A Feedback-Enhanced Learning Approach for Routing in WSN

Anna Egorova-Förster
University of Lugano, Switzerland
Email: anna.egorova.foerster@lu.unisi.ch

Amy Murphy
University of Lugano, Switzerland
Email: amy.murphy@unisi.ch

University of Lugano
Faculty of Informatics
Technical Report No. 2006/03
May 2006

Abstract

Much research in sensor networks focuses on optimizing traffic originating at multiple sources destined for a single, base station sink. Our work reverses this assumption, targeting scenarios where individual sensor data is sent to multiple destinations. In this case, the data path that produces the least network cost is unlikely to overlap completely with any of the optimal routes between the individual pairs of source/destination nodes. If the entire topology is known, an offline approach can likely find this minimum path. However this is an unrealistic assumption. Instead, our approach uses only local information and converges toward optimal. The novelty of our approach is a technique for actively exploring alternate data routes, sharing *feedback* regarding route fitness, and *learning* better routes. While non-optimal choices are made during the discovery phase, the resulting, learned path has lower cost than the initial path. Further, our protocol identifies multiple paths with equal cost, providing additional opportunities for saving energy by switching among alternate routes throughout the lifetime of the application. This paper describes our feedback-based protocol, shows simulation results demonstrating its benefits and explores the future opportunities of the learning technique presented.

I. INTRODUCTION

Technological advancements in small hardware and wireless networking have recently raised interest in studying wireless sensor networks. Much effort focuses on what the currently available technology can achieve and mainly concentrates on collecting vast amounts of sensed information at a single, powerful base station. While a variety of applications can exploit this setup, network algorithms supporting it have limited use when the data from the network can be exploited inside the network, as in the case of sensor actuator networks. In such scenarios, actuators are dispersed along side the sensors, collect data gathered by them, and take some action. The classic example is that of sprinkler actuators using information from moisture sensors. While it is possible to use a centralized approach, collecting information at a base station and resending it to the actuators, this unnecessarily increases both the amount of data transmitted in the network and the latency between data collection and actuation.

By taking this sensor/actuator scenario as our motivation, our work essentially reverses the typical sensor network assumptions and instead of sending multiple pieces of data to a single sink, we target sending a single piece of data to multiple sinks. Within this context, our goal is to minimize the overall cost to the network, thus increasing its lifetime.

Recent research efforts have resulted in a wealth of routing protocols to discover optimal paths between data sources and destinations, considering energy constraints, path length, etc. While these are quite effective in reaching a single destination, the minimal tree connecting a source and multiple sinks may not contain any of the minimal routes between the source and any single destination. The challenge we face therefore is to discover this minimal tree of routes without requiring global information about the network topology.

Our approach is an in-network protocol that incrementally *learns* about better paths in the network. To achieve this, we assume that sink nodes advertise their data needs with a simple broadcast protocol, and that basic routing information is collected during propagation of this broadcast message. This yields initial information about possible paths for data to the sink nodes, however, as noted previously, these may not be the best paths for reaching all sinks. To find these better paths, neighbors exchange information about the fitness of the selected route. Application of this fitness information allows incremental improvement to the routes, decreasing the overall network cost for the data to reach all sinks. During this process, each node actually learns multiple, equal cost routes. By cycling through these routes, we are able to increase the lifetime of the network, essentially by spreading out the load on the nodes involved in the routing process.

This paper continues in Section II with an overview of our routing protocol then Section III presents the details. Section IV provides a discussion and simulation results demonstrating the effectiveness of our approach. We then outline related work before Section VI concludes the paper with our plans for future work.

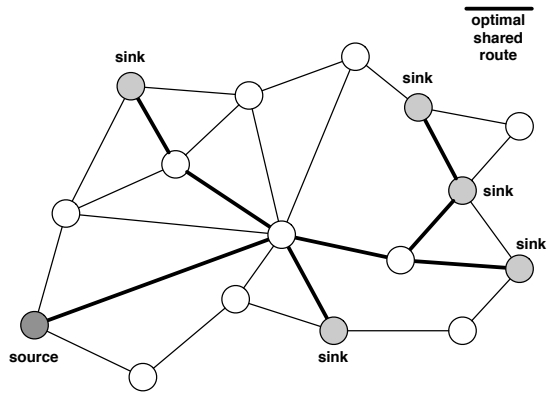


Fig. 1. A sample network with 5 sinks and one source. The lowest cost shared route is a tree, rooted at the source and spanning all sinks.

II. OVERVIEW

The goal of our protocol is to find the shortest possible path for data to follow from its source to all interested sinks. One possible path is formed by the union of the individual paths from the source to each sink, however a shorter path often exists. This shorter path takes the form of a tree, as shown in Figure 1. While the discovered tree may not be the theoretical optimum, it is likely to be an improvement over the individual paths. The challenge is to globally identify this tree without full topology information and without exchanging global information. The main task of our protocol is to update local information regarding "next-hops" to reach sinks from each node such that the resulting tree is as small as possible.

We accomplish this with a *three phase protocol*. The first establishes basic information at each node regarding the identity of the sinks and initial path information to each sink. The second phase sends data, explores routes, and learns the accurate shared costs of these routes. The third phase is steady-state, in which packets are routed along the best available routes without exploration. The sequence of phases is not strict, meaning that if a new sink requests data, or if a new, shorter path is introduced, the system will asynchronously return to the first and/or second phases.

The first phase requires each sink to broadcast a packet announcing its intent to receive data of a certain type. As this packet propagates, nodes receive information about the cost along available paths to the sink nodes.

The second phase begins when a data source starts sending packets. To decide which neighboring nodes to forward a packet to, each node constructs and consults what we refer to as a *Path Sharing Tree* (PST), explained in detail in Section III-B.2. Essentially the PST shows the estimated cost of sending the packet to all sinks through different next hops. For each data packet, the node selects a set of next hops to reach the sinks. By selecting a different next hop each time a new data packet arrives, the space of routes is explored. The node also calculates a *feedback* value indicating how good this node was as a choice for routing. This value is processed by the node that sent the data to the current node and is used to update the

TABLE I

DATA REQUEST AND DATA PACKET FORMATS.

