FROMS: A Failure Tolerant and Mobility Enabled Multicast Routing Paradigm with Reinforcement Learning for WSNs

ANNA FÖRSTER
University of Lugano, Switzerland
and
AMY L. MURPHY
FBK-IRST, Trento, Italy

University of Lugano
Faculty of Informatics
Technical Report No. 2009/004
June 2009

Abstract

A growing class of wireless sensor network (WSN) applications require the use of sensed data inside the network at multiple, possibly mobile base stations. Standard WSN routing techniques that move data from multiple sources to a single, fixed base station are not applicable, motivating new solutions that efficiently achieve multicast. This paper explores in depth the requirements of this set of application scenarios and proposes, FROMS, a machine learning-based approach. The primary benefits are the flexibility to optimize routing on a variety of properties such as route length, battery levels, etc., ease of recovery after node failures, and native support for sink mobility. We provide extensive simulation results supporting these claims, clearly showing the benefits of FROMS in terms of low routing overhead, extended network lifetimes, and other key metrics for the WSN environment.

1 Introduction

The 1998 SmartDust [1] project is commonly used to mark the beginning of wireless sensor network (WSNs) research, as it identified the vision for large autonomous networks for monitoring environmental and industrial parameters. Since then the price of individual sensors has been decreasing, while memory, processing and sensory abilities have been increasing, simultaneously expanding the potential application scenarios. Researchers and practitioners from many scientific and industrial areas have already leveraged the achievements of the WSN community, deploying of sensor networks with applications ranging from scientific monitoring of active volcanos [2] and glaciers [3], through agricultural monitoring [4], military and rescue applications[5, 6], to the futuristic vision of the InterPlaNetary Internet [7, 8], designed to connect highly heterogeneous devices such as satellites, Mars and Moon rovers, sensor networks, space shuttles, and common handheld devices and laptops into one holistic network.

The growing number of applications for WSNs and especially their heterogeneous requirements and properties demand new communication protocols and architectures. Routing for WSNs has attracted a lot of research in recent years, and many different protocols has been developed for various application scenarios and data traffic schedules. However, lately this area has attracted extensive criticism: application scenarios are too restricted or not carefully described, experimental setups are unrealistic, and simulation environments are too abstract [9]. Further, despite the overwhelming number and variety of routing protocols, key problems remain unsolved, importantly energy efficiency in various application scenarios and for multiple traffic patterns, and tolerance against failures and mobility has not been sufficiently addressed. Additionally, the problem of sending data to multiple, possibly mobile sinks via optimal paths (multicast) has not been solved efficiently.

This paper presents a novel multicast routing protocol called FROMS (Feedback ROuting to Multiple Sinks), which exploits reinforcement learning. Our target scenario includes any applications with periodic or long-lasting sensory data reporting to multiple, mobile sinks in a multi-hop environment. FROMS easily accepts different cost metrics such as hops, latency, remaining battery etc.. Its most salient advantages are: the ability to find globally optimal multicast routes; to incorporate different cost metrics and thus optimization goals; and to quickly recover in case of failures and sink mobility. The main goal of FROMS is to provide the WSN developer with *one* routing solution, able to be tuned to many different application scenarios.

This paper presents a comprehensive view of Froms, including a theoretical model and an analysis of its complexity and overall behavior; a complete evaluation both in simulation and on real hardware; and a challenging comparison against geographic based multicast routing protocol MSTEAM [10] and a multicast variation of Directed Diffusion [11]. The presented simulation environment uses sophisticated radio propagation models and realistic MAC protocols. In contrast to our previously reported results [12, 13], this paper offers significantly more depth to the characterization of the Froms parameter space and its properties, and a complete comparison to other multicast routing protocols both in simulation and on real hardware.

Next, Section 2 motivates the work and our approach, describing major challenges and related works. Section 3 gives an intuitive introduction to the Froms routing protocol, before Section 4 models multicast routing as a reinforcement learning problem and presents our solution. Section 5 makes a theoretical complexity and convergence analysis. Section 6 proceeds with defining the Froms protocol and discussing implementation details, before Sections 7 – 10 present our evaluation environment and discuss the simulation and testbed experiments. Finally, Section 11 outlines future research directions and challenges.

2 Motivation and related efforts

In the next paragraphs we concentrate on the requirements and properties of well-known various WSN deployments and use them for defining our own target application scenario. Then we discuss the current state of the art of WSN routing protocols and how they meet the needs and challenges of our identified scenario.

2.1 Target application scenario

Real deployments of wireless sensor networks usually implement one of three general applications: periodic reporting, event detection [14], and data-base like storage [15]. Periodic reporting is by far the most used and simplest application scenario: at regular intervals the sensors sample the environment, store the sensed data, and send it further to the base station(s). Actuators are often directly connected with those sensor networks, for example automatic irrigation systems or alarm systems.

In this work we consider periodic reporting scenarios, since they make up the major part of current and future WSN deployments. The main property of periodic reporting applications is the predictability of the data traffic and volume. More precisely, we consider wireless sensor network applications to disaster relief and military operations [5, 6], environmental monitoring and surveillance [2, 16, 17, 4, 18] and the InterPlaNetary Internet [7, 8]. Although these scenarios are very different in their nature and goals, they share a lot of properties. In the next paragraphs we derive the properties of the application scenario for our routing protocol.

1. *Network size.* The sample WSN deployments we use in this work span a wide variety of network sizes and densities. Some of them are randomly deployed with hundreds of nodes (e.g. military or disaster recovery applications [5, 6]), others are thoroughly planned and include only few to a dozen of nodes (volcano monitoring [2]).

   Thus, we conclude that the number of nodes is unknown and can vary from only several nodes to hundreds or even thousands randomly organized into a multi-hop topology.

2. *Energy restrictions.* One of the main challenges of wireless sensor networks are the highly restricted power reserves of the sensor nodes. They typically have on-board low capacity batteries, which are used for sensing, processing and communication. However, the primary power consumer is the radio [19, 20], which drains the node's battery quickly for active listening of the wireless medium and data transmission. In addition, many WSN deployments need to run unattended over weeks or even months and batteries cannot be replaced. This is the case, for example, for disaster relief operations [5] or for sensor networks as part of the InterPlaNetary Internet [7, 8]. On the other hand, failing of some sensor nodes might disconnect the network and stop data delivery. This event is often referred to as network death. Thus, one of the major design goals and requirements for data dissemination protocols is the efficient use of energy reserves and network life prolongation through on-board optimization and node-wide balancing of communication overhead.

3. *Node failures.* Node failures are a direct consequence of the limited energy availability on the nodes. With dwindling battery reserves, the

node's behavior becomes first very unreliable in terms of communication and then the node fails completely. In unattended environments the node will never recover. However, in agricultural monitoring [4, 18] exchange of batteries is possible and the node will re-enter the network. Node failure or restart can happen also for other reasons, for example because of loose contacts, defect hardware or bad environmental conditions. A data dissemination framework needs to cope well with all these events and to guarantee continuous data delivery during the full network lifetime. It also needs to accommodate new nodes to make efficient use of all network resources.

4. *Sink mobility.* Sensor nodes in all our sample applications are usually simple, static entities. Current deployments often plan only one fixed base station. However, this approach has various drawbacks: the base station is a single point of failure and other data consumers in the sensor network have to retrieve the data directly from the base station. The second argument is often considered an inconvenience rather than a real risk. However, imagine a disaster relief scenario as described in [6], where a sensor network has been deployed to observe the environment, estimate risks and discover people. The rescue workers are equipped with wireless handheld devices, which usually are able to communicate with the base station (the emergency habitat). In the "normal" situation they can get sensory data from it directly. However, what happens when they move around and their handheld devices go out of range of the base station? Usually no functioning infrastructure is available to ensure communication. In such cases the sensor network itself can take over the communication among the sensor network, the base station and the rescue workers. The consequence for data dissemination protocols is that multiple mobile sinks are present in the network.

   Nearly the same situation arises in the InterPlaNetary Internet [7, 8]. For environmental monitoring the need of mobile sinks is not that urgent, but it would be helpful to unobtrusively replace the base station in case of failure or to receive the data directly from the sensor network in case the used device has no access to the base station.

   Thus, the routing protocol needs to support mobile sinks and to be able to route data between heterogenous devices considering non-uniform costs of the links.

5. *Data generation, delivery and traffic.* Usually there are many different data types available in a sensor network, e.g. temperature, humidity, light, gas concentration, acceleration. Sinks need to be able to choose between different data types, data sensing intervals, reporting intervals, compression parameters, etc. The sensing and reporting can be continuos or temporary. The achievable throughput of a network depends mostly on the Medium ACcess (MAC) protocol in use. The contribution of the data dissemination protocols to managing data traffic is to generate as few packets as possible. This lowers the overall latency, and increases the delivery rate and reliability. At the same time, sinks' requirements on data quality need to be met (see next point).

   We assume that a suitable MAC protocol is used and the volume of data traffic can be anything between few readings from a single node to a single sink to all nodes reporting to several sinks.

6. *Quality of service requirements.* In addition to the data requirements above, the sinks have also quality of service requirements. Different applications have different requirements. For example, disaster relief operations [5] need reliable minimum delay delivery of sensory data for ensuring fast response. In contrast, agricultural monitoring [4] is a delay-tolerant application where efficient energy use and long network lifetimes are more important to keep maintenance effort and costs low.

   In summary, the designed routing protocol needs not only to support all of these quality of services requirements, but to be able to switch between them quickly and efficiently. The most important requirements are support of minimum delay, minimum energy expenditure, and high reliability (delivery rate).

Additionally, there are some important design criteria concerning the quality and the credibility of the conducted work. Unlike the requirements outlined above, which arise directly from the described deployments and applications, the design criteria and their fulfillment are important for practitioners in the area and other researchers. They guarantee the real world applicability of the implemented routing protocol.

- *Simplicity.* The protocols must be easy to understand and implement, in order to be feasible for real-world deployments.

- *Memory and processing requirements.* The implementation must fit comfortably onto a typical sensor node, leaving space for other protocols and applications.

- *Flexibility.* The protocol must be easily adaptable to different applications and optimization goals.

- *Scalability.* The implemented protocols must be scalable in terms of network size, number of sources, and number of sinks.

In order to design and implement the routing protocol, we need to make some assumptions about the rest of the communication stack:

1. *Sink announcements (data requests).* We assume that sinks announce themselves via a network-wide broadcast in which they state their optimization goal and data requirements. During this announcement, the nodes in the network are able to gather some initial routing information and to calculate in a localized manner their cluster membership. Propagating sink announcements is a very common approach in WSNs.

2. *MAC layer.* Routing protocols rely heavily on the lower layer protocols' performance. We consider a simple broadcast-enabled MAC protocol without re-transmissions and without delivery guarantee, basically any sensor network MAC protocol.

3. *Neighborhood management.* We do not assume any neighborhood management protocol - the neighbors' reliability and quality needs to be managed by the routing and clustering protocols directly, in order to be able to manage failures and mobility in an efficient and holistic way.

This section presented and analyzed the most important application requirements for this work. In summary, our routing protocol needs to cope with different network sizes, multiple mobile sinks, failing nodes, restricted energy reserves, and various data and quality of service requirements.

Our first intuition is that machine learning seems a good choice for solving the above problems in an autonomous, self-organized, and energy-efficient way. In the next Section 2.2 we will explore related efforts on multicast routing for WSN and machine learning approaches for routing in WSNs.

2.2 Application scenario challenges and related works

While a large body of different routing protocols [5] has emerged in the last years, there is still no general and well-performing routing protocol for WSNs. Real deployments often decide for a simple, already implemented routing protocol based on hops like MintRoute [21] for TinyOS. However, they often also change the protocol according to their needs [16, 4, 2], for example by using a different neighborhood management protocol or a custom cost metric. Thus, the resulting protocols are highly specialized and optimized solutions for the targeted network rather than a standard protocol for a broad variety of scenarios.

Multicast routing for WSNs. Many routing protocols have emerged from routing protocols for Mobile Ad Hoc Networks (MANETs). They build a full routing path table at all nodes and each node keeps the full route to each possible destination. The main disadvantage of such an approach is that route information needs to be propagated throughout the network (from the source to the destination and back). Second, a complicated route repair procedure needs to be started in case of topology changes or failures to re-build the routes. There are specialized multicast routing MANET routing protocols, like MAODV [22], LAM [23], and ADMR [24]. Mesh-based routing protocols are a popular solution to multicast routing too, for example ODMPR [25] and PUMA [26]. They proved to be very efficient in high mobility scenarios, but cause great communication overhead for constructing and maintaining the mesh and thus cannot be successfully applied to WSNs. Such experiences were reported by various researchers while implementing MANET routing protocols for WSNs, like the implementation of ADMR on MicaZ motes [27]. There are some recent works using swarm intelligence [28] (see the next section), but again the overhead from sending ants is unbearable for wireless sensor networks.

Location-based (or geographic) network routing is based on the location-awareness of the nodes. Traditional geographic routing protocol is GPSR [29], which selects next hops based on their progress to the destination. In case the routing is stuck (a node is reached with no progress to the sink), a special face routing procedure is started to route the packet around the void region. GMR [30] and MSTEAM [10] are both geographic based multicast routing protocols. The main disadvantage of geographic routing protocols is the length of the selected routes, especially in case of void regions. Another problem with traditional geographic routing schemes is their preference of long unreliable hops. In case no separate link protocol is used, geographic routing selects next hops only based on their progress to the sink - thus, mostly long lossy connections. An extensive study of this problem and a comparison of various other location-based metrics on simulation and real hardware is presented in [31].

Another approach for WSNs for multicasting is what we call "fake multicast": unicast protocols, which are slightly optimized for multicast routing. Such protocols just build paths from a source to each of the sinks without really considering sharing of paths or finding globally optimal ones. For example, Directed Diffusion [11] is a very popular and powerful routing paradigm, where routes from the source to the destinations are established on-demand based on interests that are flooded through the network. This flooding establishes gradients for data to follow from multiple sources to the sinks. It can be easily extended to multiple sinks, but the resulting multicast routes are not optimal. Nevertheless, Directed Diffusion has inspired a lot of other routing protocols for WSNs, like Rumor routing [32] or GRE-DD [33]. MintRoute [21] from TinyOS[1] is very similar to Directed Diffusion, but includes also a neighborhood management protocol.

Sink mobility management in WSNs. Some routing protocols assume that the mobility pattern of the sinks is known a-priori at the sensor nodes. One such protocol is the spatiotemporal mobicast routing algorithm in [34]. This protocol is rather an overlay routing protocol, which decides *when* to forward the data through a geographic routing protocol to which neighbors. In this way it guarantees spatiotemporal delivery of needed data to needed regions.

TTDD [35] is a layered routing protocol, developed especially for high mobility scenarios. The authors concentrate on efficient delivery to multiple mobile sinks through building a routing overlay. The network is clustered into cells and mobile sinks flood their requests in the local cell only. Thus, the overlay is always aware of the current position of the sinks and routes the data to them. This approach proved to be very effective in high mobility scenarios. However, the nodes building the overlay (a cell structure) drain their power quickly and the overlay has to be rebuilt with high communication overhead. Thats why the protocol is better suited for event-detecting sensor networks

[1] www.tinyos.org

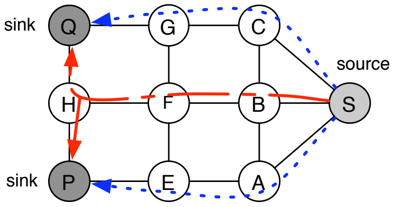
with only sporadic traffic rather than continuous monitoring.

