MVSink: Incrementally Building In-Network Aggregation Trees

Leonardo L. Fernandes[1,2] and Amy L. Murphy[2]

[1] University of Trento
[2] Fondazione Bruno Kessler – IRST
Trento, Italy
{leiria,murphy}@fbk.eu

Abstract. In-network data aggregation is widely recognized as an acceptable means to reduce the amount of transmitted data without adversely affecting the quality of the results. To date, most aggregation protocols assume that data from localized regions is correlated, thus they tend to identify aggregation points within these regions. Our work, instead, targets systems where the data sources are largely independent, and over time, the sink requests different combinations of data sources. The combinations are essentially aggregation functions. This problem is significantly different from the localized one because the functions are initially known only by the sink, and the data sources to be combined may be located in any part of the network, not necessarily near one another. This paper describes MVSink, a protocol that lowers the network cost by incrementally pushing the aggregation function as close to the sources as possible, aggregating early the raw data. Our results show between 20% and 30% savings over a simplistic approach in large networks, and demonstrate that a data request needs to be active only for a reasonably short period of time to overcome the cost of identifying the aggregation tree.

1 Introduction

In-network data aggregation is rapidly becoming the accepted mechanism to reduce the amount of transmitted data in a wireless sensor network without significant loss of data quality [1]. This requires both the selection of the aggregation function, a highly application-dependent task, as well as the identification of the *best* node at which to apply the function. Most existing approaches assume that data from a localized region can be fused together, thus they focus on identifying one or more nodes in each region, rotating the aggregation function evaluation among them.

While many applications fit this scenario, we are motivated by a different class of applications in which the sensors are more heterogeneous, and neither the required data nor the aggregation function are known in advance. An aggregation function is a function that combines data from different sources. Aggregation functions can be as simple as an average of many readings or more sophisticated

functions that take data from different types of sensors and make some application specific decision, for example, combining data from smoke detectors and temperature sensors to infer if there is a fire. We identified this problem during our prior work on the MiLAN middleware [2], which, in summary, takes as input a large set of sensors, and over time selects different subsets that, when combined according to user-defined functions, meet a minimum quality constraint. This choice of the subset and its duration of use is made to maximize the total system lifetime. In MiLAN, we assume that the functions operating on data are applied at the sink, but in this work we recognize that moving the functions into the network reduces the amount of information that needs to be transmitted, thus increasing system lifetime.

Our overall goal, therefore, is to identify the nodes where the aggregation function should be applied, building a cost-effective aggregation tree. For this, we take an incremental, in-network approach, assuming that the sink node is initially the only node with knowledge of the function. The function is then incrementally *pushed* farther into the network until the *best* locations are found, forming an aggregation tree. The novelty presented in this paper is a set of heuristics that recognize when the function should be split, with distinct parts moving toward different sets of sources. The optimal aggregation solution is a Steiner tree that includes all the sources and the sink. Finding a Steiner tree in an arbitrary graph is known to be NP-hard [3] even for centralized algorithms. Therefore, our completely distributed approach aims to identify an approximation of the optimal Steiner tree using novel heuristics to incrementally identify better locations for the aggregate functions. Nevertheless, this approximation is an improvement over a solution that applies all functions at the sink or that applies a shortest paths tree (SPT), as detailed in our evaluation.

Section 2 describes MVSink, our novel protocol for incrementally moving an aggregation function from the sink closer to the data sources that provide its input. The section emphases the heuristics used for pushing the function to increasingly cost effective locations. Our results, presented in Section 3, indicate that our approach provides significant benefit at reasonable cost. Section 4 places this work in the context of related efforts while Section 5 ends the paper with brief concluding remarks.

2 MVSink

This section begins with brief descriptions of the system model and basic definitions upon which our description of MVSink relies. We then overview the protocol operation, then detail four heuristics we use for identifying the lowest cost aggregation tree.

2.1 System Model

We assume a standard network environment in which nodes are connected with bidirectional links. Any unidirectional links are pruned by a standard MAC

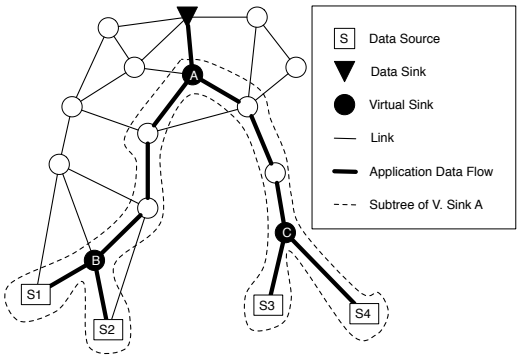


Fig. 1. Example of a network with nodes acting as sources, sink and virtual sinks.

protocol. We further assume that all nodes operate in promiscuous mode, overhearing transmissions by their one-hop neighbors, whether or not the packet is destined for them.

2.2 Basic Definitions

We begin with some basic definitions illustrated in Figure 1 and used throughout the paper. How these components combine in a coherent protocol is described next.

