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

ANNA FÖRSTER
NetLab, ISIN-DTI, SUPSI, Manno, Switzerland
and
AMY L. MURPHY
FBK-IRST, Trento, Italy

Abstract

[1] 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 and handle mobility. This paper explores in depth the requirements of this set of application scenarios and proposes FROMS, a machine learning-based multicast routing paradigm. Its primary benefits are flexibility to optimize routing over 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 theoretical, simulation and experimentation results supporting these claims, showing the benefits of FROMS in terms of low routing overhead, extended network lifetime, and other key metrics for the WSN environment.

Key words: Sensor networks, routing, energy-aware, multicast, mobile sinks, node failures, reinforcement learning.

[1]This work is partially supported by the EU Cooperating Objects Network of Excellence (CONET). It was partially conducted while both Anna Förster and Amy L. Murphy were with the University of Lugano in Switzerland and were supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

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 sensor networks for applications ranging from scientific monitoring of active volcanos [2] and glaciers [3], through agricultural and environmental monitoring [4, 5, 6, 7], military and rescue applications [8, 9], to the futuristic vision of the InterPlaNetary Internet [10, 11], 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 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 have been developed for various application scenarios and data traffic patterns. However, recently this area has attracted also criticism: application scenarios are too restricted or not carefully described, experimental setups are unrealistic, and simulation environments are too abstract [12]. Further, despite the overwhelming number and variety of routing protocols, key problems remain unsolved, important among them are energy efficiency for various application scenarios and for multiple traffic patterns, as well as tolerance of failures and mobility. 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 applications with periodic, long-lasting data reporting from several sources to multiple, mobile sinks in a multi-hop environment. FROMS easily accepts a range of cost metrics such as hops, geographic distance, latency, remaining battery, etc. Its most salient advantages are:

- Ability to find globally optimal multicast routes;
- Incorporation of different cost metrics and thus optimization goals;

- Quick recovery in case of failures and sink mobility.

The main goal of FROMS is to provide the WSN developer with *a single* 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 both the geographic based multicast routing protocol MSTEAM [13] and a multicast variation of Directed Diffusion [14]. The presented simulation environment uses sophisticated radio propagation models and realistic MAC protocols. In contrast to our previously reported results [15, 16], 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. Sections 4 and 5 present the theoretical aspects, first modeling multicast routing as a reinforcement learning problem and presenting our solution, then offering a theoretical complexity and convergence analysis. Section 6 gives a glimpse into the protocol implementation details before the simulation and testbed evaluations are discussed in Sections 7—9. Finally, Sections 10 and 11 discuss the versatility of FROMS and future directions.

2. Motivation and related efforts

This section outlines the requirements and properties of multiple, well-known WSN deployments. We then discuss the current state of the art in WSN routing protocols and how they meet the needs and challenges of the identified scenario.

2.1. Target application scenario

Real WSN deployments typically follow one of two application styles: periodic reporting or event detection [17]. Our focus is on the first, exemplified by disaster relief and military applications [8, 9], environmental monitoring and surveillance [2, 4, 5, 6, 7] and the InterPlaNetary Internet [10, 11]. We use these applications and scenarios to guide our requirements analysis, but also extend them to incorporate future opportunities. The following details several key parameters and assumptions.

Network size. Deployments range in size from a few, carefully placed nodes (e.g., volcano monitoring [2]) to hundreds of randomly placed nodes (e.g., military or disaster recovery [8, 9]). We target large systems with hundreds of nodes.

Data sources and sinks. An interesting class of applications collects data from a subset of the nodes (possibly changing over time), and delivers the data to a small number of possibly mobile sinks. In very large deployments the set of reporting nodes is usually small (in contrast to convergecast applications with few nodes, all reporting), but the data must be delivered to several, possibly mobile sinks.

Directly supporting sink mobility enables applications such as disaster recovery [9] to send data directly to mobile users inside the network, rather than collect data at a sink, then redirect it back into the network, incurring additional costs. This increases the reliability of the full system.

Energy constraints. One of the main advantages of WSN nodes is freedom of placement without wiring. This, however, introduces a primary constraint: namely reliance on batteries in scenarios where replacement is difficult or impossible. It is widely demonstrated that radio communication, for both listening and sending, is the primary power consumer [18, 19]. Therefore, data dissemination protocols must consider energy consumption, efficiently balancing usage throughout the network, avoiding the loss of any portion of the network.

Node failures. Node losses are commonly caused by the exhaustion of battery reserves or hardware faults, leading to nodes dropping out of the network. A routing framework must cope with such failures and guarantee continuous data delivery throughout the system lifetime. It should also accommodate new nodes and make efficient use of all available resources.

Data rate and requirements. Generally data can be categorized as high (e.g., from accelerometers) or low rate (e.g., from temperature sensors). We target low rates that do not congest the network but can fluctuate under application control.

2.2. Design and implementation requirements

In addition to application constraints, we consider several design criteria for our protocol development process, with the overall goal to ensure the real world applicability of our results.

Simplicity. The protocol must be easy to understand and implement, making it feasible for deployments.

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

Flexibility. The protocol must be adaptable to different applications and optimization goals, such as different cost metrics.

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

2.3. Assumptions

Our work resides at the routing layer, assuming standard functionality from other components of the communication stack:

Sink announcements. The sink uses a simple controlled flooding mechanism to inform all network nodes about itself and its data requirements (e.g., data type and rate). During this controlled flooding, each node sends the packet exactly once and gathers initial routing information such as hops to the sink or sink location.

MAC layer. We assume a simple broadcast-enabled MAC protocol without re-transmissions and without delivery guarantees. This class of MAC protocols is broad and many WSN MAC protocols meet these criteria [19]. Using a retransmission based MAC protocol is also possible, but would require some adjustments to FROMS, as detailed in Section 10.

Neighborhood (link) management. Neighborhood management protocols keep consistent information at each node about the identity of its neighbors as link qualities change and nodes fail. In this paper, we intentionally do not employ such a protocol, but instead demonstrate that our approach alone copes with link unreliability and the associated packet loss. Notably, however, the combination of FROMS with a neighborhood management protocol would increase the overall delivery rate by forcing FROMS to use only good links. The implications of using a link management protocol and the needed adjustments to FROMS are discussed in Section 10.

2.4. Related work

Many WSN routing protocols have emerged in recent years. We first discuss traditional widely used routing protocols, then focus on solutions that address the two key aspects of our application scenario: 1) routing to multiple destinations in large networks, 2) mobility of those destinations. We also devote space to machine learning-based techniques.

Traditional WSN routing. Most of the applications outlined here [2, 4, 5, 7] use protocols from the same family, namely MintRoute [20], MultihopLQI [21], or the Collection Tree Protocol (CTP) [22]. These protocols target convergecast scenarios, with all nodes reporting to a single static base station. While they perform well in this scenario, they cannot be easily extended to support multiple and mobile sinks. CTP, for example, uses neighborhood beacons to update ETX-based routing costs to a sink and to detect loops and/or failures. Beacons are exchanged throughout entire subtrees when routs need to be updated. Extending this mechanism to multiple and/or mobile sinks will likely induce beacon storms for continuous route cost updating.

Multicast routing for WSNs. Although multicast has been well studied in the resource constrained environment of mobile ad hoc [23] and mesh networks [24], these protocols generate an unacceptable amount of communication overhead to construct and maintain the routing infrastructure and thus cannot be successfully applied to WSNs [25].

One alternative comes with geographic routing solutions, which rely on the location-awareness of the nodes and use only next hop information to route packets. For example, GPSR [26] selects the next hop for a packet based on its progress to the destination in terms of distance to the sink. A special face routing procedure provides a mechanism to circumnavigate void regions. GMR [27] and MSTEAM [13] provide geographic-based routing multicast solutions, however, the resulting paths are typically long with long, lossy hops [28], especially when void regions exist. Their implementation is memory and processing intensive. Additionally, mobile sinks pose a major challenge to these algorithms, since the new sink position needs to be propagated to all network nodes.

Another approach is to adapt unicast protocols to send to multiple destinations. Such solutions build paths from a source to each sink without explicitly considering the sharing of paths or finding globally optimal ones. For example, Directed Diffusion [14] can be easily extended to support multiple sinks, but the resulting multicast routes are not optimal.

The work described in [29] attempts to find shared routes from multiple sources to multiple sinks. It works by merging next hops locally and does not explore alternative routes, which can lead to sub-optimal multicast routes. Furthermore, it does not support sink mobility nor recovery of routes.

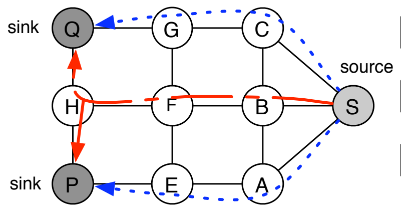
Sink mobility. While most routing protocols assume fixed sinks, several solutions consider sink mobility. For example, the spatiotemporal mobicast routing algorithm [30] is an overlay routing protocol that decides *when* to forward the data through a geographic routing protocol and to which neighbors by using the a-priori known mobility pattern of the sink. In this way it guarantees timely delivery of data to needed regions. A similar approach is taken by [31]. Unfortunately, these approaches exploit a prediction-based routing approach that requires a-priori information of the sink mobility and fails if the sink takes a sightly different route.

In TTDD [32] the authors concentrate on efficient delivery to multiple mobile sinks by clustering nodes into cells. Mobile sinks flood requests only in their local cell, and an overlay routing approach keeps track of the current cells of the sinks for routing data to them. While effective in high mobility scenarios, the overhead to build and maintain the overlay is significant, especially in periodic reporting scenarios, which are more traffic intensive than event-based reporting. Therefore, TTDD is better suited to event-detecting sensor networks with sporadic rather than continuous traffic.

SEAD [33] and its successor DEED [34] attempt to optimize routing from a single source to multiple mobile sinks by allowing each sink to select an *access sensor node*. A data delivery tree is built between the source and all access nodes based on a geographic location heuristic. When the sink moves, a path between its current nearest neighbor and the access node is maintained, eliminating the need to rebuild the tree. However, if the sink moves far away, a new access node is selected and the tree is rebuilt. The approach shows good results in comparison to Directed Diffusion [14] and TTDD [32] in terms of dissipated energy for data packets. However, no evaluation of the control overhead with mobile sinks is presented, and this value is expected to be high.

The authors of [35] present a unicast and multicast enabled protocol for WSNs with strictly limited route length and memory requirements. Although the work offers solid theoretical analysis, it has not been implemented or even evaluated in simulation. Further, the protocol assumes static topologies and does not scale well to large networks, as it maintains routing information to a large subset of network nodes (not only to next hops).

Machine learning based solutions. Machine learning has gained much attention in recent years for solving challenging problems such as routing in wireless ad hoc networks. In a recent survey, reinforcement learning was identified as a well-suited technique for routing in WSNs due to its low overhead, high flexibility and robustness [36]. It has been successfully used for multiple routing problems including geographic routing [37], discovering routes between two nodes [38] and finding optimal compression routes in a convergecast scenario [39]. While these works show clear advantages from



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.

using ML techniques, they do impose moderate cost. Specifically, online learning of the real route costs forces the protocols to sporadically use also longer, non-optimal routes. Therefore, ML should only be applied when the application data requirements remain consistent for *long enough* to overcome the learning costs with the exploitation of the shorter, learned routes. Fortunately this is common among many WSN application scenarios (see also Section 2.1), making ML an appealing approach.