SEAD [36] and its successor DEED [37] optimize routing from single source to multiple mobile sinks. Each sink selects an "access sensor node", to which data from the source is routed. A tree is built based on a geographic location heuristic between the source and all access nodes. When the sink moves away, a path between its current nearest neighbor and the access node is maintained, so that it is not necessary to rebuild the tree. If the sink moves too far away, a new access node is selected and the tree is rebuilt, but only with high communication overhead. The approach shows very good results compared to Directed Diffusion [11] or TTDD [35] in terms of dissipated energy for data packets. However, no extensive evaluation of the control overhead under mobile sinks is presented, which is expected to be high. An analytical evaluation of virtual infrastructure routing protocols (TTDD [35], SEAD [36] and others) is presented in [38].

2.3 Machine Learning applications to routing in WSNs

Machine learning has gained a lot of attention in latest years for solving hard problems in wireless ad hoc networks such as routing [39, 40, 41]. In WSNs, reinforcement learning (RL) has been already applied to point-to-point routing in different settings - to support geographic routing [42]; for discovering routes between two nodes [43], for finding optimal compression routes between many sources and one sink [44] etc. All of these works show the great advantage of using ML techniques for routing. However, to the best of our knowledge, there are no works on applying machine learning (especially RL) to multicast routing, which requires changes to the original ML algorithms, while keeping their advantageous properties.

Another well known machine learning algorithm for routing is swarm intelligence and especially ant colony optimization. It has been applied to routing in ad hoc networks in AntHocNet [28] and for point to point routing in WSNs, but with less success. The main challenge is overcoming the communication overhead caused by the traveling ants. These algorithms are well suited for highly mobile, energy-rich domains like MANETs and less for energy-restricted, but rather static environments like WSNs.



routing table : node S

Neighbor A	sink P	3 hops
	sink Q	5 hops
Neighbor B	sink P	4 hops
	sink Q	4 hops
Neighbor C	sink P	5 hops
	sink Q	3 hops

Figure 1: A sample topology with 2 sinks, the main routes to them from source S and its initial routing table.

3 Protocol intuition and overview

The goal of our protocol is to find the optimal possible path for data to follow from its source to all interested sinks. Optimal can be defined as either minimum delay, minimum hop count, minimum geographic distance, maximum remaining batteries or a combination of some of the above. Here, we will use number of hops as an example.

Consider the sample network from Figure 1 with one source and two sinks. One possible path from the source to the sinks is formed by the union of the individual paths from the source to each sink (the dotted lines in the figure), however a shorter path often exists. This shorter path takes the form of a tree, as the one through nodes B, F and H. The challenge is to globally identify this tree without full topology information and using only local information exchange. The main task of our protocol is to update local information regarding "next-hops" to reach sinks from each node such that the cost of the resulting tree is optimal.

During an initial sink announcement phase, as proposed in Section 2.1, all nodes gather some initial routing information and register known sinks in the network. In our example from Figure 1 node S gathers hop information for each sink *individually* as shown in its routing table in the figure. When data packets arrive at the node for routing, the node needs to select one or more next hops towards the sinks. However, instead of simply choosing the best looking one (in this example: node C for sink Q and node A for sink P), it also *explores* non-optimal routes in the assumption that some of them might have lower costs than in its own routing table. This is because its neighboring nodes may be able to share next hops too. For example, the source node S estimates that node A needs 7 hops to reach both sinks: 5 hops to sink P, 3 hops to sink Q and the first hop is shared, thus the minus

1 or a total of 7 hops. However, node S does not know whether node A will be able also to share the next hop or will need to split the packet and send it through two different neighbors. In our example, node A is in fact able to share the next hop. It calculates that it can reach the sinks through node E (see the routing table of node A in the figure) in $(2+4)-1=5$ hops. Thus, node S will be able to reach both sinks in 1 hop to node A plus 5 hops from node A to all sinks or a total of 6 hops, which is 1 hop less than the initial information on the source node. Thus, node A needs to inform node S about its own estimation of the costs to both sinks. It can do so while sending the data packet further to the sink by making use of the broadcast environment and piggybacking its own cost estimation.

Similarly, node E piggybacks its cost estimation and informs node A and so on. There are four important observations to make: these piggybacked values, which we also call feedbacks, propagate exactly one step back until they reach the sinks, where the packet stops. Thus, the source needs to send several data packets to node A before its own cost estimation for node A represents the real hop cost of the route.

Second, the source needs to send data packets not only to node A, but to all neighboring nodes a sufficient number of times, before all of its cost estimations converge. The neighbors of the nodes need to also *explore* their neighbors and so on. Third, feedback can be used not only by the previous hop, but by all overhearing nodes of the transmitter and thus deliver additional information to the nodes. And fourth, keeping all of the routes at all nodes and always giving feedback to the neighbors with the current cost estimations, innately handles recovery and mobility. For example, in case node E fails, node A will switch to another route, for example through node B, will update its cost estimations and will inform the source S via feedback on the next data packet about its current costs. The information propagates together with the data packets, without incurring any additional communication overhead and update automatically the routes and their costs on all involved nodes.

The above made observations form a reinforcement learning based routing protocol. In the next section we formalize the ideas discussed here and present the details of the Q-Learning model, solving the multicast problem in WSNs.

4 FROMS: Solving Multicast with Q-Learning

The main goal of this section is to model the multicast routing problem and solve it with reinforcement learning, as sketched in the pervious section. This will not only build the basis of our protocol, but also give us the possibility to make a theoretical analysis of the protocol in terms of complexity, correctness and convergence.

4.1 Problem definition

We consider the network of sensors as a graph $G = (V, E)$ where each sensor node is a vertex v_i and each edge e_{ij} is a bidirectional wireless communication channel between a pair of nodes v_i and v_j. Without a loss of generality, we consider a single source node $s \in V$ and a set of destination nodes $D \subseteq V$.

Optimal routing to multiple destinations is defined as the minimum cost path starting at the source vertex s, and reaching all destination vertices D. This path is actually a spanning tree $T = (V_T, E_T)$ whose vertexes include the source and the destinations. The cost of a tree T is defined as a function over its nodes and links $C(T)$. For example, it can be the number of one-hop broadcasts required to reach all destinations or in other words the number of non-leaf nodes in T. Further cost functions are presented in Section 6.8 and evaluated in Section 9.3.

4.2 Multicast Routing with Q-Learning

Finding the minimum cost tree T, also called the Steiner tree, is NP-hard, even when the full topology is known [45]. Our goal, therefore, is to approximate the optimal solution using localized techniques. We turn to reinforcement learning and especially to Q-Learning [46].

In our multiple-sink scenario, each sensor node is an independent learning agent, and actions are routing options using different neighbor(s) for the next hop(s) toward a subset of the sinks, $D_p \subseteq D$, listed in the data packet. The main challenge in our application is to model the actions of the nodes, since they contain not a single next hop (route to some neighbor n), but a-priori unknown number of next hops. The following provides additional detail for the Q-Learning solution.

Agent states. We define the state of an agent as a tuple $\{D_p, routes^N_{D_p}\}$, where $D_p \subseteq D$ are the sinks the packet must reach and $routes^N_{D_p}$ is the routing information about all neighboring nodes N with respect to the individual sinks. Depending on this state, different actions are possible.

Actions. In our model, an action is one possible routing decision for a data packet. However, the routing decision can include one or more different neighbors as next hops. Consequently, we need to change the original Q-Learning algorithm and define a possible action, a, as a set of sub-actions $\{a_1 \dots a_k\}$. Each sub-action $a_i = (n_i, D_i)$ includes a single neighbor n_i and a set of destinations $D_i \subseteq D_p$ indicating that neighbor n_i is the intended next hop for routing to destinations D_i. A *complete* action is a set of sub-actions such that $\{D_1 \dots D_k\}$ partitions D_p (that is, each sink $d \in D_p$ is covered by exactly one sub-action a_i).

Continuing with our example from Figure 1, consider a packet destined for $D_p = \{P, Q\}$. One possible complete action of the source S is the single sub-action $(B, \{P, Q\})$, indicating neighbor B as the next hop to all destinations. Alternately, node S may choose two sub-actions, $(A, \{P\})$ and $(C, \{Q\})$, indicating two different neighbors should take responsibility to forward the packet to different subsets of sinks.

The distinction between complete actions and sub-actions is important, as we assign rewards to sub-actions.

Q-Values. Q-Values represent the goodness of actions and the goal of the agent is to learn the *actual* goodness of the available actions. Here we differ from the original Q-Learning, which randomly initializes Q-Values, and where Q-Values serve only for quantitative comparison.

In our case, we bound the Q-Values to represent the real cost of the routes, for example, if the cost function is number of hops, the Q-Value of a route is also the number of hops of this route. To initialize these values, we use a more sophisticated approach than random assignment, which calculates an estimate of the cost based on the individual information about the involved neighbor and sinks. This non-random initialization significantly speeds up the learning process and avoids oscillations of the Q-Values.

For example, without loss of generality and continuing our example with a hop-based cost function, it estimates the route cost by using the hop counts available in a standard routing table, such as that in Figure 1. We first calculate the value of a sub-action, then of a complete action. The initial Q-Value for a sub-action $a_i = (n_i, D_i)$ is thus:

$$Q(a_i) = \left(\sum_{d \in D_i} hops_d^{n_i} \right) - 2(\mid D_i \mid -1) \tag{1}$$

where $hops_d^{n_i}$ are the number of hops to reach destination $d \in D_i$ using neighbor n_i and $\mid D_i \mid$ is the number of sinks in D_i. The first part of the formula calculates the total number of hops to individually reach the sinks, and the second part subtracts from this total based on the assumption that broadcast communication is used both (hence the 2) for transmission to n_i as well as by n_i to reach the next hop. Note that this estimation is an *upper bound* of the actual value, as it assumes that the packet will not share any links after the next hop. Therefore, during learning, Q-Values will always decrease and the best actions will be denoted with small Q-Values.

The Q-Value of a complete action a with sub-actions $\{a_1, \ldots, a_k\}$ is:

$$Q(a) = \left(\sum_{a_i \in a, i=1 \ldots k} Q(a_i) \right) - (k - 1) \tag{2}$$

where k is the number of sub-actions. Intuitively this Q-Value is the broadcast hop count from the agent to all sinks.

The above is an example of calculating the Q-Values when using the specific hop-based cost. We will explore further cost metrics in Section 6.8.

Updating a Q-Value. To learn the real values of the actions, the agent must receive the reward values from the environment. In our case, each neighbor to which a data packet is forwarded sends the reward as feedback with its evaluation of the goodness of the sub-action. The new Q-Value of the sub-action is:

$$Q_{new}(a_i) = Q_{old}(a_i) + \gamma(R(a_i) - Q_{old}(a_i)) \tag{3}$$

where $R(a_i)$ is the reward value and γ is the learning rate of the algorithm. We use $\gamma = 1$ to speed up learning. Usually a lower learning rate needs to be used with randomly initialized Q-Values, since otherwise they will oscillate heavily in the beginning of the learning process. However, since our values are guaranteed to decrease and not to oscillate, we can avoid the learning rate and the resulting delay in learning. Therefore, with $\gamma = 1$, the formula becomes

$$Q_{new}(a_i) = R(a_i) \tag{4}$$

directly updating the Q-Value with the reward. The Q-Values of complete actions are updated automatically, since their calculation is based on subactions (Equation 2).

Reward function. Intuitively the reward is the downstream node's opportunity to inform the upstream neighbors of its actual cost for the requested action. Thus, when calculating the reward, the node selects its *lowest (best) Q-Value* for the destination set and adds the cost of the action itself:

$$R(a_i) = c_{a_i} + \min_a Q(a) \tag{5}$$

where c_{a_i} is the action's cost (always 1 in our hop count metric). This propagation of Q-Values upstream eventually allows all nodes to learn the actual costs.

In contrast to the original Q-Learning algorithm, low reward values are good and large values are bad. This is because we define the Q-Values to represent the real hop costs of some route and thus the lowest Q-Values are the best. Furthermore, rewards from the environment are generated and sent out without real knowledge of who receives them. Note also that the reward values are completely localized and simply indicate the current best Q-Value at the rewarding node.

Exploration strategy (action selection policy). One final, important learning parameter is the action selection policy. A trivial solution is to greedily select the action with the best (lowest) Q-Value. However, this policy ignores some actions which may, after learning, have lower Q-Values, resulting in a locally optimal solution. Therefore, a tradeoff is required between *exploitation* of good routes and *exploration* among available routes. A simple, though efficient strategy is ϵ-greedy, which selects the best available action with probability $1 - \epsilon$ and a random one with probability ϵ. There are also variants of ϵ-greedy, where ϵ is decreased with time or where the range of random routes are restricted to the most promising ones. Section 6.9 gives more details about the exploration strategies we use for FROMS.

Parameter	Description
D	number of destinations
M	diameter of the network
Y	network density (maximum number of 1-hop neighbors)
$\|N\|$	number of nodes in the network
A	Maximum number of possible actions at each node
S	Maximum number of action steps (sent packets) at the source before convergence

Table 1: Summary of network scenario and complexity parameters, as used in the discussion of FROMS.

5 Theoretical analysis of FROMS

In this section we concentrate on the theoretical analysis of FROMS: on its convergence, complexity, memory, and processing requirements. First we explore an idealized model of the environment and later we introduce realistic properties like asymmetric links and link failures.

5.1 Worst-case complexity and convergence

We show first the worst-case complexity of FROMS (time to stabilize) and thus also implicitly its convergence. In our scenario, convergence means that first, the protocol is stable and the Q-Values do not change any more, and second and more importantly, that the optimal route has been identified. The original Q-Learning algorithm has been shown to converge after an *infinite* number of steps [46]. Here we need to show that our Q-Learning based protocol converges after a *finite* number of steps. For this, we start by calculating the number of steps until convergence.

First, we assume a Q-Learning algorithm like the one we presented in the previous Section 4 with $\gamma = 1$, hop-based cost metric, and deterministic exploration strategy, which chooses the routes in a round-robin manner. We further assume a network N with the following properties: D is the number of destinations, M is the diameter of the network (the longest shortest path in the network between any two nodes in N) and Y is the density of the network (the maximum number of 1-hop neighbors at any node in N). The parameters are summarized in Table 1. We also assume static nodes and sinks and perfect communication between the neighbors. Without loss of generality, we assume a single source, since the routes are constructed depending on the destinations, not on the sources. We will discuss multiple sources at the end of this section.

Further, the maximum number of possible actions A at any node is, according to the definition of actions in Section 4, the number of permutations of size D over all neighbors Y with repetitions (because we are allowed to use the same neighbor to reach multiple sinks) or:

$$A \leq Y^D \tag{6}$$

In the worst case the source of the data or the initiator of the learning process is at maximum distance M from all of the sinks. Our goal is to compute how many action selection steps have to be taken on all nodes in N, so that the Q-Values stabilize. With $\gamma = 1$ the feedback of any 1-hop neighbor is used for direct replacement of the old Q-Value. Thus, in order to learn the real costs of any route of length M we need exactly $M - 1$ steps. However, the source has to first wait for all other nodes to stabilize their Q-Values before it can be guaranteed that its Q-Values are stable too. In the worst case it has to explore the full network and all possible routes in it. Let us count the number of action selection steps S we need for the whole system to converge.

Assuming the learning is always initiated by the source, we know that we need to select each of the routes available $M - 1$ times. Using Equation 6 we have:

$$S \leq (M - 1) \cdot Y^D$$

The 1-hop neighbors of the source need to do the same. Their distance to the sinks is also at most M. Note this is the worst case and it actually cannot exist in a real network: if all of the neighbors of some node are at the same distance from the sinks as the node itself, the network is disconnected. Thus, all of the nodes in the network have to select each of their routes at most M times. Thus, we have for the complexity:

$$S \leq (M - 1) \cdot |N| \cdot Y^D = O\left((M - 1) \cdot |N| \cdot Y^D\right) \tag{7}$$

This is the worst-case number of actions across all nodes (packet broadcasts) for the protocol to converge. After convergence, exploration can be stopped and the algorithm can proceed in a greedy mode, as the best route has been identified and has the best Q-Value among all available. If there are more than one best routes, they can be alternated to spread energy expenditure.