- *Aggregation Tree:* We represent the sensor network itself as an undirected, connected graph $G(V, E)$ with a single sink node and a set of data sources $S \subset V$. An aggregation tree is a connected subgraph of G containing the sink, the sources S and any other nodes and edges needed to connect these nodes without creating cycles. Our protocol seeks to minimize the size of this tree. The bold edges in Figure 1 identify the edges of the minimal aggregation tree for this network.
- *Virtual Sink:* A virtual sink is a node in the aggregation tree that receives data from two or more sources and applies an aggregation function, shrinking the amount of data forwarded to the sink. Figure 1 shows three virtual sinks, A, B, and C .
- *Candidate Node:* A candidate node is a *candidate* for serving as a virtual sink, meaning it has the potential to aggregate a subset of the sources currently aggregated by a virtual sink. All candidates are within k hops of a current virtual sink, where k is a tunable parameter.
- *Subtree:* The subtree of a virtual sink or candidate node n is the tree that contains n as a root and all the nodes and edges that compose the paths between each source n aggregates (or proposes to aggregate) and itself. Figure 1 highlights the subtree of virtual sink A.

2.3 Protocol Operation

In a nutshell, MVSink seeks to minimize the size, in number of edges, of the aggregation tree by incrementally moving the virtual sink from its initial location at the data sink to a location closer to the sources. If necessary, the virtual sink can be split into multiple pieces, creating some sinks that continue to migrate deeper in the aggregation tree, closer to the source, while leaving one virtual sink behind to combine the data from these deeper virtual sinks.

The protocol executes the following steps repeatedly, until no better aggregation tree can be found:

1. The virtual sink announces in broadcast to its k-hop neighbors its id and the set S of data sources it currently aggregates. k is a protocol parameter;
2. Nodes receiving such an announcement and that have overheard data flowing from two or more nodes in S identify themselves as virtual sink candidates as they can aggregate data from some set of sources. This candidacy is communicated to the current virtual sink along with the information about the subset of S that can be aggregated and the hopcount from the candidate to each of these nodes. Intuitively, we include individual hop counts as it allows the virtual sink to select a candidate based on the cost for any set of sources it can aggregate. Sections 2.5 through 2.6 provide further details.
3. After receiving the candidacy messages, the virtual sink decides, based on one of the heuristics we define later, which of the candidates should assume the role of virtual sink and start aggregating data. There are multiple options. For example, a single candidate can be chosen to aggregate all the sources in S. Alternately, the virtual sink can be split among multiple candidates, having each of them aggregate a subset of S. If no candidates propose a cost/effective solution or if there are no candidates, the protocol ends with the current virtual sink.
4. Any newly assigned virtual sinks start this process from the beginning.

The primary contribution of this paper is the proposal of four novel heuristics applicable in step 2 described above. The following sections outline each.

2.4 Largest Set Heuristic

Intuitively, the more data a virtual sink aggregates, the more effectively it can reduce the amount of data flowing in the network. Therefore, our first, and most straightforward heuristic, simply identifies the candidate node with the largest set of proposed sources. If multiple candidates can aggregate the same number of sources, the choice is arbitrary or the best local gain heuristic, discussed next, can be applied as a tie-break.

After the first candidate is selected and the sources it can cover are removed from S, S may not be empty. The suitability of the remaining candidates is evaluated to cover these sources, identifying additional virtual sinks. This first heuristic employs very small candidacy messages, containing only the set of sources proposed to be aggregated by the candidate.

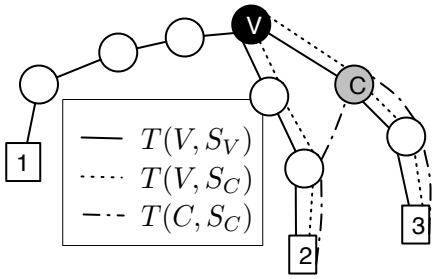


Fig. 2. Representation of the terms in (1). The set of sources for virtual sink V is $S_V = \{1, 2, 3\}$ and for candidate C is $S_C = \{2, 3\}$.

2.5 Best Local Gain Heuristic

Given that our goal is to reduce the size of the aggregation tree, our next heuristic selects candidate virtual sinks based on the size of the subtree they yield. For example, in Figure 2, the candidate node C proposes to aggregate sources $S_C = \{2, 3\}$ with a subtree size of 4. As the cost for the current virtual sink to aggregate this set of sources is 6, this represents an improvement in the overall size of the aggregation tree of 1, as the link between V and C must be considered. Formally, if $T(X, S_Y)$ is the number of edges in the subtree that connects a node X to the sources node Y aggregates, S_Y, where $X = Y$ is allowed, then the cost if candidate C is selected by virtual sink V is given by:

$$Cost(C) = T(V, S_V) - T(V, S_C) + T(C, S_c) + d(V, C) \tag{1}$$

where $d(V, C)$ is the distance from V to C, in this case 1. Figure 2 shows the T costs. This formula is applied to each candidate, and the one with the lowest $Cost(C)$ is chosen. In case of a tie, the node farthest from the virtual sink (i.e., with the largest $d(V, C)$) is selected, with the goal of moving the aggregation as close to the sources as possible in a single step.