This paper represents the first application, to our knowledge, of machine learning, reinforcement learning in particular, to multicast routing. As we show in the remainder of this paper, this requires the design of a new ML algorithm to correctly reflect the requirements arising from WSNs.

3. Protocol intuition and overview

To satisfy the application needs identified in Section 2.1, our goal is to develop a protocol that finds the optimal path for data to follow from a source to multiple, interested sinks. Optimal can be defined in many ways, e.g., according to delay, hop count, geographic distance, remaining battery level or any combination of the above. This section uses the number of hops as a simple to understand metric, while Section 10 discusses other options in depth.

To illustrate the potential benefits from identifying the optimal route, consider the sample network in 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 shortest paths from the source to each sink (the dotted lines in the figure). However going through nodes B, F and H yields a shorter route. The challenge is to identify this route without full topology information. The main task of our protocol is to exchange *next-hop* routing information among neighbors such that the optimal route is discovered.

Using Figure 1 as an example, the main functionality of our protocol is as follows. An initial sink announcement phase allows all nodes to gather preliminary routing information toward sinks. As shown, S gathers *individual* hop counts for each sink through each neighbor (right table in Figure 1). When a data packet arrives, a node must select one or more nodes to serve as the next hop(s) towards the destinations. While a node could simply choose the best nodes with the shortest individual paths (in our example: C to reach sink Q and A to reach sink P), the node optionally *explores* non-optimal routes based on the assumption that these might have lower costs than those calculated from the individual costs in the routing table. Such lower costs arise because neighboring nodes may be able to *share* subsequent hops to reach the sinks.

To show how information about the shorter paths propagates, consider that initially S estimates that node A requires 7 hops to reach both sinks: 3 hops to sink P, 5 hops to sink Q and the first hop is shared (removing 1 from the total). However, A's initial estimate to reach both sinks through E is only $(2+4)-1=5$ hops, implying that S can reach both sinks though A in 6 hops (the 5 hops of A plus the one hop from S to A). By propagating the cost of 5 from A to S, S can update its routing estimate. This information is piggybacked on all data packets. Therefore, by exploiting the broadcast environment, data can be simultaneously sent to E and routing information propagated back to S. Similarly, E piggybacks its cost estimate, informing A, and so on.

We observe that piggybacked values, which we call feedback, propagate backward in the direction from the sink to the source. Therefore, for accurate hop count information to arrive at the source, multiple packets must be sent down the same path. Further, a node must send packets to all neighboring nodes to explore all possible paths for their real costs. We also note that keeping all routes at all nodes and always supplying current cost estimation as feedback innately allows our protocol to support recovery of failed nodes and mobility.

As described, this solution is a reinforcement learning based routing protocol. The next section formalizes the ideas discussed here and presents the details of the Q-Learning model, offering a multicast solution for WSNs.

4. FROMS: Solving Multicast with Q-Learning

This section offers a model for our multicast routing problem and describes a reinforcement learning solution. This foundation enables our theoretical analysis of complexity, correctness and convergence in Section 5 and

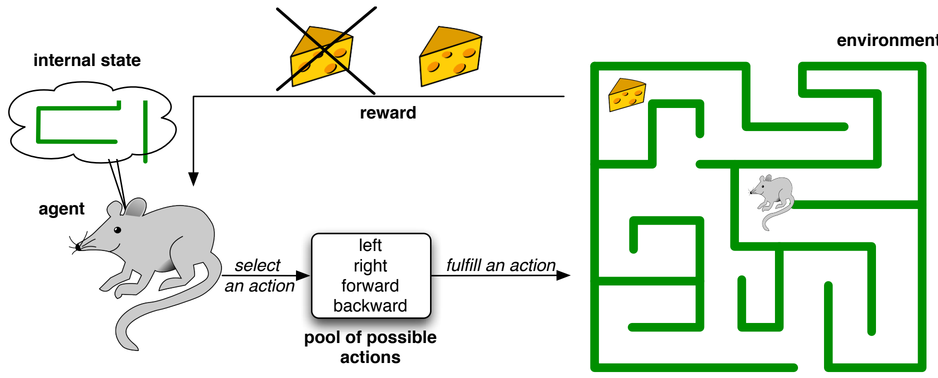


Figure 2: Reinforcement learning. In each step, the agent selects one action and executes it. The environment responds to the agent with a reward, which is used to learn the best possible sequence of actions to find the goal.

is the basis for the implementation described in Section 6.

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 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 vertices include the source and the destinations. The cost of a tree T is defined as a function over its nodes and links $C(T)$, e.g., the number of one-hop broadcasts required to reach all destinations. Alternate cost functions and their usage are discussed in Section 10.1.2.

4.2. Multicast Routing with Q-Learning

Finding the minimum cost routing tree T is analogous to identifying the Steiner tree, therefore finding the exact solution is NP-hard even when the full topology is known. As outlined in Section 3, our goal is to find the optimal route by exploiting only local interactions. For this, we turn to reinforcement learning [40], extending the model that has been successfully applied to other routing problems [39, 38] to accommodate the multiple mobile sink scenario.

Q-learning [40] is a model-free reinforcement learning technique exemplified in Figure 2. It assigns each possible action a *Q-value* representing the approximate *goodness* of the action. During the learning process, the agent selects and executes one action and receives a scalar reward from the environment. This reward is used to update the Q-value. Over time the agent learns the actual action goodness values and is able to select the most appropriate.

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

Agent states. We define the state of an agent as a tuple $\{D_p, routes_{D_p}^N\}$, where $D_p \subseteq D$ are the sinks the packet must reach and $routes_{D_p}^N$ 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. Note that the routing decision can include one or more different neighbors as next hops. Consequently, we need to modify 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 the 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, 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. The original Q-Learning approach randomly initializes Q-Values, and exploits them only for quantitative comparison among actions. Instead, in our case Q-Values represent the real cost of the routes. For example, if the cost function is the number of hops, the Q-Value of a route is also the number of hops of this route. Further, to initialize Q-Values, we use a more sophisticated approach than random assignment. Specifically we calculate a cost estimate based on the individual information known about the involved neighbor and sinks. This non-random initialization significantly speeds up the learning process and avoids oscillations of the Q-Values.

Continuing our example with a hop-based cost function, we estimate the route cost by using the hop counts in a standard routing table, such as that shown in Figure 1. We first calculate the value of each sub-action, then of the 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 $hops_d^{n_i}$ is 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 for transmission to n_i as well as by n_i to reach the next hop, hence subtracting double. Note that this estimation is an *upper bound* on the actual value, as it assumes that the packet will not share any links after the next hop. During learning in a static system, Q-Values will always decrease and the best actions are identified as those with small Q-Values.

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

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

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

It is worth noting that we use hop counts as an easy to understand example, but the Q-Value calculations can be easily modified to incorporate alternate cost metrics. Further discussion appears in Section 10.1.2.

Updating a Q-Value. Agents learn the real values of the actions by incorporating rewards received from the environment. In our case, each neighbor to which a data packet is forwarded sends the reward as feedback describing 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. With randomly initialized Q-Values, a low learning rate is used to avoid heavy oscillation at the beginning of the learning process. However, since we initialize the values with over-estimations of the cost values, and further guarantee that the values only decrease, we avoid the learning delay associated with a low γ and instead use $\gamma = 1$, updating Equation 3 to:

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

which directly updates the Q-Value with the reward. Further, the Q-Values of complete actions are automatically updated as their calculation is based on sub-actions (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 our Q-Values represent the real costs of a route and low hop counts (Q-Values) are better. Furthermore, rewards from the environment are generated and sent in broadcast, and therefore 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 may result in a locally optimal solution as it will ignore some actions that may, after learning, have lower Q-Values. Therefore, a tradeoff is required between *exploitation* of good routes and *exploration* among available routes. A typical widely used efficient strategy is ϵ-greedy, which selects the best available action with probability $1 - \epsilon$ and a random one with probability ϵ. Several variants are considered in Section 7.4.2.

Parameter	**Description**
$\|N\|$	number of nodes in the network N
D	number of destinations
M	network diameter
Y	maximum network density (maximum number of 1-hop neighbors)
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.

5. Theoretical analysis of FROMS

Next we concentrate on the theoretical analysis of FROMS, namely its convergence, complexity, memory, and processing requirements. First we explore an idealized model of the environment and then introduce realistic properties such as asymmetric links and link failures.

5.1. Worst-case complexity and convergence

We discuss first the worst-case complexity of FROMS (time to stabilize) and thus implicitly its convergence. In our scenario, convergence means that first, the protocol is stable and the Q-Values no longer change, 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 [41]. 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 such as the one just presented in Section 4 with $\gamma = 1$, a hop-based cost metric, and a deterministic exploration strategy that chooses routes in a round-robin manner. We further assume a network with nodes in the set N with the following properties, summarized in Table 1: D is the number of destinations, M is the diameter N (the longest shortest path between any two nodes in N) and Y is the maximum density of N (the maximum number of 1-hop neighbors of any node in N). We assume stationary nodes and sinks and perfect, stable communication among neighbors. Without loss of generality, we assume a single source. This is possible because the routes are constructed depending on the destinations, not on the sources. Nevertheless, we discuss multiple sources at the end of this section.

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 a maximum of Y neighbors with repetitions (because we can use the same neighbor to reach multiple sinks) or:

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

In the worst case the source of the data, the initiator of the learning process, is at the maximum distance M from all of the sinks. Our goal is to compute how many action selection steps must be taken by all nodes in N, such that the Q-Values stabilize to the optimal, minimum cost. With $\gamma = 1$ the feedback of any 1-hop neighbor directly replaces the old Q-Value. Thus, to learn the real cost of any route of length M we need exactly $M - 1$ steps, as the real cost propagates backward from the destination one step each time the path is used. However, the source must wait for all other nodes to stabilize their Q-Values before it can be guaranteed that its Q-Values are also stable. In the worst case it must fully explore all possible routes in the whole network.

To count the number of action selection steps S for the whole system to converge we assume the learning is initiated by the source, and by the previous reasoning we know that we must select each of the available routes $M - 1$ times. Using Equation 6 we have:

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

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

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

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, *optimal*, route has been identified as that with the lowest Q-Value. If more than one route have the same Q-Value, a node can alternate between them to spread energy expenditure.

This, admittedly, is a very loose upper bound of the complexity as no real networks have the worst-case properties such as *all neighbors being M hops away from the destinations.* On the other hand, this analysis does offer an idea of scalability and expected performance. In the next paragraphs we discuss how the convergence behavior changes with various network parameters and the consequences for the protocol. Later, in Sections 7–9, we use experimental evaluations to show the real behavior of the protocol.

Parameter analysis. The number of destinations D and the density Y are *neither* dependent on the number of network nodes $|N|$ nor on its diameter M. To understand the expected performance of FROMS as these parameters vary, we explore how they individually influence the protocol.

The number of sinks D is application defined, and the relationship to other parameters is $D < |N|$. With a growing number of sinks, complexity grows exponentially, as D is in the exponent of Equation 7.

With a growing number of nodes $|N|$, it is most common that either the *diameter M* or the *density Y* grow, or both increase but at a lower rate. In either case, according to Equation 7, the complexity has polynomial growth.