However, this is a very loose upper bound of the complexity - no real networks have the worst-case properties like "all neighbors are M hops away from the destinations". However, it gives us an idea about the scalability of the approach and its expected performance. In the next paragraphs we discuss in detail how the convergence behavior changes with various network parameters and what are the consequences for the protocol. We use experimental evaluations to show the real behavior of the protocol in Sections 4-9.

Parameter analysis. The number of destinations D and the density Y are *not* directly dependent on the number of nodes $|N|$ in a network or on the diameter M. To understand better the expected performance, we explore these individual cases for each of the parameters:

The number of sinks D is completely independent from any of the other network properties, $|N|$, M, or Y, as it is a requirement of the application. The only limitation is that $D \leq |N|$. With a growing number of sinks the complexity grows exponentially, because D is in the power (see Equation 7).

With growing number of nodes $|N|$, usually either the *diameter M* or the *density Y* are growing, or both, but at a lower rate. In both cases, we expect the complexity to have a polynomial growth (from Equation 7).

In a network with constant number of nodes $|N|$, M and Y depend on each other. When the diameter is growing, the number of neighbors is decreasing; and vice versa. In the extreme case we have $M = |N| = c, Y = 2$, where we have a chain network with maximum number of neighbors 2. In this case we have:

$$S = O\left(|N|^2 \cdot 2^D\right) \tag{8}$$

The other extreme case is when the density or Y grows towards $|N|$ and M decreases towards 2 - note that the case $M = 1$ does not make sense, because then any source will be exactly one hop from any sink and routing would be trivial. In the case of $M \to 2$ we have:

$$S = O\left(2|N|^{D+1}\right) \tag{9}$$

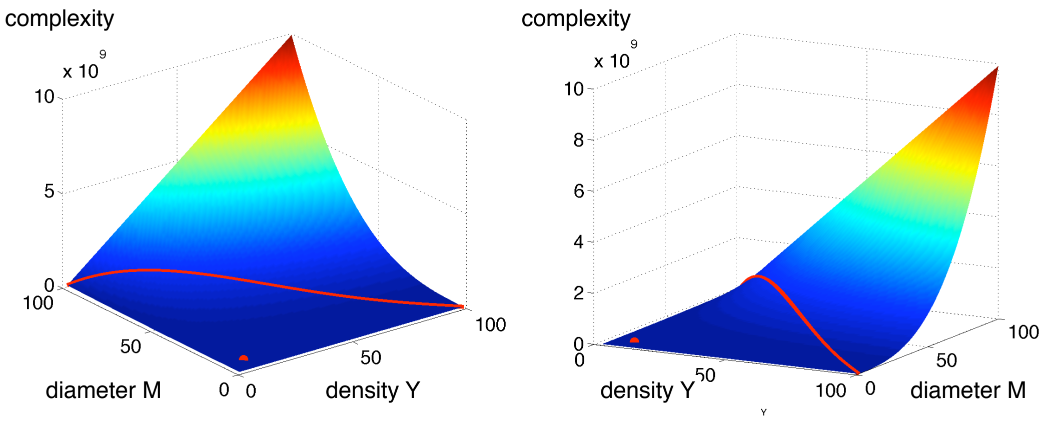


Figure 2: Worst-case complexity for some M and Y values from different views. The number of sinks is fixed to $D = 3, |N| = 100$. The thick line at the welding of the graph corresponds to maximum expected complexity and the single point near the origin to a real dense network with $M = 10$ and $Y = 10$.

However, these equations do not consider the behavior inbetween. It is more interesting to explore the complexity in a network with constant $|N|$ and different M and Y values. Figure 2 shows a case study for a network of 100 nodes, 3 sinks and different densities and diameters. The worst-case complexity is presented from two different points of view. Of course, as expected, with growing M and Y, the complexity grows. However, the thick line shows exactly the development when M is growing and Y decreasing - it shows that the function has a maximum between the two extreme cases. As a rule of thumb for practical networks it can be generalized, that having a lower density is always a good idea, since Y is in the power of D (see again Equation 7), unless M is very low, as the complexity decreases again. Note also that the extreme case of Figure 2 where both M and Y are growing towards $|N|$ is impossible in practice [47]. Realistic values for a network with 100 nodes will be $M = 10$ and $Y = 10$, which corresponds to the single point in Figure 2.

Probabilistic exploration strategy. The above complexity is given for a deterministic round-robin exploration strategy. However, both the original Q-Learning algorithm, as well as our protocol, use probabilistic exploration strategies - for each route r there is a probability p_r to be chosen at any step

s_t. If the probabilities of all routes are $p_r > 0$, convergence is guaranteed. However, complexity is hard to compute because of the non-deterministic nature of the algorithm. Instead, we will show experimental evaluations in the next sections.

Realistic communication environment. The above proof is built under the assumption of perfect communication. However, the real world of WSNs is seldom perfect. Packet losses are usual and have to be considered.

However, assuming some probability p_m for delivering a message between two nodes is enough to maintain the convergence criterium of the algorithm. The convergence will take longer, but the correctness is not violated if the probability p_m is non-zero. In the special case of $p_m = 0$ for some link(s), the network model changes: these links are actually non-existing and under the new network model the algorithm will converge.

A scenario with asymmetric links is slightly more complex. Here, two neighboring nodes may have a one-way communication only. Thus, one of the nodes may hear from the other, but not vice versa. Consequently data packets may be forwarded through some node, but feedback will never be received by the sender. If the node with the asymmetric link happens to be on the optimal route, the sender of the packets will never learn its real costs and the protocol will not converge to the optimal route. However, in practice just links are often considered are not-existing at all because of their unreliable nature. If we assume this and come back to the above discussion of packet loss, convergence is guaranteed again. It is the responsibility of the protocol's implementation to recognize asymmetric links and to delete them and we will discuss how we do this in the next Section 6.

Multiple sources. In the above paragraphs we assumed a single data source learning the optimal routes to all sinks. However, what happens when more sources are present in the network? In fact, this speeds up the convergence process of all nodes in terms of data packets sent by one source. Imagine a network with 2 sources, sending data at the same rate to 3 identical sinks. In this case, nodes on the routes of both sources to the sinks receive double feedback from sending data packets from both sources. This is because our feedback is delivered to all neighboring nodes.

5.2 Correctness of Froms

The correctness of FROMS is easily deducible from the definition of the used Q-Learning model in Section 4. The goal is to show that after convergence, the Q-Values of the full actions at any node will accurately reflect the hop-based costs. We use simple induction to sketch the proof in sufficient detail for our purposes. We begin by showing the correctness of FROMS for one sink, then expand the proof to multiple sinks.

Assumptions. We assume perfect communication, static network, and the Q-Value calculation and update equations from Section 4.

Initial step. The induction starts with the sinks and we define the cost of the sinks of routing to themselves to be always 0, since no forwarding is needed any more. Thus, the reward of the sinks for routing to themselves is always $r = 0 + c_a$ with $c_a = 1$ from Equation 5. For $\gamma = 1$, the neighbors update the Q-Value for the corresponding sub-action to $Q = r = 1$, which we know is the correct cost of this sub-action, since the sink is exactly one hop away.

Induction step. Assume that a node N (sink or any other node) has a correct estimation of the costs to the sink Q_N. Its reward is always computed as $r = \min_a Q(a) + c_a$, where $\min_a Q(a)$ is necessarily the above Q_N and $c_a = 1$. When node N sends its reward to its direct neighbors, they will update their corresponding Q-Values for this node to $Q_N + 1$, which is the correct estimation of the cost through node N, since they are exactly one hop further away from the sink than node N. Thus, for any node N with correct estimations of the cost, its direct neighbors also have correct cost estimations.

We showed above that FROMS converges to the correct hop-based costs for one sink in the network. In fact we know that FROMS is correct for one sink also because of the sink announcement propagation. During this network-wide broadcast, every node easily learns about the best routes in terms of hops to a single sink. Thus, we have both a practical and a theoretical proof that FROMS converges to the correct costs for one sink. This is the beginning of the second induction proof, which shows that FROMS converges to the correct hop-based costs also for more than one sink.

Assume a network with 2 sinks that the Q-Values for each sink individually at all nodes have already converged (see the discussion above). For simplicity we call the sinks A and B. The costs of B to reach itself is 0 and to reach sink A is a constant $v = \min_a Q_B(a)$, which is the minimum Q-Value for A at node B. Thus, the cost of reaching *both* A and B at B is $0 + v$ and the reward of B is $r_B = (0 + v) + c_a = v + 1$. The direct neighbors of B will update their own Q-Values to this reward value, which is the right cost: they need one hop to reach sink B and further v costs to reach sink A. This trivially extends to the next hops, as shown already above. It also intuitively extends to more than 2 sinks.

Summarizing Sections 5.1 and 5.2, we have shown that Froms converges to the correct hop-based costs of the routes after finite number of steps.

5.3 Memory and processing requirements

Before explaining the implementation details of FROMS and showing its experimental evaluation, we analyze the theoretical memory and processing requirements of the algorithm for each node in the network.

Each node has to store all locally available routes. According to Equation 6 the expected storage requirement is $O(Y^D)$. The processing requirements include selecting a route and updating a Q-Value. The first function requires in the worst case to loop through all available routes to compare them in terms of their costs and is thus bounded by $O(Y^D)$. The update of a Q-Value is itself an atomic action: given the old Q-Value and the reward, it calculates the new one. Assuming a data structure, organized by neighbor, we need as worst case for searching $O(Y + D)$.

6 Protocol implementation details and parameters

The multicast routing protocol FROMS is built upon the formal Q-Learning model, presented in Section 4. A pseudo-code of the resulting protocol is given in Figure 3. Basically, the routing protocol consists of three main processes: sink announcement and initialization of routes (lines 3-4), selection of routes (lines 9-12), and learning and feedback (lines 8 and 14). Additionally, there are some parameters of FROMS like exploration strategy (line 12) and cost functions (line 2), and the sink mobility management module (line 7).

We step through all of these and give details in the following sections.

```
1:  init:
2:    init_cost_function();

3:  on_receive(DATA_REQ req):
4:    add_nexthop(req.sinkID,req.neiID,req.hops,req.battery);

5:  on_receive(DATA d):
6:    // snoop on all incoming packets
7:    sinkControl.update(d.sinkStamps,d.neiID);
8:    add_feedback(d.feedback, d.neiID);
9:    // route packet to next hop(s)
10:   if (d.nexthops.includes(self))
11:     routes = get_possible_routes(d.my_sinks,cost_function);
12:     route = strategy.select_route(routes);
13:     d.routing = route;
14:     d.feedback = best_route_cost;
15:     broadcast(d);
16:   end if
```

Figure 3: The main FROMS algorithm

6.1 Sink announcement

Recall from our application scenario described in Section 2.1 that we assume each of the sinks announces itself via a network-wide broadcast of a DATA_REQ message, during which initial routing information like hops to the sink is gathered (lines 3-4 in Figure 3). Additionally, position information, battery status of neighbors, etc, can be delivered to the nodes.

6.2 Feedback implementation

A substantial part of FROMS is the exchange of feedback (reward). This is what enables FROMS to learn the global cost of the routes and to use the globally optimal paths. We piggyback the feedback, which is usually only a few bytes, on usual DATA packets (line 14 in Figure 3). There are several advantages of this implementation: feedback is sent only on-demand and only to local neighbors; and overhead is kept minimal because no extra control packets need to be exchanged.

Note that feedback is accepted and route costs are updated even if the feedback is negative and the previously known costs were better. Thus, mobility and recovery are handled automatically. The feedback is usually received by all overhearing neighbors, which speeds up the learning process. However, feedback can also be delivered to the previous hop only, thus avoiding energy expenditure for overhearing of packets. This implementation requires a multicast MAC layer protocol, able to send the message only to some subset of neighbors. Unfortunately there is no such protocol designed for low-energy WSNs to the best of our knowledge and its implementation is not trivial, since it requires a well-designed scheduling together with variable-length preamble packets. We consider designing such a protocol and testing it with various routing techniques in the future.

6.3 Data management

One of the implementation challenges of Froms is to design an efficient multi-destination routing data structure. This data structure is different from usual routing tables like the one in Figure 1 since it not only holds next hops for individual sinks and their costs, but also combines shared paths to multiple sinks. In other words, we need a data structure to hold the sub-actions as described in Section 4. For example, the possible sub-actions for node S from Figure 1 for each of the neighbors n_i are: $\{n_i, (P)\}$, $\{n_i, (Q)\}$ and $\{n_i, (P, Q)\}$.

Data structure API

As shown in the algorithm pseudocode from Figure 3, the multi-destination routing data structure used by Froms has to implement efficiently and reliably the following API:

```
add_nexthop(sinkID, nexthop, hop_cost, battery)
```

This function is called when a DATA_REQ arrives, or when a feedback for an unknown sub-action arrives. The second case happens, when sink announcements were lost and some next hops are unknown at the node. However, the first time when the unknown neighbor broadcasts a data packet the node will repair its routing table.

```
add_feedback(feedback, previous_hop)
```

This is called every time the node hears a data packet. The data structure has to find the required sub-action and to update its cost. The cost is updated always and not only when it is better than before. The costs are expected to be higher than previously known when a node fails or when a sink moves away. All routes' full cost, using this sub-action, have to be updated. Additionally, if this sub-action cannot be found, it should be recovered (see `add_nexthop`).

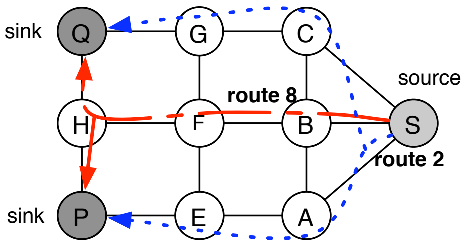
```
get_possible_routes(sinks, cost_function)
```

This is called by the exploration strategy and should return all possible routes, which fulfill some requirements, like maximum hop cost, maximum total cost etc (for loop management, see below). The routing strategy will then select one of them for usage.

PSTable. Our Froms implementation uses an instantiation of the above defined data structure called **PSTable**, or **P**ath **S**haring **Table**. Let us continue with our example of Figure 1. Figure 4 presents the resulting data structure for node S. For easy reference we have copied also the network topology. The PSTable consists of two simple tables, for the sub-actions and the routes (full actions), and three management variables. Note that this sample PSTable contains the initial Q-Values for all sub-actions and full actions and is based on hops for simplicity. Note that cost calculation for sub-actions occurs only once: at initialization. After that, feedbacks are used to update the Q-Values. Q-Values of full actions (Table `allRoutes`), which we also call Q-full, are computed according to Equation 2 from the Q-Values of the included sub-actions. Further details are given below:

- `subActions`: This table holds all available sub-actions for each of the neighbors. They are organized by neighbor ID for speeding up search in case of feedback. For each of the sub-actions, the table holds the Q-Value of that action and assigns an ID, which is used as a pointer to that sub-action. The grey-shaded fields are pruned sub-actions to save memory and will be explained later in Section 6.4.

- `allRoutes`: This table holds basically all possible combinations of sub-actions, such that in each route all sinks are covered exactly once. The table holds the total Q-Value of the full action, computed from



routing table : node S

Neigh	sink	hops	...
A	P	3	...
	Q	5	...
B	P	4	...
	Q	4	...
C	P	5	...
	Q	3	...

