MVSINK: Incrementally Improving In-Network Aggregation

Leonardo L. Fernandes[1,2]
[1]University of Trento, Italy
leiria@itc.it

Amy L. Murphy[2]
[2]FBK-IRST, Italy
murphy@itc.it

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 significant savings over a simplistic approach, 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 best aggregation point.*

1 Introduction

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. 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. We discovered this problem during our prior work on the MILAN middleware [5] 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 functions should be applied. For this, we take an incremental, in-network approach, assuming that the sink node is initially the only node with knowledge of the functions. These functions are then incrementally *pushed* farther into the network until the *best* location is found. Because we do not perform a global, offline maximization, we recognize that this may not be the absolute best location, but rather it is likely to be a local minimum. Nevertheless, it is an improvement over a solution that applies all functions at the sink.

Section 2 describes MVSINK, our novel protocol for incrementally moving a single function from the sink, closer to two data sources that provide its input. As such, it is a restriction of more general multiple source, multiple function problems, which instead, represent the future direction of our work, outlined in Section 4. Nevertheless, our initial results, presented in Section 3, indicate that our approach provides significant benefit at reasonable cost. Section 5 places this work in the context of related efforts while Section 6 ends the paper with brief concluding remarks.

2 MVSINK

To clearly illustrate the problem we face, consider the simple example in Figure 1 showing a small network with two sources (A and B) and one sink. Initially, all data from A and B is sent to the sink, where an aggregation function is applied. However, the system would be more efficient if the aggregation were performed at the intermediate node

1-4244-1455-5/07/$25.00

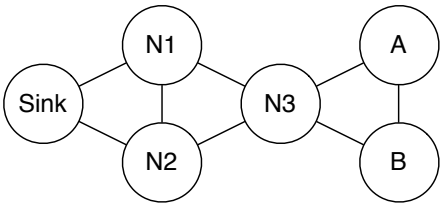


Figure 1. A simple scenario with one sink and two sources, A and B.

$N3$, and only the aggregated data sent from $N3$ to the sink. It is the job of MVSINK to identify $N3$, starting only with the knowledge at the sink of the aggregation function and the identities of the source nodes.

Our protocol assumes that initially the sink node applies the aggregation function. It then incrementally moves the function application role through the network, each time reducing the total transmission cost calculated as the paths from the two sources to the aggregation point, and from the aggregation point to the sink. Because the aggregation point acts as an intermediate sink node, we refer to it as the *virtual sink* and name our protocol MVSINK, because it incrementally moves the virtual sink. This section provides the details of MVSINK, starting with a description of the assumptions, then outlining the various components of the protocol.

2.1 System Model

Our work assumes a set of *connected* sensor nodes, where all pairwise links between nodes are bidirectional. Although in a real system unidirectional links may exist, we rely on a MAC protocol to filter out messages arriving on unidirectional links. We further assume that all nodes operate in promiscuous mode, overhearing all transmissions by their one-hop neighbors, whether or not the packet is destined for them.

2.2 Protocol Operation

The protocol initiates when the sink node knows which data sources are required and the aggregation function. To establish routes from the sources to the sink, a sink announcement message (`SinkAnn`) is broadcast from the sink to all nodes in the network. Initially the sink doubles as the virtual sink, performing the aggregation on arriving data. The remainder of the protocol, as outlined in Figure 2, describes the active role taken by the virtual sink to identify candidate virtual sinks that result in lower transmission cost, and inform the best candidate that it should become the next virtual sink.

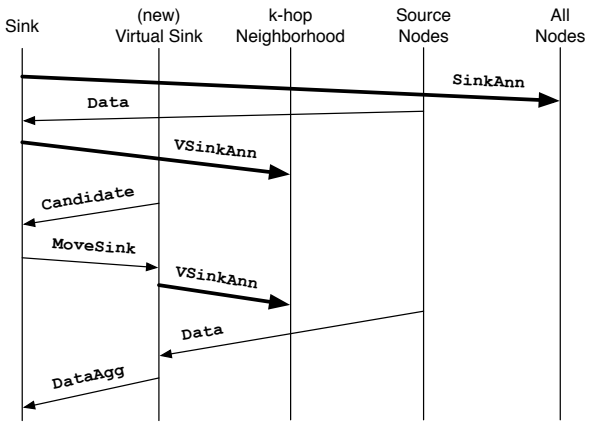


Figure 2. Messages exchanged during MVSINK to select the first virtual sink. Nodes are represented symbolically by their role w.r.t. the sink, then the (new) Virtual Sink after it is selected. Broadcast messages are represented by darker lines than unicast.