Instead, in a network with a *constant number of nodes $|N|$*, M and Y depend on each other. As the diameter grows, the number of neighbors decreases; and vice versa. In the extreme case of a chain of nodes where nodes have at most two neighbors, $Y = 2$, and the diameter is approximately the same as the network size $M \approx |N|$, we have:

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

Another extreme case is when the density Y grows toward $|N|$ and M decreases toward 2. Note that the case $M = 1$ does not make sense, as any source will be exactly one hop from any sink and routing is trivial. In the case of $M \to 2$ we have:

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

Nevertheless, these equations do not consider behavior for intermediate values. We, therefore, need to explore complexity in a network with constant $|N|$ and different M and Y values. Figure 3 shows a case study for a network of 100 nodes, 3 sinks and varying densities and diameters. The worst-case complexity is presented from two different points of view. As expected, with growing M and Y, the complexity grows. 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, we can generally state that having a lower density is

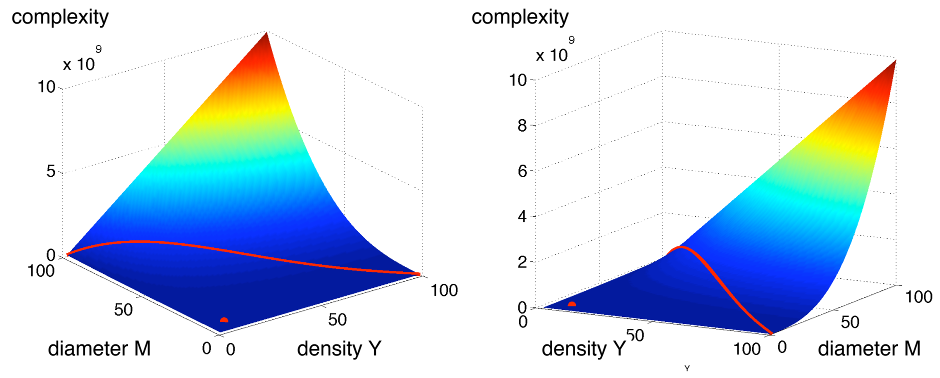


Figure 3: Worst-case complexity for some M and Y values from different views. The number of sinks is fixed to $D = 3, |N| = 100$. The thick line at the welding of the graph corresponds to maximum expected complexity and the single point near the origin to a real dense network with $M = 10$ and $Y = 10$.

always a good idea, since Y is in the power of D in Equation 7, unless M is very low, as the complexity decreases again. Note also that in the extreme case of Figure 3 where both M and Y grow towards $|N|$ is impossible in practice [42]. Realistic values for a network with 100 nodes will be $M = 10$ and $Y = 10$, which corresponds to the single point shown in Figure 3.

Probabilistic exploration strategy. The above complexity analysis applies to a deterministic round-robin exploration strategy. However, both the original Q-Learning algorithm and our protocol use probabilistic exploration strategies: for any route r there is a probability p_r to be chosen at any step s_t. If the probabilities of all routes is $p_r > 0$ over time, convergence is guaranteed. Nevertheless, complexity is difficult to compute because of the non-deterministic nature of the algorithm. Therefore we offer experimental evaluation in the next sections.

Realistic communication. The previous analysis assumes perfect communication. To work in the real world, however, we must consider the effect of packet losses. Fortunately, to maintain eventual convergence, it is enough to assume a non-zero probability of delivering a message between two nodes. While convergence will take longer, eventually messages will be delivered and the Q-Values will be updated.

Scenarios with asymmetric links, in which the delivery probability is lower in one direction, actually reduce to the above scenario. As long as the probability is non-zero, the convergence criteria holds. This covers most practical scenarios [43] and in the rare case that the delivery probability in one direction is zero, the neighbor is deleted. We discuss the practical and implementation challenges of asymmetric links further in Section 6.7.

Multiple sources. In the preceding paragraphs we assumed a single data source learning the optimal routes to all sinks. However, our goal is to consider multiple sources. Such a situation, in fact, speeds up the convergence process in terms of the number of messages each sink must generate to reach convergence. Consider a network with 2 sources sending data at the same rate to 3 sinks. In this case, nodes on the routes between both sources and the sinks receive double feedback from forwarding data packets from both sources. This is because we deliver feedback using broadcast, thus it is received by all neighboring nodes.

5.2. Correctness of FROMS

The correctness of FROMS derives from the definition of the adapted Q-Learning model in Section 4. The goal is to show that after convergence, the Q-Values of the full actions at any node accurately reflect the hop-based costs. We use simple induction to sketch the proof, first showing the correctness of FROMS for one sink, then expanding to multiple sinks.

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

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

Induction step. Assume that a node N (a sink or any other node) has a correct estimation of the cost 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 receive correct cost estimations when a reward is sent.

With this, we have shown 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 hop count 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 sketch 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 where the Q-Values to reach each sink individually have converged at all nodes (according to the above discussion). For simplicity we label 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 correct 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 done above. It also intuitively extends to more than 2 sinks.

Optimality. Combining the results for convergence of Section 5.1 with the correctness above shows that FROMS *converges to the correct hop-based costs* of the routes after a finite number of steps and thus finds the optimal route(s).

5.3. Memory and processing requirements

Two final aspects to consider are the memory and processing requirements at each network node.

Specifically, each node must store all locally available routes. Following Equation 6, the expected storage is $O(Y^D)$. The required processing includes 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 $O(Y^D)$. Updating 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, this yields a worst case for searching $O(Y + D)$.

Note again that the memory requirements for each node do not depend on the network size, but on the number of neighbors. For networks with reasonable densities, the needed memory requirements are moderate, and for high density scenarios we have developed special pruning techniques, described in Section 6.4.

6. Protocol implementation details

The implementation outlined in this section is based on the reinforcement learning model of Section 4 and introduces practical elements such as event-based processing and data structures tuned to the resource restricted environment of WSNs.

Pseudo-code of the Froms protocol is given in Figure 4 and can be considered as three interleaved processes: sink announcement and initialization of routes (lines 1-4), selection of routes (lines 9-11), and learning and feedback (lines 8, 13 and 18). Concrete explanations and implementation details of all sub-processes of Froms are provided in the following sections.

```
1:  on_receive(DATA_REQ req):
2:    new = add_nexthop(req.sinkID,req.neiID,req.costs);
3:    if (new)
4:      broadcast(req);

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:   if (d.nexthops.includes(self))
10:     // route packet to next hop(s)
11:     routes = get_possible_routes(d.my_sinks);
12:     d.routing = strategy.select_route(routes);
13:     d.feedback = best_route_cost(routes);
14:     broadcast(d);
15:   else
16:     // reward neighbors, even if not routing
17:     d.routing = null;
18:     d.feedback = 1;
19:     broadcast(d.header);
20:   end if
```

Figure 4: Froms pseudocode.

6.1. *Sink announcement*

As mentioned in the application scenario of Section 2.1, we assume each sink announces itself via a controlled flooding of a DATA_REQ message. Each node broadcasts this message exactly once, ensuring it reaches all nodes and avoiding a broadcast storm. During this dissemination process, initial routing information such as the number of hops to the sink is gathered (lines 1-4 in Figure 4). Other information including position, battery status, etc, can also be collected.

6.2. Feedback implementation

A key element of FROMS is the exchange of feedback (rewards) among nodes. This enables FROMS to learn the global route costs and to use the globally optimal ones. The feedback information, usually a few bytes in length, is piggybacked on normal DATA packets (lines 13 and 18 in Figure 4), a choice that offers multiple advantages: feedback is sent only on-demand and only to local neighbors; and overhead is minimal because no additional control packets are created. Note that feedback is accepted and route costs are updated even if the feedback is negative indicating that the previously known costs were better. This choice allows FROMS to automatically handle mobility and recovery. As feedback is received by all overhearing neighbors, the learning process is faster than a unicast feedback sent only to the node that originally transmitted the data packet.

Our feedback mechanism is in some ways similar to CTP's [22] neighbor exchange of ETX information, but with two differences that significantly affect the behavior of the routing protocol. First, FROMS feedback is piggybacked in the headers of regular data packets and is overheard by all neighbors, avoiding control packets or beacons even in unstable scenarios. While CTP also exploits this process and uses data packets to relay routing information, it uses also beacons throughout its lifetime, even when the network is stable. The frequency of sending beacons is lower, but never zero, but increases the energy expenditure significantly. Second, precisely because we piggyback feedback on data packets, feedback is exchanged only in the areas of the network with routing activities. Uninvolved network sectors remain silent, expending no energy. In contrast CTP's feedback is exchanged to both update the routing cost and to verify route correctness. Any deviation of the CTP feedback costs from the currently known ones is interpreted as arising from network failures or other significant changes and triggers beacons throughout the subtree. Consequently, directly extending CTP's beacon-based recovery mechanism to multiple and/or mobile sinks will likely trigger beacon storms in the updated areas, increasing congestion. Further, costs will be updated at all nodes in the network. While this is reasonable for convergecast scenarios, the overhead is not acceptable for either unicast or multicast.

6.3. Data structure API

One implementation challenge in FROMS is to design an efficient data structure to support multi-destination routing. This data structure is different from a typical routing table, such as the one in Figure 1, since it not only holds next hop and cost information for individual sinks, but also tracks the costs of shared paths to multiple sinks. In other words, we need a data structure to hold the sub-actions described in Section 4.

The multi-destination routing data structure used by FROMS must efficiently and reliably implement the following interface:

```
add_nexthop(sinkID, nexthop, cost)
```

This function is called when a DATA_REQ arrives, or when feedback for an unknown sub-action arrives. The second case occurs if sink announcements are lost and a potential subsequent hop is unknown at the node. The first time the unknown neighbor broadcasts a data packet the node will repair its routing information.

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

This is called every time the node overhears a data packet and updates the stored cost for a known sub-action. If the sub-action cannot be found, it should be recovered using `add_nexthop`. As previously mentioned, costs are always updated, even if the new value is *worse* than the stored cost, as is expected when a node fails or a sink moves.

```
get_possible_routes(sinks)
```

This is called by the exploration strategy and returns all possible routes. The routing strategy must then select one.

The initial version of FROMS [15], implemented only in simulation, used a data structure called the **PSTree**, **P**ath **S**haring **Tree**, as detailed in [44]. The main idea was to organize the sub-actions into full actions using a tree-like structure. While this was intuitive and easy to describe, the PSTree requires dynamic memory management. As this was not available on our real sensor hardware (see Section 7.2 and [45]), we moved to the static, table data structure as it is both simple and efficient to implement. The most recent implementation of FROMS, as presented here, leverages the so called **P**ath **S**haring **Table** or **PSTable**. Additional implementation details are available in [46].

6.4. Route storage reducing heuristics

As noted in Section 5, storage for routes grow exponentially with the number of sinks and polynomially with the number of neighbors. In practice this means that for large numbers of sinks and neighbors we simply cannot store all routes. The consequence is that we can no longer guarantee the optimality of learned routes. However, near-optimality can be preserved by wisely managing which routes to store and which to drop. In our previous work we developed two pruning heuristics for the PSTree [15] and showed that the optimality of FROMS is only slightly affected, while the memory requirements are reduced significantly. Interestingly, for the scenarios we evaluate in Section 9, we did not need to apply any pruning heuristics, as the full data structure fit comfortably in memory. Nevertheless, for scenarios with higher densities, the pruning heuristics are a valuable tool for reducing memory consumption.

6.5. Loop management

Because FROMS explores non-optimal routes to find the globally best route, it may choose a route with a potentially unlimited length. In other words, it may be that a packet travels in a loop. We manage this problem by adding a simple TTL (time-to-live) to all data packets. Implementation details are available in [46].

