FROMS: Feedback Routing for Optimizing Multiple Sinks in WSN with Reinforcement Learning

Anna Egorova-Förster[†] and Amy L. Murphy[†‡]
[†]University of Lugano, Switzerland, anna.egorova.foerster@lu.unisi.ch
[‡]FBK-IRST, Italy, murphy@itc.it

Abstract

In the domain of wireless sensor networks (WSNs), information routing is both a fundamental and challenging problem. In this work, we describe how information local to each node can be shared without overhead as feedback to neighboring nodes, enabling efficient routing to multiple *sinks. Such a situation arises in WSNs with multiple, possibly mobile users collecting data from a monitored area. We formulate the problem as a reinforcement learning task, and apply Q-Routing techniques to derive a solution. Evaluation of the resulting* FROMS *protocol demonstrates its ability to significantly decrease the network overhead over existing approaches.*[1]

1. Introduction

A challenging application in the wireless sensor network (WSN) domain is to support multiple mobile users, each requiring information from the environment to make future movement decisions. This problem differs from the majority of WSN applications, as the data is not centrally collected, but rather used at multiple, internal points in the network. The simple solution to keep distinct point to point paths for all source-destination pairs unnecessarily wastes energy, and instead, a favorable solution shares as many links as possible as the data flows from each source to all destinations. In this work, we provide a novel routing protocol that builds and maintains this distribution tree in the presence of mobility and failures.

This routing path is similar to a Steiner Tree whose cost is calculated considering broadcast communication. As solving the Steiner Tree offline is not appropriate, our goal is to define a low-overhead, distributed approach. To cope with the system dynamics, our solution exploits *reinforcement learning*, incrementally learning at each node sufficient network knowledge to identify the next, best hop. We exploit the broadcast nature of communication to share local information among neighbors. Previous work has applied reinforcement learning to routing [1], [2] and outlined solutions for multicast communication MANETs [7], [8]. In comparison, our work applies learning to multicast through a mechanism efficient enough for use in WSNs.

The work described here represents a significant extension over our earlier introduction of the idea of applying reinforcement learning to multiple sink routing [3]. This paper updates both the feedback mechanism and the network cost function, and, most significantly, formally frames the problem through a reinforcement learning model (Section 2). We also outline new extensions for mobility and failure recovery (Section 5), and include an evaluation of the full parameter space (Section 6). A comparison to related work is presented in Section 7.

2. Multiple Sink Routing as a RL problem

We begin by defining multiple sink routing, then model it as a reinforcement learning problem solvable with Q-learning.

A. 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. We consider a single source node $s \in V$ and a set of destination nodes $D \subset V$.

Routing to multiple destinations is defined as the minimum cost path starting at the source vertex s, and reaching all destination vertices D. This path is actually a spanning tree $T = (V_T, E_T)$ whose vertexes include the source and destinations. The cost of a tree T is defined as the number of one-hop broadcasts required to reach all destinations.

B. Q-learning approach

Finding the minimum cost tree T is NP-hard, even when the full topology is known. Our goal, therefore, is to approximate the optimal solution using localized techniques. For this, we turn to reinforcement learning [4], as it has been successfully applied to various routing problems [1], [2]. Here we extend the model to accommodate the multiple sink scenario and use Q-learning to solve it sub-optimally inside the network.

Q-learning [4] is a model-free reinforcement learning technique, based on receiving scalar rewards from the environment. It assigns *Q-values* to each possible agent action, representing the approximate *goodness* of the action. In the learning process, the agent selects and executes one action and receives a reward representing the goodness of that action. This reward is used to update the Q-value. Over time the agent learns the real action values and is able to select the most appropriate.

In our multiple-sink scenario, each sensor node is an independent learning agent, and actions are routing options using

[1]The work described in this paper is 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.

different neighbor(s) for the next hop(s) toward a subset of the sinks, $D_p \subseteq D$, listed in the data packet. The following provides additional detail for the Q-learning solution.