DATA REQUEST	
sinkID $[int]$	A
time_stamp $[sec]$	78
coordinates $[x, y, r]$	$[100, 45, 8]$
expire_after $[sec]$	1000
TTL $[int]$	10
hops $[int]$	1

DATA PACKET	
Info:	
data	...
sourceID $[int]$	20
time_stamp $[sec]$	78
Feedback:	
forID	34
RLE $[int]$	12
Routing:	
neighborID	45
sinks$(array)$	(B, C)
max_cost	4
neighborID	12
sinks$[array]$	(A, D)
max_cost	9

PST, allowing nodes to *learn* the real costs of different paths to the sinks.

After some time, no new information is being learned and it is no longer worth while to explore potential paths. At this point, the nodes switch into the third phase, using only the best routes found during the exploration phase. To balance the costs among all the nodes in the network, a node uses all paths of equal cost, thus increasing the lifetime of the network.

It is worth noting that our routing protocol tolerates both node failure and loss of data and feedback packets. Also, while the protocol does not prevent routing loops during the exploration phase, it still ensures that the packet eventually reaches the destination, unless it is corrupted during transmission.

III. APPROACH

To explain the details of our approach, we step through the phases of the protocol outlined in the previous section.

A. Phase I - Sink announcement

The first phase of our protocol requires each sink node that desires to receive data of a particular type to broadcast its request to the entire network. The request message itself is fairly straightforward and it is outlined in Table I. The most important information it contains are the identifier of the sink node and the time to live (TTL) for the packet, indicating the limit of the dissemination of the request message. Also, the hops entry, initialized to 1 at the sink, increments each time the request is forwarded.

When a node receives a request message, it consults its routing table, shown in Figure 2 for the arbitrary node with identifier 20. If an entry already exists in the table for this sink/neighbor combination with a smaller number of hops traveled, the message is not processed. Otherwise, if no entry exists or the hops entry in the table is greater, the table is updated and the packet forwarded in broadcast with a decreased TTL and an increased hop count. This forwarding process creates the routing tables on all nodes, each containing several options for paths to each sink.

B. Phase II - Exploration

Phase II represents the core of our protocol. It is during this phase that the best possible route from the source to all sinks is discovered. The key points to our approach are the function that estimates the effectiveness of the route, the *fitness function*, the data structures used to make routing decisions for data packets, and the *exploration strategies* determining which routing decisions are taken.

Before getting into the details, it is important to note that all packets are sent in one-hop broadcast mode, not unicast, to all neighbors. Therefore, if a packet should be sent to two neighbors, e.g., to nodes 45 and 12, a single packet is sent with information inside specifying which nodes should process it. In Table I, this information appears in the *Routing* section of the data packet. Due to the natural broadcast nature of wireless communication, this clearly reduces the costs when sending the same data to two or more neighbors.

Phase II initiates when a data packet is received at a node and it must make a decision concerning how to route that packet. Each data packet carries information concerning which sinks it is expected to reach by going through this node. For example, in Table I, the data packet must reach sinks (B, C) by going through node 45 and sinks (A, D) through node 12.

1) Fitness Function (Route Length Estimate): Given that the goal of our protocol is to find the *best possible* route to all sink nodes, we must define precisely the metric for evaluating routes. By studying the environment, several properties emerge. For example, remaining battery power at the nodes can affect the quality of the path. If any node has low power, paths including it should be less favorable than alternatives. Similarly, if a path includes a low quality link or a link near the bandwidth limit, it should be less desirable. In general, the fitness function can be calculated based on multiple variables related to the appropriateness of the path for routing packets to a particular sink or set of sinks.

For this paper we use a simple, intuitive fitness function: *Route Length Estimate* (RLE). This metric is the total number of hops that a message is expected to travel to reach all sinks and generally corresponds to the overall energy requirements for routing the packet through the network. In the rest of the paper we assume this fitness function is used, however other metrics such as path energy and quality could be easily folded into our approach by changing the fitness function.

After Phase I, all nodes know locally the cost to send to a given sink through a given node. Based on this information, we can estimate at any node the cost to send a packet to a *set* of sink nodes through some of its neighbors. However, because the topology is not known, this cost is only an approximation, hence the word *estimate* in our fitness function metric. Using Figure 2 as an example, it is known that from node 20, neighbor node 45 can be used to reach sinks (A, B, C) but it is not known whether node 45 has an outgoing shared link to use for these packets, or if it must split the packets to three different neighbors to reach all of the sinks. Therefore, we estimate the worst case remaining cost to send a packet to sinks (A, B, C) through node 45 as the cost to send the packet one

Neighbor ID	Route to Sink ID	# Hops
45	A	1
	B	3
	C	2
54	B	2
12	A	6
	D	4

Fig. 2. Routing information maintained by sensor node 20 with path lengths to all known sinks obtained in Phase I.

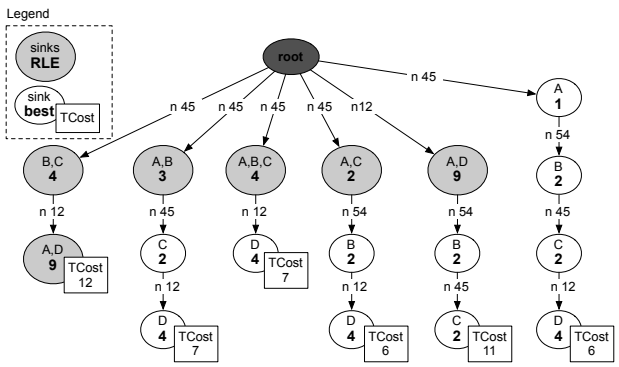


Fig. 3. The initial path sharing tree (PST) for the neighbors of node 20, constructed with information from Figure 2. A path from the root to a leaf represents a single routing option to reach all sinks from node 20 with the cost indicated on the leaf. Shaded nodes identify sharing of the next hop.

shared hop to node 45 and then to split the packet along three different paths after node 45 for the remainder of its route to the sinks. Therefore the cost is $1+(1-1)+(3-1)+(2-1) = 4$. This equation is generalized as:

$$RLE = \left(\sum_i cost_i\right) - (n-1) \quad (1)$$

where i is the set of sinks ((A, B, C) for this example), $cost_i$ is the path length to send a packet through the given neighbor (1, 3, and 2 above) and n is the number of sinks (3 above).

