30% of a regular node's energy capacity.

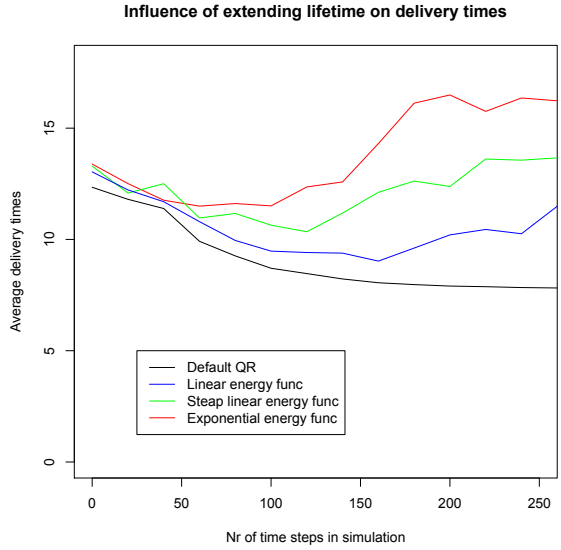

Figure 6.11: Delivery times for lowest energy on path algorithm on the large homogeneous topology. Network performance is comparable to that off the parent energy algorithm.

Experimental setup:

- Initialisation: $Q_i^0(j)$ = Shortest path, $E_i^0 = 100\%$ for regular nodes, $E_i^0 = 30\%$ for modified nodes.
- Learning rate: $\alpha = 0.5$
- Message generation: $p = 0.05$
- Topology: Large heterogeneous topology 1

We ran both the parent energy and the lowest energy on path variations of the energy aware algorithms). In this scenario the lowest energy on path algorithm is able to deliver a better lifetime, as it can propagate the energy information further into the network. This allows the lowest energy on path algorithm to offload nodes that are adjacent to the modified node. The results for the network lifetime of both algorithms, with all different weighing functions, are given in table 6.6.

Delivery times for energy aware algorithms are worse at start because they route traffic along a longer route in order to avoid the node with limited energy. The results are shown in table 6.13.

Large heterogeneous topology 2 In this experiment we use a large grid based topology shown in figure 6.14. During the experiment nodes marked with an X have a three times higher probability of generating a data packet for the sink. This causes a significant increase in energy consumption for these nodes which are located around the sink. Our experiments show that the lowest energy on path algorithm is able to increase the network lifetime significantly longer than the parent energy algorithm because it is able to offload the nodes which are adjacent to the modified nodes.

In table 6.7 we compare the achieved network lifetime for the shortest path algorithm and combinations of the shortest path algorithm with both energy metrics. By using a shortest

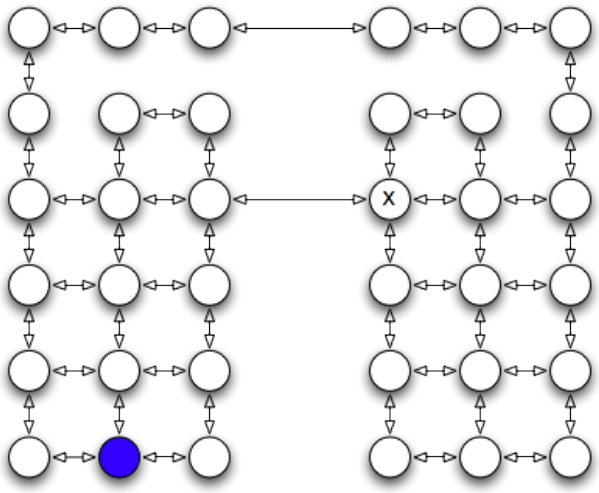

Figure 6.12: Large heterogeneous topology 1. The node marked with an X in the topology has only 30% of the normal energy capacity.

Energy weighing function	Parent energy	Lowest energy on path
Regular QR (No energy)	73 (±2)	73 (± 2)
Linear weighing	114 (± 4)	125 (± 5)
Steep linear weighing	141 (± 5)	156 (± 4)
Exponential weighing	181 (± 6)	194 (± 3)

Table 6.6: Lifetime for parent energy and lowest energy on path in large heterogeneous topology 1 (figure 6.12). By propagating energy information beyond the modified node's direct neighbours the lowest energy on path algorithm is able to further increase the network lifetime.