6.6. Node failure and mobility management

Our Q-Learning based protocol has the innate ability to manage changing network conditions. They simply appear as feedback and Q-Values are updated during the usual learning process. However, practical challenges arise from the fact that increasing the costs of some route could either mean a mobile sink is moving away or a sink is disconnecting. The first case is to be expected, however the second will cause packets to loop as they travel forever searching for non-existent sinks.

Properly managing mobility can be described in two steps: identification of node failures and maintaining sink freshness. The first is used also for general neighbor failure recognition and is accomplished by simply keeping a timestamp at each node of when it last overheard a packet from each neighbor. If the timestamp becomes too old (a parameter), then the neighbor is considered dead. A special case arises when the failing neighbor was a sink. In this case, the direct connection to the sink might have been lost, but the sink could be still alive and moving away. To identify this situation we also keep a timestamp for each sink. Direct neighbors of the sink propagate the last time they overheard a packet from a sink to inform other nodes that the sink is still alive. Recall from Figure 4, lines 18-19, that sinks implicitly acknowledge the receipt of data packets when sending feedback.

With these two interleaved mechanisms, neighbor failure detection and sink freshness tracking, each node assesses its neighborhood and can choose to use alternative routes, to re-learn optimal routes or to stop data delivery to some sinks. For example, when a node, A, deletes a neighbor, D, because of D's freshness value, A also deletes all routes that include D. However, alternate routes remain with up-to-date Q-Values that can be selected by A immediately to route messages. Recall that each forwarded packet contains the reward, equal to the current, best available Q-Value. If the new, best Q-Value at A is the same as before D's deletion, A's neighbors will not change their Q-Values, even though the actual route has changed. If the reward is worse (higher), A's neighbors will update their costs and the updated values will propagate on future packets until all nodes involved in routing update their costs. Recall that only routing neighbors send feedback, thus limiting the scope of the updates to those on the path as well as their neighbors that overhear the feedback.

It is worth noting that our failure recognition mechanism is not part of the Q-Learning protocol. It is simply a supporting module that sits beside the routing protocol.

6.7. Low quality and asymmetric links

Another major challenge for any routing protocol is to handle low quality and asymmetric links. A recent study has shown that a relatively high percentage of links in real networks can be considered asymmetric or even unidirectional [43]. Various link quality management protocols have been developed either as stand-alone solutions (e.g. Arbutus [47]) or as part of other routing protocols (e.g. CTP [22]). Their main goal is to evaluate the link quality in both directions and provide this data to the higher layer protocols, allowing them to avoid poor links.

Previous applications of RL to routing in WSNs (see Section 2.4) do not exploit such protocols, but instead have assumed perfect links while simultaneously arguing that perfect links are not a requirement to find optimal paths. Nevertheless, the behavior of RL protocols in the face of unreliable links has never been demonstrated. Therefore, for our evaluation of FROMS we show that with realistic, unreliable links and *without* a link management protocol, FROMS does perform properly, learning the best routes.

Despite our choice for this paper, a link quality protocol could efficiently be integrated with FROMS, as our protocol operates above the link layer and can utilize its information. This is discussed in depth in Section 10.

6.8. 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 this section, we concentrate on how the exploration strategy fits into the FROMS implementation rather than what kind of exploration strategies are possible. Section 7.4.2 discusses multiple strategies and evaluates them in the context of the application scenario.

The exploration strategy is used at exactly one place in the FROMS algorithm: when selecting a route. After line 11 returns all possible routes that meet the requirements such as decreasing the total hop count to the sinks, the exploration strategy selects one route to use in line 12. For example, a randomized strategy could decide with some probability to select the optimal (lowest cost) route or to randomly select among all available routes. Another exploration strategy might decide to use the optimal route with some probability or to round-robin among all available routes, etc.

6.9. Cost functions

Similarly to the exploration strategies described above, the cost function is a crucial part of FROMS and here we explain how to implement it instead of what kind of cost functions are appropriate, which instead is discussed in Sections 7.4.2 and 10.

The cost function appears in the implementation in line 2 and is used in lines 11 and 13. Line 2 registers next hops and their costs to the sink. The cost can involve metrics such as battery level of the next hop, ETX or hops to sink. Note that the storage of the cost components can be separate to enable metric-specific requirements (e.g., never use a node with a battery level lower than 10%, irrespective of all other metrics). However, to compare routes based on costs, all metrics need to be combined in a single numerical value, which is either stored separately or computed on the fly.

Line 11 uses the above cost components or total cost to select possible routes to the sinks. Line 13 selects the best among all available routes to reward the nodes' neighbors and to enable learning. These three methods are implemented in a modular way to enable easy substitution of the cost functions.

6.10. Summary

This section presented implementation details of FROMS. The main parameters which need to be specified before deploying FROMS are its cost function and exploration strategies. We explore examples in Section 7.4. Additionally, node failure management is required in nearly any WSN. However, all other presented modules represent special features, such as sink mobility support or route pruning heuristics for extremely memory-restricted 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

In addition to the *theoretical analysis* of Section 5, we offer *evaluation through simulation and on real hardware* to show the properties of FROMS. We show a wide range of metrics for many different network scenarios and protocol parameters.

7.1. Simulation environment.

We use the OMNeT++ network discrete event simulator, together with its Mobility Framework and probabilistic radio propagation model extension [48]. Unfortunately, this combination lacks a true energy expenditure model and realistic MAC protocols, thus we implemented the following additional simulation models:

- **Linear battery model.** A linear battery model that 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 2.
- **MAC protocols.** We implemented BMAC [49] and LMAC [50] as representatives of low power listening MAC protocols and TDMA based protocols. Frame and slot durations were chosen experimentally so that all traffic models are accommodated without MAC buffer overflow. In LMAC we reserved 5 node IDs for mobile nodes to avoid continuous slot changing.

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 BMAC	**slots**: 32 **slot length**: 60 ms **preamble length**: 12 bytes **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	**Carrier frequency**: 868 MHz **Signal attenuation threshold**: -110 dBm **Path loss coefficient alpha**: 3 **Max transmission power**: 1 mW **Maximum transmission range**: 300 m (calculated from above)
General	Rectangular field Network	**Size**: 2000 x 2000 m **Nodes**: 50 – 200 **Sources**: 1 – 5 **Sinks**: 1 – 5

Table 2: Summary of simulation environment model for our experiments.

7.1.1. Simulation settings

FROMS targets routing from one or few sources to multiple mobile sinks in large networks. To show its performance in these application scenarios, we defined the parameter settings of the simulation environment as shown in Table 2. We place the nodes in a square field of 2000 x 2000 m and set the radio transmission parameters to obtain a maximum transmission range of approximately 300 m, thus obtaining networks with 5 to 7 hops in diameter. Note that the real transmission range varies significantly, as the radio propagation model is probabilistic-based.

We vary the number of nodes from 50 to 200, thus covering medium to large networks. We vary the number of sources from 1 to 5 or approximately 2 – 10 % of the network nodes, which is reasonable for multicast scenarios.

We vary the number of sinks from 1 to 5. With these settings, we cover a large parameter space and the obtained results present sufficient information to be able to predict the behavior of the protocol on even larger scales.

In terms of data rate, we fix it to 1 packet every 10 seconds, which is reasonable for our low rate application scenarios. MAC protocol settings are chosen such that no congestion ever occurs. We do not vary the data rate, as the maximum supported throughout of the network depends more on the MAC protocol than the routing protocol. Nevertheless, we evaluate the processing time of FROMS in a testbed in Section 7.2. How data rate affects the mobility scenarios is discussed in Section 8.3.

7.1.2. Evaluation metrics in simulation

We selected five evaluation metrics to discuss the protocol behavior. These metrics are gathered over the full network lifetime, including sink announcements and learning, and are presented as the mean, normalized values over at least 50 connected topologies and 30 random runs on each topology.

- **First node death** is measured in seconds and represents the death of the first node in the network. This is a standard metric that provides a good measure of how long the network is guaranteed to perform, as after the first node death the network behavior may be unpredictable due to disconnection.

- **Routing overhead** is measured as the number of packet transmissions for all nodes in the network. Again, this is widely used to evaluate the length of the routing paths in the networks. Note that when using a MAC protocol without retransmissions the routing overhead metric is the same as the expected number of transmissions.

- **Total energy spent** for all nodes is measured in mW and gives a broader picture of the routing costs than the routing overhead as it also considers receiving and overhearing packets and node sleeping times.

- **Standard deviation of energy at first node death** is used to evaluate the ability of the routing protocols to spread the routing task among all nodes in the network. High deviation implies poor spreading with only few nodes carrying out the routing task, while low deviation implies good spreading among all nodes.

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 5: Characteristics of the MSB430 sensor nodes

- **Delivery rate** is used to evaluate the performance of the protocols in case of node failures and sink mobility. Higher delivery rates imply up-to-date routing information at the nodes and faster recovery (availability of backup routes).

We do not measure latency or throughput as these metrics are either unreliable under simulation (latency is usually underestimated under simulation) or are irrelevant for our application scenario (throughput).

7.2. Hardware testbed.

We implemented and tested FROMS on a real hardware testbed consisting of 10-15 MSB430 ScatterWeb [51] nodes. Their main characteristics are summarized in Figure 5. We used the OS-like ScatterWeb[2] library, which provides interfaces for sending/receiving messages, setting timers, reading sensory data, etc. We use the provided non-persistent idle CSMA MAC protocol without acknowledgments. Further implementation details are provided in [45].

7.2.1. Hardware testbed settings

Our evaluation requires meaningful topologies, where multiple, non-overlapping routes to individual sinks exit and their union is longer than the optimal shared route. While such a scenario can be easily found in real, large scale, sparse topologies, it is difficult to find in a testbed environment. For example, in typical testbeds such as Harvard's MoteLab [52], networks have only 3-4 hop diameters with sporadic 5 hop links. Even in the later case, individual routes often overlap.

A general solution is to use a link layer protocol to eliminate unreliable links and at the same time increase the diameter of the network. However,

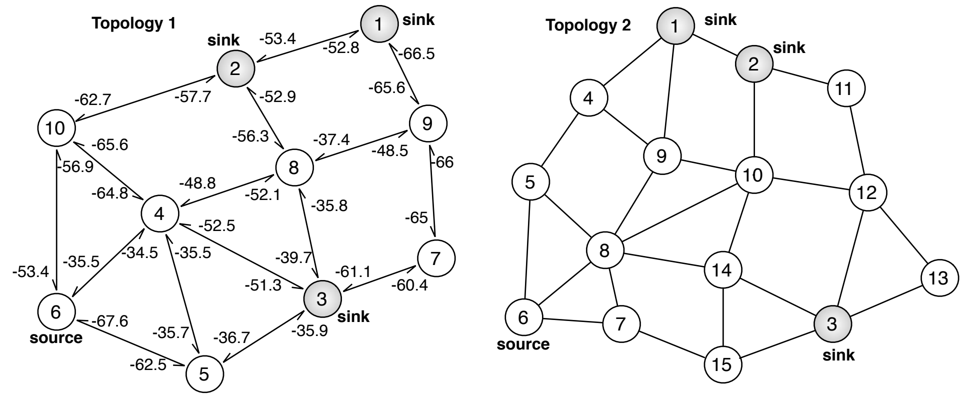


Figure 6: Testbed topologies 1 (left) and 2 (right). Sinks are shaded. Source is node 6. Mean RSSI values in dBm of links (in both directions) are provided for a sample run of topology 1.

as discussed in Sections 2.3 and 6.7, one of the goals of this work is to show that reinforcement learning based protocols such as Froms can, themselves, efficiently cope with WSN-specific challenges such as packet loss and unreliable or asymmetric links. To preserve this requirement also in the hardware testbed and at the same time achieve the required multihop topologies, we turned to another solution. In the hardware testbed, we explicitly ignore some of the available links. However, rather than eliminate only poor-quality links, we randomly select links to ignore, resulting in topologies with varying link qualities. To concretely illustrate this, the left of Figure 6 shows the mean RSSI values obtained from a sample run of topology 1. It can be seen that the quality varies significantly among links, and even some asymmetric links are created.