It is important to emphasize that this is the worst case estimate for the fitness of the path to sinks (A, B, C) through node 45. It is entirely possible that the packet can continue to share a path after node 45 for at least two of its sinks, in which case the actual cost will be lower. To *learn* a more accurate estimate of the cost, the estimates for the same routes on other nodes are conveyed back to the node where the original estimate was made, from nodes 45 and 12 to node 20 in this case. This is done as *feedback* during the routing process, as described in Section III-B.4.

2) Path Sharing Tree: When a node receives a data packet, it must ensure that the data will reach all of the sinks listed inside the data packet by delegating responsibility for routing the data to one or more of its neighbors. If a packet contains only a single sink destination, the routing decision can be taken only by looking at Figure 2, and selecting the entry with the shortest path to that sink.

However, if the data must reach several sinks, the decision is quite complex. Consider example node 20 in Figure 2. If a data packet must reach all four sinks we have at least the following distinct options: *(i)* 45 for (A, B, C), 12 for (D), *(ii)* 12 for (A, D), 45 for (B, C), or *(iii)* 45 for (A, C), 54 for (B), 12 for (D). This is not a complete enumeration of the options, but it gives the reader an impression of the complexity of how many options are available.

To manage this complexity, we devised a data structure called the *Path Sharing Tree* (PST) whose goal is to organize the available options for selecting the next hops for a data packet to take, making the path selection task manageable.

Figure 3 shows a sample PST, based on the routing information in Figure 2. In the PST, a single path from the root to any leaf indicates which nodes should be used to send data to which sinks. Two sinks in the same shaded oval indicate a shared link to the next hop. For example, the leftmost path of Figure 3 indicates that data should be destined to node 45 for sinks (B, C), and to node 12 for sinks (A, D). The leaf node shows the current route length estimate for this option is 12 according to Figure 2 and Equation 1.

The PST has two main parts, the top one contains only shared paths (shaded PST nodes), and the bottom one contains any sinks that do not occur in the shaded nodes above it (non-shaded PST nodes). For example, after the shaded node (A, B, C) on level one of the PST, there is no possibility to share any paths for the remaining sink (D). Therefore, the tree continues with a non-shaded node covering the remaining sink individually. The RLE for all non-shared nodes is the shortest path to that sink in Figure 2.

The tree is constructed in a brute force manner, with some optimization shown later. The root of the tree has no special meaning, it is simply a starting point for the enumeration of routing options. The first row includes all combinations of two or more sinks sharing the same next hop. The second row chooses among the remaining combinations of sinks that do not overlap with the previously covered sinks and do not use same next hop. Although not demonstrated in this simple PST, any node can have more than one child. The subsequent rows in the PST follow the same rules, containing combinations of sinks not already covered and using next hops not already used. After all combinations are exhausted, the remaining sinks are covered individually as previously described.

The total cost indicated on the leaf is the cost to use the path formed by the nodes from the root to this leaf node. It is analogous to Equation 1, namely the sum of the individual RLEs of all nodes on this path, minus the number of PST nodes on the path, minus one because the first hop is broadcast and therefore shared.

This total cost represents the estimated cost for sending a packet to all sinks, however it is possible that a packet arrives and must only be forwarded to a subset of the sinks. Fortunately, the information required to analyze the forwarding options is already contained in the PST. This is best described

by example: consider a packet that must be routed only to sinks (A, B, C). In this case, any shared paths that include the other paths, namely (D), should not be considered. To do this with the same PST, we eliminate from the first row any nodes that contain (D). Therefore we consider the first four nodes on the first level, ignoring (or rather, virtually pruning) the last shared node, (A, D). If a shaded node not containing (D) is encountered further down in the PST, the branch containing it is pruned. On the other hand, if a non-shaded (D) is encountered, it is simply ignored, not pruned. This is correct because the non-shaded nodes in the PST appear in no particular order, therefore we can consider (D) as the last node without affecting correctness. Finally, the total cost for the possible paths must be adjusted to ignore the (D) nodes. This is trivially accomplished by subtracting the cost of (D) from the total cost on the leaf node and adding 1 because one node has been removed from the PST path. Analogously, the same PST can be used to route to any subset of the known sinks by only adjusting the total cost relevant to the subset.

The size of the PST is related to the number of sink nodes and the number of neighbors with routes to those sinks. Given the assumption that the number of sink nodes is not large, the size of the tree is quite manageable. Further, although we have described this as a verbose tree data structure, the structure itself can be greatly compressed for use on resource constrained sensor devices, e.g., by using two bit vectors to represent the sinks and next hops, and storing the tree as a table, eliminating the storage of tree pointers. Additionally, the tree can be pruned to remove duplicate paths and to ignore paths with high estimated costs. These and other options are further explored in Section IV-A.

3) Managing Loops: While the PST presents several options for the routing of the packet to a set of sinks, it does not recognize if the selected route will contain loops. Therefore, we address this separately and deal it when it arises.

Usually, it is sufficient to know the number of hops that the previous node expected this node to use when routing, and to select a path with lower cost. For example, the node of the PST tree shown in Figure 3 received a request to route to sinks (A, B, C, D), this request also includes the maximum allowed cost for forwarding the packet, *max_cost*. If this value is 8, the node must eliminate the two possible routing options with cost higher than 7, the maximum allowed cost minus one because one hop was already taken to reach node 20. As long as the max_cost decreases with each hop, the data is guaranteed to eventually reach the sink.

In most cases, a node will be able to locate a path in its PST with an acceptable cost. If not, the packet is returned to the sender, who must choose a different path to reach the sink(s).

It is possible that a packet will travel in a loop while a large cost path is being explored during Phase II. However, paths containing loops will not be used after the exploration phase ends because if a path with a loop exists, the PST also contains a path without the loop. The cost without the loop must be smaller than the one with the loop, and therefore the loop-free path will be used after the exploration stage ends.