If the selected candidate does not aggregate all the sources aggregated by V, the remaining sinks and candidates are considered, always selecting the highest *Cost* value until either there are no more sources, or there are no more suitable candidates.

This heuristic can also be used as a tiebreak strategy for other heuristics.

2.6 Affinity Based Heuristics

We developed two affinity based heuristics in order to avoid the local minima trees that are occasionally found in the previous heuristics. Such local minima occur due to the lack of global information available at the virtual sink. A virtual sink, with access to information from its neighborhood only, cannot identify a

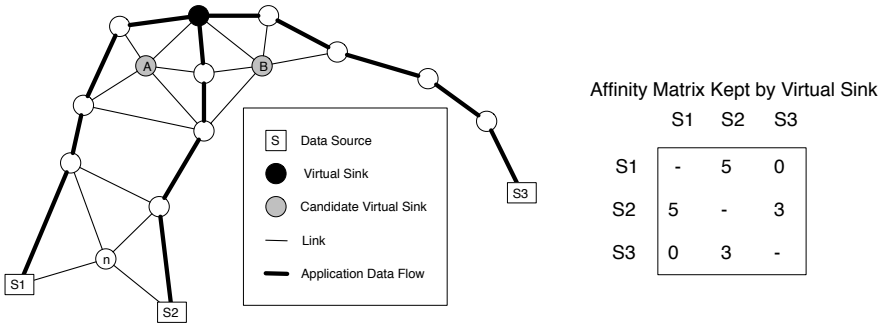


Fig. 3. Example use of affinities. With only local (1-hop neighborhood) information, the virtual sink cannot decide whether it is better to select candidate A or candidate B as the next virtual sink. Candidate A proposes to aggregate data from $S1$ and $S2$, whereas candidate B can aggregate sources $S2$ and $S3$. In the previous heuristics, candidates A and B are equivalent choices, but the figure clearly shows that choosing candidate B would result in a dead end, since there would be no further suitable candidates in the network to aggregate $S2$ and $S3$, while candidate A is a much better choice, since the protocol would then be able to proceed until reaching node n as a virtual sink.

candidates potential to make progress with subsequent virtual sinks selection closer to the data sources.

Figure 3 shows the motivation for the next two heuristics. In the picture, there are two virtual sink candidates. Candidate A can aggregate sources $\{S1, S2\}$, and candidate B proposes to aggregate $\{S2, S3\}$. From the figure, it is clear that sources $S1$ and $S2$ are more closely related than $S2$ and $S3$, making candidate A the best choice. However, relying only on local (1-hop) information, the virtual sink cannot identify the best option. We refer to this relationship between $S1$ and $S2$ as *affinity*, and develop a heuristic to identify it.

We define the affinity between two given sources as a numeric value kept by every node i, reflecting approximately the distance from node i to a potential aggregation point. In the figure, node n is a potential aggregation point for $\{S1, S2\}$ and is farther from the virtual sink than node B, which is the farthest possible aggregation point for $\{S2, S3\}$. Therefore, the affinity perceived by the virtual sink is higher for $\{S1, S2\}$ than it is for $\{S2, S3\}$.

Affinity information is introduced in the network by nodes that overhear messages from two or more sources, a situation which indicates their ability to serve as a virtual sink. Unfortunately, if such a node is not transmitting data, this affinity information will not reach the current virtual sink. However, because the node has already overheard application messages, a single, one-hop broadcast of this affinity information is enough to reach a node that *is* transmitting data. This broadcast, referred to as an affinity message, carries affinity information that is then forwarded with application messages toward the sink.

Affinity information is kept and transmitted as an $n \times n$ matrix, where n is the number of sources. If all sources are not known in advance, a data structure containing only the information about known sources can be used. To save memory and reduce packet size, a triangular matrix can be used instead of a full matrix, since the values in the upper and lower parts of the matrix are equivalent (e. g. $M_{a,b} = M_{b,a}$).

Each node keeps the affinity values of all pairs of sources it knows, initially all affinity values are zero. When a node overhears messages from two sources with affinity zero between them, it sets the affinity to 1 and sends its affinity matrix to its neighbors. The neighboring nodes update their local matrices, which are transmitted with subsequent application messages. When a node receives an affinity matrix $A = (a_{i,j})_{n \times n}$ in an application message, the local matrix $B = (b_{i,j})_{n \times n}$ updates all of its entries as follows:

$$b_{i,j} = \begin{cases} \max(b_{i,j}, a_{i,j} + 1), & \text{if } (a_{i,j} > 0) \\ b_{i,j}, & \text{otherwise.} \end{cases} \tag{2}$$

Since application messages are always forwarded through the current aggregation tree, virtual sinks receive updated affinity information. Since the positive values are incremented at each hop of the application messages, the potential candidates that are farther from the virtual sink receive higher priority, as desired.