Although our network topology and its link properties can be considered typical for WSNs, there are scenarios in which transient links exist. While Froms can generally handle such links, the overall performance of the system will be increased with the usage of a link layer protocol or other means of recognizing these links. This is discussed in detail in Section 10.2.

7.2.2. Evaluation metrics in hardware testbed

On the hardware testbed we re-use some of the evaluation metrics of the simulation environment to allow for comparison between simulation and hardware results. We also add metrics that are unreliable in simulation, yet important for hardware experiments. Note that these metrics are gathered for the full network lifetime, including sink announcements and learning.

- **Routing cost** is measured as the mean number of hops per packet. We differentiate between *routing cost per generated packet* and *routing cost per received packet* to evaluate the protocol both in terms of the length of the selected routes and delivery rate. We also give the theoretical shortest route costs (the Steiner tree cost) as a point of comparison.
- **Delivery rate** is measured as the ratio between generated and received (at the sinks) packets.
- **Memory usage** is measured in bytes of RAM and ROM. Our data is obtained from the compilation process and includes all run-time data structures, as they are all statically allocated.
- **Processing time** is measured as the time in milliseconds needed to select a route to the sinks at the source node.

7.3. Comparative protocols.

To conduct a comparative analysis of FROMS, we have implemented three well known state-of-the-art routing protocols with similar goals and application scenarios:

- MSTEAM [13] is a geographic multicast routing protocol. We consider two versions. The original MSTEAM selects the next hop based on two components: geographic progress to the sinks and the cost of the link. The cost is calculated based on geographic distance between the two nodes. However, since transmission cost is often constant in a network (constant transmission power for all nodes), we implemented also a simplified version MSTEAM-CONST, where the cost of any link is considered 1 and the next hops are consequently selected only in terms of their progress to the sinks.
- UNICAST DIRECTED DIFFUSION (uDD) [14] is a well-known, simple and efficient routing paradigm in which each node 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, which was designed primarily to handle a single sink. We also consider MULTICAST DIRECTED DIFFUSION (or mDD), as explained next.
- MULTICAST DIRECTED DIFFUSION (mDD) is a multicast-optimized variation of uDD of our own design [53]. Instead of keeping only a single best gradient for each sink (as in uDD), mDD keeps all best

cost gradients and searches locally at the nodes for shared paths for multiple sinks.

We compare the performance of all protocols under simulation, however on the hardware testbed we have implemented only Froms and mDD. We decided against the original Directed Diffusion primarily because it is designed to support unicast. For Msteam, instead, the implementation is processing and memory intensive and did not fit on the selected hardware.

When evaluating the protocols, we used the same application layer implementation. Additionally, each protocol populates its routing data structure as sink announcements propagate, ensuring that all protocols are initialized the same way and that none use other a-priori routing information, as it is usual for geographic protocols.

7.4. Parameter settings for Froms

As discussed in Section 6, there are two important parameters of Froms that need to be defined a-priori and can significantly change the behavior of the protocol. Unlike traditional scalar parameters (such as link costs in Msteam, see above), the exploration strategy and the cost function of Froms allow the protocol to meet completely different optimization goals in varying application scenarios and to perform cross-layer optimization with other communication stack layers (see Section 10).

7.4.1. Cost function for mobile multicast

Recall from Section 2.1 that our core scenarios require multicast routing towards multiple, mobile sinks. We assume a retransmission-free MAC protocol and no neighborhood management protocol. Thus, the best cost function to represent shortest paths is hop count. Note that with a retransmission-free MAC protocol there is exactly one transmission per hop and therefore the number of hops is equal to the number of expected transmissions (ETX). Thus, we use the hop-based cost functions presented in Section 4.

7.4.2. Exploration strategy for mobile multicast

The exploration strategy must balance the potentially large cost to explore routes against the exploitation of the best ones. Many different exploration strategies are available in the reinforcement learning community, each suitable to different settings. Unlike cost functions, which can be intuitively selected to meet some optimization goals (see the previous section), exploration strategies require deeper evaluation, supported by experimentation. We use both sample runs and a large parameter space evaluation

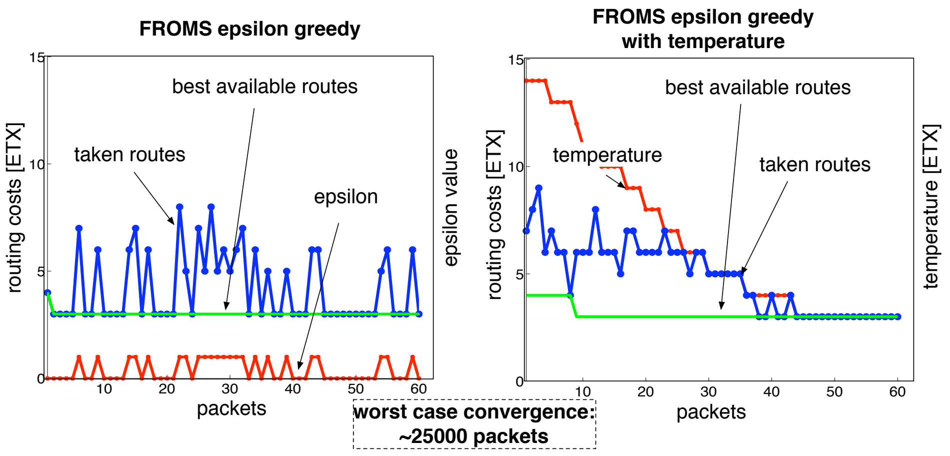


Figure 7: The route selection behavior at the source with different exploration strategies in a sample 50 node topology with 3 sinks and 1 source.

to compare several exploration strategies in the mobile multicast scenario. These evaluations help to select the best suited strategies for this specific scenario, and provide us with intuition for alternate application scenarios.

In our preliminary studies [15], we applied two different techniques for exploration: greedy and stochastic. 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 complicated to implement since they require updating the probabilities after each reward.

Here, we turn to a new set of exploration strategies, motivated by the need to make them more intuitive and efficient to implement.

ϵ - greedy. This strategy is the original Q-Learning strategy and is 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 $\epsilon = 0$ this becomes the greedy strategy from [15].

decreasing ϵ - greedy. This strategy is the same as the above, but in addition, ϵ decreases over time. This is motivated by the observation that typically, at the beginning of the execution, 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. This strategy corresponds to a widely used strategy in the RL community, where the learning constant decreases with time.

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 available, 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 the best of our previously evaluated strategies [15] and is used for comparison. 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.

Figure 7 presents sample runs for ϵ-greedy and ϵ-greedy with temperature to further explain their behavior. It can be seen for both strategies, that the best routes are found very quickly, specifically after only 2 packets with ϵ-greedy and 8 packets when temperature is included. This should be compared to the theoretical worst case described in Section 5, which is around 25,000 packets for the examined network. This large difference is due to the initialization of the Q-Values in our protocol, which estimates the initial Q-Values with the individual costs to the sinks. This speeds up learning significantly as the real route costs are very close to the initial estimates.

Figure 8 shows an evaluation of all strategies in a simulation environment, normalized by *decreasing ϵ-greedy*. The deviation of the first node death time is insignificant, never exceeding 1%. First we observe that the differences between the exploration strategies are small: only a few percentage points for network lifetime (shown as time to first node death in the left plots) and 10% for routing overhead (shown in the right plots). From this, we conclude that a simple to implement and tune exploration strategy such as *ϵ-greedy* or *decreasing ϵ-greedy* is a reasonable choice. This choice is also supported by the absolute performance of these two strategies as in most cases they show the lowest routing costs and the longest network lifetimes. This is particularly clear in the cases of increasing network sizes and number of sources in the network. Only as the number of sinks increases does

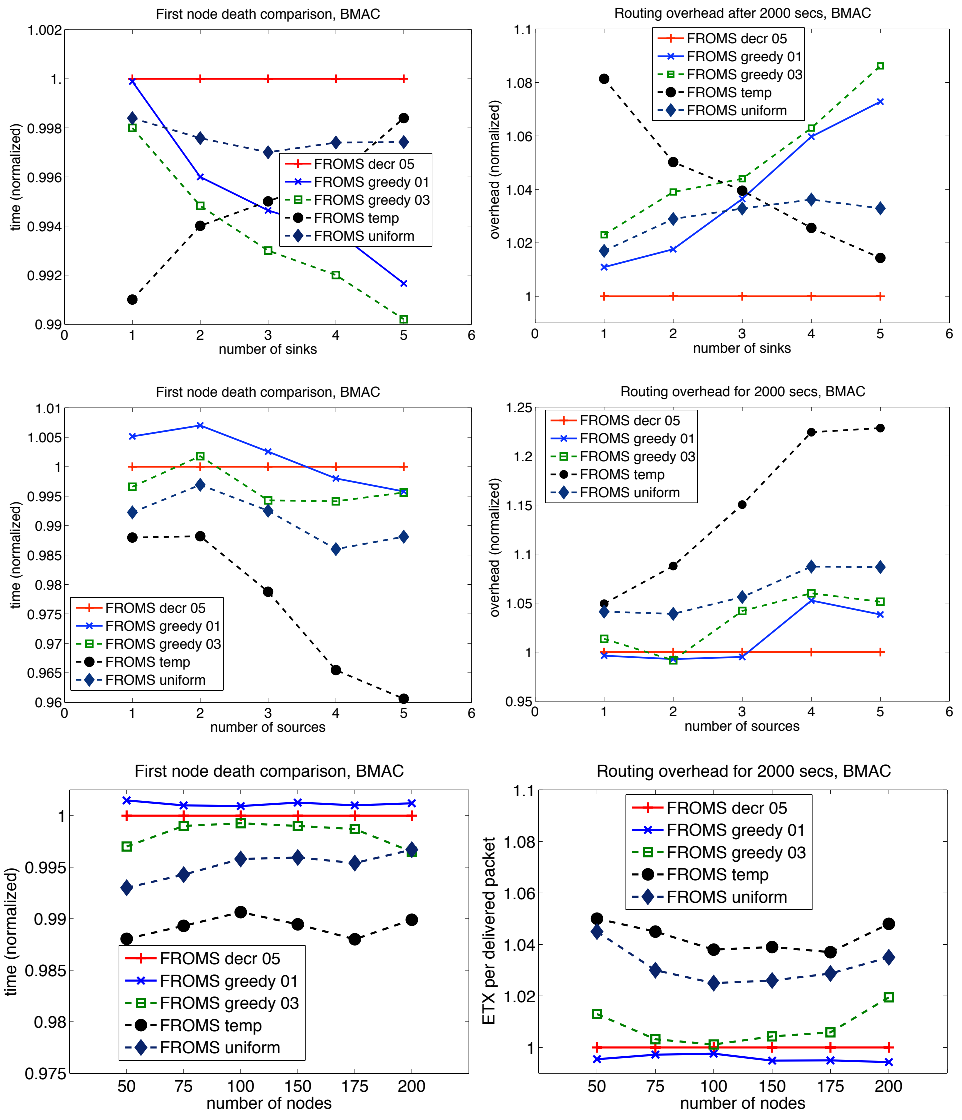


Figure 8: Evaluation of exploration strategies. The network consists of (top) 50 nodes, 1 source and 1-5 sinks; (middle) 50 nodes, 1-5 sources and 2 sinks; (bottom) 50-200 nodes, 1 source and 2 sinks.