As our goal is to incrementally move the virtual sink closer to its best position, we must determine the region from which candidates are drawn. In an ideal situation we could consider the whole network, identifying the node that minimizes the data propagation cost, however collecting sufficient information to make such a global decision is costly. Therefore, we restrict the region from which candidates are drawn to a k-hop neighborhood around the current virtual sink, where k is a configurable parameter. Our experimental analysis showed that $k = 2$ provides a reasonable trade-off between the cost to identify the candidate and the number of incremental steps before the protocol stabilizes.

To find candidates, the current virtual sink sends a broadcast message (`VSinkAnn`) both identifying itself as the current node to which data should be sent and requesting nodes to identify themselves if they can serve as a *better* virtual sink. A node *can* serve as a virtual sink only if it has the possibility to merge data from both sinks. To evaluate this, a node listens to all neighborhood communication, and remembers the identities of the original data sources of all packets. In the event that a node overhears messages originating from *both* sources, even if it is not on the path to the sink, it *could* insert itself in that path to serve as a virtual sink. For example, in Figure 1, $N3$ may forward all packets through $N1$, but because $N2$ listens to all communication, it hears data from both A and B. Therefore, either $N1$ or $N2$ can serve as the next virtual sink.

A node evaluates its cost to serve as the virtual sink

by estimating the size of the routing tree connecting the sources, itself, and the sink. The size is known because all data packets from the sources contain their hop count and the original sink announcement provides the number of hops to the sink. These path lengths are returned to the current virtual sink via unicast (`Candidate`). In our example, the tree with $N1$ as the virtual sink has a cost of $2 + 2 + 1$, where 2 is the cost from both A and B to $N1$, and 1 is the cost from $N1$ to the sink.

The current virtual sink, after requesting the candidates, waits a predetermined time limit, collects all candidate responses, and selects one with the lowest cost. If multiple candidates have the same cost, the one with the larger distance from the sink is selected. A unicast message (`MoveSink`) is then sent to this node, informing it to take on the role of virtual sink, completing one incremental virtual sink movement.

Upon receipt of the `MoveSink` message, the newly selected virtual sink immediately starts the process to find a better sink. At this point the protocol considers two possibilities, one that simply re-starts candidate selection with the k-hop broadcast and the other that short circuits the k-hop broadcast based on locally known information.

In the first case, the virtual sink announcement (`VSinkAnn`), has two effects. As indicated above, it triggers nodes to send candidate responses, but it also informs nodes to forward data packets toward the new virtual sink, instead of along the routes either toward the previous virtual sink or toward the sink. This allows the virtual sink to actually apply the aggregation function. In our example, if $N2$ is chosen, its `VSinkAnn` message informs $N3$ to direct all packets to it, instead of $N1$.

In the second case, a node already knows that one of its neighbors is a better virtual sink than itself, therefore the k-hop can be short-circuited, and its cost saved. Consider the situation in Figure 1. If $k = 1$, in the first round, $N1$ is selected as the next virtual sink. While $N1$ could continue by broadcasting the candidate request message, it actually knows locally that $N3$ *must* have a lower cost than its own. This is based on the observation that messages from both sources A and B arrive at $N1$ via one single node, namely $N3$. Therefore, it is clear that the network cost is reduced by performing aggregation at $N3$. The k-hop broadcast is unnecessary. Although the example in Figure 1 is simple, this situation arises frequently in sparse networks, and results in huge savings with respect to the basic protocol that broadcasts `VSinkAnn` in each iteration.

When a node does not receive any candidate response messages within the timeout, the protocol terminates, and the current node remains the virtual sink until the data request is terminated.

It is worth noting that the first `SinkAnn` may provide a node with multiple, equivalent routes to the sink. Nevertheless, MVSINK requires that a single route be selected and used for all packets. If we loosen this restriction, it is likely that a node will send some packets along a path that will intersect with the virtual sink, and others will follow independent paths to the sink without aggregation. While this improves the load balancing, it actually increases the cost in terms of the number of hops taken by the messages. Since our goal is to identify the best aggregation point, we accept this trade-off. Nevertheless, if this is a significant concern for an application, an additional phase can be added to MVSINK, such that if a node has been acting as such for an extended period of time, it broadcasts a message toward the sources to establish multiple routes toward itself. The nodes can then alternate among these routes, as long as the virtual sink does not move.