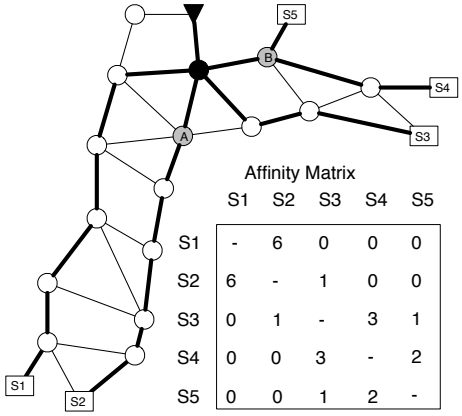
The broadcast of affinity messages by nodes that are not already transmitting data represents an extra cost to MVSink. Although affinity messages are limited to a single one-hop broadcast per node, the number of such nodes can be large, increasing the total cost of the protocol. This cost is discussed in the evaluation section, where the benefits it provides clearly outweigh the cost.

Next, we briefly describe two heuristics that rely on affinity information to choose between virtual sink candidates.

Best Affinity Candidate Heuristic This heuristic simply selects the candidate that proposes to aggregate the set of sources with largest affinity. When candidates propose to aggregate more than two sources, the affinity value of the set is the average of the affinity values of each individual pair of sources in the set. When affinity values are equal, the best local gain heuristic can be used as a tiebreak.

Best Affinity Set Heuristic This heuristic is similar to the best affinity candidate heuristic, but it tries to redistribute the sources among the candidates in a way that increases the total affinity in the sets rather than simply choosing between the sets proposed by the candidates.

This strategy is motivated by situations in which candidates propose to aggregate a large set S which effective "steal" sources that have more affinity with sources that are not part of S. For example, in Figure 4(a) a virtual sink aggregates five sources $S1$ through $S5$. $S1$ and $S2$ have affinity 6. $S3$ and $S4$ also have a good affinity value of 3. Now suppose there are two candidates A and B



Affinity Matrix

	S1	S2	S3	S4	S5
S1	-	6	0	0	0
S2	6	-	1	0	0
S3	0	1	-	3	1
S4	0	0	3	-	2
S5	0	0	1	2	-

(a) Example virtual sink and its affinity matrix.

Pair	Set of Sets	Feasibility
$S1, S2$	$\{\{S1, S2\}\}$	$A\{S1, S2\}$
$S3, S4$	$\{\{S1, S2\},$ $\{S3, S4\}\}$	$A\{S2, S1\}$ $B\{S3, S4\}$
$S3, S2$	{{S1,S2,S3,S4}}	Not feasible, undo.
$S3, S5$	$\{\{S1, S2\},$ $\{S3, S4, S5\}\}$	$A\{S2, S1\}$ $B\{S3, S4, S5\}$

(b) Best affinity set heuristic execution. Pairs added in decreasing order of affinity and the resulting subsets according to algorithm 1.

Fig. 4. Motivating example for the best affinity set heuristic. Note that according to the protocol there would be more candidates in this example, but for simplicity we consider only candidates A and B.

proposing to be virtual sinks. Node A proposes to aggregate $S1$, $S2$ and $S3$ and node B can aggregate $S3$, $S4$ and $S5$. The previous heuristic would let node A aggregate its three sources, even though there is little affinity between $S3$ and the other sources in A's set. A better solution would be to let A aggregate $S1$ and $S2$, and node B aggregate $S3$, $S4$ and $S5$, to take advantage of both good affinity pairs.

Instead of considering the candidate sets, the virtual sink, using its local affinity matrix, incrementally builds the best affinity sets and tries to find candidates that can aggregate them.

Algorithm 1, executed on each virtual sink, outlines the heuristic. Essentially, the algorithm adds each pair of sources, in decreasing order of affinity, to a set of sets S to be assigned to virtual sink candidates. Adding a pair of sources to S means that both sources should be in the same set of S, since there is good affinity. The operation can result in adding both sources to a set in S or in the merging of two sets of S, as detailed in algorithm 2. Figure 4 shows an example execution, with the pairs of sources added and the resulting set of sets generated at each step. Sets are built in order to maximize the total affinity between all sets. After the addition of each pair, the algorithm tests if there are candidates that can cover S, adding only pairs that produce sets feasible to be distributed among the available candidates.

The `feasible` procedure returns a boolean value of true if there are candidates available that can cover the sets in `doneSources`.