PSTABLE for Node S

subActions

neighbor	sinks	Q-Value	subaction ID
A	P	3	1
	Q	5	2
	P,Q	6	3
B	P	4	4
	Q	4	5
	P,Q	6	6
C	P	5	7
	Q	3	8
	P,Q	6	9

allRoutes

route ID	subaction IDs	Q full
1	1,5	(3+4)-1=6
2	1,8	(3+3)-1=5
3	4,8	(4+3)-1=6
4	2,4	(5+4)-1=8
5	2,7	(5+5)-1=9
6	5,7	(4+5)-1=8
7	3	6-0=6
8	6	6-0=6
9	9	6-0=6

validSinks = {P,Q} **costsChanged = false** **routesChanged = false**

Figure 4: The PSTable for node S from Figure 1. Grey-shaded boxes are ignored sub-actions (not stored), which saves memory after applying route storage pruning heuristics $C = 1, Nr = 3$ (see Section 6.4).

> the Q-Value of the included sub-actions according to Equation 2. Two examples are emphasized in the figure, route 2 and 8. Route 2 (marked bold in the figure) consists of two sub-actions with IDs 1 and 8 and corresponds to the dashed route in the network topology in the same figure. Its full route costs (its full Q-Value) is 5, which is the cost in terms of hops for this route. In contrast, route 8 consists of only 1 sub-action with ID 6 and its full cost is also 6 hops.

Note that these two tables must be separate: rewards are assigned and delivered by sub-actions, but full routes are needed when routing data packets. Putting them together will increase significantly the search time for incoming rewards, because sub-actions will be presented several times in different routes and the full table would need to be traversed to find them.

- `validSinks:` The sinks, for which the full Q-Value is computed and stored. We apply lazy evaluation of routes to speed up the route selection. For example, if a route to only one of the sinks is desired (e.g. for sink Q), the Q-Values of the routes will be re-computed as to include only the desired sinks. If this computation is impossible, for example as it is for route 8, the Q-Value is marked with -1. This is the case when needed and unneeded sinks are combined into one sub-action: for example, sub-action 6 of route 8 contains both sinks P and Q and thus separated computation of the cost to sink Q only is impossible.

- `routesChanged:` This variable indicates that the `allRoutes` table has to be rebuilt because new routes are available or old ones lost.

- `costsChanged:` This indicates that the costs of some routes have changed and have to be recalculated or that the costs are not valid any more (`validSinks` has changed). This happens usually when new feedback arrives, which in fact changes the routes' Q-Values. Then all routes which use the updated Q-Value become invalid. For example, if sub-action 1 from our Figure 4 gets updated, routes 1 and 2 become invalid. However, instead of immediately searching for those routes and recalculating their costs, we mark the whole table as invalid and wait until a data packet arrives for routing. This saves processing effort when the node is overhearing a lot of feedback from its neighbors, but does not route data packets.

In the simulation environment (described in Section 9) we use dynamic memory allocation for *subActions* and *allRoutes* and memory pointers to the sub-actions. In the real hardware environment (described in Section 8) we do not have dynamic memory allocation and use a static array of `subActions` items and a static array of `allRoutes` items. The size of both of them are large enough to accommodate all possible sink combinations and routes. Instead of memory pointers we use IDs, like in the example in Figure 4.

6.4 Route storage reducing heuristics

As pointed out in Section 5, the storage requirements for all routes grow exponentially with number of sinks and polynomially with number of neighbors. In practice this means that for large number of sinks and neighbors we are not able to store all routes. The consequence is that we cannot guarantee any more that the algorithm is optimal. However, its near-optimality can be easily preserved by wisely managing which routes to store and which not.

We have developed two route pruning heuristics: C - *cost over best maximum* and Nr - *maximum number of routes to sink*. The first one checks what is the currently best cost to the sink in question and if the newly arrived route has cost more than this best one plus the threshold C, it ignores the route. The second one is a limit over the number of routes per sink - when this number is exceeded, the newly arrived route is ignored. In Figure 4 ignored entries after applying $C = 1, Nr = 3$ are shown in grey. Note that these heuristics not only limit the memory requirement at the nodes, but also the convergence time, since less routes need to be explored.

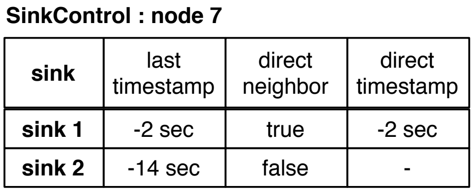
6.5 Loop management

FROMS explores non-optimal routes for finding the globally best route. This means that it chooses a route with a non-limited length. Thus it can happen that a packet travels in a loop, even forever. In order to manage this, we have introduced the maximum allowed hop cost for a neighbor. Each node receives the data packet together with the subset of sinks which it has to care of, and a maximum hop cost for the selected route. We set this maximum allowed cost to the currently known cost for this sub-action. Thus, if the cost estimate is right and the node has no better routes, it will be forced to use the best one. The reason for requiring this is that if the cost estimate is right the probability that this estimate is also the real cost is very high.

6.6 Mobility management

The Q-Learning algorithm has the innate ability to manage changing network conditions. They will be delivered as feedback and the Q-Values will be updated accordingly in the usual learning process. However, practical challenges arise: growing costs of some route could either mean mobile sinks moving away or a disconnection from some sinks. The first case is normal and should be handled as usual. The second one, however, will cause looping packets, traveling forever and searching for non-existing routes.

An important special case for managing moving sinks is when a node is a direct neighbor of a sink. In this case we exclude this sink from learning and always send directly to it. However, this causes problems when the sink



SinkControl : node 7

sink	last timestamp	direct neighbor	direct timestamp
sink 1	-2 sec	true	-2 sec
sink 2	-14 sec	false	-

Figure 5: SinkControl for node E (direct neighbor of sink P from Figures 1 and 4).

moves away and the sink needs to be included in normal learning again. Thus, we need a technique to recognize alive sinks moving out of range.

SinkControl is a simple data structure whose goal is to detect moving or disconnected sinks. It does not affect the Q-Learning algorithm, but manages the available routes, erasing invalid ones. It stores information about each known sink in the network. Figure 5 presents it for the sample topology of Figure 1. The feedback delivers a last timestamp for each included sink; this is the last time this neighbor has heard of the sink. If this timestamp is too old (a threshold parameter), the sink is deleted. This is the case when either the sink itself has failed or disappeared from the network or the network is disconnected between the sink and the node. In both cases the application layer has to be notified to delete the data delivery task for those sinks and routing to them has to be stopped.

On the other hand, while the sink is “fresh” data delivery can continue even if the routes’ costs to it are growing. In order to detect sinks in the direct neighborhood, we also store the last time the node has heard from a sink directly. If the threshold is exceeded, the flag for direct neighbor is deleted and Froms is notified.

This simple module enables detection of sink mobility and learning of new routes with minimum communication overhead, the additional last timestamp feedback. Despite using timestamps, Froms does not require a time synchronization protocol or any other means of global time. It is enough to use timestamps like in Figure 5: $(now - n \cdot sec)$. The goal is to detect sinks, which are not responsive for a long time.

Obviously, this sink mobility detection can be implemented for any routing protocol. However, it is not sufficient to handle sink mobility: it only checks whether a route can exist or not. Finding the optimal route is still performed by Froms and its learning and feedback mechanism. Most importantly, delivery of data to the sinks continues while recovering the routes and learning the new costs.

6.7 Node failures

Node failures are managed the same way as sink mobility. Each node stores the last time it heard from any 1-hop neighbor. Additionally, it stores the last time it routed something to that neighbor. In case the difference between both timestamps exceeds some threshold, the neighbor is deleted. Note that if this happens by mistake, the next time the node hears again from this neighbor, the route will be recovered.

Note that unlike many link management protocols, FROMS does not use any beacons or periodic full-network broadcasts. Only overhearing of data packets is used to check the status of neighbors.

6.8 Cost metrics

Here we present FROMS innate ability to incorporate different cost functions to reach different optimization goals. The cost function is used to calculate the initial Q-Values in FROMS. A simple hop-based metric was presented already in Section 4 with Equations 1 and 2. Its optimization goal is to find the shortest shared path for multiple sinks in terms of hops. The hop-based cost function can be easily exchanged with any other cost-per-link metric, like energy needed to reach the farthest neighbor, geographic distance or geographic progress to the sinks, etc. Various cost metrics and their properties are summarized in Table 2.

Another example for a cost-per-link function is a latency-based cost metric. Here we need to gather latency information during sink announcement to the sensor nodes. The latency needs to represent the radio propagation latency (where the differences will be negligible for usual sensor networks) and the latency caused by the packet queues on the nodes. However, note that such a cost metric is what we call here a *dynamic cost metric*: it is expected to change during network lifetime and to change fast. For FROMS this means that it will never globally converge, nor stay in a converged state. However, we show in the next paragraphs other dynamic cost functions and how to handle their behavior. In fact, we make use of this non-converging behavior and turn it into an advantage.

Cost metric	**Calculation of initial values**	**Optimization goal**	**Convergence**	**Dynamic**	**Best Q-Values**
simple metrics					
Hops	$\sum hops$	shortest shared path (Steiner tree)	guaranteed	no	lowest
Latency	$\sum latency$	least latency path	no	yes	lowest
Transmission energy	$\sum energies$	least energy path	guaranteed	no	lowest
Geographic distance	$\sum dist$	shortest shared path	guaranteed	no	lowest
Aggr. rate	$\sum rates$	maximum aggr. path	slow	no	highest
combined metrics					
Hops & rem. battery of nodes	$\sum hops \cdot hcm(bat_{hops})$	shortest shared path through nodes with high battery	no	yes	lowest

Table 2: Different possible cost metrics for FROMS and their main properties.

Here we concentrate on one *combined* cost metric to demonstrate how to use such complex cost metrics with FROMS. We use a combination of remaining battery on the nodes and minimum hops. In this case we calculate the Q-Values as a combination of two metrics as follows:

$$Q_{comb}(route) = f(E_{hops}, E_{battery}) \tag{10}$$

where E_{hops} is the estimated hop cost of the route exactly as we calculate it in equations 1 and 2, and $E_{battery}$ is the estimated battery cost of this route, which we define as the minimum remaining battery of all nodes along it:

$$E_{battery}(route) = \min_{n_i \in route} battery \tag{11}$$

The function f that combines the two estimates into a single Q-Value is based on a simple and widely used function:

$$f(E_{hops}, E_{battery}) = hcm(E_{battery}) \cdot E_{hops} \tag{12}$$

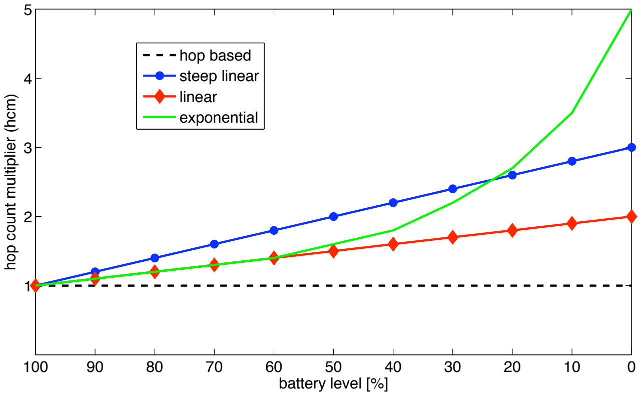


Figure 6: Hop count multiplier (hcm) functions for different optimization goals.

hcm is the hop-count-multiplier, a function that weights the hop count estimate based on the remaining battery. For simplicity we drop the "estimation" and denote the Q-Value components as $hops$ and $battery$.

Figure 6 shows four different hcm functions. If the battery level is completely irrelevant, then $hcm(battery)$ is a constant and $f(hops, battery)$ is reduced to a hop-based function only. Instead, if the desired behavior is to linearly increase f as the battery levels decrease, a linear hcm function should be considered. Figure 6 shows two linear functions. The first (labeled linear), has minimal effect on the routing behavior. For example, a greedy protocol which always uses the best (lowest) Q-Values available, when faced with two routes with $f(1, 10\%) = 1.9$ and $f(2, 100\%) = 2$, will select the shorter route even though the battery is nearly exhausted. Even when faced with longer routes of length 2 and 3 respectively, it will use the shorter route until its battery drops to 40%. Only when their values become $f(2, 40\%) = 3.2$ and $f(3, 100\%) = 3$, the protocol will switch to the longer route. Thus, this trade-off of weighing the hop count of routes (their length) versus the remaining batteries must be taken into account when defining hcm.

The main drawback of linear hcm functions is that they do not differentiate between battery levels in the low and high power domain. For example, a difference of 10% battery looks the same for $20 - 30\%$ and for $80 - 90\%$. Thus, to meet our goal of spreading the energy expenditure among the nodes, we require an exponential function that starts by slowly increasing the value of *hcm* with decreasing battery, initially giving preference to shorter routes. However, as batteries start to deplete, it should more quickly increase *hcm* in order to use other available routes, even if they are much longer, thus maximizing the lifetime of individual nodes. Of course, such a function gives preference to longer energy-rich routes, and will increase the per packet costs in the network.

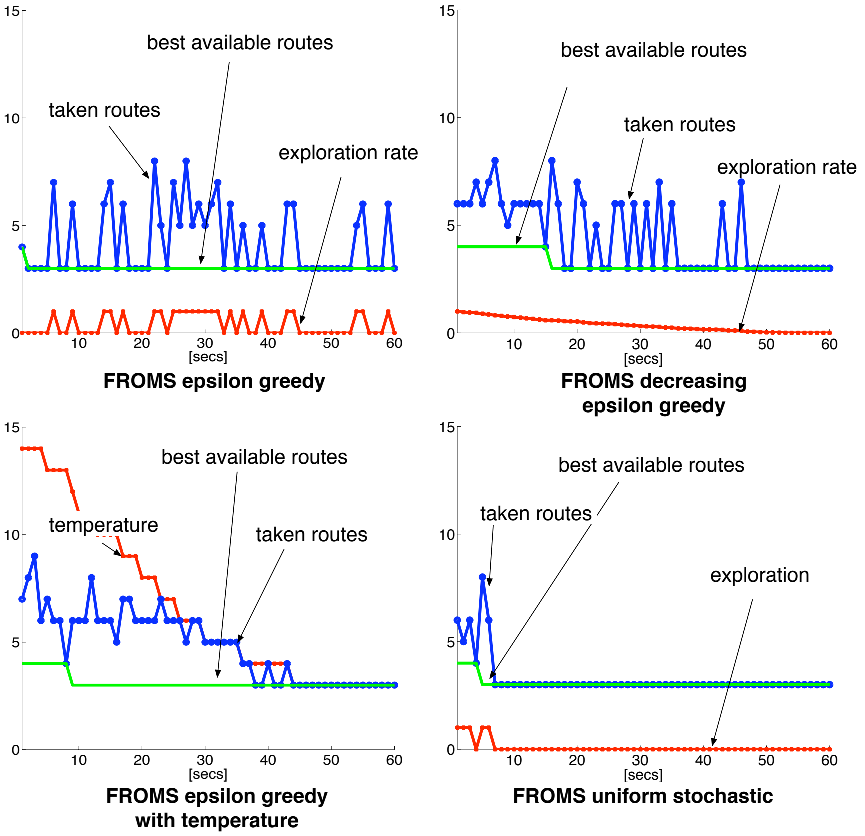
The presented battery and hop based function is a dynamic function, which means that it is expected to change during the network lifetime. Obviously, the remaining batteries of the nodes will change and thus the Q-Values as well. The major consequence of this is that FROMS does not stabilize, because the Q-Values never stabilize. However, this is not necessarily a disadvantage: FROMS will just continue exploring routes throughout the network lifetime. Combining a dynamic cost function with a mostly greedy exploration strategy will ensure that FROMS is not spending too much energy on exploration of routes and is mostly using the best available routes. On the other side, we need to ensure that FROMS is still able to find the best routes. For this, we use the advantage of a dynamic cost function. The Q-Values change because of the dynamic nature of the cost metric and force FROMS to use different routes (because it mostly selects the best ones): thus, it implicitly forces FROMS to explore new routes.