ϵ-*greedy* seem to perform not as well, nevertheless, it remains similar to the other more sophisticated strategies.

Our experience in carrying out these simulations and the results support our hypothesis that reinforcement learning and machine learning in general can be easily and efficiently implemented on memory and processing restricted sensor nodes. In the rest of the our evaluations we use only ϵ-*greedy* and *decreasing* ϵ-*greedy*.

8. Evaluation of FROMS via simulation

Next we turn to the performance comparison of FROMS against three state-of-the-art routing protocols: multicast Directed Diffusion (mDD), unicast Directed Diffusion (uDD) and MSTEAM, whose details are provided in Section 7.3. We fist show scalability in terms of the number of sinks, sources and network nodes. Then we turn to specialized scenarios such as node failure and mobile sinks.

8.1. Scalability analysis

First we evaluate the scalability of FROMS in terms of the number of sinks, sources and network nodes. Results in Figure 9 are normalized by the first point of FROMS *decreasing* ϵ-*greedy* (specifically for one sink, one source, and 50 nodes). Such normalization allows us to simultaneously offer scalability and comparative analysis. Results for FROMS ϵ-*greedy* are omitted since they overlap with those from Figure 8.

Overall, our simulations show that FROMS spends the least amount of energy across all networks for all parameter settings. These benefits arise from two factors: first, its ability to find optimal multicast routes and second, its limited use of broadcast sink announcements.

In comparison, the high energy consumption by MSTEAM arises from face-routing to avoid void areas, which tends to create long routes. As all packets follow the same face-routed path, these long routes incur excessive overhead, while instead the reinforcement learning techniques in FROMS avoid cases where the packet would be sent to the void nodes and back. Notably, MSTEAM-CONST which assumes a constant cost for communication between any two nodes performs better than the original MSTEAM. This is because the original MSTEAM uses a special cost function that increases the cost when sending a packet longer distances, forcing the protocol to take many, short hops. Instead, in deployments where the radio transmission power is often fixed, longer hops lead to overall energy savings, even when considering retransmissions.

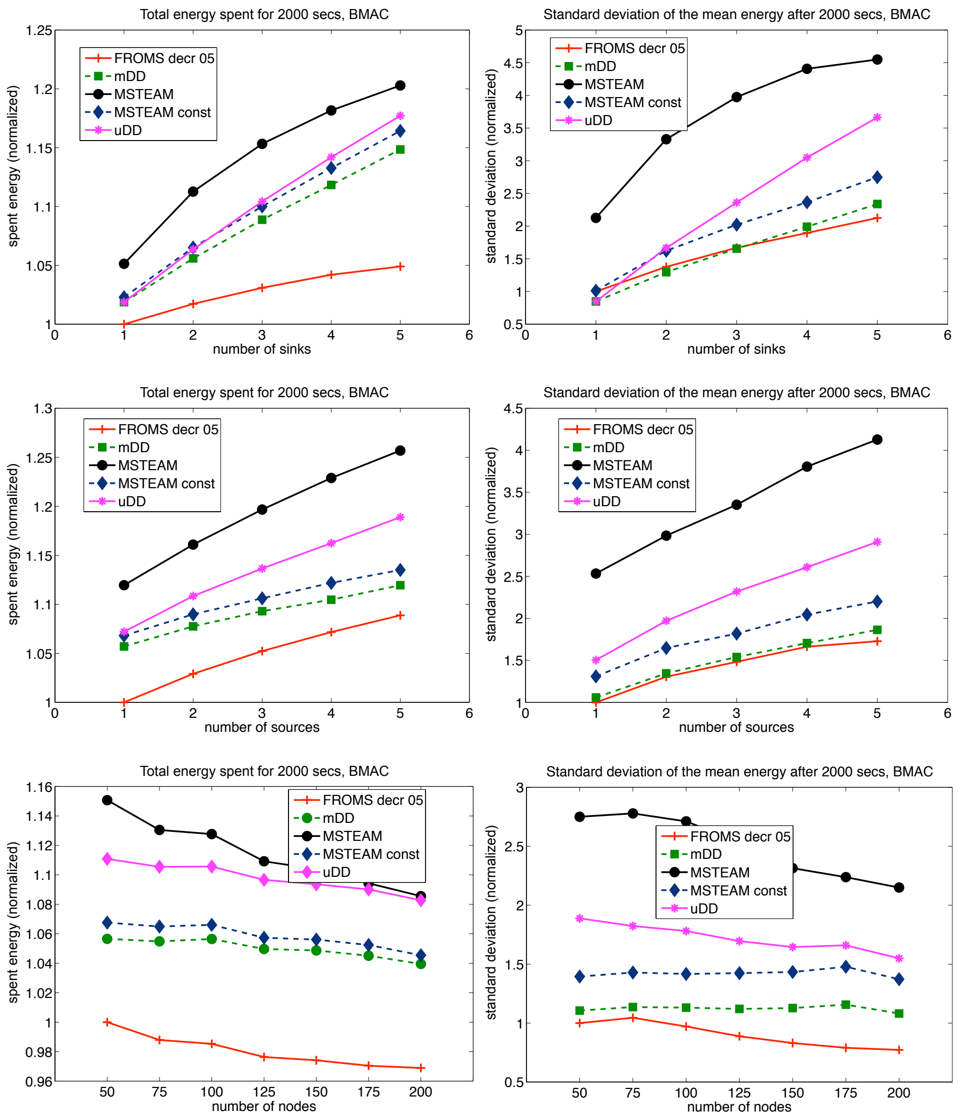


Figure 9: Evaluation of total spent energy and standard deviation of remaining energy after the first 2000 seconds. The network consists of (top) 50 nodes, 1 source and 1-5 sinks; (middle) 50 nodes, 1-5 sources and 3 sinks; (bottom) 50-200 nodes, 1 source and 2 sinks.

In terms of the overall scalability, we note that all protocols scale well with all network parameters. This is due to their localized nature, which makes them independent with respect to the size of the network or the number of sinks and sources.

In summary, FROMS achieves between 10 and 22% longer network lifetimes in terms of first node death, incurs around two times less routing overhead, spends between 5 and 15% less energy, and shows 2 to 3 times lower standard deviation among the remaining node energies. Plots for lifetime and routing support the observations made here but are not shown due to space reasons. We also compared the performance with LMAC and BMAC, achieving similar results.

8.2. Recovery after failure

A challenging situation arises when one or more nodes fail. Routes from sources to the sinks need to be repaired as quickly as possible to prevent data loss. To study the behavior of the analyzed routing protocols in this situation we conducted simulations in which we assign lower initial battery levels to a random subset of nodes, causing them to fail prematurely. 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.

Figure 10 shows the total energy spent and the delivery rate for all protocols, with each point averaging 50 topologies with 30 different random sets of failed nodes. We note that we ignore the results where the failures resulted in disconnected topologies, as in such scenarios the data cannot be delivered. The standard deviation is 2-3.3%.

For both metrics, FROMS achieves the best performance. This is due to its ability to recover quickly after node failures. As explained in Section 6.6, it tracks which neighboring nodes are responding. In case some neighbor is no longer reachable, FROMS switches to the next best route. The new costs are propagated as feedback through the network and the new, best routes are learned at all affected nodes. This flexibility and innate ability to handle topology changes are two of the main advantages of the FROMS learning mechanism.

In terms of energy expenditure, FROMS ϵ-greedy performs best because of its continuous exploration. Instead of exploring only on demand, ϵ-greedy tracks all possible routes and proactively updates their costs. 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 non-optimal routes is avoided, delivery rate is increased because

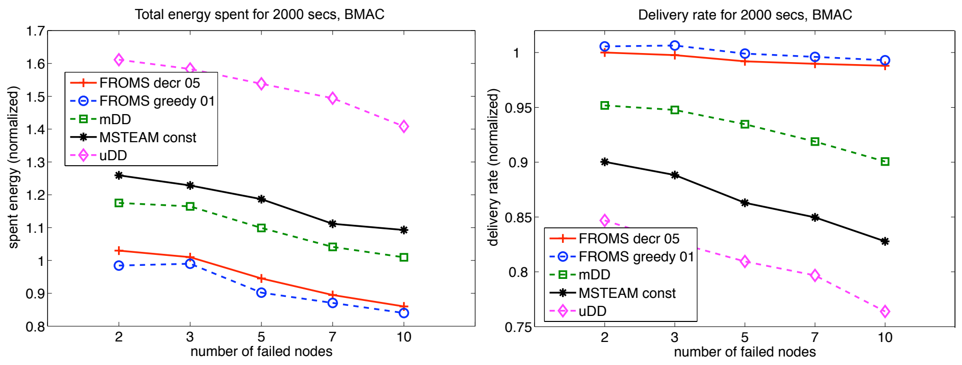


Figure 10: Comparison of delivery rate and spent energy for different routing protocols with varying number of failed nodes in the network. The network consists of 50 nodes, 1 source and 3 sinks

of shorter routes (Figure 10 right), and spent energy is minimized (Figure 10 left).

Similarly, mDD also monitors the neighborhood and maintains alternative routes. Its delivery rate is 2-5% less than that of Froms due to the learning behavior of Froms. On the other side, Msteam-const uses longer routes that incur more packet loss. Additionally, the neighborhood failure detection is less efficient as Msteam uses the same route over and over. Thus, in the case of failures of some nodes on a route, the route 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 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), are good strategies to quickly recover after failures.

8.3. Sink mobility

Another challenging situation arises with mobile sinks whose mobility models are unknown, e.g., rescue workers in a disaster recovery scenario. The routing protocol needs to detect this situation, to differentiate it from node failures and to maintain the routes to ensure continuous data delivery.

To evaluate the behavior of the routing protocols in such a scenario, we designed two different experiments: one with different numbers of mo-

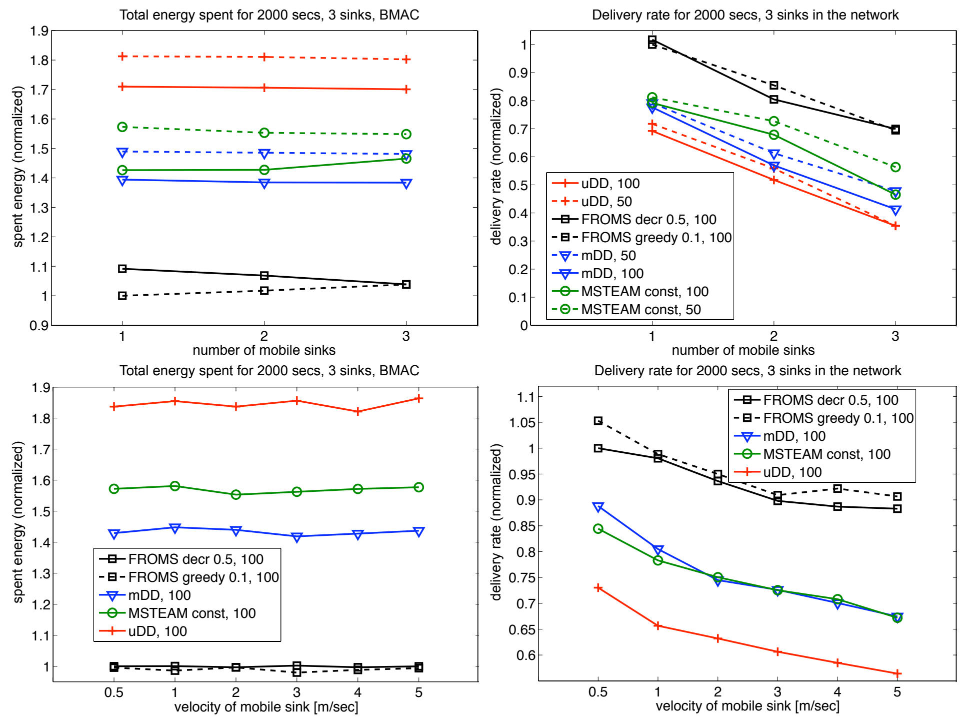


Figure 11: Evaluation of all routing protocols with various number of mobile sinks with constant velocity of $1m/sec$ (top) and with various velocities of the mobile sink (bottom). The network consists of 50 nodes, 1 source and 3 sinks (of which 1 to 3 are mobile).

bile sinks, and a second with different sink velocities. In both cases, the sinks move according to the random waypoint model. The experiments were conducted over 50 random topologies, with 10 random runs on each and achieved a standard deviation of $1.6 - 1.9\%$.