Algorithm 1 The best affinity set heuristic

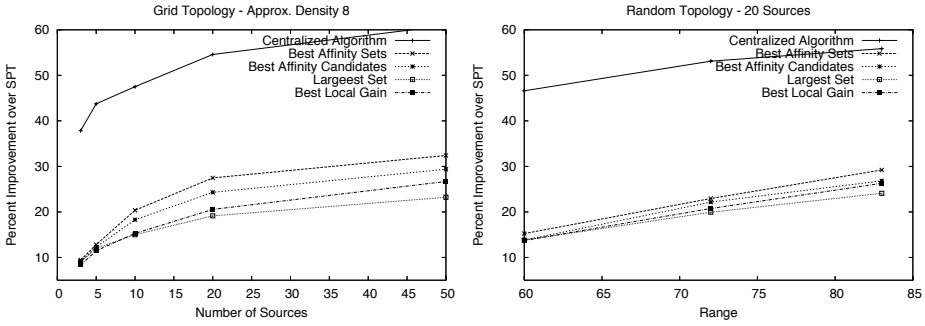
```
Set mySources = aggregatedSources
SetOfSets doneSources = ∅
Matrix aff = affinityMatrix
while |mySources| > 1 and hasNextHigher(aff) do
  pair (a,b) = getNextHigher(aff)
  doneSources = addPair(a,b,doneSources)
  remove(a,b,mySources)
  if !feasible(doneSources) then
    undo last addPair and remove calls
  end if
end while
Distribute sets to resp. candidates
```

Algorithm 2 The addPair function.

```
Require: setOfSets and sources a and b as parameters
  if setOfSets contains both sources a and b then
    merge sets of a and b in setOfSets
  else if setOfSets contains source a then
    add b to the set that contains a
  else if setOfSets contains source b then
    add a to the set that contains b
  else {setOfSets does not contain a or b}
    add new set {a, b} to setOfSets
  end if
  return setOfSets
```

3 Evaluation

Intuitively the movement of the virtual sinks closer to the sources lowers network cost, and thus has the potential to improve system lifetime. This section supports this intuition with an evaluation through simulation showing both costs and benefits. Two settings were used for simulations. Random deployment of sensors and distribution of sensors in a grid topology. The main difference between the two topologies is the uniformity of grids. In random deployments there are often connectivity holes and antiholes (subgraphs in which every vertex is adjacent to every other) [4]. The presence of holes may result in local minima for virtual sink placement. Antiholes can increase the cost of protocol operation due to broadcasting in high density areas. With grids, node connectivity is uniform throughout the network, avoiding both holes and antiholes. In both topologies, data sources are randomly chosen among the nodes and data sinks are placed at a corner of the simulated area. In the random topology, the deployment area is a 1000×1000 square. In the grid topology, 1000 nodes are arranged in a 40×25 grid with nodes uniformly distributed. In all simulations, we use $k = 2$ for the virtual sink announcements. We assume a perfect aggregation function



(a) Variable number of sources (b) Variable range

Fig. 5. Percent improvement of the heuristics over the shortest path tree.

(e.g. average), meaning that a packet of aggregated data has the same size as a raw data packet.

We vary the communication range, the number of data sources and the number of nodes in our simulations in order to evaluate the performance of MVSink in different network sizes and densities. We compare MVSink's different heuristics: largest set, best local gain (BLG), best affinity candidate and best affinity set; against shortest paths trees (SPT) and against a centralized algorithm.

The shortest paths tree, or opportunistic aggregation tree, is formed by each source sending application messages to the sink along a shortest path between the two. Overlapping shortest paths are combined to form the aggregation tree.

The centralized approach is based on the nearest participant first algorithm by Takahashi and Matsuyama [5]. The heuristic starts with the tree containing only the sink. In each step, the closest source to the current tree is connected. This process is repeated until all sources are connected. The output of this heuristic is an approximation to the optimal Steiner tree containing all the sources and the sink. The performance ratio of the algorithm is 2, in the worst case. Although the centralized approach is significantly better than MVSink, it requires global information which is not readily available and is very costly to obtain.

Each graph represents an average of 250 simulation runs. Error bars show standard deviation. The large values for standard deviation are due to results being largely dependent on the placement of data sources in the network. Networks with most sources close to the sink or to each other present results significantly different from cases with most sources far away from the source and sparsely distributed.

Since the main focus of this work is on the algorithm behavior, we used Sinalgo, a simulator for network algorithms [6], abstracting away from low-level network concerns. We assume reliable, constant delay, bidirectional channels, and while we acknowledge that limitations exist in real deployments, they will not affect the fundamental correctness of MVSink. We have also run simulations with up to 10% message loss (except for the sink transferring messages, which

must be reliably delivered) and obtained results only slightly worse than those presented here.

3.1 Improvement Over Shortest Paths Tree

Since SPT is a common way of performing aggregation in sensor networks, in this section we compare the percent improvement over SPT of the different heuristics and the centralized approach. The comparison allows us to clearly identify the performance differences, in terms of tree size, between the four proposed heuristics. The percent improvement is equivalent to the reduction in the tree size of each heuristic compared to SPT.

Figure 5(a) shows this comparison in a grid with a density of 8 neighbors per node, except at the edges. As expected, the best affinity sets approach is the best of our heuristics, due to its characteristic of selecting better aggregation sets at each step and its capability to collect information from nodes closer to the sources. Best affinity sets trees were close to 9% smaller than SPT with 3 sources and approximately 30% better for the case with 50 sources. The best local gain heuristic obtained better results than the largest sets approach, however it is still inferior to both affinity based heuristics.