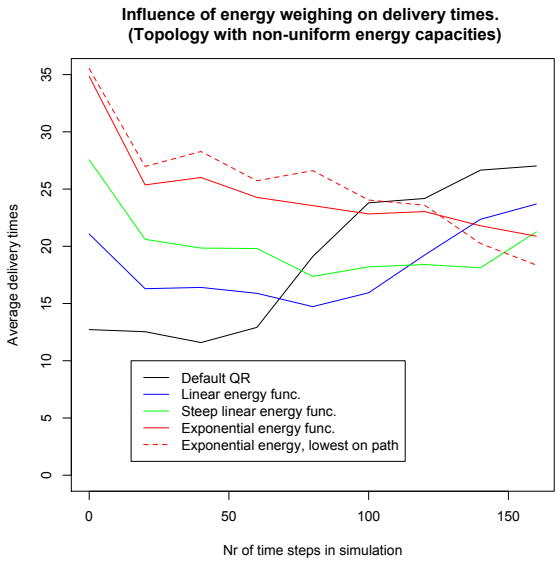

Figure 6.13: Network latency in heterogeneous topology 1 . Regular Q-routing performs significantly better in the early lifetime of the network because it ignores the nodes energy levels. However once it depletes the modified node's energ the algorithm looses a critical path to the sink and latency times suffer significantly.

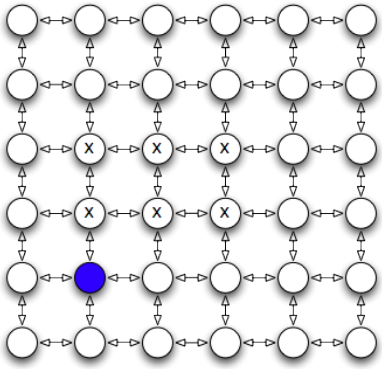

Figure 6.14: Large heterogeneous topology 2. All nodes have the same energy capacity. Nodes marked with an X are three times more likely to generate data packets than other nodes.

Algorithm	Lifetime
Shortest Path (SP)	304 ($\pm$ 6)
SP + Parent energy	315 ($\pm$ 4)
SP + Lowest energy on path	341 ($\pm$ 5)

Table 6.7: Network lifetime on large heterogeneous topology 2. Both energy algorithms used the exponential weighing function, as it always results in the best lifetime. Average time of death over 30 runs is reported together with 95% confidence interval.

path algorithm we prevent the delay metric from influencing the lifetime results. Because we only report on network lifetime we only report results from using the exponential weighing function, which always results in the best lifetime.

Experimental setup:

- Initialisation: $Q_i^0(j) =$ Shortest path, $E_i^0 = 100\%$ for regular nodes.
- Learning rate: $\alpha = 0.5$
- Message generation: $p = 0.05$ for regular nodes, $p = 0.15$ for modified nodes.
- Topology: Large heterogeneous topology 2

Table 6.8 shows the results of simulations when using the regular delay based Q-routing algorithm. Using load balancing already results in a significant improvement of the lifetime and the lifetime can be even further extended by using an energy aware algorithm, however the difference between both energy aware algorithms becomes much smaller.

Irregular topologies: When we use a ring based topology with heterogeneous nodes (figure 5.5) the lowest energy on path algorithm is able to propagate information about a weak node along the path, allowing for a perfect balance of traffic and a maximisation of the network lifetime. The results for the network lifetimes are shown in table 6.9.

In the bottleneck topology the lowest energy on path algorithm is able to deliver a significant improvement in the lifetime over the regular parent energy algorithm. The network lifetimes for this algorithm are available in table 6.10.

Network traffic	Low Load	High Load
Regular QR	413 (409 - 417)	306 (302 - 309)
Regular QR + Parent energy	430 (424 - 435)	324 (322 - 326)
Regular QR + Lowest energy on path	440 (433 - 445)	325 (323 - 327)

Table 6.8: Network lifetime on heterogeneous topology. Average time of death over 30 runs is reported together with 95% confidence interval.