Finally, we note that throughout the operation of the protocol to find the best virtual sink, data is flowing to the final destination; albeit at higher cost than after the best sink is identified. As just noted, prior to aggregation, data follows only one of the routes set up by the initial sink announcement. After aggregation and within the k-hop neighborhood, however, data can follow *any* path to the sink, meaning we no longer restrict data forwarding to a single outgoing link. Therefore, post-aggregation data is tagged as such (`DataAgg`) and can be forwarded to any neighbor closer to the sink than the current node.

3 Evaluation

Intuitively the movement of the virtual sink 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. Unless otherwise noted, all simulations were run with nodes in a 1000×1000 area, a communication radius of 100 and $k = 2$. As a baseline, we consider a protocol that performs no in-network aggregation, and simply uses the independent routes from both sources to the sink. We also consider a variant of MVSINK in which only the short-circuit sink movement is applied, without exploiting the k-hop broadcast of the `VSinkAnn`.

We consider two primary ways to stress MVSINK. First we increase the size of the network while keeping the density even. Then we fix the network size, and increase the density. Our experiments show improvement in network cost over the base case, and more importantly, can be used to show the so-called *break even point*, a measure of the number of times the aggregation point must be used before the overhead energy required to find it is recovered. Since when the protocol stabilizes every packet is aggregated, the number of aggregations is the same as the number of packets sent by each source. Our analysis assumes all packets have the same fixed size, and thus treats control packets the

same as data packets. In most applications, data packets are likely to be much larger, therefore weighting messages by size would actually improve our results.

Since the main focus of this work is on the algorithm behavior, we chose Sinalgo, a simulator for network algorithms [1], 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.

Each data plot in the graphs represents the average results from 50 random topologies. In each topology, the sink is located in a corner of the region in which the sensors are deployed. As our target application domain is one in which a sink node collects data, we believe it is reasonable to assume this node will be placed at the edge, rather than at the center of the sensed region. Admittedly, this choice positively biases our results, as the placement of the sink in the corner provides more opportunities to move the aggregation point with respect to situations with the sink in the center. Sources are randomly placed.

3.1 Increasing Network Size

To evaluate the scalability of MVSINK, we considered scenarios ranging from 200 to 1000 nodes, keeping the network density approximately constant. We accomplish this by tuning the communication range from 135 to 75.

The most fundamental measure for performance is the tree size along which data is transmitted, or the sum of the path lengths from both sources to the virtual sink and from the virtual sink to the sink. Figure 3(a) compares the tree sizes resulting from the three protocol variations. In general MVSINK achieves significant improvements in tree size, and with the selected density, approximately half of the benefit arises from short-circuiting and the other half as a result of the k-hop `VSinkAnn` broadcast and resulting candidate messages.

This improvement comes at the cost summarized in Figure 3(b). As expected, the broadcast `VSinkAnn` messages form a significant part of the overhead, but they are approximately equal to the candidate messages. This is because almost all nodes reached by the broadcast can potentially serve as new virtual sinks. Part of our future work is to reduce the number of candidate messages, either by aggregating them as they flow toward the virtual sink, or eliminating the transmission of some candidates if better candidates are overheard before the message is transmitted. Also, it is worth noting that the total overhead does not significantly increase as the network size increases. This is because the overhead is tied to the number of iterations through the protocol, or the number of times the virtual sink moves. As Figure 3(a) shows, the size of the tree does not dramatically increase with the number of nodes, and we further note that the branch of the tree between the sink and the virtual sink is likely to be the shortest branch. Taken in combination, these explain the only moderate increases in overhead.

Finally, Figure 3(c) shows the break even point of MVSINK. Again, this value does not vary significantly with the number of nodes, most likely for the same reasons that the overhead does not significantly increase. Further, it is likely that the sources will send more than 40 data values, thus the overhead of the protocol is justified.