Agent states: For multiple sink routing, we define the state of an agent as a tuple $\{D_p, hops_{D_p}^N\}$, where D_p are the sinks the packet must reach and $hops_{D_p}^N$ is the routing information through all neighboring nodes N 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. Specifically, we 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 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). For example, consider a packet destined for $D_p = \{A, B, C\}$. One possible complete action is the single sub-action $(N_1, \{A, B, C\})$, indicating neighbor N_1 as the next hop to all destinations. Alternately, a node may choose two sub-actions, $(N_1, \{A, B\})$ and $(N_2, \{C\})$, 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. In our case, Q-values represent an estimation of the cost of the route, specifically the broadcast hop count to reach all sinks from the agent. To initialize these values, we could use random values, as is common in many learning approaches. However, we use a more sophisticated approach that calculates an estimate of the hop count cost based on the individual hop counts available in a standard routing table, such as that in Figure 1, thus speeding up the learning process. Note that the table information in Figure 1 is different from our target as it contains routes to *single* sinks, not multiple sinks simultaneously.

We first calculate the value of a sub-action, then of a complete action. Using the simple routing information, the initial Q-value for a sub-action $a_i = (n_i, D_i)$ is:

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

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

The Q-value of a complete action a with sub-actions $\{a_1 \dots a_k\}$ is:

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

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

Clearly, other cost metrics are possible such as including a node's battery level or, for power-aware routing, using a node's transmission power. We intend to explore these in future work.

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

$$Q_{new}(a_i) = Q_{old}(a_i) + \alpha(R(a_i) - Q_{old}(a_i))$$

where $R(a_i)$ is the reward value and α is the learning rate of the algorithm. We use $\alpha = 1$ to speed up learning. Also, our direct interpretation of the Q-values as real costs (broadcast hop-count) forces us to use 1 to keep the Q-values meaningful throughout learning. Therefore, with $\alpha = 1$, the formula becomes $q_{new}(a_i) = R(a_i)$, directly updating the Q-value with the reward. The Q-values of complete actions are updated automatically, since their calculation is based on sub-actions.

Reward function: Intuitively the reward function 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 Q-value* for the destination set and adds the cost of the action itself:

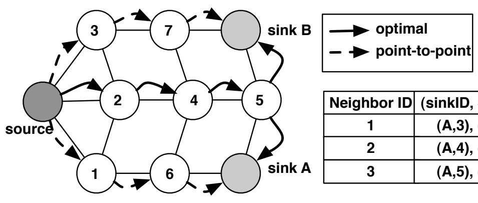
$$R(a_i) = c_a + MIN_a Q(a)$$

where c_a 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.

Exploration strategy (action selection policy): One final, important learning parameter is the action selection policy. A trivial solution is to greedily select the action with the best (lowest) Q-value. However, this policy ignores some actions which may, after learning, have lower Q-values, resulting in a locally optimal solution. Therefore, a tradeoff is required between exploitation of good routes and exploration among available routes. This problem has been extensively studied in machine learning [4]. In Section 6 we compare several exploration strategies: greedy, stochastic, and weighted stochastic.

3. FEEDBACK ROUTING FOR OPTIMIZING MULTIPLE SINKS (FROMS)

The benefits of finding shared routes to multiple sinks are shown in the sample network of Figure 1 with one source and two sinks. Considering broadcast hop count, the best route from the source to both sinks goes through nodes 2, 4, and 5 with a total cost of 4. Finding this shared, lowest cost route is our primary goal. This section describes FROMS, Feedback



Neighbor ID	(sinkID, #hops)
1	(A,3), (B,5)
2	(A,4), (B,4)
3	(A,5), (B,3)

Fig. 1: Sample network with one source and two sinks with best shared route (solid arrows) and best point-to-point routes (dashed arrows), together with the Neighbor Table of the source.

```
1: routeData(DATA):
2:   PST.updateQ(DATA.Routing, DATA.reward);
3:   if (myAddr in DATA.Routing)
4:       possActions =
             PST.getAllActions(DATA.Routing.sinks);
5:       action = explore.select(possActions);
6:   DATA.reward =
         PST.getBestQ(DATA.Routing.sinks) + 1;
7:   DATA.Routing = action;
8:   sendBroadcast(DATA);
```

Fig. 2: Processing of one DATA packet.

Routing for Optimizing Multiple Sinks, the implementation of our Q-learning based approach. Subsequent sections provide details and variations of the protocol.