Algorithm	Average lifetime
Delay QR + Parent energy	63 ($\pm$ 2)
Delay QR + Lowest energy on path	94 ($\pm$ 2)

Table 6.9: Lifetime for energy metrics in circular topology. Using lowest energy on a path in the circular topology can extend lifetime the lifetime an additional 39%.

Experimental setup:

- Initialisation: $Q_i^0(j)$ = Shortest path, $E_i^0 = 100\%$ for regular nodes, $E_i^0 = 30\%$ for modified nodes.
- Learning rate: $\alpha = 0.5$
- Message generation: $p = 0.05$
- Topology: Irregular topologies from figure 5.5

6.5 Reducing communication overhead

As a final contribution to the original work provided in this thesis provide a delta optimisation. This technique has been described in section 4.5 and attempts to increase the networks lifetime by only sending feedback messages when the reported feedback values have changed significantly since the previously transmitted feedback.

Because the algorithm needs to determine when a value has changed significantly, it requires a threshold in the algorithm that specifies when a feedback value has changed enough to warrant the transmission of a feedback message. This threshold is referred to as the x-value in the results.

Small homogeneous topology, $x = 0.02$ The optimised algorithm was tested on the small homogeneous topology. The results in table 6.15 compare the achieved lifetime for the optimised and the non-optimised algorithms. The optimised algorithm is able to achieve significantly better network lifetimes.

Algorithm	Average lifetime
Delay QR + Parent energy	136 ($\pm$ 6)
Delay QR + Lowest energy on path	174 ($\pm$ 3)

Table 6.10: Using lowest energy on a path in the bottleneck topology can extend lifetime 28%.

	Earliest failure	x = 0.02
Linear energy function	253	317 (± 8)
Steep linear energy func.	265	332 (± 7)
Exponential energy func.	275	346 (± 5)

Table 6.11: Lifetime for delta optimisations in the small homogeneous topology. Lifetime can be extended an additional 26% by using smart feedback messages.

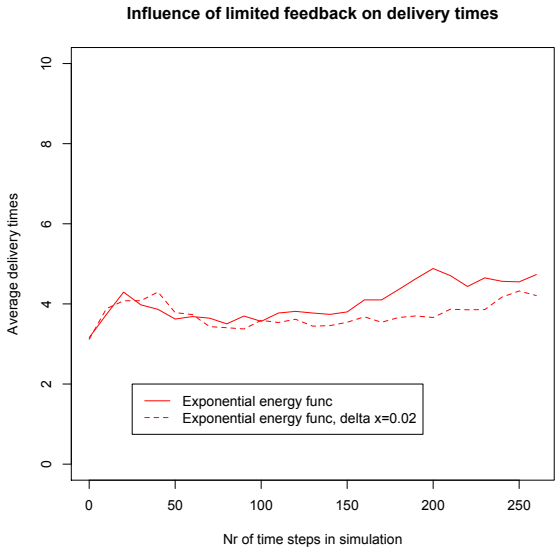

Figure 6.15: Network performance for the delta optimised algorithm in the small homogeneous topology. Comparing exponential energy function without and with delta updates. There is no significant drawback on delivery times.

Experimental setup:

- Initialisation: $Q_i^0(j) =$ Shortest path, $E_i^0 = 100\%$.
- Learning rate: $\alpha = 0.5$
- Message generation: Simultanious interval every 10 time steps.
- Topology: Small homogeneous topology

6.6 Summary

Before discussing the most important findings from our results in the next chapter we provide an overview of the results reported in this section. Table 6.12 provides lifetime and network performance for all combined algorithms on the small homogeneous topology. We observe that in this scenario, the best lifetime is achieved by the parent energy algorithm, while the best network performance is achieved by the delay based Q-routing algorithm. When combining both energy and delay metrics both energy algorithms result in about the same performance, however when using the lowest energy on path approach the algorithm achieves a slightly better network performance.