In Figure 5(b) a similar graph is shown for a random topology and variable communication range yielding densities between 10 and 20 neighbors per node. Here we observe the improvement of the protocols as density grows. In this scenario, the affinity based heuristics are superior to the other two.

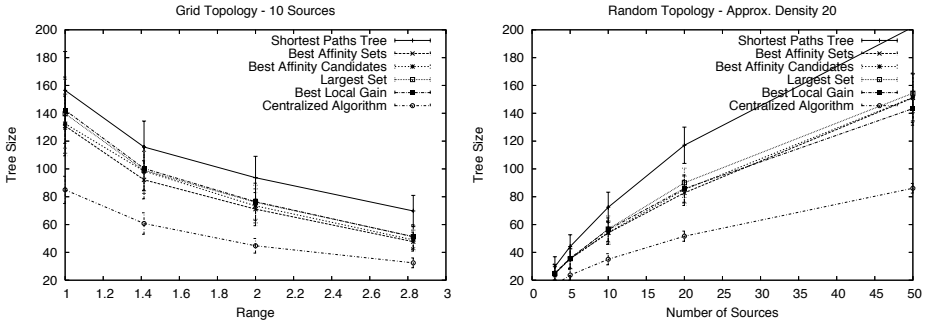
Our results also lead us to conclude that the protocol works better in the grid topology than in the random one. The results for the grid with density 8 and 20 sources in Figure 5(a) are better than for the random topology with 20 sources and density 10, the leftmost points in Figure 5(b). This difference was expected due to the regularity of grids.

3.2 Reducing Transmission Cost

The aggregation tree size is equivalent to the number of transmissions it takes to transmit data from all of the sources to the sink. In this sense, minimizing the tree size is crucial for energy savings and increasing network lifetime.

The graph in Figure 6(a) shows the comparison between our heuristics, the centralized approach, and SPT. There is a clear tendency for MVSink to approach the centralized algorithm as communication range grows. This improvement is due to the virtual sink announcements reaching more nodes in higher density scenarios. There are also more possible candidates, since more nodes can overhear application messages in dense networks. The larger groups of candidates to choose from allows MVSink to make better decisions and to get closer to the quality of the centralized algorithm.

Figure 6(b) demonstrates the consistent advantage of MVSink over the opportunistic aggregation scheme for different numbers of sources. MVSink tree sizes are significantly better than SPT in all cases. The protocol scales well for all



(a) Variable range

(b) Variable number of sources

Fig. 6. Tree sizes of MVSink heuristics, shortest paths tree and centralized approach.

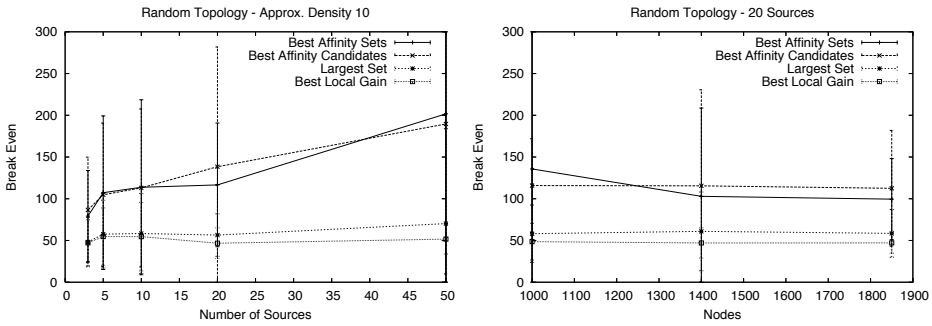
ranges of sources simulated. Figure 6(b) shows, for example, that for runs with fifty data sources MVSink on average achieves a tree size with approximately 140 edges, while SPT in the same scenario builds trees with size over 200.

We note a clear advantage of the affinity based heuristics over both the BLG and the largest set heuristics. As expected, the extra information collected provides the affinity oriented virtual sinks the possibility to make better decisions. But the extra information imposes higher costs. The affinity messages transmission cost is significant and must be taken into consideration. In the next section we discuss the costs of each heuristic and provide a cost/benefit analysis.

3.3 Overhead Evaluation

We measure the cost of MVSink as the number of transmissions it takes to: perform 2-hop broadcasts of all virtual sink announcements, send all unicast candidacy and virtual sink assignment messages, and, in the case of the affinity based heuristics, send all affinity messages. The cost is shown as the number of times that the final aggregation tree must be used until the message savings (compared to SPT) compensate for the costs of MVSink. We call this value the break even point. For this analysis we set the cost of MVSink messages equal to that of application messages. In many systems, application messages would be larger than control messages and in such cases, the gains of using MVSink would pay off even earlier.

In Figure 7(a), we can see that the break even is quite stable as the number of sources increases. The affinity based heuristics get more expensive with more sources, but still obtain a quite good average break even value of less than 200 for the considerable number of 50 data sources. For the BLG and largest sets heuristics, the situation is even better, since the break even does not grow as the number of sources increases. The small number of extra virtual sink announcements, assignment and candidacy messages it takes those heuristics to deal with larger numbers of sources (i.e. larger trees) are completely compensated by the extra aggregation gain of smaller trees discussed in Sections 3.1 and 3.2.