The results from the first experiment with a single source are presented in Figure 11 (top). The total number of sinks is fixed at 3, and we vary the number of mobile sinks from 1 to 3, leaving the rest static. Sink velocity is constant at $1m/s$. The frequency of sink announcements is one of the factors influencing the ability of mDD, uDD, and Msteam-const to maintain correct, short routes to mobile sinks. Therefore, we experimented with two different frequencies, sending a sink announcement every 50 or 100 seconds. With the higher frequency, the delivery rate increased as expected (top right), but correspondingly the energy consumed also increased (top left). However, energy expenditure increases non-proportionally to the achieved gain in delivery rate, thus it is not worthwhile.

In terms of energy expenditure (Figure 11 top left), all protocols scale well with an 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 (top right). This is because the mobility affects the quality of the selected links and some links disappear.

Among the selected protocols, FROMS has the lowest energy expenditure and still achieves the best delivery rates. This is 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 better, while uDD spends the most energy and achieves the lowest delivery rate.

In our second experiment presented in Figure 11 (bottom) we vary the velocity of a single mobile sink from $0.5m/s$ to $5m/s$, corresponding to slow human walking ($2km/h$) and slow vehicle movement ($20km/h$). In terms of energy expenditure (Figure 11 bottom), the behavior of the routing protocols is the same as in the previous experiment. FROMS has a significantly lower energy expenditure, followed by mDD, MSTEAM-CONST, and finally uDD. The reasons are the same as before.

The delivery rate trend in the case of higher velocities is also as expected, dropping with higher velocities. This is due to the fact, that nodes route data to the sink when it is already away from their transmission radius. FROMS performs slightly better due to its learning mechanism, which not only substitutes the sink announcement re-broadcasts, but enables faster recovery of routes. Notably, with very high velocity of the sinks the data delivery will break completely and other, proactive mechanism from the side of the sink will be needed for all protocols.

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

9. Evaluation of FROMS on the hardware testbed

To demonstrate concretely the feasibility of applying reinforcement learning on real WSN nodes with concrete restrictions on processing, memory, and communication, we turn to testbed evaluation of FROMS. The results

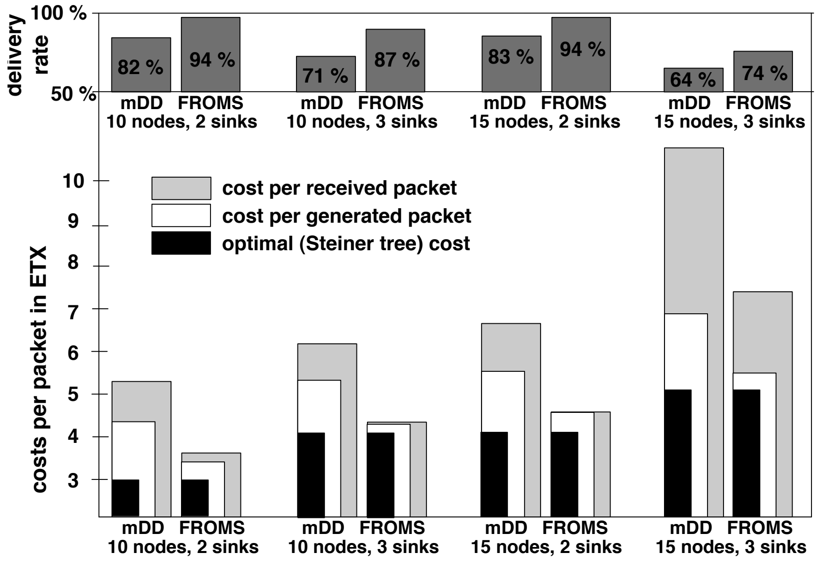


Figure 12: Routing costs and delivery rates for FROMS and mDD in various network scenarios.

presented in this section conform to our simulation study and thus we do not explore scalability issues here in terms of very large topologies or high number of sources. Instead, we concentrate on implementation and behavior. The tested topologies have only one source node, as increasing the number of sources will not affect the implementation complexity and will not add significant value to the main objectives of this study.

We have implemented FROMS with the ϵ-greedy exploration strategy and our multicast version of Directed Diffusion [53]. We chose not to implement uDD because it is fundamentally a unicast protocol. Further, our initial implementation of MSTEAM failed due to its complexity (it simply did not fit on our hardware platform).

9.1. Comparative analysis

First, we compare the routing performance of FROMS and mDD on real hardware in terms of delivery rate and routing costs, summarizing the results in comparison also to an optimal Steiner tree in Figure 12. As expected based on our simulation experiments and theoretical analysis, FROMS incurs lower routing costs. This is attributed to the learning algorithm that actively explores the network for optimal routes.

In simulation we are unable to accurately measure delivery rates since transmission failures cannot be reliably simulated. Here, instead, we confirm

	ROM (bytes)	RAM (bytes)
FROMS + Scatterweb^2	33610	3345
mDD + Scatterweb^2	29648	2952
Scatterweb^2	19628	1476

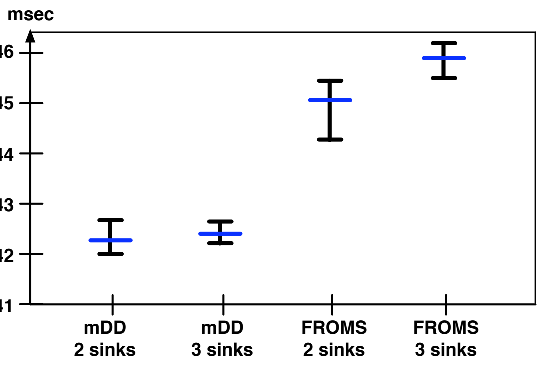


Figure 13: (left) Memory usage at compile time. (right) Processing time (milliseconds) and min/max intervals to find a route in mDD and Froms.

our theoretical expectation that Froms is able to achieve a higher delivery rate in any network scenario. Data is lost in mDD mainly due to the higher network communication caused by the periodic sink announcements and the longer routes to the sinks. These increase the traffic and collision probability, leading to packet losses. Figure 12 supports these observations, showing that the delivery rate of both protocols clearly drops in networks with larger numbers of nodes and sinks.

9.2. Memory and processing requirements

Two of the main difficulties when implementing RL or in general machine learning based algorithms on sensor networks are algorithm efficiency and complexity. To assess this complexity, we measured the memory requirements and needed processing time for both protocols, as shown in Figure 13. The left shows the memory needs in terms of ROM and RAM memory. The memory requirements of Scatterweb^2, the operating system of the sensor nodes, and of the implementation of acknowledgments are given for comparison. The right shows the measurements of the processing time to find a route at the data source for 2 and 3 sinks respectively.

It is interesting to observe that the memory requirements for Froms are not significantly larger than for mDD. Recall that both implementations of Froms and mDD use static data structures because there is no dynamic memory management implementation for the MSB430 platform. No route storage heuristics are used for Froms, and all possible routes are kept at all times. Therefore the data structures are already included in the memory footprints of Figure 13. Although Froms's data structures are more complex and larger than the routing table of mDD, its memory requirements are not significantly larger. mDD has a small data structure, but the space required for the implementation is non-negligible. When analyzing this difference we realized that the complexity of the protocols comes not from the routing data structures, but from trivial functionality such as constructing and analyzing packets, constructing routes, passing packets to the application and to the MAC layer, etc.

Additionally, we wanted to know how long it takes for both protocols to find a route for a packet. We suspected that Froms would need more time because of the higher complexity of its data structure and the need to iterate through all possible routes. However, as Figure 13 right shows, this difference is less than 10% for 2 and for 3 sinks.

These results are an important demonstration 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 such as mDD.

10. Adjustments and Further Applications of FROMS

As pointed out in previous sections, Froms is more than a single, specific routing protocol – it is routing paradigm. It can be applied to a multitude of application scenarios, reach various optimization goals and integrate with different communication stacks. In this section we discuss some of these options and outline which changes or adjustments are required for Froms to be successfully applied.

10.1. Application requirements

Froms has been designed in a modular way with a variety of components that can be tuned, modified or replaced to meet application needs. The cost function can be defined to incorporate a critical application resource, such as remaining battery power. Similarly, the routing overhead can be tuned by changing or tuning the exploration strategy. This section discusses how to select appropriate cost functions and exploration strategies to match application requirements, first addressing changes in the application scenario such as the number of sources or sinks.

10.1.1. Application scenarios

While the version of Froms presented here targets multicast applications, other scenarios are also possible. Unicast can be seen as a special case of multicast with a single sink and is thus trivially supported. Notably, the implementation will be memory and processing efficient, as the stored routing information regards only a single sink. Some cost functions (e.g. hops) will perform similarly to non-RL-based protocols, such as uDD with the additional advantage that such a unicast-FROMS will handle mobility and failure recovery.

Convergecast is also a special case of the multiple source scenario evaluated in the previous sections, and requires no changes to FROMS. Note, however, that a convergecast towards multiple sinks can be solved in a more efficient way than routing to all sinks simultaneously. In this situation, it is nearly always more efficient to route data to a single, master sink, which then disseminates the data to all other sinks.

10.1.2. Cost functions

Recall from Figure 4 that the cost function is exploited by lines 2, 11 and 13 for respectively setting route costs, identifying viable routes, and choosing a single route.

Table 3 offers alternate cost metrics, together with the optimization goals they meet and their properties. Some functions are *static* while others are *dynamic*, referring to whether or not, in a static network, the function varies over time. For example, geographic distance between nodes does not change in a static network, thus this function is identified as non-dynamic. Latency, however, can change over time.

The choice of a cost metric is very important for the final behavior of FROMS. While throughout this paper we have concentrated on hops for comparative and readability reasons, some of the cost functions in Table 3 might be better suited for some real world deployments. Fro example, the existence of bursty, long range sporadic links, which temporarily provide shorter paths in the network, are of particular interest. On one hand, these shorter paths can be successfully exploited for minimizing the While most of the cost functions in Table 3 involve only a single metric, some combine several metrics. Here we concentrate on one example that incorporates both remaining battery levels and hop counts to find the shortest routes with high remaining energy levels on the forwarding nodes. This example demonstrates how a complex cost metric can be naturally used with FROMS.