3.2 Increasing Density

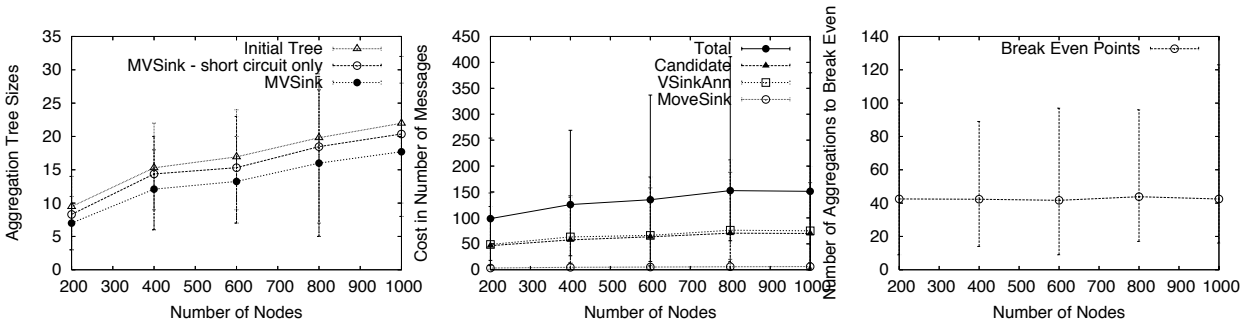
Experiments in the previous section selected a high network density of approximately 20 neighbors. This is motivated by our desire to show the difference between MVSINK and the short-circuit-only approach. Sparse network topologies tend to be characterized by many small "islands" of connectivity with a few "bridge" nodes. Short-circuiting clearly takes advantage of such cases, while in more dense networks, there are more routing options, and short-circuiting is less applicable.

Therefore, we turn our analysis to study multiple densities. Our simulations fix the number of nodes at 1000, and vary the communication radius to achieve network densities from 6 to 80 neighbors.

As expected, Figure 4(a) shows smaller differences in tree sizes between MVSINK and short-circuit-only for low densities, and larger differences with high densities, indicating that the k-hop broadcast has little benefit in low density settings. As we increase density, the short-circuit-only approach tends to achieve results close to the Initial Tree, while MVSINK keeps a reasonable level of transmission savings.

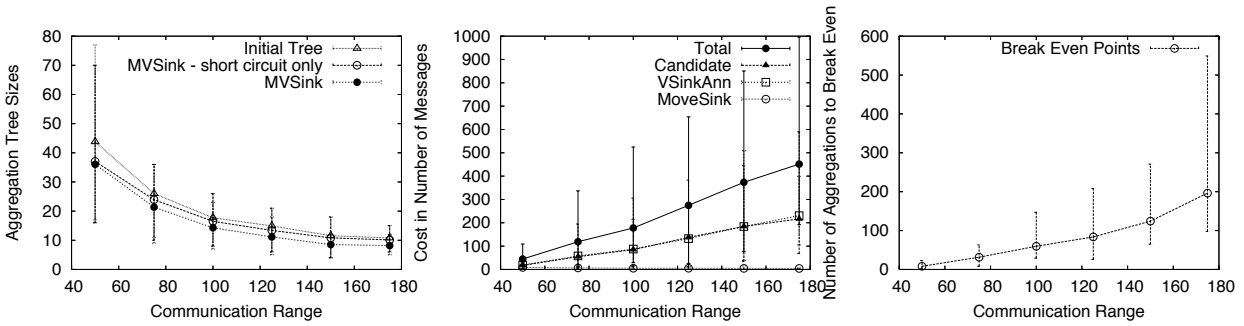
Figure 4(b) again reports the protocol costs, showing a significant increase with density. This is because as the network becomes more dense, both the number of nodes receiving and re-broadcasting the `VSinkAnn` increases and more nodes overhear data messages from the source and candidate themselves in response to the `VSinkAnn`.

Despite these growing costs, Figure 4(c) shows that MVSINK still provides reasonable break-even points. On average, we can recover the investment in MVSINK in less than 200 aggregations even for very dense networks. Further, if we weight data messages more than control messages, this value would reduce even more. Comparison of this and the corresponding graph of Figure 3(c) shows that density plays a significant role in determining the *break even point*, while tree size does not.



(a) Tree size (b) Overhead (c) Break even

Figure 3. Varying network size, constant density (approximately 20 neighbors).



(a) Tree size (b) Overhead (c) Break even

Figure 4. Varying network density (6 to 70 neighbors), constant network size (1000 nodes).

4 Extensions

The results in the previous section indicate that MVSINK has reasonable performance and yields significant improvement. Nevertheless, our general application target are systems with more than two sources and more than one aggregation function. To address this, MVSINK requires several extensions, and extensive simulation. This section presents our initial thoughts.

For systems involving one function applied to more than two sources, MVSINK needs to be able to identify different subsets of sources, generating one or more virtual sinks to aggregate such subsets. For example, assume that a virtual sink is currently aggregating data from four different sources A, B, C and D. If this virtual sink receives messages from two candidates: one offering to aggregate A and B, and the other one willing to be a virtual sink to C and D, the two candidates would both be new virtual sinks, one for each subset of sources. Of course it is very likely that such non-disjoint subsets arise, and a proper algorithm to find the best virtual sink splitting strategy needs to be developed.