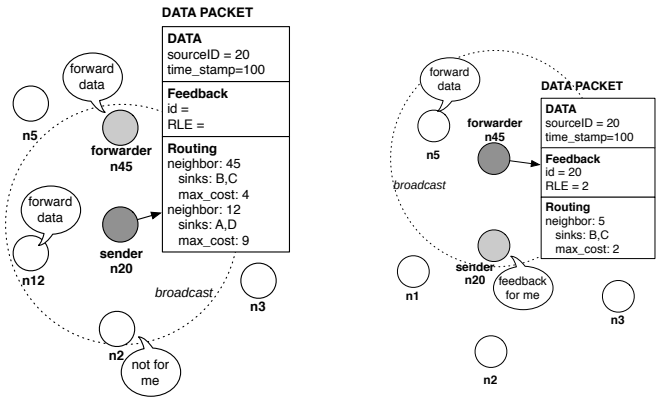


Fig. 4. A demonstration of the routing and feedback mechanism between two nodes. One hop broadcast is used to send and while every node receives the packet, it processes it only if its identifier appears in the routing section of the packet.

4) Feedback and Learning: While the PST is able to estimate the network cost to all sinks, its initial data is based only on information gathered during Phase I and is therefore the worst-case estimate of the possible sharing, assuming that the packets to different sinks will follow different paths after reaching the next hop. As this is clearly not always the case, it is necessary for any downstream nodes to provide updated route costs to the upstream nodes, providing more accurate route length estimates. This information is shared in the form of feedback, returned to the upstream node when the packet is re-broadcast by the next hop. The sending and receiving of the feedback information and the updating of the RLEs throughout the network is our actual *learning mechanism* that results in improved routing decisions over time.

Consider the example shown in the left of Figure 4 in which sender node 20 forwards a packet to nodes 45 and 12 for sinks (B, C) and (A, D) respectively. Note that the packet includes the *max_cost* mentioned in the previous section. When node 45 receives this packet, it must find a path for routing to sinks (B, C) with cost less than 4. It identifies another neighbor, 5, as an appropriate next hop for both sinks (B, C), and recognizes that the cost for this path, 2, is less than the maximum allowed by sender node 20. This implies that node 20 has too high an estimate for the route through node 45. Node 45 sets feedback information in the packet that it forwards, indicating that node 20 should update its route length estimate to 2, the current estimate of the path known at node 45. When node 20 receives this feedback, it updates the PST tree, setting the individual cost for using node 45 as the next hop for (B, C) to the feedback value and updating the total costs for paths using this routing option.

In general, feedback information updates information at the previous hop, however in order to find the *overall* best routes, this information must propagate throughout the network; our mechanism does precisely this. Observe that with the new path information, node 20 now has a better cost estimate that will affect its perception of the route fitness: it has *learned* a new RLE for this route. Therefore, the next time node 20 is

requested to send data along this same route, it will forward the updated cost estimate from its PST to its predecessor node. Thus, the information eventually propagates the full reverse path from the sinks to the data senders. This means that for a route to some sink with n hops, the same route must be used exactly $(n-1)$ times in order for all nodes on the path from the source to the sink to have the most accurate information.

5) Exploration Strategies: While the PST information shows several possible routes and combinations, it is the job of the routing protocol to decide which route to use. Clearly several options are available. We could simply choose a single path with the minimum cost. For example the PST of Figure 3 indicates that sharing node 45 for sinks (A, C) and using other nodes for (B) and (D) has a cost of 6, lower than all other options. However, it may be that this is a poor estimate, and one of the other options has lower actual cost.

Another option is to cycle among all options, trying them all, and receiving feedback from the other nodes to update the RLE. Yet a third option is to assign weights to the options. For example, assigning larger weights to paths with low cost. With these weights, we can then probabilistically select a path to use, selecting low cost path with higher probability. As with round-robin, nodes are expected to send feedback, updating route estimates. The expectation, however, is that paths with low initial estimates will also have low actual cost, therefore it is beneficial to select these paths to explore before trying a path with very high cost.

Other route exploration strategies are possible, with the choice depending on application requirements and network properties. The exploration strategy works together with the loop management and ensures that only *allowed* routes are chosen for exploration. A route is allowed as long as it is strictly less than the max_cost contained in the data packet.

Finally, every exploration strategy must define the duration of the exploration phase. Again, several options are available based on time, on the received feedback, on the number of explored routes, etc. In our implementation, the exploration phase ends after a given number of packets do not trigger an update to the PST. With a constant data rate, this is equivalent to a time-based approach, however it is more flexible for varying data rate applications.

C. Phase III - Stable Data Gathering

According to the selected exploration strategy, eventually the protocol will stop exploring new paths and will work with the information collected thus far. When this happens, the PST is likely to have several routes with the same minimal cost. These are considered the *best* routes available, and therefore should be used during the stable data processing phase to route all remaining data packets.

The fundamental choice, however, is whether to use one or more of these available routes. As the costs are equal, we observe that alternating among the available paths is likely to spread out the load of data forwarding among the available network nodes. Therefore Phase III is characterized by randomly selecting one of the best paths and sending the data packet along it.

It should be noted that if the PST changes for any reason, Phase II is re-initiated. Therefore, the protocol may exit the stable phase, but assuming the PST eventually stops changing, the system will spend most of its time in Phase III.

IV. DISCUSSION AND EVALUATION

In this section, we discuss the overall properties of the algorithm, then present our simulation environment and results.

Throughout this section, we examine the behavior of our protocol with two different exploration strategies: RANDOMEXPLORE and NOEXPLORE. RANDOMEXPLORE assigns probabilities to each leaf in the PST according to its total cost, then selects allowed routes for exploration with these probabilities. It stops exploring, when ten consecutive data packets receive no valuable feedback for the PST. NOEXPLORE uses only the best found routes in the PST after Phase I, meaning no exploration is ever performed.

As a baseline for comparison we use an implementation of Directed Diffusion's "one phase pull" protocol, DIRECTED-DIFFUSION, as described in [1], [2]. We selected this protocol as it is both similar to our approach and has demonstrated good experimental results.