This property of dynamic cost functions we call the *dynamic cost advantage of implicit exploration*, which is a very important property of FROMS. It allows FROMS to use a very simple greedy or ϵ-greedy exploration strategy with very low probability for exploration (see next section) and still ensures that the optimal routes are found. This simplifies significantly the implementation of FROMS both in terms of processing and memory requirements and make FROMS much more intuitive.

Similarly, one can easily design and implement other cost metrics, both simple and combined. The used cost function depends on the application scenario and needs to be revisited for each deployment. However, the power of FROMS is its innate ability to accommodate nearly any cost function. The changes to the protocol are marginal and do not affect its basic functionality.

6.9 Exploration strategies

The exploration strategy controls how FROMS chooses between the available routes. It also controls the exploration/exploitation ratio, which is responsible for both finding the optimal route and minimizing routing costs. In



FROMS epsilon greedy

FROMS decreasing epsilon greedy

FROMS epsilon greedy with temperature

FROMS uniform stochastic

Figure 7: The route selection behavior at the source with different exploration strategies in a sample 50 node topology with 3 sinks and 1 source.

our preliminary studies [12], we have applied two different techniques for exploration: greedy and probabilistic. The *greedy* strategy simply ignores exploration and always chooses between the best available routes. *Stochastic* exploration strategies on the other hand assign a probability to each of the routes, depending or not on their current or initial Q-Values, and choose the routes accordingly. These exploration strategies showed good results, but are very complicated to implement since they require updating the probabilities after each reward.

Here, we turn to a new set of exploration strategies for two main reasons for the main reason of making them more intuitive and simple to implement. The behavior of the considered strategies is depicted in Figure 7.

ϵ - greedy. This strategy is taken directly from the original Q-Learning algorithm and is very simple to apply and implement: with probability ϵ select any of the available routes; with probability $1-\epsilon$, select one of the best routes. Note that when $\epsilon = 0$ we have the old greedy strategy from [12].

decreasing ϵ - greedy. This strategy is the same as before, but additionally decreases ϵ with time. The reason for this is that usually at the beginning of the algorithm the Q-Values change a lot, but with time these updates become more rare and eventually stop. After convergence it is more appropriate for FROMS to be greedy, since no changes are expected and the routing costs should be as low as possible. ϵ increases again in case of failures or mobility.

ϵ - greedy with temperature. This strategy is again a variation of ϵ-greedy, but instead of decreasing ϵ itself, it limits the set of routes presented to the strategy. At the beginning, with high temperature T, all routes are presented to the strategy, independent from their current Q-Values. With decreasing T, however, only routes with better Q-Values are presented and with $T = 0$ only the best routes are presented. ϵ remains constant and the temperature is increased in case of failures or mobility.

uniform stochastic with stopping strategy. This strategy is taken from our previous work [12] (it performed the best out of all compared stochastic strategies there) and is used for comparison reasons. It assigns the same probability to each sub-action and updates it every time a reward arrives for it, decreasing it with neutral rewards, increasing it with negative rewards, and leaving it the same with positive rewards. It stops exploration completely after some number of continuous neutral rewards to the node and starts it again with negative/positive rewards.

6.10 Summary

In this section we presented all parameters and implementation details of FROMS. The main parameters which need to be specified before deploying FROMS are its cost function and exploration strategies. Additionally, node failure management is a necessary option in nearly any WSN. However, all other presented modules implement special features, like sink mobility support or route pruning heuristics for extremely memory-resticted hardware systems, and need to be deployed only when necessary. In the following sections we present an extensive evaluation of FROMS and all of its components and features both under simulation and on real hardware.

7 Evaluation methodology and environment

Additionally to the *theoretical analysis* in Section 5, we use *evaluation through simulation and on real hardware* to show the most of the aspects and properties of FROMS. We use a wide range of evaluation metrics across many different network scenarios and parameters. Of course, an exhaustive analysis under any possible environmental conditions is not possible for time and space reasons.

Simulation environment. We use the OMNeT++ network discrete event simulator, together with its extensions Mobility Framework and probabilistic radio propagation models [48]. This is a user friendly environment with active development community, easily extendable with our own models. Unfortunately, there are no energy expenditure models, nor realistic MAC protocols for the Mobility Framework. Thus, we needed to implement the following additional simulation models:

- **Linear battery model.** A linear battery model which accounts for different energy expenditures for radio sleeping, receiving and sending, is sufficient for the evaluation of a routing protocol. We use the energy expenditure model of Mica2 nodes, see Table 3.

- **MAC protocols.** We have implemented BMAC [49] and LMAC [50] as representatives of low power listening MAC protocols and TDMA based protocols. Both have been used for real WSN deployments and are widely accepted by the WSN community. Frame and slot durations were identified experimentally so that all evaluated data traffic models are accommodated without MAC buffer overflow. In LMAC we reserved 5 node IDs for mobile nodes to avoid continuous slot changing.

Hardware testbed. We implement and test the developed routing protocol FROMS on a real hardware testbed, consisting of 10-15 MSB430 nodes from ScatterWeb [51], organized in multi-hop topologies (see Figure 9). Their main characteristics are summarized in Figure 8. For implementation we use

Layer	**Protocol/model**	**Parameters**
Application	regular data report	**data rate**: every 10 sec **sink announcement**: every 100 sec
Routing	Froms unicastDD multicastDD MSTEAM	see text
Medium access	LMAC	**slots**: 32 **slot length**: 60 ms **preamble length**: 12 bytes
	BMAC	**slot length**: 50 ms **preamble length**: 120 bytes
Energy expenditure	Linear battery (Mica-2)	**SLEEP**: 36 mW **RX, TX**: 117 mW
Radio propagation	1-Nakagami	-

Table 3: Summary of simulation environment model for our experiments.

the OS-like ScatterWeb[2] library, which provides simple interfaces for sending/receiving messages, setting timers, reading sensory data etc. We use the provided non-persistent idle CSMA MAC protocol without acknowledgments.

Comparative analysis. For conducting a comparative analysis of Froms, we have implemented three well known state-of-the-art routing protocols:

- Msteam [10] is a recent state-of-the-art geographic multicast routing protocol. The comparison between Msteam and Froms is especially interesting and challenging, as they require different available information on the nodes to achieve the same goal. Thus, also a general performance comparison between hop-based and geographic based protocols is presented. We use the same application layer and sink announcement broadcasts for both Froms and Msteam. They have several advantages against the typical pre-known set of sink coordinates, used by many geographic-based protocols. First, it replaces the use of beacons for discovering and maintaining neighbors and, second, it enables recovery and mobility, which are not covered by the original version

MSB430	
Provider	ScatterWeb, Berlin, Germany
Processor	MSP430
Frequency	8MHz
Memory	5 KB RAM + 55 KB Flash
Radio	ChipCon 1020
OS	ScatterWeb[2], TinyOS, Contiki, etc.
Other	SD-card slot

Figure 8: Characteristics of the MSB430 sensor nodes

of the protocol. Two versions of Msteam are evaluated: the original Msteam uses cost over progress to sinks metric to evaluate possible next hops, where the cost is a function of the geographic distance between the nodes. We also implemented a simplified version, called Msteam-const, where the cost is always 1 and thus only the progress to the sinks is considered.

We decided to add Msteam to our analysis since it represents a very well performing class of protocols for multicast applications. Indeed, most of the multicast protocols for WSNs are location-based and we desire to have a direct comparison with one of them.

- Unicast Directed Diffusion (udd) [11] is a well known, simple, and efficient routing paradigm, where each of the nodes builds gradients towards the sinks. We label this version of Directed Diffusion ”unicast” (or udd for short), since we consider the original one-phase pull version of the protocol, as opposed to multicast Directed Diffusion (or mdd), as explained next.

- Multicast Directed Diffusion (mdd) is a multicast-optimized variation of udd of our own design [52]. It is searching locally on the nodes for shared paths for multiple sinks. It can be considered a simplified version of greedy-Froms, which keeps only the best hops to individual sinks, does not explore, and the cost function is based on hops only. However, it does not incorporate the learning mechanism

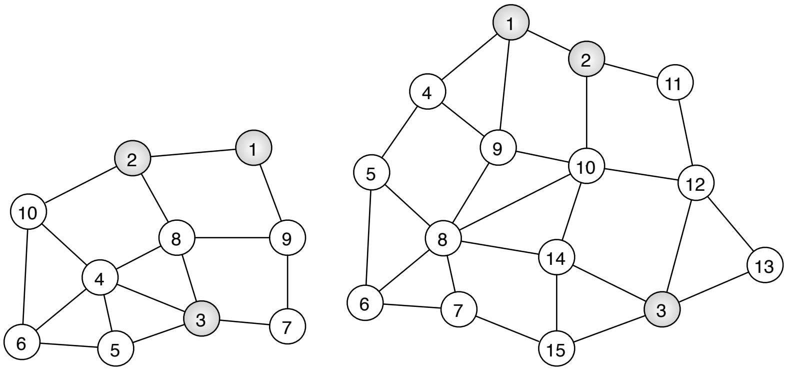


Figure 9: Topologies 1 (left) and 2 (right) for the real hardware testbed, sinks are shaded, source is node 6.

of Froms, nor it is able to find the globally optimal path unless by chance.

We compare the performance of all here presented routing protocols under simulation. On the hardware testbed we have implemented only Froms and MDD. We decided against the original Directed Diffusion, because it is a unicast routing protocol and against Msteam, because its implementation is very processing and memory intensive and did not fit on the used hardware.

8 Evaluation of FROMS on the hardware testbed

We compare the performance of Froms with ϵ-greedy exploration strategy against the multicast version of Directed Diffusion of our own design [52]. We use two controlled network topologies as given in Figure 9. In these experiments, we allowed the nodes to process packets only from some predefined set of nodes and to drop immediately all others. We were forced to do this, because we were unable to create a natural multi-hop topology, which is essential for the evaluation of the routing protocol [53].

8.1 Memory and processing requirements

Here we present and discuss the memory and processing requirements as obtained from the real hardware testbed. No simulated results are presented,

	ROM (bytes)	RAM (bytes)
FROMS + ACKs	33610	3345
FROMS	31986	3326
mDD + ACKs	29648	2952
mDD	28024	2932
Scatterweb^2	19628	1476

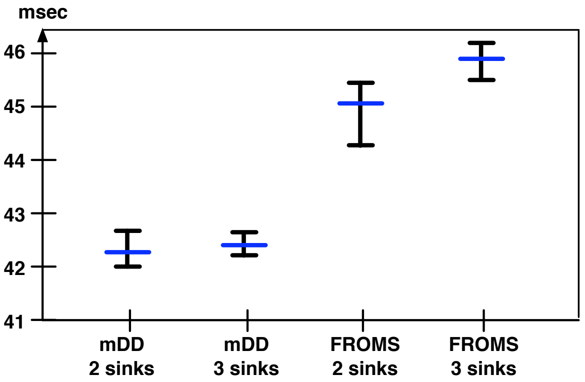


Figure 10: (left) Memory usage at compile time. The Scatterweb library alone is given for comparison. (right) Processing time to find a route in milliseconds for MDD and FROMS and max-min intervals.

since they depend strongly on the simulation environment used and are not comparable to real hardware requirements.

Memory usage. Figure 10 (left) presents the memory footprints at compile-time for MDD and FROMS together with the application layer. It shows the memory reserved for the flash ROM and the RAM. The footprint of the ScatterWeb^2 library alone is given for comparison. The vanilla implementation of MDD on top of ScatterWeb^2 e.g. consumes roughly 3KB of RAM at compile time, leaving 2KB for stack allocation and application-level protocols. Compared to the standalone implementation of ScatterWeb^2, this is negligible.

Both implementations of FROMS and MDD use static data structures, because there is no working dynamic memory management implementation for the microprocessor of the used hardware platform MSB430. No route storage heuristics are used for FROMS and thus all possible routes are kept at all times. Thus the data structures are already included also in the memory footprints in Figure 10. Although FROMS's data structures are more complex and large than the routing table of MDD, its memory requirements are not significantly higher. MDD has a very tiny data structure, but despite this, its implementation size is not negligible. In fact, the majority of its memory is used for the functionality of the protocol, not for data structures.

Processing time. We measured the time needed to find a route for each

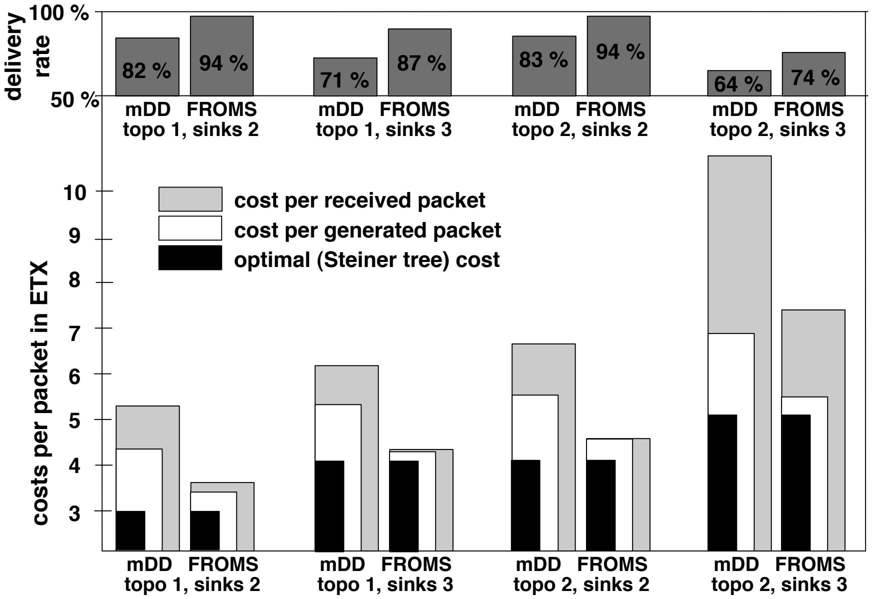


Figure 11: Routing costs and delivery rates for Froms and mdd in various network scenarios.

packet in the network at every node in milliseconds. Basically, we discovered that it takes slightly longer to find a route to more sinks but the difference between the protocols is very small. The results in Figure 10 (right) are summarized based on the number of sinks in the network. They are obtained from experiments with topology 2 only. The reason why Froms needs more time to find a route for a data packet is its routing data structure. We need to search through all of the available routes to find the best available one. Consequently, with 3 sinks the processing time increases further.

These results are an important proof of the applicability of Froms and in general of reinforcement learning based communication protocols on real hardware. They show that Froms is easily implementable and that its memory and processing requirements are negligibly higher than those of a very simple routing protocol like mdd. The comparative evaluation of both protocols on the real testbed is discussed later in Section 8.2.