A. Gathering initial routing information

For estimating the initial Q-values of the available sub-actions, hop counts to all known sinks are required. This knowledge is collected when a sink broadcasts an announcement indicating its interest to receive a particular data type. The Neighbor Table (see Figure 1) is filled and later used for Q-value calculations. This approach is common to many protocols [5].

B. Learning the real Q-values

When data begins to flow in the network, agents start to learn the real values of the shared paths in the network. Figure 2 outlines one iteration of our routing protocol, detailed below.

Identifying and selecting actions: A learning agent with a data packet to route must select one action among those available, as explained in Section 2. Continuing our example, one possibility could be selecting two sub-actions $(1, \{A\})$ and $(3, \{B\})$. To manage effectively the options, we use a data structure called the Path Sharing Tree, PST. Implementation details are provided in [3]. Here, it is sufficient to understand that the PST provides a set of actions for reaching the desired sinks (line 4). The exploration strategy chooses one option from this set (line 5), as discussed in Section 4.

The major drawback of the PST is its size, due to the large number of possible sub-actions and actions. In Section 6 we explore and evaluate some size reduction heuristics.

Sending back rewards: We assume that when messages are wirelessly transmitted, they can be overheard by all neighbors. Packets contain all required routing information (e.g., which neighbors are responsible for routing to which sinks), allowing recipients to decide whether to process or drop the message (line 3). This broadcast model allows us also to piggyback additional information for all neighbors, for example, to provide reward values.

The reward value is calculated according to Section 2. In our example, consider a packet sent to node 2, then to node 4. When node 2 is forwarding the packet to 4, it piggybacks its reward for the source: its current best Q-value for both sinks, $3 + 3 - 2 = 4$ plus 1. The source updates its Q-value for the sub-action $(2, \{A, B\})$ (line 2) accordingly.

By repeating this process at each hop, accurate information propagates *backward* from the sinks to the sources, while the data flows *forward*. Note that rewards are exchanged only with one-hop neighbors, forcing the source to use the same route several times in order to receive rewards from distant nodes. Nevertheless, nodes other than the previous hop can benefit from the feedback and update their Q-values accordingly (line 2).

C. Behavior after convergence

After a finite number of steps the learning protocol converges, meaning the Q-values no longer change. Assuming that the network topology is stable, the balance between exploration and exploitation must be updated, as further exploration of actions with sub-optimal Q-values is unnecessary. Thus, our exploration strategy should switch to greedy action selection, minimizing routing costs (see Section 4).

If topology changes occur, e.g., due to node failure or mobility, the exploration rate must be adjusted. Details are discussed in Section 5.

4. Exploration strategies

The *exploration strategy* or *action selection policy* is the heart of the FROMS learning mechanism. It decides which action proposed by the PST to select for each data packet. Its most important properties are the exploration/exploitation ratio and convergence behavior. Exploitation of the best routes guarantees low network costs, however, to avoid local minima, exploration of non-optimal routes is required. Thus a balance is needed to achieve both energy-efficiency and near-optimality.

A. Choosing a route to explore

We focus on two techniques: greedy and stochastic.

Greedy exploration: This is a pure exploitation policy, always selecting the actions with the best (lowest) Q-values. We evaluate greedy because it provides a baseline for comparison and demonstrates the benefits of explorative learning.

Stochastic exploration: Stochastic techniques select among available routes, where each route has an assigned probability to be chosen. These probabilities change continuously during the learning process, giving priority to different routes over time. There are several options to initialize the route probabilities. One is to assign equivalent values, with the intention to treat all routes equally. Alternately, we set the initial probability to the inverse of its initial Q-value, giving preference to initially "good" routes.

At runtime, we adjust the probabilities by a tunable factor, f, each time the route is used and according to the reward received. When a route is used, its probability is decreased proportionally by f to give precedence to less-explored routes.

Second, we adjust the probabilities as rewards arrive. We differentiate between three types of reward, *positive*, *neutral* and *negative*. We respond to both positive and negative rewards by increasing the probability and priority by the factor f, since they signal a less-explored region or a change in the topology. A neutral reward, on the other hand, means that the current Q-value is correct and the probability of the route is not changed.