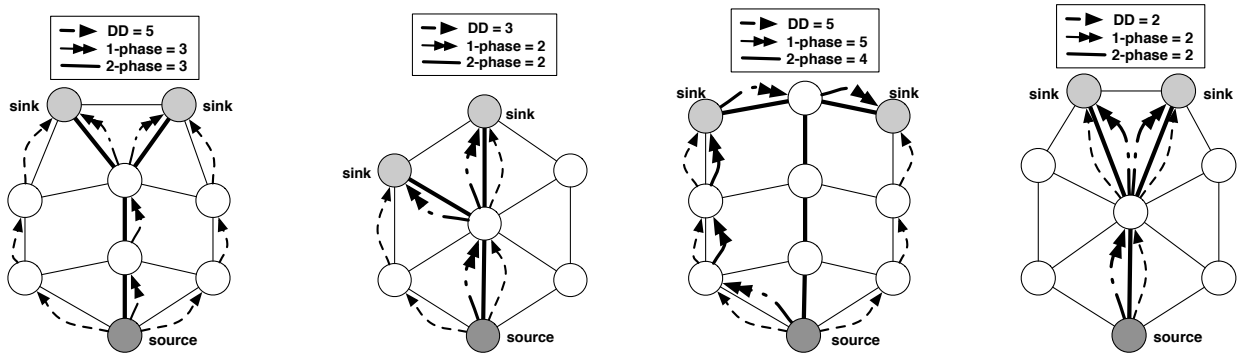
A. Discussion

The feedback-enhanced routing protocol presented thus far is based on a non-deterministic, probabilistic learning approach. As discussed in Section III-B, it guarantees data packets eventually reach the sinks in all phases, however, it does not guarantee to find the optimal route.

To always find the optimal route in the network, we would need to implement a blind search through all possible routes and guarantee that the feedback from all routes completely propagated throughout the network. These suffer from a single disadvantage: they require time. For this reason, we apply a standard machine learning technique: randomize and limit the exploration to all heuristically good routes. This is manifest in the assignment of probabilities to routes in the PST during exploration such that the paths with low current cost estimates are likely to be explored and thus to receive feedback.

The success of the exploration phase is dependent on the topology of the network. With some topologies, the optimal route to all sinks is the same as the individual routes to each sink separately, which is the information in the routing table created in Phase I. We refer to this routing option as *best single routes to sinks*. If these routes are optimal, the exploration phase will not find any better options. Nevertheless, in other cases, benefits in network cost will be found in the exploration phase. Figure 5 shows several small networks with similar topologies illustrating both scenarios where exploration can provide benefits and those where it will not.

Figure 5(a) shows that the network costs when using best single routes to sinks will be 5, considering one-hop broadcast for all packets outgoing from a given node. If we consider



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4

Fig. 5. Different network topologies exhibit different properties of the routing protocol. The gradients for DIRECTEDDIFFUSION and the feedback-enhanced route are given. The RLE values are for DIRECTEDDIFFUSION (DD), RANDOMEXPLORE (after stabilization), and NOEXPLORE.

shared routes, both RANDOMEXPLORE and NOEXPLORE result in the same route through the middle nodes in the network with the cost of 3, a savings of 40% over the best single routes.

Figure 5(b) shows a similar situation with 50% gain for both RANDOMEXPLORE and NOEXPLORE, however the figure emphasizes another property of the network: by *good luck* the best single paths established in Phase I may overlap with the best shared route in the network, as their costs are the same. In this case, the exploration cannot discover any better routes.

Figure 5(c) shows a topology where the exploration phase of RANDOMEXPLORE finds a better route than NOEXPLORE. At initialization, after Phase I, the shared route through the middle nodes in the network will have costs of 7: the known costs to both sinks is 4, thus the first RLE for this route is $(4 + 4) - 1 = 7$. The routes through the left and right nodes will also have initial RLEs of 7. Thus, the stable phase will choose one of these routes nondeterministically. If the middle one is chosen, the best route will be found by chance. If not, the network cost will stabilize to 5, which is the same as the costs for the best single routes, but not the overall optimal.

The topology in Figure 5(c) illustrates another property of our exploration mechanism: it is *not* guaranteed to find the optimal route. Because we choose randomly among the available routes during exploration, it may happen that the middle route is not explored before the end of the exploration phase. In this case, the optimal route will never be found. This depends on the exploration strategy as it defines the order and how often certain routes are explored, as well as the phase duration. Therefore, there is a small probability that RANDOMEXPLORE will stabilize to less optimal routes than NOEXPLORE optimization.

Figure 5(d) shows the final case where neither RANDOMEXPLORE nor NOEXPLORE can find a better route because the best single routes already use the optimal one. In a real network, with longer paths to the sinks, such a situation is unlikely to occur.

This analysis raises the question whether the exploration phase is needed since, as just demonstrated, it does not necessarily find better routes. The answer is yes for two reasons. First, *most of the time* we do find routes with better fitness through exploration (see the simulation results in the next section). Second, the exploration phase exhibits an auxiliary property, namely it finds many routes with the same minimum cost. This can be used for actively managing network lifetime by alternating among routes and sharing energy among forwarding nodes. We return to this property in the simulation results section.

a) Extreme scenarios: It is worth abstractly considering some extreme scenarios including both very few and very many sinks. With only one sink, the PST tree will not be built at all because there are no shared routes and the best route to this sink from the routing table will be directly used. In this extreme scenario of one sink, the results will be exactly the same as using DIRECTEDDIFFUSION.

With the other extreme, a large number of sinks, several properties in the routing protocol change. First, the exploration time increases because more shared routes are possible through a single neighbor. Second, the network cost savings also increase for the same reason. On the negative side, the size of the PST also increases in order to track all possible paths.