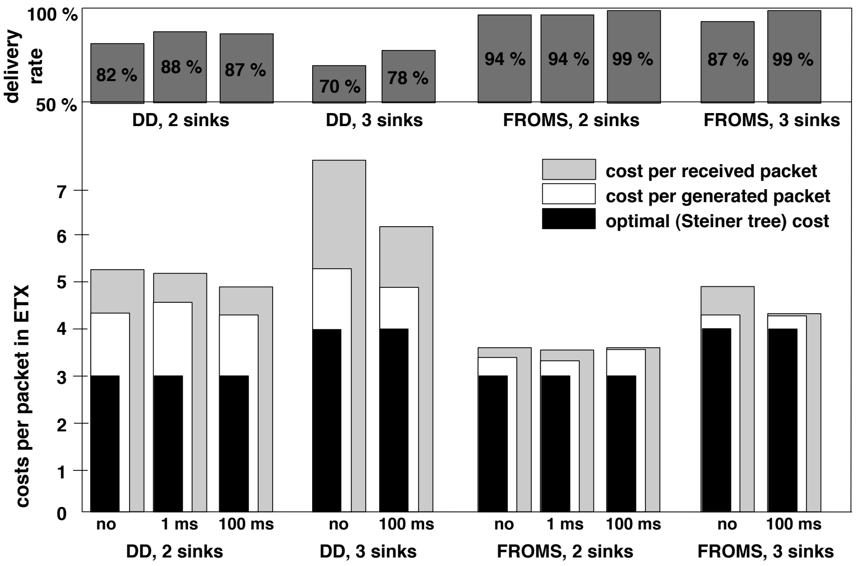


Figure 12: In-network performance when applying transmission backoff, results from topology 1.

8.2 Comparative analysis

Next, we compare the performance of FROMS and MDD on real hardware in terms of delivery rate and routing costs. Figure 11 summarizes the results for several network configurations. As expected from our simulation experiments and theoretical analysis, FROMS achieves lower routing costs. This can be attributed to its learning algorithm which actively explores the network for optimal routes. We also compare the performance against the theoretically optimal cost of the Steiner tree.

In simulation we are unable to measure accurate delivery rates since transmission failures cannot be reliably simulated. Here, instead, we confirm our theoretical expectation that FROMS is able to achieve higher delivery rate in any network scenario. Data is lost in MDD mainly due to the higher in-network communication caused by the periodic sink announcements (see Section 9) and the longer routes to the sinks. This increases the traffic and collision probability leading to packet losses. Figure 11 supports these observations, showing that the delivery rate of both protocols clearly drops in networks with larger numbers of nodes and sinks.

Figure 12 presents the results when using transmission backoff, clearly

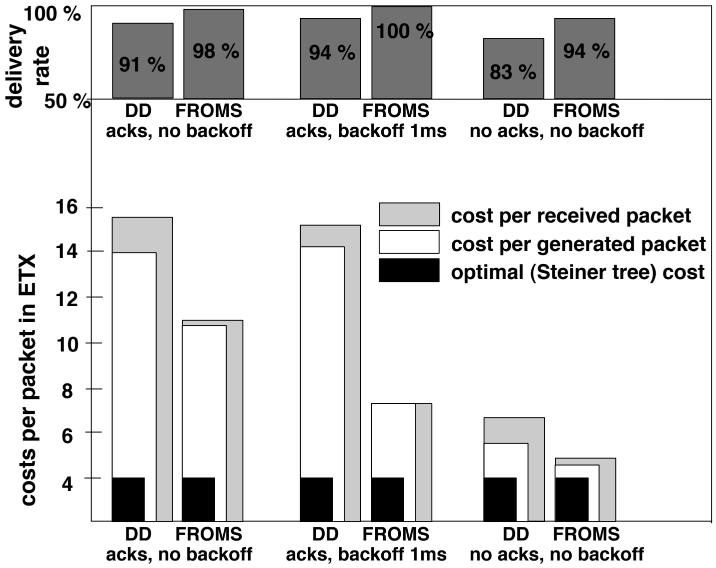


Figure 13: In-network performance when using acknowledgments, results from topology 2.

showing that the technique is highly effective at improving delivery rates. We implemented a simple algorithm in which a parameter (in our case 0, 1 or 100 ms) is multiplied with the node's ID and this delay is applied before forwarding any packet. This backoff reduces packet collisions and thus increases successful delivery.

Another common mechanism to increase the delivery rate is to force packet acknowledgments. We use overhearing of DATA packets as implicit acknowledgments, avoiding additional costs. Figure 13 shows how routing costs skyrocket, while the delivery rate also increase. The incurred overhead stems from re-sending unacknowledged packets. Communication failures cause not only data loss, but also loss of acknowledgments. This results in resending packets which were actually received, but not acknowledged. Consequently, the communication traffic explodes, leading to even higher loss rates.

	heuristic		number of sinks				
	Nr	C	2	3	4	5	6
PST size [bytes]	**10**	**3**	25	115	737	2469	17326
	5	**1**	29	77	369	1437	6590
	4	**1**	19	51	253	671	4682
	2	**1**	10	36	192	215	1731
Overhead [norm.]	**10**	**3**	1	1	1.03	1.03	1.06
	5	**1**	1	1.07	1.08	1.09	1.12
	4	**1**	1.05	1.06	1.04	1.06	1.12
	2	**1**	1	1	1.02	1.03	1.15

Table 4: PSTable pruning heuristics, evaluated in terms of PSTable size (in bytes) and achieved overhead per packet (normalized by optimal Steiner).

9 Parameter analysis of FROMS (simulated environment)

In this section we explore the performance of FROMS under various parameter settings. We conduct these experiments on simulation, since it allows us to more efficiently explore larger parameter spaces.

9.1 Route storage heuristics

As discussed in Section 6.4, different heuristics can be applied to the PSTable, limiting its size and thus saving memory on the nodes and speeding up the learning process. We consider two PST route pruning heuristics: limiting the number of routes per sink to Nr, and limiting the maximum route cost to a sink to $bestCost + C$. Both types of information refer to the routing table (see Figure 4), *before* the sub-actions and actions are computed and initialized. As the PST size decreases, fewer actions are available for selection. Because the best route may be among those pruned, we expect the protocol performance to decrease as the size of the PST decreases. This trend is shown for FROMS ϵ - *greedy* in Table 4 for various values of Nr and C and for multiple numbers of sinks. In this experiment we compare the routing overhead of FROMS in terms of number of transmissions (ETX) against an optimal Steiner tree.

Interestingly the largest table (with $(Nr = 10, C = 3)$ does not always discover the best routes. This is due mainly to packet loss, especially when the number of sinks in the network increases. This causes higher data traffic and thus more data loss.

In the remainder of our experiments, however, we do NOT use any route pruning heuristics in order to limit the number of used parameters and simplify evaluation and understanding of the results. Furthermore, as we already showed in the previous Section 8.1, we are able to implement FROMS with no route pruning heuristics on real hardware. In case the implementation needs to be restricted because of a very large number of sinks or very high density of the nodes, we suggest using a moderate size for the PSTable with $(Nr = 4, C = 1)$ that yields route costs close enough to optimal.

9.2 Exploration strategies

In the next paragraphs we explore the behavior and performance of FROMS with different exploration strategies. We consider the following four types of strategies: *ϵ - greedy*, *ϵ - greedy with temperature*, *decreasing ϵ - greedy* and *uniform-stochastic* (see also Section 6.9).

The experimental results are shown in Figure 14. In the top graphs we fix the number of nodes to 50, the number of sources to 1 and vary the number of sinks from 1 to 5. All exploration strategies are normalized by FROMS *decreasing ϵ-greedy*. The deviation of the first node death time (left graph) is only insignificant and does not exceed 1%. On the other hand, the differences in the routing overhead (number of ETX per delivered packet in the network) reaches 10%. This deviation is a result of the MAC layer. BMAC uses only broadcast transmissions. This diminishes small differences in the number of sent packets (ETX) from the routing layer, and shows how important the MAC layer is for minimizing the energy spent and maximizing network lifetime. On the other hand, less traffic is always an advantage since it increases the delivery rate and thus the overall efficiency of the network.

The rest of the graphs in Figure 14 present experiments with varying number of sources and nodes respectively. They support the above made observations. Given the results obtained in this step, we will consider two exploration strategies in our comparative analysis experiments: decreasing ϵ-greedy with $\epsilon = 0.5$ and ϵ-greedy with $\epsilon = 0.1$.

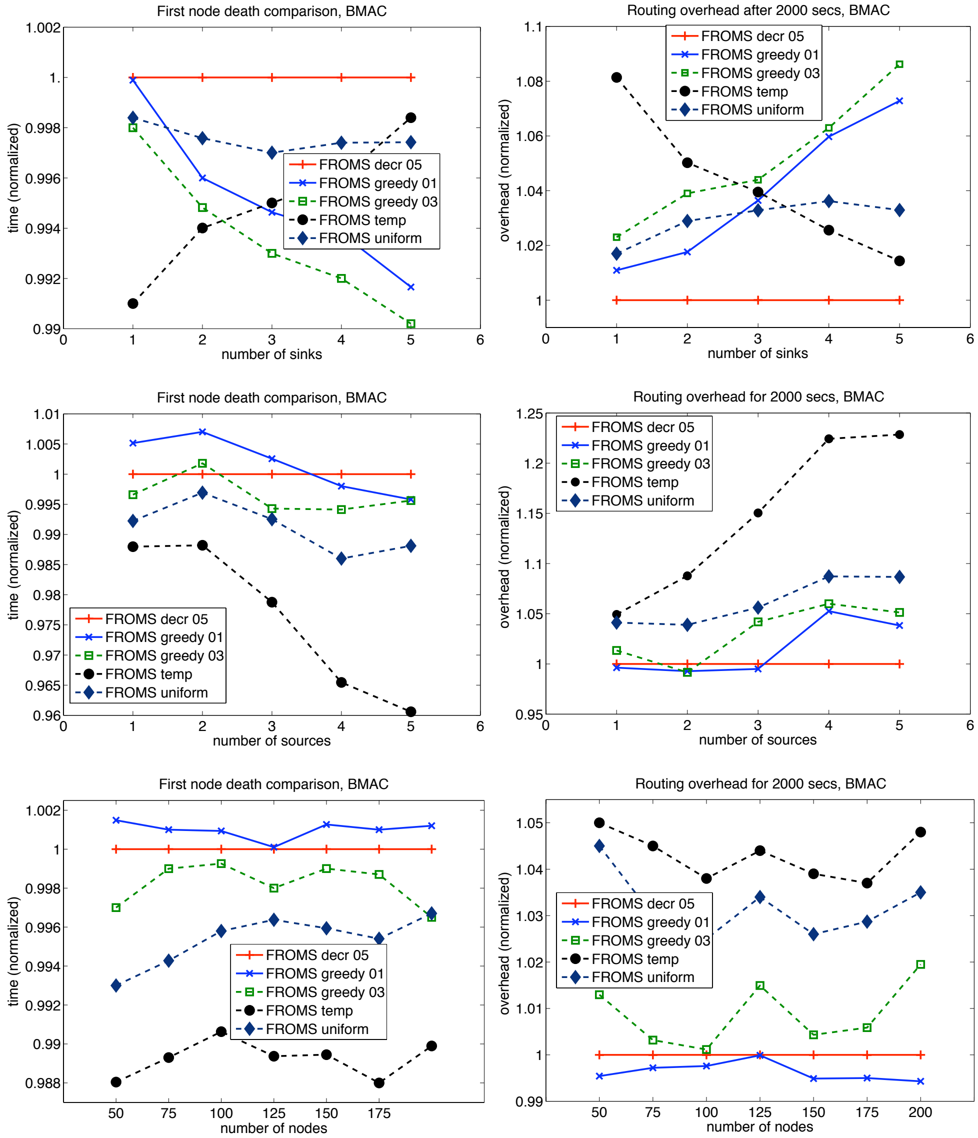


Figure 14: Evaluation of exploration strategies, mean over 50 different topologies, 5 runs each; the network consists of (top) 50 nodes, 1 source and 1-5 sinks; (middle) 50 nodes, 1-5 sources and 3 sinks; (bottom) 50-200 nodes, 1 source and 2 sinks. All experiments performed with BMAC.

9.3 Cost functions

As we already showed in Section 6.8, FROMS can work with nearly any cost function: hops, location information, remaining energy on the nodes, delay, etc. An important property of the used cost function is its localized nature, as FROMS allows direct exchange of information only among one-hop neighbors. The cost function in FROMS is used in three places - initialization of route costs, computation of costs to reach some neighbor and comparison between routes. All these functions are independent from the rest of FROMS and can be easily exchanged.

In this section we concentrate on two main cost functions: a hop-based one and a combined hop and remaining energy based one, as already introduced in Section 6.8. Recall that the goal of the first one, *hop function*, is to select globally shortest routes to reach all sinks. Instead, the second one, *hop-battery function*, favors shorter routes with higher remaining batteries of the involved nodes, thus spreading the energy expenditure throughout all nodes of the network. Everything else in the FROMS implementation remains the same: data structures, exploration strategies, feedback, etc.

Figure 15 presents different metrics for two exploration strategies of FROMS with both cost functions. The hop-battery cost functions slightly extends the network lifetime: however, only by at most 1% (top left graph). This is due to the nature of the cost function: on one side, it uses nodes more uniformly and this can be observed in the standard deviations of remaining energies at the nodes from the top right graph. In fact, the hop-battery cost functions decreases the standard deviation of the remaining energies by 10-15%. However, at the same time FROMS is forced to use longer routes and the routing overhead increases (bottom left graph). These two effects combine into a slightly increased lifetime and slightly decreased spent energy, but the difference to the hop-based cost metric is only insignificant.

Our previous study in a less realistic simulation environment (MATLAB) in [13] showed different results. In that study we achieved a significantly longer network lifetime (by nearly 80%) because the nodes in the network were used more uniformly for routing. However, in a more realistic wireless simulation with a real implementation of a MAC protocol, overhearing of packets among neighbors cannot be avoided and spends a lot of energy. Even when switching to alternative routes with higher batteries, the hop-battery cost function selects routes near to the last used ones, thus further draining the batteries of the nodes. An extreme example are the direct neighbors

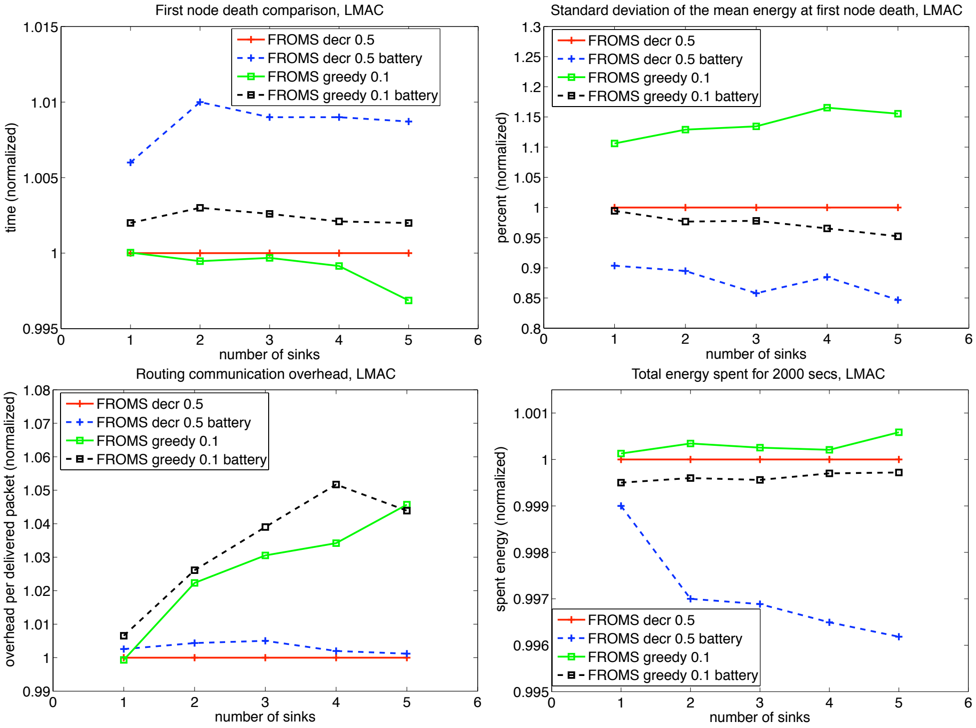


Figure 15: Comparison for two different exploration strategies of FROMS with *hop-based cost* and *hop-battery based cost*. All experiments performed with LMAC.

of the source: whatever route is taken to the sinks, the neighbors of the source always overhear the packets and drain their batteries. In fact, we discovered that usually either the source itself, neighbors of the source, or direct neighbors of the sinks are the first to die.