In table 6.13 we give a similar overview of results for the second large heterogeneous topology. On this topology the Lowest energy on path algorithm is able to deliver best lifetime. When combing metrics however combing the delay metric with the parent energy metric results in a slightly better performance while combing the delay and the lowest energy on path metric results in a slightly better network lifetime.

The discussion in the next chapter will address some notable differences in the results between the different types of topologies in more detail.

Algorithm	Average lifetime	Average latency	Total # messages
Shortest Path	191 (± 6)	4.94 (± 0.08)	222 (± 4)
Regular QR	245 (± 12)	3.37 (± 0.14)	293 (± 11)
Parent energy	317 (± 8)	4.04 (± 0.05)	378 (± 2)
Lowest energy on path	302 (± 7)	3.66 (± 0.12)	354 (± 4)
QR + Parent energy	291 (± 8)	4.33 (± 0.10)	346 (± 6)
QR + Lowest on path	290 (± 7)	3.82 (± 0.09)	342 (± 5)

Table 6.12: Overview of different algorithms on small homogeneous topology. Reported values are: Network lifetime (time of death of first node), average latency and total amount of messages received at the sink during the network lifetime. 95% confidence intervals reported between brackets.

Algorithm	Average lifetime	Average latency
Shortest Path	304 (± 6)	19.62 (± 0.21)
Regular QR	413 (± 4)	12.39 (± 0.54)
Parent energy	435 (± 4)	20.45 (± 1.00)
Lowest energy on path	474 (± 5)	23.60 (± 1.05)
QR + Parent energy	430 (± 6)	15.17 (± 0.46)
QR + Lowest on path	440 (± 6)	16.00 (± 0.32)

Table 6.13: Overview of different algorithms on large heterogeneous topology. Reported values are: Network lifetime (time of death of first node) and average travel time of the messages arrived at the sink. 95% confidence intervals reported between brackets.

Chapter 7

Discussion

This chapter discusses some of the results that are reported in the previous chapter in more detail.

7.1 Delay based QR: Offloading bottlenecks

A key characteristic of the regular Q-routing algorithm is that it attempts to distribute the traffic load over the topology. We discuss the influence of this behaviour on both the network performance and the network lifetime.

The images in figure 7.1 and figure 7.2 show the amount of messages that pass through each node based on an algorithm's routing behaviour.

Latency When we look at the way the nodes are spread on the small homogeneous topology and we take into account the nodes limited communication range, it becomes clear that the shortest routes to the sink suffers from higher traffic loads. The central node has to forward a lot of data from other nodes (7 messages per data run, including its own generated message). When we keep in mind that a node can only forward a single message per time step but that it can receive multiple messages per time step, we find out that some messages will lose time waiting to be processed in the queue of this centrally located node.

When we look at the routing behaviour of a delay based Q-routing algorithm after conversion, we notice that it is able to offload the centrally located node. By spreading some traffic along the outside of the topology, the average latency of the network improves significantly. Even if some messages have to travel an extra hop, they gain time by not having to wait in any queues.

This shows that we can considerably improve some of the longer delivery times by sending messages along routes that are actually a hop longer, but will still result in a faster delivery.

Lifetime Another important finding is that while it may seem more costly to add an extra hop to a path, it may still result in improved network lifetime. This might seem counter-intuitive, but by offloading a central node, we improve its lifetime. Instead we put an extra load on multiple other nodes. In an homogeneous network, all the nodes along the outside have the same energy capacity while they barely need to forward other messages besides their own. Because of this these nodes use much less of their available resources. So by routing traffic along longer paths that traverse these nodes, an algorithm will inevitably increase the

total energy cost for routing a message message, but this cost is payed by nodes with an excess of energy.

Conclusion In high load situations, a delay based routing algorithms is able to improve both the network latency and network lifetime. However having a lot of traffic is often undesirable in WSNs, so while this is a positive characteristic of Q-routing algorithms, it is also important for algorithms to deliver good results in scenarios with limited traffic. For this purpose it is desirable to include a metric that accounts for a nodes energy level regardless of its current capacity usage.