Even in the moderate cases with a small number of sinks, we must apply some heuristics to the PST tree to manage its size. As discussed in Section III-B.2, one option is to save only routes to some particular sink whose costs are near the minimal single paths costs. Another possibility is to keep the routes, but after building the tree, prune it only to some number of leafs with near-minimal costs. In this case, more memory is needed to keep the routes, but the PST is more flexible to changes in the network (new requests, expiration of old requests, changing connectivity, mobile nodes, etc.) and more efficient when identifying good shared routes as it constructs the whole tree before pruning it to the best routes found.

b) Multiple sources and data types: To this point we have only considered a single source in the network. However, when the number of sources changes, neither the protocol nor the properties of the exploration change. Although more packets will be routed through the network, the feedback relates to the sinks and not the sources, therefore it remains

unchanged. Moreover, the algorithm is likely to converge faster, as feedback is exchanged more frequently due to the higher data rate. Still, only one PST is needed per node and this PST holds all possible routes.

Also, when multiple data types (predicates over the sensor data) are requested, the PST remains the same because, as previously mentioned it contains only information relevant to the sinks, not the data. Nevertheless, building separate, smaller PSTs for each predicate may be considered as an optimization for managing the size of the PST.

c) Repair after failure: Our protocol is also tolerant to connection failures. If some neighbor is no longer reachable, all corresponding routes are deleted from the PST of the node that discovered the failure and the next best routes can be used. If the cost of the new route is larger than before, this information is passed as feedback to the neighbors, updating their PSTs, triggering a new exploration phase, and causing new routes to be explored and used.

In the case we have several routes with the same minimal costs, which is often the case, the network costs stay the same even after node failures. Such an option is not possible when only one next hop or gradient is kept for each sink. Thus, our approach immediately repairs the routes after failures without the need to resend requests.

d) Bandwidth requirements: Finally, we consider the bandwidth requirements of our feedback-enhanced protocol. We have already shown that the protocol does not increase network costs in the stable phase since feedback information is piggybacked on normal data packets. In the exploration phase, the network costs are higher because of the exploration of non-optimal routes, but these temporary increases are compensated for in the stable phase, as seen in the simulation results.

B. Simulation setup and results

Our simulation study was conducted using the OMNeT++ discrete event simulator [3], version 3.2 with the Mobility Framework extension[1]. Both our experimental results as well as the simulation code are available on our website[2].

The key component of our simulation is the implementation of our routing protocol. The remainder of the environment settings are options within the Mobility Framework such as a dummy application layer for receiving the data, a simple CSMA MAC layer, and a simple wireless medium simulation at the physical layer. This physical layer uses a single parameter for managing bit errors and medium interference. Details are available with the Mobility Framework documentation. Admittedly, this simulation environment is not completely satisfactory, mostly due to the overly simplified physical layer simulation and the selection of MAC protocols, both of which are non-typical for wireless sensor networks. Nevertheless, the simulation is more than sufficient to provide a meaningful first evaluation of our ideas. In the future, we plan not only to work with a more satisfactory simulation environment, but to also implement the protocol on real sensor nodes.

[1]http://mobility-fw.sourceforge.net/

[2]http://www.inf.unisi.ch/projects/mics

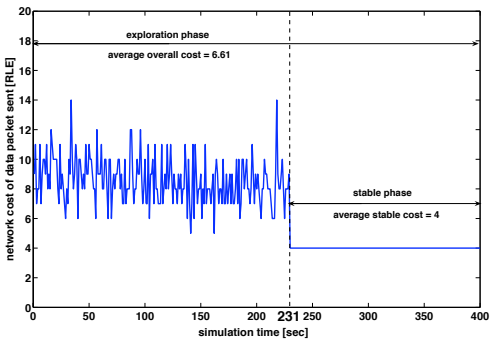


Fig. 6. The network routing costs for data from a single source to all sinks in a sample, random topology with 6 sinks. The two phases of RANDOMEXPLORE are shown: exploration until time 231 followed by stable.

As previously mentioned, our evaluation compares two versions of our feedback-enhanced protocols NOEXPLORE and RANDOMEXPLORE against an implementation of the "one phase pull" DIRECTEDDIFFUSION.

All of the experiments reported here consist of 500 runs over 125 different random topologies of 50 nodes. Each topology is guaranteed to be connected and is fixed at the beginning of the simulation. To simulate the lifetime of a node, we assign each node an initial energy quota and decrement this value only when sending a packet. We choose not to decrement this value on receiving a packet because receiving costs are likely to be uniform for all nodes in the network. Even though a node may not need to process a packet, it must be received to make this determination.

In this section we report results for overall network costs and network lifetime, showing the improvements of our protocols over DIRECTEDDIFFUSION. We also report the convergence time for the exploration phase of the protocol to give an idea for the amount of time the system is spending exploring non-optimal routes before stabilization.

Before showing any comparisons, we intuitively show the behavior of the protocol as it progresses through the exploration and stable phases. Figure 6 shows a single run for a single topology. Before stabilizing, the network costs are irregular because packets are sent through different routes with varying costs. Once the stabilization phase begins, only the best routes are used and the network cost is considered minimal.

Based on these observations, we define two metrics used for comparison: *overall cost* and *stable cost*. The first denotes the average network cost during the whole run (both phases), 6.61 in this case. The stable cost averages only the cost during the stable phase, 4 in this case. While clearly the additional costs during the exploration phase are important as they are part of the total energy expended by the system, this value becomes less relevant with long simulation runs. The second metric, on the other hand, shows clearly the gains that our approach can achieve once the small, shared routes are identified. Our comparison results show both values, but it is important to

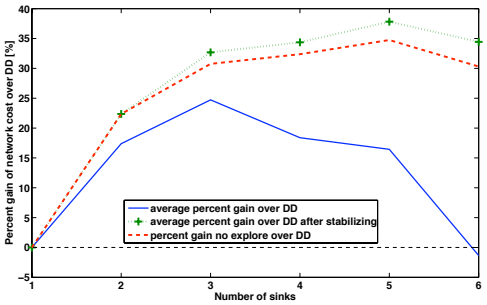


Fig. 7. Percent gain over DIRECTEDDIFFUSION in network costs for RANDOMEXPLORE after stabilizing, RANDOMEXPLORE total and NOEXPLORE.