(a) Variable number of sources (b) Variable number of nodes

Fig. 7. Protocol Overhead

Figure 7(b) shows the break even as a function of the network density. To achieve increasing density, we simulated runs with different numbers of nodes in the same area. The picture shows that as density grows, the break even costs remain stable for all heuristics. In higher density scenarios, an increasing number of messages is necessary to run the protocol. Denser 2-hop broadcast announcements and the consequently larger number of candidacy messages, for instance, affect the costs significantly. The number of affinity messages is also heavily influenced by density, since more potential virtual sinks are likely to be found. Despite all the extra cost required to run the protocol on higher density networks, the aggregation gains improve enough to keep the break even costs reasonably stable for all heuristics.

In both break even graphs it is clear that the affinity based heuristics are more expensive than the other two, due to the need for affinity messages. These results, combined with the observation from the previous sections that the affinity based heuristics provide better trees, indicate that the affinity heuristics are more suitable for longer lived queries, i.e. that use the tree for longer periods, while BLG and largest sets heuristics are more appropriate for building trees that will be used a smaller number of times.

4 Related Work

Aggregation has been extensively studied in the literature, and in general has been shown to provide significant performance gains in a wide range of scenarios [1].

Some aggregation oriented protocols require additional information to be known in advance or transmitted throughout the network for a large cost. Greedy Incremental Tree (GIT) [7, 1], for example, incrementally builds an aggregation tree by first selecting the shortest path connecting the sink to the closest source and then connecting other sources to the tree, one at a time, at the closest point between of the current tree. However, to do so GIT requires that "exploratory

samples have initially and repeatedly been flooded throughout the network" [7]. GIT uses the large amount of global information this makes available, but the flooding is costly, even prohibitively so on large networks. Another approach, by Kansal et. al. [8] requires node location information, which is often unavailable in sensor networks. The work by Ramachandran et.al. [9] requires specific data fusion roles to be known in advance and needs some topological information for achieving a good initial placement of such fusion points. Our protocol does not require any additional knowledge above what is provided by a simple broadcast sink announcement to establish routes, such as that provided in standard versions of directed diffusion [10] and many other protocols.

Other proposed solutions [11–13] build approximations to a minimum spanning tree (MST) to be used as an aggregation tree. Minimum spanning trees are good solutions for the case in which all nodes are sources, or when using the same structure to deal with any arbitrary query. MVSink, instead, deals with a different problem: approximating the optimal tree for a specific set of sources on a per-query basis. The optimal solution in the case of MVSink is a Steiner tree containing the sink and all the sources. Also, it is often the case in sensor networks that the radio transmission radius is not tunable, meaning that essentially all transmissions have the same cost (i.e. all edges have the same weight). In such cases, any arbitrary spanning tree is an MST. Therefore, we claim that an arbitrary spanning tree is not necessarily a good solution.

Other protocols provide aggregation in hierarchical topologies [14–16]. Nodes are divided in clusters with special nodes, cluster heads, that aggregate the data from all nodes in the cluster. Such solutions rely on a hierarchy from which generating an aggregation tree is straightforward. Often cluster based approaches also rely on the capability of cluster heads to transmit data directly to the sink [14, 17], which is very costly, especially on large networks. Gao et. al. [18] propose a protocol for sparse data aggregation that forms an aggregation forest, each tree being connected to a node in the boundary of the network. The approach assumes nodes in the boundaries of the network have special capabilities to communicate directly with the data sink. Fan et. al. propose structure-free data aggregation [19], based on data-aware anycast transmissions (e.g. receivers with data to aggregate have priority) and randomized waiting to achieve aggregation. The approach is only suited for applications with high time and spatial convergence, such as event based applications and does not provide aggregation for sparse data on either time or space. Also, the approach assumes the use of geographic information, which is often not available on sensor networks. Our approach does not make any topological assumptions. It does not rely on special nodes or on one-hop communication to the sink and is thus applicable in any random, connected topology. MVSink takes advantage of spatial convergence and can also find suitable solutions for reasonably sparse data sources.

Many works address aspects such as quality of service [20], security [21] or load balancing [8] and assume the a priori existence of a proper aggregation tree over which to apply their techniques. The focus of MVSink is on the more basic

problem of finding a good tree. Therefore such techniques can be applied after MVSink generates the aggregation tree.

One of the main concerns regarding the use of aggregation is the latency it adds to the network. But, as shown by Zhu et al. [20], in extreme cases where there is too much traffic in the network, the use of aggregation can actually reduce latency in the network. It is important to notice that these are also the cases in which aggregation is most useful.