However, battery-hop cost functions are a good solution when the data delivery task to the sinks is short. In such cases, the cost function is able to spread the usage of the nodes more uniformly, thus avoiding building low-battery bottlenecks in the network.

10 Comparative evaluation of FROMS in simulation

We compare the performance of Froms against three other state-of-the-art routing protocols: multicast Directed Diffusion (MDD), unicast Directed Diffusion (UDD) and Msteam. We explore all of the routing protocols in terms of their routing overhead, network lifetime, standard deviation of the remaining energy on the nodes, and total spent energy in various network scenarios, including mobile sinks and node failures. We use the simulation environment as described in Section 7.

10.1 Multi-source multi-sink routing

In this section we make an extensive scalability analysis and comparison between Froms, Msteam, MDD and UDD, as outlined in Section 7. Similar to the stand-alone evaluation of Froms in the previous Section 9 we fix all network parameters except for one and give mean results over 50 different random connected topologies with 5 random runs each. Figure 16 presents the obtained results for different number of sinks (top), number of sources (middle) and number of nodes (bottom) while using BMAC as MAC layer protocol. The achieved results with LMAC were very similar and graphs are omitted. The first point of Froms (e.g. one sink, one source or 50 nodes) is used as the point of normalization. Unlike the experiments for the stand-alone evaluation of Froms in Section 9, here we are interested in the scalability analysis and comparison of all routing protocols. Thus, we need to use only a single point for normalizing the results and not a full line.

Coming back to Figure 16, with increasing number of sinks, all protocols spend more energy. However, Froms achieves the least energy expenditure compared to the other protocols. This is due to two reasons: its ability to find optimal multicast routes and the limited use of broadcast sink announcements. The lower energy expenditure of Froms compared to MDD are also due to these reasons. However, it is interesting to note that Msteam (both variations) reaches energy spendings well over MDD and Froms. In fact, Msteam-const (where the cost of sending a packet between two nodes is considered to be constant) performs much better than the original Msteam protocol. This is due to the fixed transmission power of the simulated nodes (which is often also the case in real hardware). The original Msteam proto-

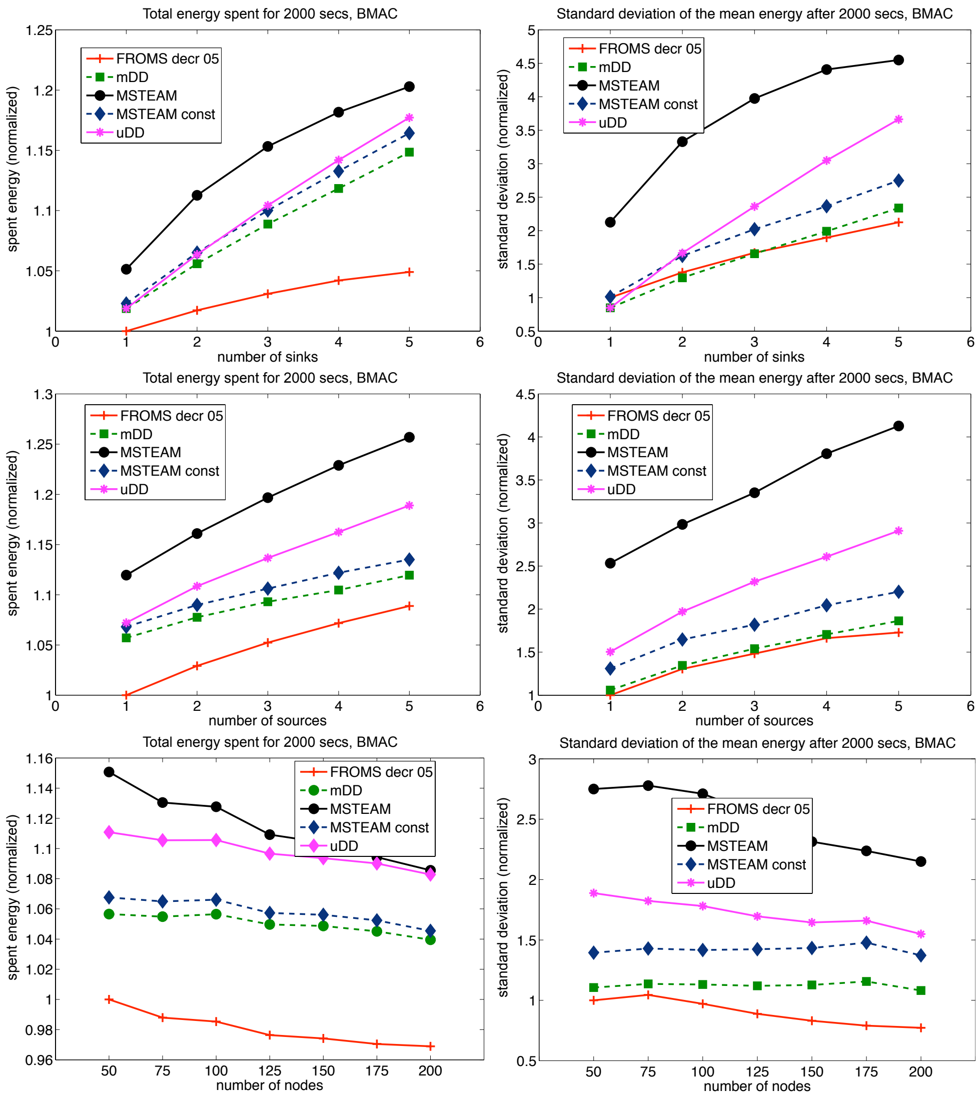


Figure 16: Evaluation of routing protocols in terms of total spent energy and standard deviation of remaining energy after the first 2000 seconds, 50 different topologies, 5 runs each; the network consists of (top) 50 nodes, 1 source and 1-5 sinks; (middle) 50 nodes, 1-5 sources and 3 sinks and (bottom) 50-200 nodes, 1 source and 3 sinks. All experiments performed with BMAC.

col uses a special cost function, which increases the cost of sending a packet with increasing distance between the nodes. This cost function is based on geographic distance rather than taken from real experimental data and thus forces the protocol to take more, shorter hops instead of less long hops.

FROMS clearly outperforms any of the protocols in this comparative analysis in terms of energy expenditure, but especially the geographic-based protocol MSTEAM. The reason for this is the so-called face routing in geographic protocols, which handles void areas (nodes with no progress against the sinks). In these cases, the packet is sent back and follows a predefined route over a "face" until reaching again a node with some positive progress towards the sinks. However, this face route is possibly very long. Second, and more importantly, the exact same route will be taken for each packet, including the sending back of the packet. This incurs excessive and unneeded routing overhead, where reinforcement learning will avoid the repetitive sending to void nodes and back.

The same observations can also be made for varying number of sources in Figure 16 (middle). In contrast, Figure 16 (bottom) shows the good scalability of all protocols when varying the number of nodes (the density of the network is constant). This is due to the localized nature of all protocols, which are independent of the size of the network. The comparative analysis, therefore, shows exactly the same trend as before. We have observed the same order of the protocols' performance also in terms of first node death and routing overhead per packet (graphs omitted for space considerations).

In summary, FROMS achieves between 10 and 22% longer network lifetimes in terms of first node death, around 2 times less routing overhead, between 5 and 15 % less spent energy and 2 to 3 times lower standard deviation of the remaining energies against the other compared routing protocols. The second best protocol in terms of these metrics is MDD. Next comes the constant-cost variation MSTEAM-CONST, then UDD and then the original MSTEAM protocol.

Last, we present a comparison of all routing protocols over BMAC and LMAC, see Figure 17. This comparison is not intended as an evaluation of the MAC protocols in use. Its goal is rather to show the importance of cross-layer design between routing and MAC protocols. In fact, LMAC achieves longer network lifetime (by 20-25%) and lower energy expenditure (by 20-25%) against BMAC in the network scenarios which we explored. Thus, in our scenarios LMAC is the better choice. However, this evaluation will probably change with different data rates in the network and another choice

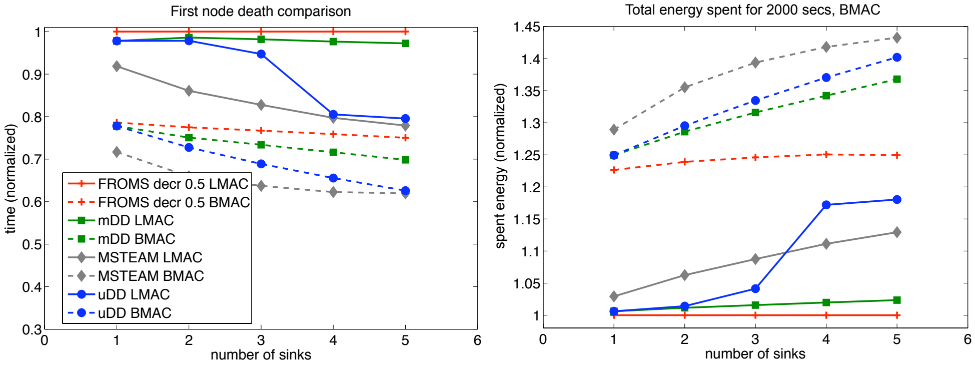


Figure 17: Joint comparison of routing protocols with LMAC and BMAC.

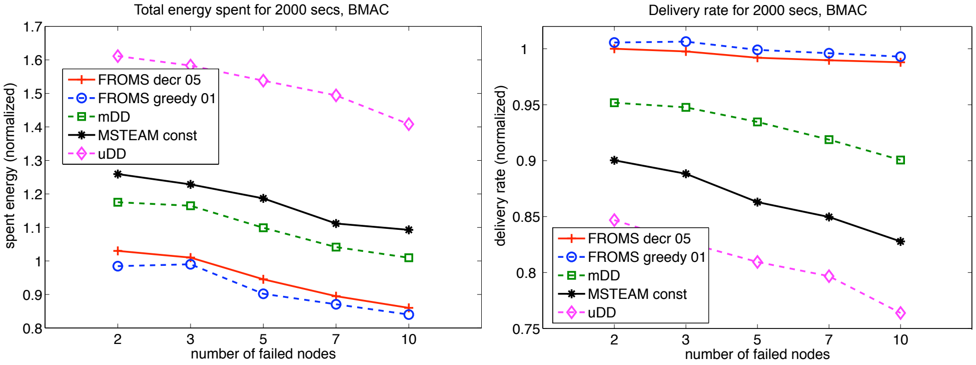


Figure 18: Comparison of delivery rate and spent energy for different routing protocols with varying number of failed nodes in the network.

of a MAC protocol might be necessary.

10.2 Recovery after failure

An important feature of FROMS is its ability to recover quickly after node failures. The protocol keeps track of which neighbors are responding and which are not, as explained in Section 10. In case some neighbor is not reachable any more, FROMS switches directly to the next best route. The new costs are propagated as feedback through the network and learned at all affected nodes. In this section we compare the performance of the four routing protocols in various node failure scenarios.

We have designed a failure experiment where all but a small set of nodes are given full battery levels. The small set of nodes is given only one third of the usual battery level and are thus expected to die quickly one after another. We consider this scenario more realistic compared to a controlled killing of nodes at some predefined time, since in real deployments nodes do not die simultaneously. The results of the experiment in terms of delivery rate achieved and total spent energy are given in Figure 18 for a set size of failed nodes between 2 and 10 (approximately $4 - 20\%$ of all nodes). Each point represents a mean over 50 different topologies, with 30 different random sets of failed nodes. Note that results are gathered only for connected topologies. In case failing of nodes actually disconnected the topology, the scenario was ignored. The achieved standard deviation of the experiments is around $2.3 - 3\%$.

Froms achieves the highest delivery rate and the least energy spent. This is mainly due to the availability of alternative routes at each node and the feedback, which quickly propagates through the network, not only recovering routes but recovering the best ones. Similarly, mdd also monitors the neighborhood through the Froms node failure detection module, and has alternative routes at the nodes. Its achieved delivery rate is 2-5% less than the one for Froms, due to the learning behavior of Froms. On the other side, Msteam (we tested here only the better performing constant cost variation of Msteam) uses much longer routes (see again Section 10), which incur more packet loss. Additionally, the neighborhood failure detection does not work as efficiently as for Froms and mdd because Msteam uses exactly the same route over and over again. Thus, in the case of failures of some nodes on a route, it will still be used until the failure detection module deletes the neighbor. Only then will an alternative be used, which might again have failed. In contrast, mdd and Froms use same-cost alternative routes in a round-robin manner and thus spread the risk of taking a failed route. For udd the scenario becomes even worse, since it relies on a single route which needs to be updated by sink announcements.

In terms of energy expenditure, Froms ϵ-greedy performs the best, because of its continuous exploration. Instead of exploring only on demand, ϵ-greedy keeps track of all possible routes and updates their costs proactively. Thus, when a failure is detected, not only an alternative route is available, but its quality is up-to-date and the best possible route can be taken. Additional exploration and taking of non-optimal routes is avoided, delivery rate is increased because of shorter routes (Figure 18 right), and spent energy is minimized (Figure 18 left).

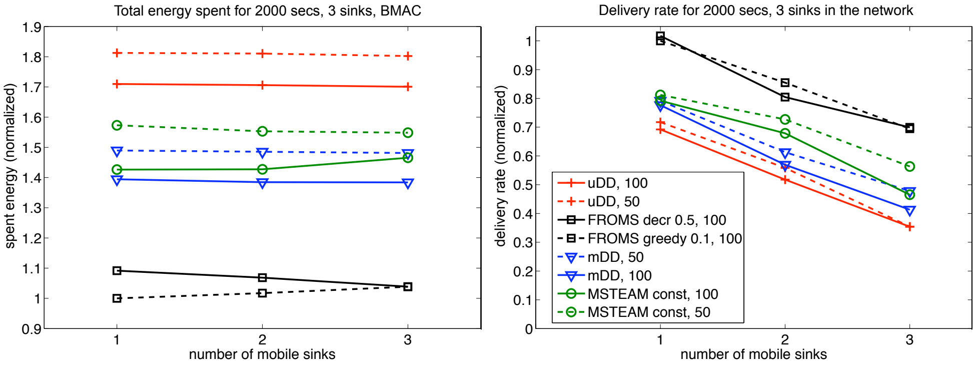


Figure 19: Evaluation of all routing protocols with various number of mobile sinks in the network.

In summary, keeping alternative routes, using shortest possible routes, and keeping track of the real length of all available routes (not only of the shortest ones), is a good strategy to be able to recover quickly after failures.

10.3 Sinks mobility

For testing the performance of the routing protocols under scenarios with sink mobility, we designed two different experiments: one with different numbers of mobile sinks, and a second with different velocities of the mobile sinks. The experiments for both of them were conducted over 50 random topologies, with 10 random runs on each. We achieved a standard deviation of the results of $1.6 - 1.9\%$.