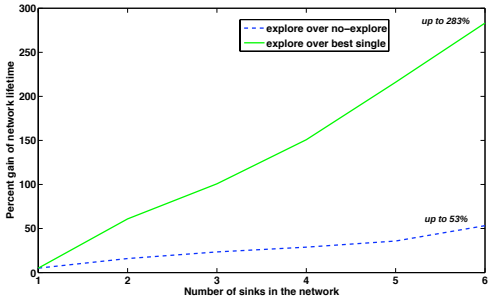


Fig. 8. Network lifetime of our protocols compared to DIRECTEDDIFFUSION.

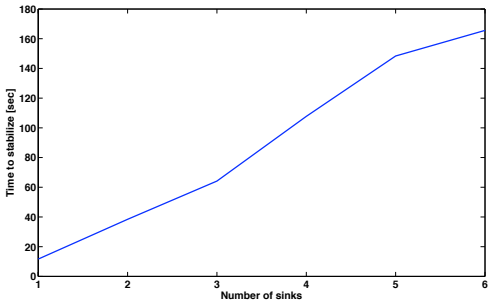


Fig. 9. Duration of the exploration phase for RANDOMEXPLORE.

realize that the first, overall cost, is highly dependent on the length of the simulation run.

Our initial hypothesis for this work is that by sharing routes to multiple sinks, lower overall network costs can be discovered, and that our feedback-based approach will be able to find such shared paths. This hypothesis is validated by Figure 7 where we report the improvements of our protocols compared to DIRECTEDDIFFUSION, reporting the percentage difference in performance on the same topology for each protocol. We use this metric, because it eliminates differences among random topologies, such as the length of the paths between source and sinks which can differ dramatically.

Overall, the stable cost of our protocol achieves significant gains, up to 35% for 5 sinks. This graph also shows that much of the gain comes simply by exploiting the PST to find shared routes but in most cases, additional gains are achieved by the exploration. However, when including the costs incurred during exploration, the overall cost, our gains are not as significant over DIRECTEDDIFFUSION. In fact, the last data point for the overall cost with 6 sinks is below that of DIRECTEDDIFFUSION. As previously explained, the overall cost value depends heavily on the length of the simulation run and over time these additional costs are amortized by the gains achieved with shorter routes discovered during exploration.

Another interesting observation from this plot is at the point for two sinks. In this case, the RANDOMEXPLORE is unlikely to find a better route than NOEXPLORE because of the limited number of alternate routes that exist and thus can be explored. As the number of sinks increases and the number of available routes correspondingly increases, the potential improvements due to exploration grow. Also the slight decrease in the trend at the last data point is most likely due to the random behavior of our protocol and noise in the network. We do not expect the downward trend to continue for more sinks.

The data shown in Figure 7 is the average gain of the compared protocols (e.g. NOEXPLORE and DIRECTEDDIFFUSION), which gives the percentage difference between the performance of the two protocols on the same random topology.

Next we consider the potential gains in network lifetime by exploiting multiple identical cost links. Again our hypothesis holds: namely that exploration finds more equal cost links and by alternating among these links the network lifetime increases, as measured by the time to the first node failure. Figure 8 shows that when comparing the stable phase after RANDOMEXPLORE against DIRECTEDDIFFUSION, our approach can achieve up to 283% longer lifetime than DIRECTEDDIFFUSION for the maximum number of sinks in the network. Compared to NOEXPLORE, our approach still achieves up to 53% improvement, demonstrating that indeed the exploration finds more route options and clearly demonstrating that our learning approach can achieve significant benefits.

Finally, because the overall cost is affected by the length of time that it takes the protocol to stabilize, it is worth showing the average time to stabilization for various numbers of sink nodes. Figure 9 shows that as the number of sinks increases, so does the time to stabilization as it takes more time to receive all the feedback. Nonetheless, considering that many real applications gather data for weeks or months, an exploration phase of a few hundred seconds is acceptable.

In addition to the plots shown here, we also ran our simulations with more dense topologies, increasing the connectivity among the nodes. Such an increase implies two trends: first, the routes to the sinks are shorter reducing the exploration phase, and second, there are more neighbors and the number of routes available to explore increases. As confirmed by simulations, these two changes compensate for one another, yielding no measurable difference for any experiments. Plots are not shown due to space considerations.

Additionally, we gathered information about the data delivery rates at the sinks, for setups with 1 to 6 sinks. We achieved

results varying from 97% to 99%, which we expected since the data rate in the network is relatively low (once per second). With increasing data rates, we expect it to decrease.

V. RELATED WORK

In this section, we compare our approach with the most relevant work from other areas, namely content based routing in sensor networks and the application of learning and feedback in the networking domain. This is not intended to be an exhaustive survey, but simply to put our work in context.

A. Content-Based Networking in WSN

One way to view our approach is as an implementation of content-based networking [4], a routing framework where data is sent from the source to the destinations based on interests expressed by the destinations to receive a particular pattern of data. Such an approach is relevant for sensor networks as it is data driven as opposed to address driven. This has been demonstrated in [5] where the authors use a distance vector protocol to construct a tree from the source node to all interested sinks. The tree uses the minimum path lengths for tree construction, however, unlike our approach it does not optimize the tree to consider path sharing.

Another instantiation of content-based networking for sensor networks is Directed diffusion [1], [2] where routes from the source to the destinations are established on-demand based on interests that are flooded through the network. This flooding establishes gradients for data to follow from multiple sources to the sinks. As the source sends low-rate data samples, the routes where data first arrives are reinforced by the sinks. In the extreme case of only one sink, our approach is actually an implementation of Directed diffusion, however it is intended for multi-sink scenarios, in particular identifying shared routes to these sinks. Our most significant deviation from the Directed diffusion work is our explicit exploration and learning mechanisms, allowing us to learn the parameters of the network with limited local exchange of information.

Additionally, while the Directed diffusion paradigm supports multiple sinks, the routes for all sink are established separately with no optimization for shared routes. While the option of using a multicast tree has been discussed in [2], this assumes the network is IP-based and the multicast tree is built with external tools.