Section 6 evaluates the parameter space of these exploration options. Depending on the scenario, however, additional strategies can be applied.

B. Convergence behavior

An important property of an exploration strategy is its convergence behavior. During the learning process, Q-values are continuously updated through feedback rewards. After some finite number of steps the Q-values no longer change and the learning process is said to *converge*. After this, we should switch to a greedy policy, as further exploration will not reveal routes with lower costs. Implementing this behavior, however, is challenging since the convergence itself must be detected.

To address this issue, we introduce *stopping strategies* to define when the system has converged. Many possible options exist depending mainly on the energy restrictions of the network, for example, using the number of neutral rewards received, using the number of already explored routes, setting a cost threshold for the whole exploration process, etc. We consider two strategies, reward-based and a combination of reward- and route-based. They are evaluated in Section 6.

Reward-based strategy: counts the number of neutral rewards received, stopping exploration after N such rewards. In some sense, this considers the final system convergence.

Route- and reward-based strategy: combines with the first, but requires that at least M routes are explored before counting the neutral rewards, making sure distant rewards have the chance to arrive.

5. Failure recovery and sink mobility

The feedback and learning techniques in FROMS enable it to naturally handle recovery and mobility of nodes with no changes or optimizations.

A. Recovery after node failure

Keeping all possible routes in the PST not only enables us to select routes for learning, but also gives alternate routing options in case a node fails. For example, when a node used in the best shared route fails, its neighbors automatically switch to the next-best alternative, thus guaranteeing continuous delivery to all sinks. Further, updates relevant to the failed node propagate through the system as rewards, since also the Q-values will change.

The only new aspect we must explicitly address is neighbor failure detection. Here we assume a node sends a special *dying* message just before switching off, emulating a scenario where node failures are due to battery discharge and can be predicted. Clearly other scenarios and solutions are available.

The behavior of our protocol in the presence of a node failure is shown in comparison to Directed Diffusion in

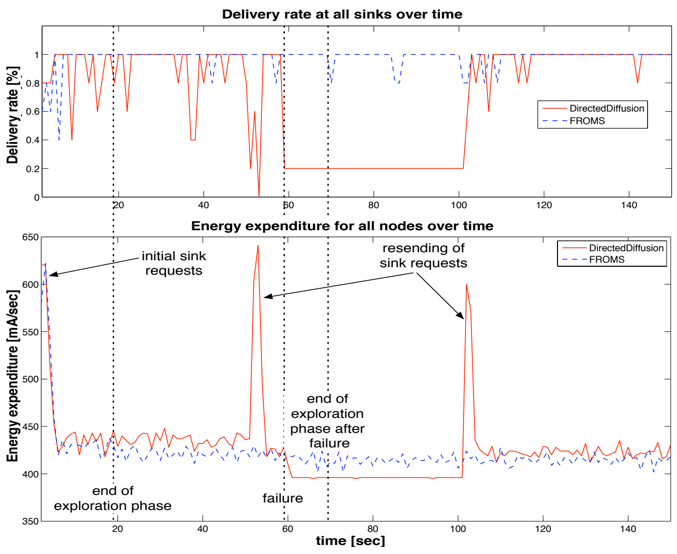


Fig. 3: Delivery rate and energy expense after failure, as handled by FROMS and DIRECTEDDIFFUSION.

Figure 3. This example shows a network with 50 nodes, one source, and five sinks. A single node on the best route to the sinks of both protocols fails (at time=59), and the topology remains connected. We show two different metrics: the data delivery ratio and the energy expenditure over time.

Before the node fails, both protocols deliver data at almost 100%, all losses are due to packet collisions. FROMS maintains this nearly perfect data rate after the node failure, because it simply switches to a different route (action) available from the PST. DIRECTEDDIFFUSION, on the other hand, does not maintain enough information to switch, thus its delivery drops to 20% until the next periodic sink announcement.

Considering system cost, the high cost of periodic sink broadcasts is evident. Additionally, the benefits of FROMS learning approach are shown by the lower, overall cost compared to DIRECTEDDIFFUSION. Although we normally expect increased costs during exploration, in this sample topology, the increases were insignificant. Finally, although DIRECTEDDIFFUSION has a lower cost between the failure and the next sink request announcement, it delivers only 20% of the packets.