5 Conclusion

The work presented in this paper, the MVSink protocol, finds aggregation trees for applying a single aggregation function to an arbitrary number of sources in a wireless sensor network. While useful for a wide orange of applications and aggregation functions, another scenario requires multiple, unique functions. We trivially solve this problem by running multiple instances of MVSink, one for each function. For example, to apply $F(G(S_1, S_2), H(S_3, S_4)$, where S_1 through S_4 are data sources and F, G and H are different aggregation functions, initially two instances of the protocol are started to solve G and H. Then, a third instance is started to solve F, considering the nodes that apply G and H as data sources.

In general, MVSink is unique in its approach to incrementally move aggregation points from the sink towards the sources, making it applicable in sink-driven data collection scenarios. This paper focused on four novel heuristics for virtual sink selection, and our evaluation showed the benefits of finding low-cost aggregation trees clearly outweighs the overhead of the protocol for moderately long-lived aggregation scenarios.

References

1. Krishnamachari, B., Estrin, D., Wicker, S.B.: The impact of data aggregation in wireless sensor networks. In: Proc. of the 22nd Int. Conf. on Distributed Computing Systems (ICDCS), Washington, DC, USA, IEEE Computer Society (2002) 575–578
2. Heinzelman, W., Murphy, A.L., Carvalho, H., Perillo, M.: Middleware to support sensor network applications. IEEE Network Magazine Special Issue (2004)
3. Gröpl, C., Hougardy, S., Nierhoff, T., Prömel, H.J.: Lower bounds for approximation algorithms for the steiner tree problem. In: WG '01: Proceedings of the 27th International Workshop on Graph-Theoretic Concepts in Computer Science, London, UK, Springer-Verlag (2001) 217–228
4. Nikolopoulos, S.D., Palios, L.: Hole and antihole detection in graphs. In: SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (2004) 850–859
5. Takahashi, H., Matsuyama, A.: An approximate solution for the steiner problem in graphs. Math. Japonica **24** (1980) 573–577
6. at ETH-Zurich, D.C.G.: Sinalgo - simulator for network algorithms. (http://dcg.ethz.ch/projects/sinalgo/)
7. Intanagonwiwat, C., Estrin, D., Govindan, R., Heidemann, J.: Impact of network density on data aggregation in wireless sensor networks (2001)

8. Kansal, A., Srivastava, M.B.: An environmental energy harvesting framework for sensor networks. In: Proc. of the Int. Symp. on Low power Electronics and Design (ISLPED), New York, NY, USA, ACM Press (2003) 481–486
9. Ramachandran, U., Kumar, R., Wolenetz, M., Cooper, B., Agarwalla, B., Shin, J., Hutto, P., Paul, A.: Dynamic data fusion for future sensor networks. ACM Trans. Sen. Netw. **2** (2006) 404–443
10. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: Proc. of the 6th Int. Conf. on Mobile computing and Networking (MobiCom), New York, NY, USA, ACM Press (2000) 56–67
11. Ding, M., Cheng, X., Xue, G.: Aggregation tree construction in sensor networks. Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th **4** (2003) 2168–2172 Vol.4
12. Khan, M., Pandurangan, G., Vullikanti, A.: Distributed algorithms for constructing approximate minimum spanning trees in wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems (2008)
13. Cheng, H., Liu, Q., Jia, X.: Heuristic algorithms for real-time data aggregation in wireless sensor networks. In: IWCMC '06: Proceedings of the 2006 international conference on Wireless communications and mobile computing, New York, NY, USA, ACM (2006) 1123–1128
14. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: An application-specific protocol architecture for wireless microsensor networks. IEEE Trans. on Wireless Communications **1** (2002) 660–670
15. Manjeshwar, A., Agrawal, D.P.: TEEN: A routing protocol for enhanced efficiency in wireless sensor networks. In: Proc. of the 15th Int. Parallel & Distributed Processing Symp. (IPDPS), Washington, DC, USA, IEEE Computer Society (2001) 189
16. Wen, Y.F., Lin, F.Y.S.: Energy-efficient data aggregation routing and duty-cycle scheduling in cluster-based sensor networks. Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE (2007) 95–99
17. Lindsey, S., Raghavendra, C.: Pegasis: Power-efficient gathering in sensor information systems. Aerospace Conference Proceedings, 2002. IEEE **3** (2002) 3–1125–3–1130 vol.3
18. Gao, J., Guibas, L., Milosavljevic, N., Hershberger, J.: Sparse data aggregation in sensor networks. In: IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks, New York, NY, USA, ACM (2007) 430–439
19. Fan, K.W., Liu, S., Sinha, P.: Structure-free data aggregation in sensor networks. Mobile Computing, IEEE Transactions on **6** (2007) 929–942
20. Zhu, J., Papavassiliou, S., Yang, J.: Adaptive localized qos-constrained data aggregation and processing in distributed sensor networks. IEEE Trans. on Parallel and Distributed Systems **17** (2006) 923–933
21. Chan, H., Perrig, A., Song, D.: Secure hierarchical in-network aggregation in sensor networks. In: Proc. of the 13th ACM Conf. on Computer and Communications security (CCS), New York, NY, USA, ACM Press (2006) 278–287