The results from the first experiment are presented in Figure 19. Here, we used a network size of 50 nodes, with 3 sinks and 1 source. We varied the number of mobile sinks from 1 to 3, leaving the rest of them static. The velocity of the mobile sinks is constant and is set to $1m/s$. We varied the sink announcement periods for MDD, UDD, and MSTEAM-CONST. The assumption is that refreshing the routes on the nodes more often would lead to better delivery rate and shorter routes in case of mobile sinks. This can be seen for all protocols in Figure 19 (right). In fact, the delivery rate compared to longer sink announcement periods increases slightly. However, this happens only at the cost of increasing data traffic and thus higher energy expenditure. Figure 19 (left) shows that energy expenditure increases non-proportionally to the achieved gain in delivery rate, and is thus not worth it.

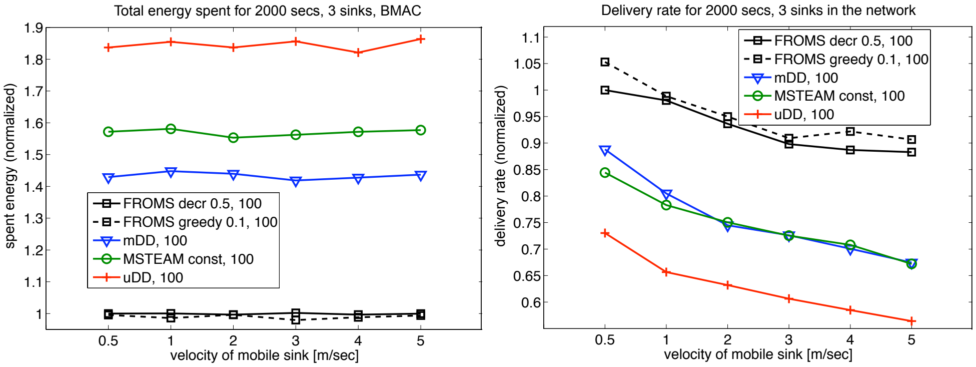


Figure 20: Evaluation of all routing protocols with various velocities of the mobile sinks in the network.

In terms of energy expenditure (Figure 19 left), all protocols scale well with increasing number of mobile sinks. The reason for this is simple: the mobility of the sinks does not invoke any additional mechanisms, such as re-transmissions, which might influence the energy expenditure. However, it can be clearly seen that for all protocols the delivery rate drops with multiple mobile sinks (Figure 19 right). This is because the mobility affects the quality of the used links and some links disappear.

Comparing the routing protocols, Froms has the least energy expenditure of all of them and still achieves the best delivery rates. This is again due to several factors: there are no regular retransmissions of sink announcements, data traffic is routed along shorter paths, and the learning mechanism keeps the routes up to date. As in our previous experiments, mdd and Msteam-const perform similarly well, while udd spends the most energy and achieves the lowest delivery rate.

In our second experiment presented in Figure 20 we vary the velocity of the mobile sink. One sink is mobile and its velocity is $0.5m/s$ to $5m/s$, which corresponds to slow human walking ($2km/h$) through slow car driving ($20km/h$).

In terms of energy expenditure (Figure 20), the behavior of the routing protocols is the same as in the previous experiment. Froms has a significantly lower energy expenditure than the others, followed by mdd, Msteam-const, and finally udd. The reasons are the same as before.

The trend of the delivery rate in case of higher velocities is also as expected. It drops with higher velocities, less for Froms and slightly more for the other protocols. The difference comes from the learning mechanism of Froms, which not only substitutes the sink announcement re-broadcasts, but enables faster recovery of routes.

In summary, these experiments show clearly the innate ability of Froms and its learning algorithm to quickly recover routes in case of mobile sinks, even for a moderate velocity of $20km/h$. Compared to all the other routing protocols, it spends significantly less energy, incurs less data traffic, and achieves considerably higher delivery rates.

11 Future directions and open issues

Current routing protocols, including Froms, consider route characteristics based on the properties of the nodes involved in routing, such as the number of hops, remaining battery levels etc. However, during our work on Froms we learned that the properties of the neighboring nodes are equally important: e.g., a node which is a neighbor of two independent routes drains its battery twice as quickly as the forwarding nodes because of message overhearing from both routes. In the future we plan to incorporate this observation into our model and spread the battery expenditure among all nodes, whether they are involved in routing or not.

Additionally, we plan to design and implement a multicast MAC protocol, as briefly discussed in Section 6. This will enable us to compare the gain from delivering rewards to all overhearing nodes against the saved energy from avoiding overhearing.

This paper presented two important contributions. First, it introduced and evaluated Froms, a highly flexible and robust multicast routing protocol. The results achieved under various environments and network scenarios clearly demonstrate its outstanding performance compared to three other state of the art routing protocols. However, even more importantly, this paper demonstrated the applicability and the potential of machine learning for solving hard problems in WSNs. We showed that learning can be efficiently implemented on real WSN hardware, that it is fully distributed and that it achieves better results in uncertain and unreliable environments compared to any traditional routing protocols. Encouraged by these results and the experience gathered with Froms, we plan to apply reinforcement learning and other machine learning techniques to other WSN problems. We will explore their potential for clustering, neighborhood management, medium access and data modeling.

References

[1] J. M. Kahn, R. H. Katz, K. S. J. Pister, Next century challenges: mobile networking for “Smart Dust”, in: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking (MobiCom), Seattle, WA, USA, 1999, pp. 271–278.

[2] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, M. Welsh, Deploying a wireless sensor network on an active volcano, IEEE Internet Computing 10 (2) (2006) 18–25.

[3] K. Martinez, P. Padhy, A. Riddoch, R. Ong, J. Hart, Glacial Environment Monitoring using Sensor Networks, in: Proceedings of the 1st Workshop on Real-World Wireless Sensor Networks (REALWSN), Stockholm, Sweden, 2005, p. 5pp.

[4] K. Langendoen, A. Baggio, O. Visser, Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture, in: Proceedings of the 20th International Symposium on Parallel and Distributed Processing Symposium (IPDPS), Rhodes Island, Greece, 2006, p. 8pp.

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

[6] E. Cayirci, T. Coplu, SENDROM: sensor networks for disaster relief operations management, Wireless Networks 13 (3) (2007) 409–423.

[7] I. F. Akyildiz, O. A. Akan, C. Chen, J. Fang, W. Su, Interplanetary internet: state-of-the-art and research challenges, Computer Networks 43 (2) (2003) 75–112. doi:http://dx.doi.org/10.1016/S1389-1286(03)00345-1.

[8] B. Malakooti, H. Kim, K. Bhasin, Human & robotics technology space exploration communication scenarios: Characteristics, challenges & scenarios for developing intelligent internet protocols, in: Proceedings of the 2nd IEEE International Conference on Space Mission Challenges

for Information Technology (SMC-IT), Pasadena, CA, USA, 2006, pp. 322–329.

[9] B. Raman, K. Chebrolu, Censor networks: A critique of “sensor networks” from a systems perspective, ACM SIGCOMM Computer Communication Review 38 (3) (2008) 75 – 78.

[10] H. Frey, F. Ingelrest, D. Simplot-Ryl, Localized minimum spanning tree based multicast routing with energy-efficient guaranteed delivery in ad hoc and sensor networks, in: Proceedings of the 9th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM), Newport Beach, CA, USA, 2008, pp. 1–8.

[11] F. Silva, J. Heidemann, R. Govindan, D. Estrin, Frontiers in Distributed Sensor Networks, CRC Press, Inc., 2003, Ch. Directed Diffusion, p. 25pp.

[12] A. Förster, A. L. Murphy, FROMS: Feedback routing for optimizing multiple sinks in WSN with reinforcement learning, in: Proceedings 3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Melbourne, Australia, 2007, pp. 371–376.

[13] A. Förster, A. L. Murphy, Balancing Energy Expenditure in WSNs through Reinforcement Learning: A Study, in: Proceedings of the 1st International Workshop on Energy in Wireless Sensor Networks (WEWSN), Santorini Island, Greece, 2008, p. 7pp.

[14] G. Wittenburg, K. Terfloth, F. López Villafuerte, T. Naumowicz, H. Ritter, J. Schiller, Fence monitoring – experimental evaluation of a use case for wireless sensor networks, in: Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN), Delft, The Netherlands, 2007, pp. 163–178.

[15] S. R. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong, TinyDB: an acquisitional query processing system for sensor networks, ACM Transactions on Database Systems 30 (1) (2005) 122–173.

[16] G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, The hitchhiker’s guide to successful wireless sensor network deployments, in: Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys), New York, NY, USA, 2008, pp. 43–56.

[17] E. A. Basha, S. Ravela, D. Rus, Model-based monitoring for early warning flood detection, in: Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys), New York, NY, USA, 2008, pp. 295–308.

[18] J. McCulloch, P. McCarthy, S. M. Guru, W. Peng, D. Hugo, A. Terhorst, Wireless sensor network deployment for water use efficiency in irrigation, in: Proceedings of the Workshop on Real-world Wireless Sensor Networks (REALWSN), Glasgow, Scotland, 2008, pp. 46–50.

[19] M. Ali, U. Saif, A. Dunkels, T. Voigt, K. Römer, K. Langendoen, J. Polastre, Z. Uzmi, Medium access control issues in sensor networks, SIGCOMM Computation and Communication Review 36 (2) (2006) 33–36.

[20] K. Langendoen, Medium access control in wireless sensor networks, in: H. Wu, Y. Pan (Eds.), Medium Access Control in Wireless Networks, Volume II: Practice and Standards, Nova Science Publishers, Inc., 2007, p. 22pp.

[21] A. Woo, T. Tong, D. Culler, Taming the underlying challenges of reliable multihop routing in sensor networks, in: Proceedings of the 1st international conference on Embedded networked sensor systems (SenSys), Los Angeles, CA, USA, 2003, pp. 14–27.

[22] C. E. Perkins, E. M. Royer, Ad-hoc on-demand distance vector routing, in: Proceedings of the 2nd IEEE Workshop on Mobile Computer Systems and Applications (WMCSA), New Orleans, USA, 1999, pp. 90–100.

[23] L. Ji, M. S. Corson, A lightweight adaptive multicast algorithm, in: Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM), Vol. 2, Sydney, Australia, 1998, pp. 1036–1042.

[24] J. G. Jetcheva, D. B. Johnson, Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks, in: Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobHoc), Long Beach, CA, USA, 2001, pp. 33–44.

[25] S. J. Lee, W. Su, M. Gerla, On-demand multicast routing protocol in multihop wireless mobile networks, Mobile Networks and Applications 7 (6) (2002) 441–453.

[26] R. Vaishampayan, J. J. Garcia-Luna-Aceves, Efficient and robust multicast routing in mobile ad hoc networks, in: Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), Fort Lauderdale, FL, USA, 2004, pp. 304–313.

[27] B.-R. Chen, K.-K. Muniswamy-Reddy, M. Welsh, Ad-hoc multicast routing on resource-limited sensor nodes, in: Proceedings of the 2nd International Workshop on Multi-hop ad hoc networks: from theory to reality (REALMAN), Florence, Italy, 2006, pp. 87–94.

[28] G. Di Caro, F. Ducatelle, L. Gambardella, AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks, European Transactions on Telecommunications 16 (2005) 443–455.

[29] B. Karp, H. T. Kung, GPSR: greedy perimeter stateless routing for wireless networks, in: Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom), Boston, MA, USA, 2000, pp. 243–254.

[30] J. A. Sanchez, P. M. Ruiz, I. Stojmenovic, Energy-efficient geographic multicast routing for sensor and actuator networks, Computer Communications 30 (13) (2007) 2519–2531.

[31] M. Zamalloa, K. Seada, B. Krishnamachari, A. Helmy, Efficient geographic routing over lossy links in wireless sensor networks, ACM Transactions on Sensor Networks 4 (3) (2008) 1–33.

[32] D. Braginsky, D. Estrin, Rumor routing algorithm for sensor networks, in: Proceedings of the 1st Workshop on Sensor Networks and Applications (WSNA), Atlanta, GA, USA, 2002, pp. 1–12.

[33] Z. Li, H. Shi, 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 (IWCMC), Honolulu, Hawaii, USA, 2007, pp. 2600–2603.

[34] Y. Chen, S. Ann, Y. Lin, Ve-mobicast: A variant-egg-based mobicast routing protocol for sensornets, in: Proceedings of the IEEE International Conference on Communications (ICC), Vol. 5, Seoul, Korea, 2005, pp. 3020–3024.

[35] H. Luo, F. Ye, J. Cheng, S. Lu, L. Zhang, TTDD: Two-tier data dissemination in large-scale wireless sensor networks, Wireless Networks 11 (1-2) (2005) 161–175.

[36] H. Kim, T. Abdelzaher, W. Kwon, Minimum-energy asynchronous dissemination to mobile sinks in wireless sensor networks, in: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys), Los Angeles, CA, USA, 2003, pp. 193–204.

[37] H. Kim, T. Abdelzaher, W. Kwon, Dynamic delay-constrained minimum-energy dissemination in wireless sensor networks, Transactions on Embedded Computing Systems 4 (3) (2005) 679–706.

[38] E. B. Hamida, G. Chelius, Analytical evaluation of virtual infrastructures for data dissemination in wireless sensor networks with mobile sink, in: Proceedings of the First ACM workshop on Sensor and actor networks (SANET), Montreal, Canada, 2007, pp. 3–10.

[39] J. Predd, S. Kulkarni, H. Poor, Distributed learning in wireless sensor networks, IEEE Signal Processing Magazine 23 (4) (2006) 56–69.

[40] M. Di, E. Joo, A survey of machine learning in wireless sensor networks, in: Proceedings of the 6th International Conference on Information, Communications and Signal Processing (ICICS), Singapore, 2007, pp. 1–5.

[41] A. Förster, 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), Melbourne, Australia, 2007, pp. 365–370.

[42] R. Arroyo-Valles, R. Alaiz-Rodrigues, A. Guerrero-Curieses, J. Cid-Suiero, Q-probabilistic routing in wireless sensor networks, in: Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Melbourne, Australia, 2007, pp. 1–6.

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

[44] P. Beyens, M. Peeters, K. Steenhaut, A. Nowe, Routing with compression in wireless sensor networks: A Q-learning approach, in: Proceedings of the 5th European Workshop on Adaptive Agents and Multi-Agent Systems (AAMAS), Paris, France, 2005, p. 12pp.

[45] H. J. Prömel, A. Steger, The Steiner Tree Problem, Vieweg, 2002.

[46] C. Watkins, Learning from delayed rewards, Ph.D. thesis, Cambridge University, Cambridge, England (1989).

[47] U. Brandes, T. Erlebach, Network Analysis - Methodological Foundations, Springer-Verlag, Berlin, Germany, 2005.

[48] A. Kuntz, F. Schmidt-Eisenlohr, O. Graute, H. Hartenstein, M. Zitterbart, Introducing Probabilistic Radio Propagation Models in OMNeT++ Mobility Framework and Cross Validation Check with NS-2, in: Proceedings of the 1st International Workshop on OMNeT++, Marseille, France, 2008, p. 7pp.

[49] J. Polastre, J. Hill, D. Culler, Versatile low power media access for wireless sensor networks, in: Proceedings of the the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys), Baltimore, MD, USA, 2004, pp. 95–107.

[50] L. van Hoesel, P. Havinga, A lightweight medium access protocol (LMAC) for wireless sensor networks, in: Proceedings of the 1st International Conference on Networked Sensing Systems (INSS), Tokyo, Japan, 2004, pp. 946–953.

[51] S. Inc., http://www.scatterweb.de/.

[52] K. Zawadzki, A. Förster, Simulating routing protocols for wireless sensor networks, bachelor thesis at the University of Lugano (2008).

[53] A. Förster, A. L. Murphy, J. Schiller, K. Terfloth, An Efficient Implementation of Reinforcement Learning Based Routing on Real WSN Hardware, Proceedings of the 4th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMOB).