An Efficient Implementation of Reinforcement Learning Based Routing on Real WSN Hardware

Anna Förster† and Amy L. Murphy‡ and Jochen Schiller⋆ and Kirsten Terfloth⋆
†University of Lugano, Switzerland, anna.egorova.foerster@lu.unisi.ch
‡FBK-IRST, Italy, murphy@fbk.eu
⋆Freie Universität Berlin, Germany, {schiller,terfloth}@inf.fu-berlin.de

***Abstract*—Efficient multi-hop data dissemination is a crucial building block to enable mature wireless sensor network (WSN) applications. Exploiting machine learning for these routing problems has received increasing attention in recent years due to its flexibility and localized mechanisms, however, with such an approach the resulting protocols often have increased memory and processing time requirements. Nevertheless, these requirements *are* within the reach of today's WSN hardware, however few substantial tests have been performed to clearly demonstrate this. This paper evaluates and discusses the results and experiences gained from implementing our reinforcement learning based multicast routing protocol (FROMS) in a testbed of ScatterWeb nodes. A comparison of our results is made to a well-known WSN routing scheme, namely a multicast variation of Directed Diffusion. Our evaluation includes several minor, but practical modifications to both protocols such as transmission backoffs and the use of acknowledgments.**

This paper offers three main contributions. First, we demonstrate that machine learning algorithms *can be* efficiently implemented on resource restricted devices and that they perform very well in multiple network scenarios. Second, we confirm the validity of simulation results obtained in a previous evaluation of FROMS, and at the same time gather delivery rates under realistic settings. Finally, we offer some general observations on properties and pitfalls of WSN implementations along with potential solutions.

I. INTRODUCTION

Research during the past several years has yielded a large number of WSN protocols and applications. Among them, those based on machine learning techniques have increased in significance because of their flexibility and performance potential [1]. Nevertheless, most of these systems have only been evaluated in simulations, often with unrealistic assumptions including perfect communication, unlimited memory and processing resources, etc. As machine learning is known to have higher memory and processing requirements, its actual implementation on resource constrained sensing devices is mandatory to not only confirm their applicability, but also to warrant continued exploration and evaluation.

This is not to say that simulation-based evaluation is not useful. In fact, there is no viable substitute to provide a general proof of concept for a new protocol or to evaluate the full parameter space in large, complex environments that are simply impossible to recreate in a real testing environment. Additionally, simulation studies provide valuable feedback for selecting reasonable parameter settings. Nevertheless, for core system software such as routing protocols, simulations must be complimented with tests on real hardware. The step of porting a protocol from simulation to real hardware reveals implicit assumptions, which, when confronted, provide opportunities to increase the robustness of even well-designed and thoroughly simulated protocols.

This paper outlines in detail the successful implementation of our reinforcement learning based multicast routing protocol FROMS [2] on ScatterWeb sensor nodes, demonstrating both the feasibility of implementing a machine learning based protocol on real hardware as well as verifying its increased performance in comparison to traditional approaches. These results confirm our previous simulation results [2] and augment them with accurate measurements of delivery rate and protocol overhead in a typical sensor network environment. We also take a step back from the evaluation, reporting challenges encountered during porting and their corresponding workarounds, and pinpointing both the influence and utility of straightforward techniques to improve network performance.

In the following, Section II reviews related work. Section III provides a basic summary of the two routing protocols we consider: FROMS and a variant of Directed Diffusion [3]. Implementation details are presented in Section IV, followed by a study of the results gathered during our experiments in Section V. Section VI offers lessons learned, before Section VII concludes the paper with a brief summary and a discussion of future research directions.

II. RELATED WORK

We consider three distinct areas of research as related to this experimental study: multicast routing protocols for WSNs, machine learning based algorithms for WSNs, and testbeds or deployments utilizing genuine sensor network hardware.

The research community has proposed a large number of routing protocols targeting various application scenarios. Among them a few multicast algorithms have emerged to route data from one or more sources to multiple, possibly mobile, sinks. Some approaches such as MSTEAM [6] and GMR [7] utilize geographic information, while others such as [8], exploit hop-based metrics. An implementation of ADMR, a protocol initially designed for MANETs, is available for the MicaZ mote platform [9].

Simultaneously, the relatively new idea of applying machine learning techniques to WSNs has received increased attention. Some flexible, localized algorithms such as ant

colony optimization or reinforcement learning offer substantial tools to e.g. efficiently implement self-organized behavior, as summarized in [1]. Representative protocols that implement machine learning techniques on WSN hardware include a single source/single sink Q-routing protocol developed for Crossbow motes [10] and an optimization for routing schemes based on link quality estimates [11] on MicaZ motes.

Finally, various deployments and testbeds have been setup and used for studies. General deployments, such as those for habitat [12], [13] or environmental monitoring [14] have been established to better understand natural processes by continuously collecting data samples, but at the same time to test sensor network hardware and software under harsh environmental conditions. On the other hand, collections of sensor nodes have also been used to evaluate isolated protocols, including those for routing [9], [11], medium access [15], [16], event-based information processing [5] or transport for network reprogramming [17]. While the number of nodes used in the deployments depends on the application scenario, testbed sizes range from 2 to 60 nodes.

Our work relates to these topics as FROMS implements a multicast routing protocol by enlisting machine learning techniques. To the best of our knowledge, it is the first protocol to combine these themes in the context of WSNs while at the same time offering a validation in a mid-size testbed.

III. PROTOCOL BACKGROUND

Next we describe the implemented routing protocols, namely FROMS [2] and Directed Diffusion [3], and the application layer used to test them. This upper layer ensures that the same traffic patterns are applied and offers functions to issue sink announcements, called DATA_REQ messages, to all nodes in the network via network-wide broadcast and to send DATA messages from sources to sinks. Note that DATA messages can be sent as multicast messages by both protocols. The routing protocol selects one or more neighbors for forwarding the packet to dedicated sinks, indicating this information in the protocol header.

A. Directed Diffusion

We implemented the one-phase pull variation of Directed Diffusion [3] (DD), a very simple but versatile protocol with small maintenance costs. The original version DD is a multi-source single sink routing protocol, where each of the nodes in the network keeps a single gradient to the sink which represents the best next hop. It consists of three phases: sink announcement broadcast, a slow data delivery phase with reinforcement, and finally a constant data delivery phase. In the first, information about potential neighbors to forward data to is gathered by each of the participating nodes to establish gradients indicating the possible next hop. In the second, data is sent infrequently to the sink utilizing all possible gradients, while the sink itself reinforces the best path to each source. Each node uses this reinforcement to identify a single gradient to reach the sink. Eventually, the sources start sending data using the selected gradients at the full data rate.

The one-phase pull version of DD used here skips the second reinforcement phase and instead builds the gradients in the initial phase by using the delay and number of hops to the sinks of the sink announcement packets. One can argue that the original version is more reliable, especially in the presence of asymmetric links. However in reality the first phase is sufficient to build good gradients. Additionally, the one-phase pull version reacts faster to topology changes and node failures, since the gradients are re-built every time a new sink announcement arrives. Since FROMS is specifically designed to address those problems, we decided from the very beginning to evaluate FROMS against one-phase pull DD.

The application of DD in a multicast scenario requires only minor modifications as opposed to the unicast use case: A node has to simply keep one gradient per sink instead of one application-level gradient. Additionally, multicast addressing is applied to DATA packets where applicable, but without any learning strategies as implemented by FROMS. Simply speaking, DD does not explore the network to find shared next hops to optimize for routing costs, but builds up gradients to sinks independent from each other without neglecting obvious shared paths to one-hop neighbors.

B. FROMS

FROMS is a multi-source multi-sink routing approach that uses Q-learning to identify routes in the network that optimize for the shortest path or the best energy efficiency [4]. A full description is available in [2]. In contrast to DD, FROMS uses the *locally* available data such as hop counts to the sinks and a feedback exchange mechanism among neighbors to explore the network and find the *globally optimal* path to all sinks.

During the sink announcement broadcast, each node generates a full neighbor routing table listing all neighbors together with their hop count to each individual sink. This is in contrast to DD which keeps only the lowest hop count. When a DATA packet arrives for routing, FROMS considers all possible routes, specifically all possible combinations of neighbors to send the packet to the desired sinks. Then, the cost of each route is estimated using the routing information and/or the node state and a dedicated cost function. Possible metrics for cost evaluation include the sum of the hops to all sinks minus the shared paths that can be saved from sending a multicast message in a broadcast medium, or simply using remaining battery level at the nodes [4].

The Q-learning protocol selects a route to use either to *exploit* the best known route or to *explore* a non-optimal route. Feedback values are piggybacked on all DATA packets and deliver the best possible route costs to all listening neighbors, which then update their own route costs. As such, exploration enables nodes to learn the real costs of shared routes whose estimates were initially too high. This exploration/exploitation ratio is controlled by a tunable parameter, ϵ. Our previous work [2] considered multiple exploration/exploitation strategies, however here we have implemented only ϵ-greedy which selects either the best route with probability $1-\epsilon$ or a random route with probability ϵ.

TABLE I
CHARACTERISTICS OF THE SCATTERWEB MSB430 SENSOR NODES

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

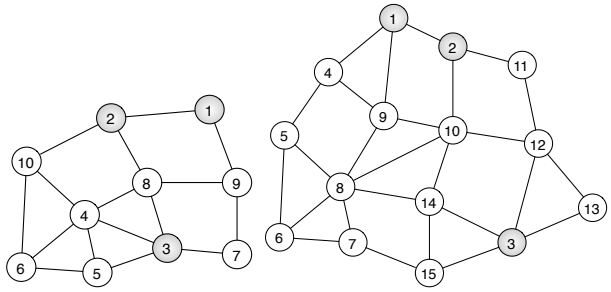

Fig. 1. Topologies 1 (left) and 2 (right), sinks are shaded, source is node 6.

A good exploration/exploitation ratio ensures that routing costs are kept low by often using the currently best available route which corresponds to the local minimum. On the other hand, it should infrequently use non-optimal routes to learn their real costs, thus lowering route costs over time and identifying the global minimum.

These three mechanisms, initial route cost estimates (Q-value estimates), exploration/exploitation and feedback (learning) are the crucial building blocks of the FROMS protocol. With them, FROMS not only finds the best shared route to multiple sinks (the optimal multicast tree), but also copes successfully with node failures and mobility. In the following, we concentrate on substantiating the first claim.

IV. IMPLEMENTATION DETAILS

This section summarizes implementation and testbed details and clarifies parameters and settings to allow for reproduction of our results and/or comparison to other experiments. As previously stated, our platform consists of ScatterWeb MSB430 sensor nodes which offer typical RAM/ROM sizes, CPU and transceiver capabilities, and a rich and freely available set of software libraries (we relied on Version 1.1), as summarized in Table I. Alternative platforms and their characteristic properties are discussed in [18] and are omitted here for brevity.

A. Network topologies and traffic model

Our tests ran on the two artificially enforced topologies (see also Section VI) shown in Figure 1. For simplicity, sinks and sources are assigned to specific node IDs at compile-time. In our case, nodes 1 to 3 serve as sinks while node 6 is the data source. The chosen topologies combine both densely deployed areas, with nodes having up to six direct neighbors, as well as border areas with low node density. They also allow for multicast optimization. This way, we were able to observe protocol behavior in different settings at the same time and ensure a fair measurement of on-node protocol performance, see Section V-B.

TABLE II
PARAMETERS FOR THE FROMS ROUTING DATA STRUCTURES AND ROUTING STRATEGY. THE VALUE USED IN OUR EXPERIMENTS IS GIVEN IN THE SECOND COLUMN.

Parameter	Values	Description
num_sinks	3	The maximum number of sinks
num_neigh	4	The maximum number of neighbors
num_routes	50	The maximum number of available routes at a single node
ϵ	0.3	The probability for exploration/ exploitation of routes

The application layer, common to both routing protocols, controls the traffic injected in the network. Initially, sinks send data requests every $packet\_interval$ seconds, and after the first DATA packet arrives, either no additional requests are sent (FROMS), or the request rate slows to one request every $10{*}packet\_interval$ seconds (DD). Data is generated and sent by the source n times every $packet\_interval$ seconds. We set $packet\_interval = 5s$ and route $n = 100$ packets from the source to all designated sinks. This setting is considered high for a WSN deployment, but it allows us to evaluate under intense traffic and to minimize testing time.

B. Routing table requirements

The DD routing table contains a single entry for each sink in the network, identifying the best available next hop to a sink inferred from incoming DATA_REQ messages.

To allow FROMS to identify routes that have low shared cost to multiple sinks, even if the route costs to each individual sink are not the minimal, it must store multiple options to choose the next hop(s) from. Further, because FROMS learns the costs of shared routes to multiple sinks, not to the individual sinks as in the DD routing table, we maintain a separate data structure for shared routes to multiple sinks and update it as feedback arrives. In our previous work, we used a dynamically allocated tree-based data structure referred to as the path sharing tree, PST. In order to minimize storage of all available routes, the PST makes intensive use of dynamic memory alocation/deallocation. However, since dynamic memory allocation is problematic in embedded programming for several reasons, see also Section VI, we utilized a fixed size statically allocated table bounded by the num_routes parameter given in Table II. A number of heuristics that can be used to limit PST size, as well as a description of their effect on protocol performance are outlined in [2].

V. RESULTS

The experiments and results are divided into three classes: on-node characteristics such as memory and processing time; in-network protocol performance in terms of delivery rate and routing costs; and in-network performance of routing protocol enhancements such as transmission backoff and acknowledgments in terms of delivery rate and routing costs.

TABLE III
MEMORY USAGE AT COMPILE TIME. THE SCATTERWEB LIBRARY ALONE IS GIVEN FOR COMPARISON.

Memory usage	FROMS + ACKs	FROMS	DD + ACKs	DD	ScatterWeb Library
ROM (bytes)	33610	31986	29648	28024	19628
RAM (bytes)	3345	3326	2952	2932	1476

A. Evaluation metrics

Comparison of the two protocols requires a set of performance metrics that accurately represents the key protocol behavior in a real hardware environment. Therefore we chose to measure the delivery rate to the sinks, the routing costs and memory and processing requirements.

In terms of routing costs, we consider two metrics: *cost per generated* packet and *cost per delivered* packet (both in terms of the number of transmissions, ETX, which is the same as number of hops because packets are always sent in broadcast). The first is the total number of DATA packet transmissions in the network divided by generated DATA packets. The second metric divides by the number of actually received DATA packets. We include both because the first considers cost per generated packet, which could be very low not only because the routing protocol uses a short route, but also because many packets get lost. The second cost, instead, could be very high not only because the routing protocol uses a very long route, but also because packets could get lost and their already made transmissions increase the overall cost. In case of very high delivery rate both costs are the same. However, we believe the second metric, cost per received data packet, is more reliable and realistic since it naturally integrates also the delivery rate.

Note that both routing cost metrics include only DATA packets, ignoring DATA_REQ packets. Thus, we compare only real DATA routing cost, ignoring the overhead of collecting routing information at the nodes. Because of our manual experimental setup (see Section VI) we could not fairly measure this overhead, and instead rely on simulation for such data.

In terms of memory and processing requirements, we consider the *memory footprint* and *processing time* for certain critical functions such as selecting a route for a packet.

B. On-node performance of DD and FROMS

Memory usage: Table III presents the memory footprints at compile-time for DD and FROMS together with the application layer. It shows the memory reserved for the flash ROM and the RAM. The footprint of the ScatterWeb Library alone is given as point of comparison. The vanilla implementation of DD on top of ScatterWeb e.g. consumes roughly 3KB of RAM at compile time, leaving 2KB for stack allocation and application-level protocols.

Note that the data structures of both protocols are static and are thus already included in the memory footprints. Despite the much more complex and large data structures of FROMS, its memory requirements are not significantly higher. DD has a very tiny data structure and despite this its implementation size is not negligible. In fact, the majority of its memory is used for the functionality of the protocol, not for data structures.

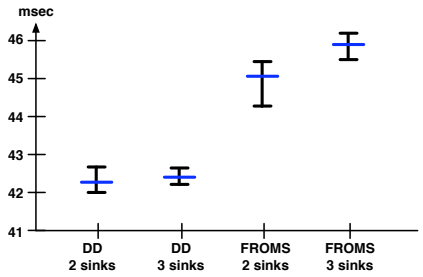

Fig. 2. Processing time to find a route in milliseconds for DD and FROMS.

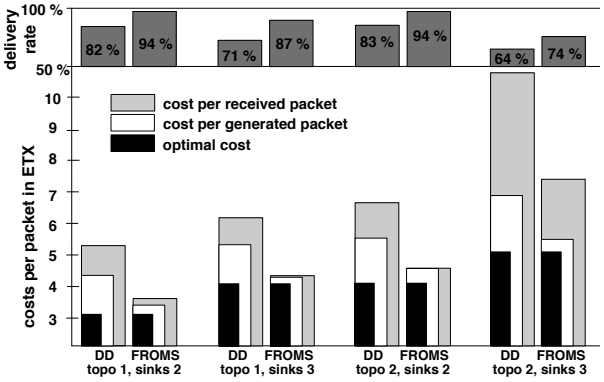

Fig. 3. Routing costs and delivery rates for FROMS and DD in various network scenarios.

Processing time: We measured the time needed to find a route for each packet in the network at every node in milliseconds. Basically, we discovered that it takes slightly longer to find a route to more sinks but the difference between the protocols is negligible. The results in Figure 2 are summarized based on the number of sinks in the network. They are obtained from experiments with topology 2 only, with 2 or 3 sinks.

C. In-network performance of DD and FROMS

Next, we compare the performance of both protocols in terms of delivery rate and routing costs. Figure 3 summarizes the results for several network configurations. As expected from our simulation experiments and theoretical analysis, FROMS achieves lower routing costs. This can be attributed to its *learning* algorithm which actively explores the network for optimal routes. We also compare the performance against the theoretically optimal cost.

In simulation we were unable to evaluate an accurate delivery rate since transmission failures cannot be reliably simulated. Here, instead, we confirm our theoretical expectation that FROMS is able to achieve higher delivery rate in any network scenario. Data is lost in DD mainly due to the higher in-network communication caused by the periodic sink announcements. As explained in Section III-A, every sink periodically sends sink announcements via a network wide broadcast, causing high traffic and thus collisions. A second factor is the increased number of forwarding nodes selected by DD compared to FROMS. This increases the traffic and

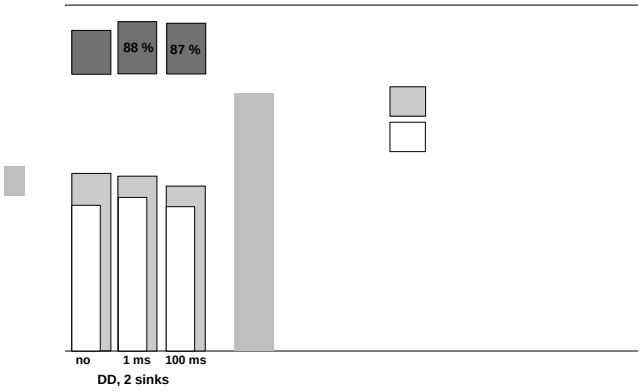

Fig. 4. In-network performance when applying transmission backoff, results from topology 1.

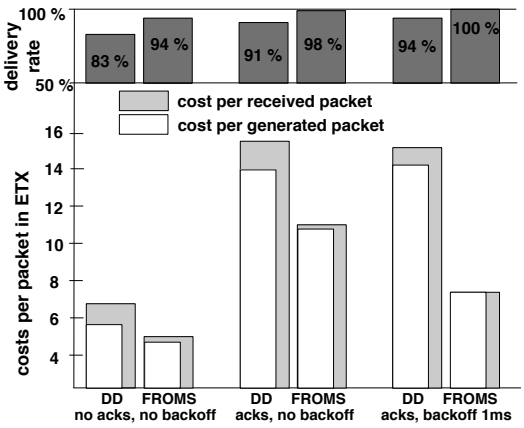

Fig. 5. In-network performance when using acknowledgments, results from topology 2.

collision probability leading to packet losses. Figure 3 supports these observations, showing that the delivery rate of both protocols clearly drops in networks with larger numbers of participating nodes and sinks.

D. In-network performance of routing enhancements

The main goal of enhancing the routing protocols is to improve the delivery rate. In our experimental setups, we study the effect of transmission backoff and as well as application-level acknowledgments on DD and FROMS.

Figure 4 presents the results when using transmission backoff, clearly showing that the technique is highly effective at improving delivery rates. We implemented a simple algorithm in which a parameter (in our case 0, 1 or 100 ms) is multiplied with the node's ID and this delay is applied before forwarding any packet. This backoff reduces packet collisions and thus increases successful delivery.

Another common mechanism to increase the delivery rate is to force packet acknowledgments. We use overhearing of DATA packets as implicit acknowledgments, avoiding additional costs. The incurred overhead stems from re-sending unacknowledged packets. Figure 5 shows how routing costs skyrocket, while the delivery rate also increase. Communication failures cause not only data loss, but also acknowledgment loss. This results in resending packets which were actually received, but not acknowledged. Consequently, the communication traffic explodes, leading to even higher loss rates.

VI. LESSONS LEARNED

While implementing and deploying FROMS and DD on the ScatterWeb platform we encountered several challenging problems. This section steps through these, ordered by their importance. We describe each problem and our solution, discussing alternatives where applicable. Although the experiences shared here have more the character of practical guidance than of algorithmic facts, we believe that especially those working on embedded programming and sensor network testing for the first time will benefit from it.

A. Multi-hop topologies do not necessarily come naturally

Since both of the implemented protocols are designed for multi-hop data dissemination, one of the first things explored was how to build different network topologies for real-world testing. Unfortunately, we discovered that constructing real multi-hop networks within our building is impossible due to the thin walls and the good communication quality between nodes. Even with minimal transmission power, we measured a maximum distance allowing direct communication between two nodes to be more than 50 meters in an indoor setting, a value that, due to less obstructions, is even bigger in outdoor areas. For gaining insights on the impact of the proposed reinforcement learning strategy, we needed at least a network diameter of 3-4 hops, a distance we were not able to achieve with a pure physical distribution of nodes. This has been observed in prior deployments, such as the fence monitoring experiment [5].

Our solution was to manually control the topology by artificially limiting the nodes' communication. Specifically, we established tables at each node that contain pre-defined IDs of neighbors whose packets a node is allowed to process. Packets received from other nodes are dropped. The main disadvantage of this approach is the additional interference on the wireless medium as compared to real multi-hop networks, a fact that may contribute to higher packet loss ratios and which makes transmission backoff even more important.

Another solution could be a quality-driven approach for topology control where a threshold is used to evaluate the link quality between nodes based on received signal strength (RSSI). Only a subset of neighbors with high reliability will be chosen for communication. The main advantage compared to static neighbor tables as described above is that this approach allows for changing topologies at runtime.

B. Dynamic memory allocation can fail unexpectedly

When porting FROMS to the ScatterWeb platform, we naturally started from the simulation code. However, we soon discovered that dynamic memory allocation and deallocation do not work properly on the nodes, a mechanism that the central PST data structure [2] heavily relies on. Since the ScatterWeb firmware does not support dynamic memory itself,

one can rely on a library available from Texas Instruments, which, however, is reported to contain several known bugs.

We decided to re-use most of our data structure but to implement it with the help of static arrays. As a result, an upper bound for memory utilization must be specified at compile-time, a disadvantage since this requires a programmer to specify the maximum possible number of sinks as well as neighbors per node a priori. Additionally, this part of the memory cannot be utilized for other things at runtime, even if it is not fully used. However, for our tests the usage of static memory was sufficient (see Section V).

C. Gathering statistical data needs to be planned

While running simulations, statistical data such as the delivery rate, costs per packet etc. is gathered from the simulation kernel itself and then presented en-block to the user. In our experimental setting, we need to log and reacquire this data explicitly on the nodes, which in turn blocks additional RAM at runtime.

Two distinct methods for logging were explored: a simple utilization of runtime variables holding the necessary information, and persistent storage of log data onto SD cards which can be used with the ScatterWeb nodes. In the first case, data is transient, thus node failures naturally lead to their complete erasure. The prime advantage is however that memory consumption can be planned at a fine granularity as opposed to an SD-card based solution in which only blocks of a minimum 512 byte can be written due to write asymmetry of flash memory. Notably, a cache this size needs to be exclusively allocated in advance, and the driver itself consumes additional resources. In compensation, data is persistently logged for multiple test runs before extracting it via debug commands and serial communication.

D. Handling lost or corrupted packets

As expected, some packets got lost or corrupted during transmission. However, this is not a major challenge to our experiments, since it is expected in any wireless scenario and the protocols do not rely on perfect communication. To cope with corrupted packets we implemented a CRC sum test.

VII. SUMMARY AND FUTURE WORK

This paper presented our experiences and results from implementing and deploying two routing protocols on a real hardware testbed. One of these is based on machine learning techniques, thus clearly demonstrating that such an approach is feasible within the constraints of real hardware. The experimental results confirmed those of our prior simulations and revealed common implementation challenges. Among our numerical results, we found out that using transmission backoff is always a good idea as it improves the delivery rate in any network scenario. In contrast, using acknowledgments unnecessarily increases the routing cost without significantly affecting the delivery rate. As a general take-away lesson it can be noted that routing costs are closely related to the delivery rate and thus the developers of sensor networks must try to minimize total communication traffic in the network to improve delivery rate and thus also routing costs.

In the future we will extend our testbed to 30 nodes and increase the experimental scenarios to include changing topologies, mobile sinks, and new cost functions, such as those based on remaining energy.

REFERENCES

[1] A. Förster, "Machine Learning Techniques Applied to Wireless Ad-Hoc Networks: Guide and Survey," in *Proc. of the 3rd Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2007.

[2] A. Förster and A. L. Murphy, "FROMS: Feedback Routing for Optimizing Multiple Sinks in WSN with Reinforcement Learning," in *Proc. of the 3rd Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2007.

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

[4] A. Förster and A. L. Murphy, "Balancing Energy Expenditure in WSNs through Reinforcement Learning: A Study," in *Proc. of the 1st Int. Wrkshp on Energy in Wireless Sensor Networks (WEWSN)*, 2008.

[5] G. Wittenburg, K. Terfloth, F. López Villafuerte, T. Naumowicz, H. Ritter, and J. Schiller, "Fence monitoring – experimental evaluation of a use case for wireless sensor networks," in *Proc. of the 4th Eur. Conf. on Wireless Sensor Networks (EWSN)*, 2007.

[6] H. Frey, F. Ingelrest, and D. Simplot-Ryl, "Localized minimum spanning tree based multicast routing with energy-efficient guaranteed delivery in ad hoc and sensor networks," in *Proc. of the 9th IEEE Int. Symp. on a World of Wireless, Mobile and Multimedia Networks (WOWMOM)*, 2008.

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

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

[9] B.-R. Chen, K.-K. Muniswamy-Reddy, and 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* . New York, NY, USA: ACM Press, 2006, pp. 87–94.

[10] M. Dela Cruz, M. Whyte, Z. Yu, and T. Hanselmann, "Q learning routing protocol," Display demonstration of real hardware implementation at the Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Melbourne, Australia, 2007.

[11] Y. Wang, M. Martonosi, and L.-S. Peh, "A supervised learning approach for routing optimizations in wireless sensor networks," in *Proc. of the 2nd Int. Wkshp on Multi-hop ad hoc networks: from theory to reality (REALMAN)*. New York, NY, USA: ACM Press, 2006.

[12] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons From a Sensor Network Expedition," in *Proc. of the 1st Eur. Wrksp on Sensor Networks (EWSN)*, Berlin, Germany, Jan. 2004.

[13] T. Naumowicz, R. Freeman, A. Heil, M. Calsyn, E. Hellmich, A. Braendle, T. Guilford, and J. Schiller, "Autonomous monitoring of vulnerable habitats using a wireless sensor network," in *Proceedings of the 3rd Workshop on Real-World Wireless Sensor Networks*, 2008.

[14] K. Martinez, P. Padhy, A. Riddoch, R. Ong, and J. Hart, "Glacial Environment Monitoring using Sensor Networks," in *Proc. of the 1st Wrksh on Real-World Wireless Sensor Networks (REALWSN)*, Stockholm, Sweden, June 2005.

[15] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *Proc. of the the 4th ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2006, pp. 307–320.

[16] G. Halkes and K. Langendoen, "Crankshaft: An energy-efficient MAC-protocol for dense wireless sensor networks," in *Proc. of 4th Eur. Conf. on Wireless Sensor Networks (EWSN)*, Delft, The Netherlands, Jan. 2007.

[17] A. Hagedorn, D. Starobinski, and A. Trachtenberg, "Rateless deluge: Over-the-air programming of wireless sensor networks using random linear codes," in *Proc. of the 7th Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2008.

[18] J. Beutel J. "Metrics for Sensor Network Platforms, "in *Proc. ACM Workshop on Real-World Wireless Sensor Networks (REALWSN'06)*, June 2006.