Clearly DIRECTEDDIFFUSION can be tuned for faster recovery, however these changes always come with increased costs. For example, the re-broadcast frequency of the sink announcement can be increased, but at the cost of a dramatic increase of the network energy expenditure even when no nodes are failing. Instead, FROMS has the innate ability to elegantly recover after node failure.

B. Sink mobility

WSN scenarios with fixed sensor nodes and mobile sinks, e.g., laptops or PDAs, are common. Such sink mobility represents a challenge to existing routing protocols, as it constantly changes the topology and makes efficient routing difficult. From the perspective of FROMS, mobility of sinks is a two-step process: first, one or more network links are broken, then new links are added. The first step, link failure, is handled directly by our

TABLE 1: PST PRUNING HEURISTICS, EVALUATED IN TERMS OF PST SIZE (IN BYTES) AND ACHIEVED OVERHEAD PER PACKET (NORM. BY OPTIMAL)

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

FROMS recovery mechanism as described above. To handle link addition, we assume sinks periodically broadcast their interest among their *one-hop neighbors*. This updates both the Neighbor Table and the PST.

Both steps only change Q-values at the one-hop neighbors of the sink, the continuous reward exchange disseminates the changes throughout the network, and triggers exploration that quickly converges because the changes are minimal. With low speed mobility, continuous packet delivery is possible.

6. SIMULATION RESULTS

Next we demonstrate the behavior of our protocol, comparing it to both an implementation of one-phase-pull Directed Diffusion[5] and a broadcast based Steiner Tree solution. Directed Diffusion is the mostly relevant comparison point, since it is defined under the same scenario and uses the same initialization process as FROMS.

For simulation, we used the Mobility Framework in Omnet++ (www.omnetpp.org). The various routing protocols are implemented using standard physical layer (interference-based, simple bit error function), MAC layer (non persistent CSMA), and battery consumption models (simulated MICA-2 nodes). The Steiner tree uses an exact enumeration algorithm in MatLab on limited size networks, and considers broadcast costs. Code and precise simulation parameters are available online (www.inf.unisi.ch/projects/mics).

Unless otherwise stated, all evaluations present averages over 50 different random, connected 50-node topologies, on a field of 1500x1500 m, with a maximum communication radius of 400 m. To even out the effects of the stochastic nature of FROMS, we ran each topology with 5 random seeds. Data is sent every second from the sources to all sinks for 500 sec.

We use three exploration strategies with varying parameters: UNIFORMEXPLORE initializes all action selection probabilities to 1, with $f = 1$; STOCHASTICEXPLORE with initial probability of 1 and varying f; and STOCHASTICWEIGHTEDEXPLORE with initial probability of the inverted Q-value and varying f. GREEDYEXPLORE provides a comparison point.

A. Tree pruning heuristics parameters

As discussed in Section 3, different heuristics can be applied to the PST, limiting its size and thus saving memory on the nodes and speeding up the learning process. We consider two PST pruning heuristics: limiting the number of routes per sink to N and limiting the maximum route cost to a sink to $bestCost+C$.

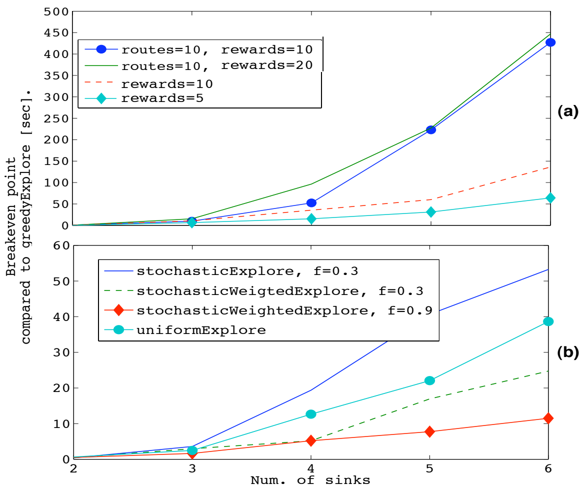


Fig. 4: The breakeven point for stopping (a) and exploration (b) strategies

Both types of information refer to the Neighbor Table (see Figure 1), *before* the sub-actions and actions are computed and initialized. As the PST size decreases, fewer actions are available for selection. Because the best route may be among those pruned, we expect the protocol performance to decrease as the size of the PST decreases. This trend is shown for UNIFORMEXPLORE in Table 1 for various values of N and C and for multiple numbers of sinks. The remainder of our experiments use a moderate size PST ($N = 4, C = 1$) that yields route costs *close enough* to optimal.