To the best of our knowledge, no protocols exist explicitly for content-based networking to multiple sinks in sensor network. Related work comes, however, from the subject-based networking work in the mobile ad hoc (MANET) domain, specifically MAODV [6]. MAODV builds and maintains multicast trees for mobile environments, and while such an approach can be applied in the sensor network domain, the network properties and protocol requirements are significantly different. For example, MANETs imply dynamic, highly mobile environments with less stringent requirements on memory, bandwidth, and energy than sensor network.

B. Learning Approaches in Networking

The cornerstone of our approach is its ability to learn about better paths over time. Our approach is partially inspired by AntHocNet [7], an approach for learning routes in wireless ad hoc networks that combines reactive path setup with proactive path probing. The key idea derives from ant colonies, which are proven to converge to shortest paths when moving from food sources to the nest by exploiting pheromones. As applied to ad hoc networks, a route is established by sending individual ants through all neighbors to discover a path to the destination. All ants follow random paths, and when a path to the destination is found, the path is reinforced. With respect to our approach, the routing tables maintained by the ants are similar to the combination of our routing table and the PST. As feedback is received, their tables are updated in a similar manner to our PST. In our case, however, the overhead of sending ants through the whole network would be considered a waste of energy, therefore we piggyback information on actual data packets. Second, and more significant, our approach finds shorter routes to multiple sinks. This would imply the cloning of the ant to follow multiple paths, an option that would break the analogy and change the properties of AntHocNet.

A different form of learning, namely Q-learning, has been applied to gather data from multiple sources, compress it, and send it to a sink node [8]. If it is possible to aggregate data along its path, this is done to minimize energy. The success of the aggregation depends on how early separate data items meet each other on their way to the sink. The best routes from the sources to the sink with respect to aggregation are *learned*. Unfortunately the authors report the results only on a rectangular grid of sensor networks and assume global topology knowledge. Also, as compared to our solution, routes are learned from multiple sources to one sink while our approach learns routes from a single source to multiple sinks. In addition, our protocol sends feedback only one hop, while theirs propagates feedback throughout the entire path.

Another effort demonstrates the idea of using reinforcement learning to find shortest paths in mobile ad hoc networks [9]. A continuous learning approach is used to explore all possible routes from a source to a single sink, however the authors do not provide sufficient information regarding the communication protocol nor how data should be exchanged. While this work is one of the first to use Q-learning for routing, the lack of details make comparison difficult. Our work takes the next steps: using the learning paradigm, describing a routing protocol, and demonstrating its benefits.

VI. CONCLUSION AND FUTURE WORK

The concept of attaching routing feedback to data packets and learning is a powerful tool in the wireless domain, especially sensor networks, as it requires only limited local knowledge and achieves significant results. Furthermore, attaching feedback information to regular data packets does not increase network costs, an important issue in this domain.

In this paper we presented a new concept of piggy-backing feedback information on regular data packets exchanged in

wireless sensor networks for the purpose of incrementally learning the properties of the network. We applied this concept in a routing protocol that learns the best route from a single source to multiple sinks in the network and demonstrated the effectiveness of the protocol.

We discussed the concepts of a fitness function, which describes the fitness of some particular route over many possible parameters such as battery status, connection quality, length of the route, etc. Additionally different exploration strategies make it possible to adjust the routing protocol to the requirements and properties of the specific application and sensor network. Our simulation results show clearly, that even simple assumptions and learning techniques achieve very good results in terms of overall network cost and network lifetime as compared to state-of-the-art routing protocols.

In the future, we plan to further experiment with different fitness functions and exploration strategies and to implement the routing protocol on real sensor nodes, as this is the best mechanism to demonstrate its actual characteristics. We plan to expand the feedback concept and to propagate the feedback information more than the current one hop, with the goal of reducing the learning time. We also plan to expand the use of the feedback beyond the original sender, namely to any interested neighbors. Again, this will make the feedback mechanism faster and more efficient.

We will also explore the opportunities to exploit learning in different layers of the protocol stack. For example, we could easily learn the connection quality for asymmetric links, where one node can communicate to some other but the communication in the opposite direction is either poor or impossible. By giving some simple feedback about the local knowledge of links, every node can adjust its links. By integrating this into the MAC layer, the result is improved connections in the network and additional network lifetime.

Another domain, to which we plan to apply our concept, is non-uniform data dissemination. In some scenarios, applications require accurate data from the direct neighborhood of the sinks and less accurate from parts of the network further away. These scenarios include many sinks in the network and routing of data to all of them. Thus it is natural to expand our protocol to support different levels of detail of the data and to learn the best possible routes for sending the data to the sinks.

ACKNOWLEDGMENT

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.

REFERENCES

[1] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," in *Transactions on Networking*, 2003.

[2] F. Silva, J. Heidemann, R. Govindan, and D. Estrin, "Directed diffusion," in *Frontiers in Distributed Sensor Networks*, 2003, pp. 573–596.

[3] A. Varga, "The omnet++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference*. Prague, Czech Republic: SCS – European Publishing House, June 2001, pp. 319–324.

[4] A. Carzaniga and A. L. Wolf, "Content-based networking: A new communication infrastructure," in *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, ser. Lecture Notes in Computer Science, no. 2538. Scottsdale, Arizona: Springer-Verlag, Oct. 2001, pp. 59–68.

[5] C. P. Hall, A. Carzaniga, J. Rose, and A. L. Wolf, "A content-based networking protocol for sensor networks," Department of Computer Science, University of Colorado, Tech. Rep. CU-CS-979-04, Aug. 2004.

[6] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 1999, p. 90.

[7] G. Di Caro, F. Ducatelle, and L. Gambardella, "AntHocNet: an ant-based hybrid routing algorithm for mobile adhoc networks," in *Proceedings of the 8th International Conference on Parallel ProblemSolving from Nature (PPSN VIII)*, ser. Lecture Notes in Computer Science, X. e. a. Yao, Ed., vol. 3242. Birmingham, UK: Springer-Verlag, September 18-22 2004, pp. 461–470.

[8] P. Beyens, M. Peeters, K. Steenhaut, and A. Nowe, "Routing with compression in wireless sensor networks: a q-learning approach," in *Fifth European Workshop on Adaptive Agents and Multi-Agent Systems (AAMAS 05), Paris, France*, 2005.

[9] J. Boyan and M. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," January 1994.