The problem becomes even more complex if the function itself can be split into pieces, for example, from an aggregation function $f(A, B, C)$ into an equivalent definition such as $g(h(A, B), C)$ or $g(A, h(B, C))$. The decision about how to best split the function to best place the virtual sinks is quite complex.

To consider such multiple-function scenarios, we face the challenge to identify multiple virtual sinks, and possibly even virtual sinks that combine data after a first round of aggregation has been performed. For example, consider a scenario with four sources, and three functions, combined as follows $f(g(A, B), h(C, D))$. We could conceive of a solution with three virtual sinks, one for each of f, g, and h, or alternately a single virtual sink that applies all three functions.

To address this, we intend to extend candidacy messages to include multiple options for combining various sinks, and allow the current virtual sink to select. In other words, the virtual sink must choose whether to simply move the virtual sink, create two virtual sinks, or assign itself part of the role of virtual sink. Our initial investigations indicate that *bet-*

ter solutions can be found with this information, but further study is required to get such solutions as close to optimal as possible.

5 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 [8].

Some aggregation oriented protocols require additional information such as node location [7], or require that extensive information is sent from aggregating nodes all the way to the sources [2]. Our protocol does not require any additional knowledge above what is provided by a simple broadcast sink announcement for establishing routes, such as that provided in simple versions of directed diffusion [6].

Other approaches address aspects such as quality of service [10], security [3] or load balancing [7] and assume the existence of a proper aggregation tree over which to apply their technique. 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. Other protocols provide aggregation in hierarchical topologies [4, 9], but they rely on an established hierarchy from which generating an aggregation tree is straightforward. Our approach does not make any topological assumptions, and is thus applicable in any random, connected topology.

Another approach [7] is similar to the short-circuiting component of our approach, however, as our evaluation shows, the full MVSINK outperforms short-circuit-only in terms of tree size in all studied scenarios, with only moderate additional cost.

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. [10], 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.

6 Conclusion

This paper presents the initial description and evaluation of MVSINK, a new protocol for identifying aggregation points in a wireless sensor network. It is unique in its approach to incrementally move this point from the sink, making it applicable in scenarios with sink-driven data collection that changes over time. Our analysis clearly showed the low break even point for overcoming the overhead of the protocol, thus motivating our continued work to extend the protocol to address additional numbers of sources and aggregation functions.

References

[1] D. C. G. at ETH-Zurich. Sinalgo - simulator for network algorithms. `http://dcg.ethz.ch/projects/sinalgo/`.

[2] P. Beyens, M. Peeters, K. Steenhaut, and A. Nowe. Routing with compression in wireless sensor networks: a q-learning approach. In *Proc. of the 5th Euro. Wkshp. on Adaptive Agents and Multi-Agent Systems*, pages 575–578, Washington, DC, USA, 2005. IEEE Computer Society.

[3] H. Chan, A. Perrig, and D. Song. Secure hierarchical in-network aggregation in sensor networks. In *Proc. of the 13th ACM Conf. on Computer and Communications security (CCS)*, pages 278–287, New York, NY, USA, 2006. ACM Press.

[4] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Trans. on Wireless Communications*, 1(4):660–670, October 2002.

[5] W. Heinzelman, A. L. Murphy, H. Carvalho, and M. Perillo. Middleware to support sensor network applications. *IEEE Network Magazine Special Issue*, January 2004.

[6] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proc. of the 6th Int. Conf. on Mobile computing and Networking (MobiCom)*, pages 56–67, New York, NY, USA, 2000. ACM Press.

[7] A. Kansal and M. B. Srivastava. An environmental energy harvesting framework for sensor networks. In *Proc. of the Int. Symp. on Low power Electronics and Design (ISLPED)*, pages 481–486, New York, NY, USA, 2003. ACM Press.

[8] B. Krishnamachari, D. Estrin, and S. B. Wicker. The impact of data aggregation in wireless sensor networks. In *Proc. of the 22nd Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 575–578, Washington, DC, USA, 2002. IEEE Computer Society.

[9] A. Manjeshwar and D. P. Agrawal. TEEN: A routing protocol for enhanced efficiency in wireless sensor networks. In *Proc. of the 15th Int. Parallel & Distributed Processing Symp. (IPDPS)*, page 189, Washington, DC, USA, 2001. IEEE Computer Society.

[10] J. Zhu, S. Papavassiliou, and J. Yang. Adaptive localized qos-constrained data aggregation and processing in distributed sensor networks. *IEEE Trans. on Parallel and Distributed Systems*, 17(9):923–933, 2006.