Interestingly the largest tree does not always discover the best routes. This is due mainly to communication failures.

B. Stopping strategy parameters

Section 4 outlined several stopping strategies based on the minimum number of routes to be explored and the number of neutral rewards received. To evaluate these, we introduce a new metric, namely the *breakeven point*, which defines the amount of time that the system must execute in order for the expense of the exploration to be compensated by the low, stable costs, compared to GREEDYEXPLORE. Figure 4(a) shows that for UNIFORMEXPLORE, as the time to converge increases because of growing number of routes to explore, the time to break even also increases. Note that the overall time to break even is low compared to typical system lifetimes.

Based on this, we select the best stopping strategy for UNIFORMEXPLORE as 5 neutral rewards. Similar experiments for STOCHASTICEXPLORE and STOCHASTICWEIGHTEDEXPLORE determined their best parameters.

C. Comparing exploration strategies

To show the performance benefits of learning, we use again GREEDYEXPLORE for comparison and consider the breakeven metric for several strategies introduced earlier.

Figure 4(b) clearly shows that some strategies minimize breakeven time better than others. Because STOCHASTICEXPLORE slowly reduces route probabilities, it needs longer to converge, and takes longer to break even. On the other hand, STOCHASTICWEIGHTEDEXPLORE uses initial Q-values

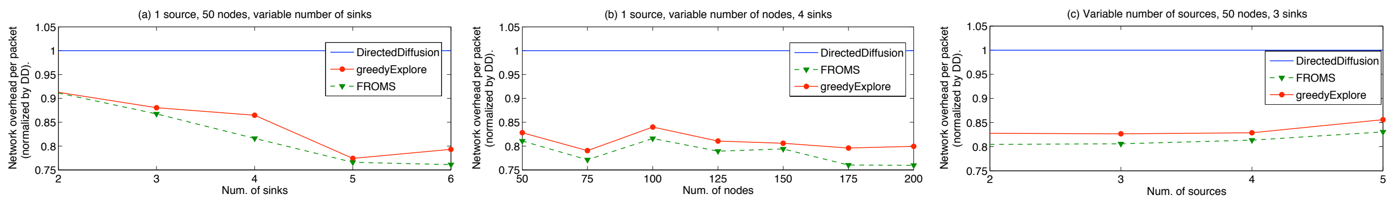


Fig. 5: Comparison between FROMS, GREEDYEXPLORE and DIRECTEDDIFFUSION in routing costs per packet.

to weight probabilities, and with $f = 0.9$ rapidly decreases all probabilities, converging faster, yet finding good enough routes to quickly break even with respect to GREEDYEXPLORE.

D. *Comparing* FROMS *to* DIRECTEDDIFFUSION

Finally we compare FROMS (UNIFORMEXPLORE with $f = 1$) and GREEDYEXPLORE against DIRECTEDDIFFUSION [5], showing the normalized routing costs per packet for various numbers of sources, sinks and network sizes *after* convergence. In all scenarios, FROMS outperforms DIRECTEDDIFFUSION, showing significant cost improvements.

Figure 5(a) shows a scenario with 50 nodes, a single source and a variable number of sinks. In general, FROMS decreases the network overhead per packet by 20% over DIRECTEDDIFFUSION, due to its ability to find shared routes. Figure 5(b) shows the behavior as the network density increases (increasing number of nodes in the same area) and thus the scalability of FROMS.

Finally, Figure 5(c) considers a multiple-source scenario, demonstrating good performance also for this task. Note that no data aggregation from different data sources is considered, which would lead to even better performance.