Fundamentally, the cost function is used to calculate Q-Values, and in this case, we define a function, f, to combine hop count and battery information into a single Q-value:

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

Cost function	**Calculation of initial values**	**Optimization goal**	**Dynamic**	**Best Q-Values**
simple functions				
Hops	$\sum hops$	shortest shared path (Steiner tree)	no	lowest
ETX	$\sum ETX$	lowest number of expected transmissions (reliable shortest paths)	yes	lowest
Latency	$\sum latency$	least latency path	yes	lowest
Transmission energy	$\sum energies$	least energy path	no	lowest
Geographic distance	$\sum dist$	shortest shared path	no	lowest
Aggr. rate	$\sum rates$	maximum compression path	no	highest
...	...	...	...	...
combined functions				
Hops & rem. battery of nodes	$\sum hops \cdot hcm(bat_{hops})$	shortest shared path through nodes with high battery	yes	lowest
...	...	...	...	...

Table 3: Different possible cost metrics for FROMS and their main properties.

where E_{hops} is the estimated hop cost of the route, calculated as in our previously defined Equations 1 and 2, and $E_{battery}$ is the estimated battery cost of this route, which we define as the minimum remaining battery among all nodes along the route.

The function f based on a simple function:

$$f(E_{hops}, E_{battery}) = hcm(E_{battery}) \cdot E_{hops} \tag{11}$$

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

Figure 14 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 14 shows two linear functions. The first (labeled linear), has minimal effect on the routing behavior. For example, a

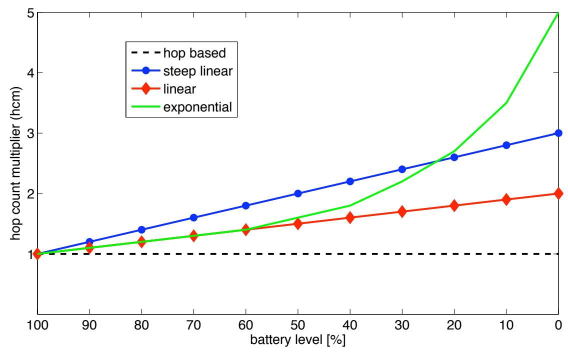


Figure 14: Hop count multiplier (hcm) functions achieving different optimization goals.

greedy protocol that 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.

We further explored the behavior of our battery-hop function experimentally to confirm that it meets its optimization goal. The results are presented in Figure 15. In the left graph, by comparing the same exploration strategy with and without the battery component, it can be seen that the inclusion of battery information reduces the mean energy level by up to 10% at first node death, meeting the intended goal. This comes at the price of slightly

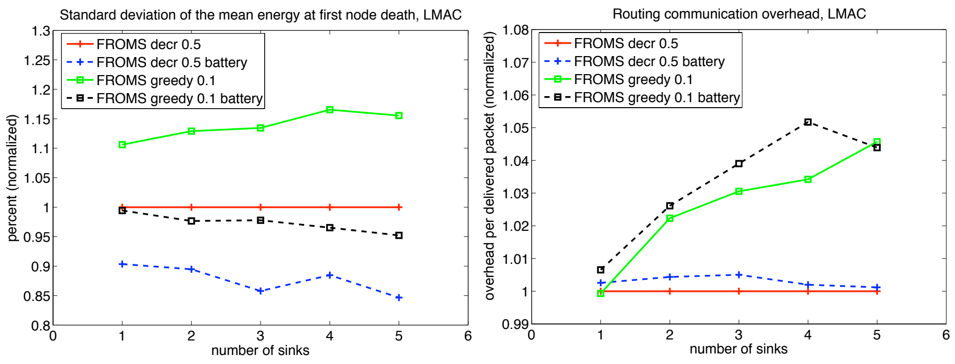


Figure 15: Comparison of two cost functions: *hop-based cost* and *hop-battery based cost*. The network consists of 50 nodes, 1 source and 3 sinks. All experiments use LMAC (BMAC experiments yielded similar results)

longer routes, as can be seen in the right plot. It is worth noting that we selected the *hcm* function to switch to longer routes only if the difference in the remaining batteries is large. Depending on the given optimization goal a more aggressive *hcm* can be chosen, further spreading the spent energy among the nodes and likely increasing more significantly the routing overhead.

10.1.3. Exploration strategies

Recall from Section 6.8 that the task of the exploration strategy is to choose between the available routes. It also controls the exploration/ exploitation ratio, which is responsible for both finding the optimal route and minimizing routing costs. Section 7.4.2 presented an evaluation of four different exploration strategies and provided intuition on their properties. The main insight from this evaluation was that the exploration rate of ϵ-greedy is the most important parameter to tune. Generally speaking, if the application scenario is very dynamic and the network topology changes often (e.g., in case of mobility), ϵ must be high in order to react more quickly. In an opposite scenario, when the network is static most of the time and only node failures generate dynamicity, a low ϵ is appropriate. If the dynamic nature of the network is expected to be sporadic, ϵ-greedy with temperature or decreasing is the best option.

The selected exploration strategy also depends on the chosen cost function. Dynamic cost functions imply dynamic networks and the above suggestions need to be considered.

10.2. Cross-layer optimization

To make the FROMS design as general as possible, we place very loose requirements on the remainder of the communication stack, such as MAC protocols or link layer management (see Section 2.3). However, sophisticated protocols at these layers would increase the performance and reliability of the complete system, for example in the presence of transient links in the network. Here we discuss the MAC and link layer.

Medium Access Protocols (MAC). The MAC protocol controls access to the medium and generally decides when to send/receive messages or to put the radio to sleep. Various protocols have been developed for WSNs [19]. FROMS is independent of the implementation details of the MAC protocol and generally can be used with any of them. However, there is space for cross-layer optimizations at both sides. Relevant features are automatic re-transmissions, acknowledgments and schedules.

If re-transmissions are implemented at the MAC layer the FROMS cost function should be adapted. For example, instead of counting the number of hops to the sinks, ETX becomes a more meaningful metric. On the other side, a re-transmission based MAC protocol can be optimized in terms of its acknowledgment mechanism. FROMS always sends rewards to its neighbors, thus, rewards can be used by the MAC protocol instead of an additional specialized ACK packet.

TDMA based MAC protocols employ schedules to send/receive messages, e.g., LMAC. In this scenario, latency becomes a meaningful metric, since a route may be short in terms of hops or ETX, but the latency may be large due to scheduling at the MAC layer.

Unicast-focused MAC protocols pose challenges to FROMS, as it exploits broadcast to allow neighbors to overhear at least the header of every packet. One option is to use the broadcast modes provided by these protocols, however this will likely incur a higher energy cost. Another possibility is to send FROMS rewards only to the previous hop (unicast), thus inhibiting learning at overhearing nodes. The tradeoff between fast learning and energy savings with unicast messages must be carefully evaluated.

Neighborhood (link) management. Section 2.3 assumed no link layer protocol is used in combination with FROMS. This allowed us to present a clean design and evaluation for FROMS without cross-layer issues. However, such a protocol will further optimize the performance of the system by providing link quality information. FROMS can easily accommodate such a protocol by using all relevant neighbor data provided by the link management.

For example, a link layer protocol can efficiently recognize and handle transient links in the network. Transient links pose difficult challenges to any routing protocol, as they confuse the routing metric with the presence of short-lived, highly efficient or short routes. Many routing protocols immediately switch to these routes, eliminating all others. When the transient links disappear, such protocols fail quickly. FROMS is inherently better suited to handle transient links, as it keeps secondary routes and quickly accommodates both new and lost links. Nevertheless, even the performance of FROMS will be improved if extremely transient links are recognized and eliminated at a lower level. This can be done by using various metrics at the link layer level, such as RSSI, LQI, packet delivery rate, etc.

On the other side, link layer protocols typically exploit beacons to maintain the freshness of the link information, e.g., the adaptive beaconing of CTP offers a tradeoff between freshness and additional cost. We can consider further optimizing adaptive beaconing by utilizing the rewards of FROMS and minimize the number of injected beacons.

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 observed that the properties of the neighboring nodes are equally important: e.g., a node that is a neighbor of two independent routes drains its battery twice as quickly as the forwarding nodes due to 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.

In this paper we were able to evaluate the performance of FROMS only in one application scenario, namely mobile multicast. While we discuss how to use FROMS in other environments, the implementation and evaluation work remains to be done. We plan, for example, to adapt FROMS to the convergecast static scenario and compare it against state of the art routing protocols such as CTP.

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 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. Among the possibilities, we intend to 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. 5.

[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. 8.

[5] 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.

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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. 7.

[17] 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.

[18] 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.

[19] 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. 22.

[20] 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.

[21] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, M. Welsh, Fidelity and yield in a volcano monitoring sensor network, in: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, USENIX Association, Berkeley, CA, USA, 2006, pp. 27–27.

[22] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, P. Levis, Collection tree protocol, in: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys), ACM, Berkeley, California, 2009, pp. 1–14.

[23] 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.

[24] R. Vaishampayan, J. J. Garcia-Luna-Aceves, Efficient and robust multicast routing in mobile ad hoc networks, in: Proceedings of the IEEE In-

ternational Conference on Mobile Ad-hoc and Sensor Systems (MASS), Fort Lauderdale, FL, USA, 2004, pp. 304–313.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] P. Ciciriello, L. Mottola, G. Picco, Efficient routing from multiple sources to multiple sinks in wireless sensor networks, in: Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN), Delft, The Netherlands, 2007, pp. 34–50.

[30] 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.

[31] B. Kusy, H. Lee, M. Wicke, N. Milosavljevic, L. Guibas, Predictive qos routing to mobile sinks in wireless sensor networks, in: Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN), San Francisco, CA, USA, 2009, pp. 109–120.

[32] 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.

[33] 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.

[34] 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.

[35] R. Flury, R. Wattenhofer, Routing, anycast, and multicast for mesh and sensor networks, in: Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM), 2007, pp. 946–954.

[36] S. Kulkarni, A. Förster, G. Venayagamoorthy, A survey on applications of computational intelligence for wireless sensor networks, IEEE Communications Surveys & Tutorials 13 (1) (2011) 25.

[37] 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.

[38] 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.

[39] 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. 12.

[40] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 1998.

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

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

[43] L. Sang, A. Arora, H. Zhang, On link asymmetry and one-way estimation in wireless sensor networks, ACM Transactions on Sensor Networks 6 (2) (2010) 1–25.

[44] A. Egorova-Förster, A. L. Murphy, A feedback enhanced learning approach for routing in WSN, in: Proceedings of the 4th Workshop on

Mobile Ad-Hoc Networks (WMAN), Springer-Verlag, Bern, Switzerland, 2007, p. 12.

[45] A. Förster, A. L. Murphy, J. Schiller, K. Terfloth, An Efficient Implementation of Reinforcement Learning Based Routing on Real WSN Hardware, in: Proceedings of the 4th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMOB), Avignon, France, 2008, p. 8.

[46] A. Förster, A. L. Murphy, FROMS: A Failure Tolerant and Mobility Enabled Multicast Routing Paradigm with Reinforcement Learning for WSNs, Tech. Rep. TR 2009/04, University of Lugano (June 2009).

[47] D. Puccinelli, M. Haenggi, Reliable data delivery in large-scale low-power sensor networks, ACM Transactions on Sensor Networks 6 (4) (2010) 41.

[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. 7.

[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] Scatterweb Inc., http://www.scatterweb.de/.

[52] G. Werner-Allen, P. Swieskowski, M. Welsh, Motelab: a wireless sensor network testbed, in: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN), 2005, pp. 483–488.

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