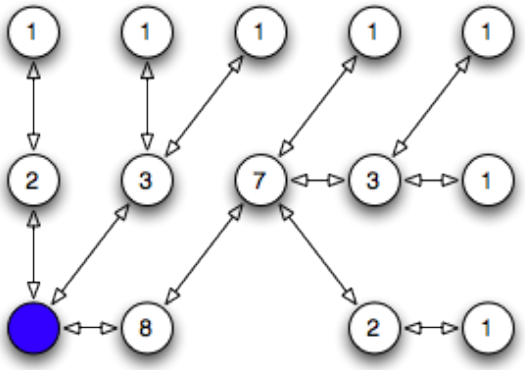

Figure 7.1: Node loads for small topology with shortest path routing.

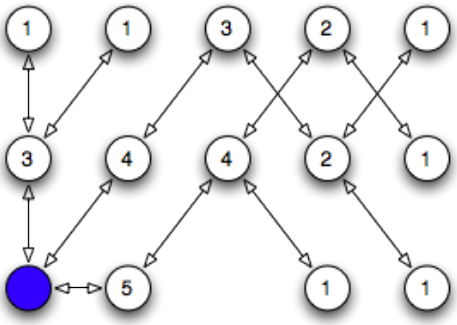

Figure 7.2: Node loads for small topology with Q-routing. Q-routing is able to offload the bottleneck along the outside of the network.

7.2 Initialisation techniques

Our experiments show that the Dijkstra initialised variation of the Q-routing algorithm performs significantly better than the constant initialised version during the initial exploration phase. However it is important to note that in simulations with average traffic, both variations need about the same amount of time steps before they start to converge to a stable policy. The main advantage for the Dijkstra initialised variation however is that its smart exploration allows it to deliver significantly better performance in the period of time before convergence.

One concern about this initialisation technique is that it might discourage the exploration of some routes that still might be interesting when we want to balance traffic load. But our experiments showed that the Dijkstra initialised Q-routing algorithm converged to the same routing policy as the constant initialised Q-routing algorithm in all tested scenarios.

From our experiments we can conclude that the cost of running a Dijkstra algorithm to initialise the estimated rewards is certainly worth the extra energy cost, even in densely connected networks where the cost of running the algorithm is at its highest. Our experiments show that the Dijkstra initialisation technique significantly improves the performance in the exploration phase before the policy converges. Furthermore the technique has no significant impact on the time the algorithm needs to converge to a stable policy or on the performance of the policy that the algorithm converges too.

7.3 Exploration and convergence

As shown in [Forster and Murphy, 2008] the need for an active exploration policy can be overcome by using energy metrics in the learning policy. In most scenarios, energy levels will gradually decrease over time. This will increase the Q-values and ensure a minimal level of exploration. Using an exponential energy function will ensure an increasing amount exploration as the networks energy level starts to worsen. Because of this constant exploration the routing behaviour will never totally converge. However because of the greedy parent selection the impact on network latency is limited as the algorithm will always consider the potentially most interesting alternative routes first, only exploring more high latency routes as the other routes start to deplete their energy.

7.4 Combining delay and energy metrics

In the introduction we explained how improving network latency and extending the lifetime of a network seem to be two conflicting goals. However, our experiments show that this is not always the case. In homogeneous topologies, when all nodes in a topology posses identical specifications and the topology layout is well connected, we observed that the optimal load balancing policy and the optimal energy balancing policy often only show minor differences. This becomes visible when we look at the results for Q-routing based algorithms in table 6.12. When we compare the differences in network lifetime and network latency for the different algorithms it is clear that these numbers all lie within a relatively close range off each other. This indicates that the different policies follow a somewhat similar routing policy.

Our algorithm which incorporates both delay and lowest energy on path energy metrics performed exactly as hoped, delivering performance closely to algorithms following the optimal load balancing policy while achieving lifetimes approaching those achieved by algorithms following an energy balancing policy.

These findings however do not negate our statement that network latency and network lifetime have conflicting goals. As our experiments on a large heterogeneous topology are an example of a scenario where there exists a much bigger difference between different routing policies. The results in table 6.13 show that an energy aware routing policy can extend the network lifetime an additional 20%, this however can have a significant impact on the network performance as this policy nearly doubles the network latency. In this topology, there is a clear gap between both energy balancing and traffic balancing policies. For this reason, this seems like a more suitable scenario for our algorithm incorporating both delay metrics and our own lowest energy on path metric. And indeed, results show that the algorithm is able to achieve an the extension of the network lifetime while keeping the increase in network latency under control.

Chapter 8

Conclusion and future work

In this chapter we summarise and explain the major observations found in this master thesis. We also elaborate on the opportunities that these observations open in regard to future research.

8.1 Parent energy vs. Lowest energy on path

One of the contributions in this master thesis is the modification of the Q-routing algorithm with an energy metric proposed in [Forster and Murphy, 2008]. The original algorithm is referred to as the parent energy algorithm while our modified version is referred to as the lowest energy on path algorithm.

A significant observation from all our experiments regarding the energy metrics is that there is no single algorithm which performs best in all of the test scenarios. In certain scenarios, the modified algorithm, based on the lowest energy on path approach, is able to increase the network lifetime beyond the lifetime achieved by the parent energy algorithm. This is especially the case in heterogeneous and irregular topologies and in these topologies the lowest energy on path algorithm performs significantly better than the parent energy algorithm, often resulting in lifetime extensions of 10% beyond those delivered by the parent energy algorithm. However in tests on multiple homogeneous topologies the original parent energy algorithm was able to deliver a slightly better performance than our proposed algorithm. This performance difference is mostly limited to about 3,5%

These observations lead us to two conclusions. First we can conclude that our modified lowest energy on path algorithm can be considered a viable alternative for the parent energy algorithm. In most homogeneous scenarios both algorithms deliver about the same lifetime, with the difference in lifetime never going beyond 3,5% in our performed experiments. In heterogeneous situations however, our modified algorithm can deliver better lifetimes, resulting in up to 10% improvement in lifetime depending on the scenario.

Our second conclusion is that based on the results from both algorithms, there still is room for improvement in the behaviour of routing algorithms. More research is needed in order to obtain a truly adaptive algorithm that can cope with both homogeneous and heterogeneous topologies.

8.2 Combining delay and energy metrics

The results of our early experiments show that applying energy metrics to a delay based Q-routing algorithm results in an algorithm that is able to achieve a better network lifetime while maintaining a high level of network performance. Even if the size of the improvements in network lifetime depends highly on the topology, adding an energy metric to the delay based Q-routing algorithm will always result in a better network lifetime. In some cases the addition of an energy metric to the Q-routing algorithm results in only a limited improvement of the network lifetime, this can be attributed to the fact that in some topologies, the optimal load balancing policy highly resembles the optimal energy balancing policy. When this is the case, adding an extra energy metric can only result in a limited increase of the network lifetime, as the load balancing policy already approaches a near optimal solution for the energy policy. This is mostly prevalent in densely connected homogenous topologies. The similarity between load balancing and energy balancing policies is also visible when we look at the network performance. A shortest path algorithm combined with an energy metric is able to achieve an excellent network lifetime while achieving a good network performance. When we replace the shortest path algorithm with a delay based Q-routing we notice that the algorithm achieves only a minor improvement in network performance. This is again due to the fact that the optimal energy policy closely resembles the optimal load balancing policy.

The results from our experiments on larger heterogeneous topologies show a bigger different in the policies of load balancing and energy balancing algorithms, leading to a bigger gap between the lifetime and performance delivered by both policies. These scenarios create a bigger opportunity for our algorithm, since it's ability to offer a balanced routing behaviour helps to bridge the gap between either a lifetime maximising or performance maximising policy.

These observations lead us to conclude that by combining both energy and traffic delay metrics our Q-routing algorithm is successful in balancing both network performance and network lifetime. In scenarios where the gap between energy and performance based policies is limited, the lifetime results for our combined algorithm closely approach the optimal results from an algorithm relying solely on energy metrics while the network performance approaches the latencies achieved by a load balancing Q-routing algorithm. In scenarios where there exists a gap in the performance of energy balancing policies and load balancing policies, the algorithm proposed in this master thesis is able to learn a routing policy that extends the lifetime of the network while maintaining a good level of network performance.

8.3 Future work

8.3.1 More heterogeneous networks

While we spent a significant amount of time with experiments on what we call heterogeneous networks, it is important to realise that this level of heterogeneity can still be increased further. An interesting opportunity for this lies in adding heterogeneity to the types of messages that are transmitted across the network. We can imagine a scenario where messages of certain types would require priority over other messages. Because of the fact that our proposed algorithm combines multiple metrics, we could modify the algorithm in such a way that each metric would be assigned a weight and that different types of messages would result in different weights for the metrics. Low priority messages could be routed along

energy efficient routes, while high priority messages could be sent along the fastest route. This design allows for an algorithm which offers a user the possibility to easily enforce a customisable level of Quality of Service.

8.3.2 Real word simulation

All the results discussed in this master thesis are obtained from discrete simulations. While the simulator uses a lot of randomisation to better resemble real world scenarios, it remains a heavily simplified representation of real word scenarios. In the simulator algorithms have a constant access to a 100% accurate energy level for each node. In a lot of real word platforms, this information is less accurate. Also, all operations of sending and receiving messages are coupled to a fixed energy cost in the simulator. This means that sending or receiving multiple messages results in a linear increasing total energy cost for each extra message. In real life platforms it is possible that certain levels of hardware optimisation allow for multiple messages to be sent or received without suffering from a linear extra cost per message. A property like this could have a possible impact of the performance delivered by an algorithm using energy metrics.

An extra step between our discrete simulator and a real word simulation is available in the TOSSIM environment. This software environment allows software designed for the TinyOS platform to be tested before deployment. The TOSSIM environment is a much more accurate simulator by, among other improvements, taking into account different noise models for radio communication. The downside of this more accurate simulation however is that it is very time and resource consuming, making it impossible to test algorithms on a topology of more then about 12 nodes.

Bibliography

[Akyildiz et al., 2002] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422.

[Boyan and Littman, 1994] Boyan, J. A. and Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems 6*, pages 671–678. Morgan Kaufmann.

[Choi and yan Yeung, 1996] Choi, S. and yan Yeung, D. (1996). Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In *In Advances in Neural Information Processing Systems 8 (NIPS8*, pages 945–951. MIT Press.

[Eberhart et al., 2001] Eberhart, R. C., Shi, Y., and Kennedy, J. (2001). *Swarm Intelligence (The Morgan Kaufmann Series in Evolutionary Computation)*. Morgan Kaufmann, 1st edition.

[Egiriva-Forster and Murphy, 2007] Egiriva-Forster, A. and Murphy, A. L. (2007). A feedback enhanced learning approach for routing in wsn. In *Proceedings of the 4th Workshop on Mobile Ad-Hoc Networks (WMAN). Bern, Switzerland*. Springer-Verlag.

[Fakir, 2004] Fakir, M. (2004). *Resource optimisation methods for telecommunication networks*. PhD thesis, Vrije Universiteit Brussel.

[Fonseca et al., 2006] Fonseca, R., Gnawali, O., Jamieson, K., and Levis, P. (2006). *TEP123, the collection tree protocol*.

[Forster, 2007] Forster, A. (2007). Machine learning techniques applied to wireless ad-hoc networks: Guide and survey. In *Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. ISSNIP.

[Forster and Murphy, 2008] Forster, A. and Murphy, A. L. (2008). Balancing energy expenditure in wsns through reinforcement learning: A study. In *Proceedings of the 1st International Workshop on Energy in Wireless Sensor Networks WEWSN*, page 7pp. Citeseer.

[Gnawali, 2006] Gnawali, O. (2006). *TEP124„ The Link Estimation Exchange Protocol (LEEP)*.

[Hedetniemi et al., 1988] Hedetniemi, S. M., Hedetniemi, S. T., and Liestman, A. L. (1988). A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349.

[Heinzelman et al., 1999] Heinzelman, W. R., Kulik, J., and Balakrishnan, H. (1999). Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 174–185, New York, NY, USA. ACM.

[Intanagonwiwat et al., 2003] Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., and Silva, F. (2003). Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11:2–16.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH*, 4:237–285.

[Kassabalidis et al.,] Kassabalidis, I., El-Sharkawi, M., Marks, R., II, Arabshahi, P., and Gray, A. Swarm intelligence for routing in communication networks. In *In IEEE Globecom*.

[Krishnamachari et al.,] Krishnamachari, B., Estrin, D., and Wicker, S. Modelling data-centric routing in wireless sensor networks.

[Langendoen, 2007] Langendoen, K. (2007). *Medium Access Control in Wireless Networks*. Nova Science Publishers.

[Levis et al., 2004] Levis, P., Patel, N., Culler, D., and Shenker, S. (2004). Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pages 2–2, Berkeley, CA, USA. USENIX Association.

[Li and Halpern, 2001] Li, L. and Halpern, J. Y. (2001). Minimum-energy mobile wireless networks revisited. In *In Proc. IEEE International Conference on Communications (ICC*, pages 278–283.

[Liu and Elhanany, 2006] Liu, Z. and Elhanany, I. (2006). Rl-mac: a reinforcement learning based mac protocol for wireless sensor networks. *Int. J. Sen. Netw.*, 1:117–124.

[Mihaylov et al., 2011] Mihaylov, M., Borgne, Y.-A. L., Tuyls, K., and Nowe, A. (2011). Desyde: Decentralized (de)synchronization in wireless sensor networks. In *Proceedings of the 20th Machine Learning conference of Belgium and The Netherlands (BENELEARN'11)*.

[Morris et al., 2004] Morris, R. T., Smith, A. C., Couto, D. S. J. D., and Couto, D. S. J. D. (2004). High-throughput routing for multi-hop wireless networks.

[Nguyen et al., 2011] Nguyen, H. A., Forster, A., Puccinelli, D., and Giordano, S. (2011). Sensor node lifetime: An experimental study. In *Pervasive Computing and Communications Workshops (PERCOM Workshops)*.

[Nowé et al., 1998] Nowé, A., Steenhaut, K., and Fakir, M. (1998). Q-learning for adaptive, load based routing. In *In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, San Diego, California, USA, 1998.*, pages 3965–3970. IEEE.

[Römer and Mattern, 2004] Römer, K. and Mattern, F. (2004). The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61.

[Rutgers, 1991] Rutgers, C. L. H. (1991). An introduction to igrp. The State University of New Jersey, Center for Computers and Information Services, Laboratory for Computer Science Research.

[Schurgers and Srivastava, 2001] Schurgers, C. and Srivastava, M. B. (2001). Energy efficient routing in wireless sensor networks. In *Proceedings of the IEEE Military Communications Conference, vol. 1*, pages pp. 357 – 361. ISSNIP.

[Sohrabi et al., 2000] Sohrabi, K., Gao, J., Ailawadhi, V., and Pottie, G. J. (2000). Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7:16–27.

[Sun et al., 2002] Sun, R., Tatsumi, S., and Zhao, G. (2002). Q-map: a novel multicast routing method in wireless ad hoc networks with multiagent reinforcement learning. In *TENCON 02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, pages 667–670. IEEE.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

[Tanenbaum, 2003] Tanenbaum, A. S. (2003). *Computer networks*. Prentice Hall Professional.

[Tiwari et al., 2007] Tiwari, A., Ballal, P., and Lewis, F. L. (2007). Energy-efficient wireless sensor network design and implementation for condition-based maintenance. *ACM Trans. Sen. Netw.*, 3.

[Zhiyu and Haoshan, 2007] Zhiyu, L. and Haoshan, S. (2007). Design of gradient and node remaining energy constrained directed diffusion routing for wsn. In *Proceedings of the international conference on Wireless Communications, Networking and Mobile Computing*, pages 2600–2603. WiCom 2007.

[Zimmermann, 1988] Zimmermann, H. (1988). Innovations in internetworking. chapter OSI reference model. The ISO model of architecture for open systems interconnection, pages 2–9. Artech House, Inc., Norwood, MA, USA.