7. RELATED WORK

Here we place our work in the context of related WSN research, specifically routing to multiple sinks and use of learning techniques in WSN.

Multiple sink routing in WSN and MANET: Most efforts in WSN routing research concentrate on the multiple sources-single sink scenario. However, some protocols implicitly or explicitly support multiple sink routing. The approach most related to ours uses feedback among neighbors to merge existing, single source-sink routes [6]. The key difference w.r.t. FROMS is our explicit exploration that eventually locally learns multi-hop information.

The multiple-sink problem can also be seen as an instance of a content-based networking system in which destinations express interest, and matching data is routed accordingly. Several efforts [5] address this for WSNs, but in contrast to our work, they do not optimize routes for multiple sinks.

Many multicast protocols have been developed in the MANET domain [7], [8]. However, they either assume different scenarios (known geographic node locations [8]) or incur large communication overhead for constructing and maintaining the multicast tree.

Learning approaches: The cornerstone of our approach is its ability to learn better path information over time. It was partially inspired by AntHocNet [9], an approach for learning routes in wireless ad hoc networks based on ant colony optimization. A similar SI-based approach is taken in [10] to build optimal multicast trees in a MANET. In WSNs, however, the overhead of sending ants through the network unnecessarily wastes energy.

Reinforcement Learning [4] has also been applied to WSN routing to single sinks [1], [2], however they either assume global topology knowledge, or solve different problems such as optimal data compression along the path to a single sink [1].

8. CONCLUSION AND FUTURE WORK

This paper presents both a formal definition of the multiple sink routing problem as a reinforcement learning task and FROMS, an implementation of our Q-learning approach. Our evaluation clearly shows that the additional expense of learning, combined with the negligible overhead to piggyback reward information significantly lowers routing cost. The observation that FROMS innately supports node failure and sink mobility further increases its applicability.

Our immediate plans include further study of FROMS in various mobility and failure environments. We are also working on a real implementation and deployment to accurately assess performance outside the inherent boundaries of simulation.

REFERENCES

[1] P. Beyens, M. Peeters, K. Steenhaut, and A. Nowe, "Routing with Compression in Wireless Sensor Networks: a Q-Learning approach," in *Proc. of the 5th Eur. Wkshp on Adaptive Agents and Multi-Agent Systems*, 2005.

[2] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," *Advances in Neural Information Processing Systems*, vol. 6, 1994.

[3] A. Egorova-Förster and A. L. Murphy, "A Feedback Enhanced Learning Approach for Routing in WSN," in *Proc. of the 4th Wkshp on Mobile Ad-Hoc Networks*. Bern, Switzerland: Springer-Verlag, 2007.

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

[5] F. Silva, J. Heidemann, R. Govindan, and D. Estrin, *Frontiers in Distributed Sensor Networks*. CRC Press, Inc., 2003, ch. Directed Diffusion.

[6] P. Ciciriello, L. Mottola, and G. Picco, "Efficient routing from multiple sources to multiple sinks in wireless sensor networks," in *Proc. of the 4th European Conf. on Wireless Sensor Networks*, 2007.

[7] R. Sun, S. Tatsumi, and G. Zhao, "Q-map: a novel multicast routing method in wireless ad hoc networks with multiagent reinforcement learning," in *Proc. of the IEEE Conf. on Computers, Communications, Control and Power Engineering*, vol. 1, 2002, pp. 667–670 vol.1.

[8] J. A. Sanchez, P. M. Ruiz, and I. Stojmenovic, "GMR: Geographic multicast routing for wireless sensor networks," in *Proc. of the 3rd Annual IEEE Conf. on Sensor and Ad Hoc Communications and Networks*, vol. 1, 2006, pp. 20–29.

[9] G. Di Caro, F. Ducatelle, and L. Gambardella, "AntHocNet: an Ant-Based Hybrid Routing Algorithm for Mobile AdHoc Networks," in *Proc. of the 8th Int. Conf. on Parallel Problem Solving from Nature*, Birmingham, UK, 2004.

[10] C.-C. Shen and C. Jaikaeo, "Ad hoc multicast routing algorithm with swarm intelligence," *Mobile Netwworks and Applications*, vol. 10, no. 1-2, pp. 47–59, 2005.