Tethys: A Distributed Algorithm for Intelligent Aggregation in Sensor Networks

Robin Snader[§], Albert F. Harris III[†], and Robin Kravets[§]
Department of Computer Science - University of Illinois at Urbana-Champaign, USA[§]
Department of Information Engineering - University of Padova, Italy[†]
rsnader2@cs.uiuc.edu, harris@dei.unipd.it, rhk@cs.uiuc.edu

***Abstract*— One key issue in wireless sensor networks is energy efficiency. Aggregation techniques attempt to minimize the energy expended in communication by processing the data in the network rather than forwarding it all to a central point for processing. However, the naive application of this technique can result in a net increase in total energy expenditure due to the computational costs of the processing. In this paper, we present a new model for aggregation in wireless sensor networks, incorporating both the amount of data reduction achieved by the aggregation function and its associated computation costs. We then use this model to create a powerful, lightweight, distributed aggregation tree creation protocol, called Tethys, suitable for implementation in wireless sensor networks. We show that our protocol provides significantly lower energy consumption than protocols which do not take the amount of data reduction and computational cost of aggregation into consideration.**

I. Introduction

With the advent of wireless sensor networks, new paradigms for communication have emerged. Since the vital element in wireless sensor network communication is the data being transmitted rather than the endpoints, much work has focused on *data-centric communication*, as contrasted with the conventional *address-centric communication*. In data-centric communication, a packet traversing the network may not include any information as to it's source or ultimate destination, but rather an abstract of the type of information it contains. Routing is determined by an association between types of information and directions of data flow; data following these directions will ultimately arrive at its destination. Another key difference between wireless sensor networks and conventional, wired networks is the overriding importance of energy efficiency. Since wireless sensor nodes are typically energy constrained, extending network lifetime is a matter of minimizing energy expenditure. Communication costs are a major drain on energy reserves, leading a large amount of the research on energy efficiency to be focused on minimizing these costs.

Combining data-centric communication with reducing communication costs naturally leads to the notion of *data aggregation*. Data aggregation consists of forwarding a digest of multiple data packets rather that the packets in their entirety. For example, if the ultimate goal of a particular data type is to determine the maximum value of a certain measurement in the network, a node can compare all of the readings it receives and forward only the highest value, thus saving energy by reducing the total amount of data transmitted. This process is repeated throughout the network, creating a tree rooted at the ultimate destination of the data whose leaves are the sources of the data measurements (though intermediate nodes can be data sources as well). Although several data aggregation algorithms and frameworks have been proposed [1], [11], [13], [14], [18], they consider only simple aggregation functions. Others require centralized calculations to provide routes [6], [8], [10]. This leaves the problem of finding optimal aggregation points in the network in the face of more complex aggregation functions an open area of research. When more complex aggregation functions are used, an aggregation tree that is designed to minimize communication costs may consume more total energy than a naive, shortest-path tree due to the increased computation energy cost for performing the aggregation. Essentially, deciding *when* and *where* to aggregate data in a sensor network becomes more complex in the face of non-trivial aggregation functions.

When considering when and where to perform aggregation, one must quantify its benefits and costs. The most important benefit is data reduction; that is, if a node receives n packets of size D and aggregates them, what is the size of the aggregated packet? Much of the research in creating aggregation trees contains the implicit assumption that aggregation is *perfect* (*i.e.*, that a node need only forward D bytes, regardless of n). However, it is easy to see that many aggregation functions do not behave in this manner. In the worst case, aggregation consists of simple concatenation, where the amount of data to send is simply $n \times D$. Additionally, for aggregation functions that are data dependent, such as the mixing of audio streams, the data reduction due to aggregation may vary for the same function. One must also consider the cost of aggregation functions in terms of energy, which is proportional to the number of cycles the CPU must spend in computing the aggregate. Again, previous work begins with the implicit assumption that this cost is negligible. While this assumption may hold for simple aggregation functions, many useful and interesting functions are not trivial to compute, requiring both the data reduction and the cost of the aggregation function to be taken into account to find energy-optimal aggregation trees [6].

Designing a protocol that takes into account the dynamic metrics presents many challenges. In addition to integrating the costs associated with communication and computation, and their non-trivial interactions, the basic problem reduces

to a hybrid node-edge weighted Steiner Tree problem, which has been shown to be NP-complete [4]. This means that any practical solution to the problem (especially in the field of very large sensor networks) must be a heuristic approach, which generates acceptably good, though not necessarily optimal, results. Adding to the complexity is the need for a distributed solution. This need arises since sensor networks are deployed somewhat at random and must self-organize; this results in the absence of any single node that knows the topology of the entire network. A distributed solution must be careful to impose a low overhead, so as not to outweigh the savings from aggregation. This complex set of requirements explains why existing solutions focus on a simplified version of the problem.

The main contribution of this work is the design and evaluation of an aggregation tree creation protocol, which we call Tethys. Tethys is a dynamically adaptive, online, distributed algorithm that decides when and where to perform aggregation in order to minimize path energy consumption by taking into account the data reduction and energy cost of the aggregation function.

The rest of this paper is as follows. Section II defines two metrics used to model the performance of an aggregation function in terms of energy consumption and data reduction. Section III presents an analysis of when data should be aggregated, making use of the metrics in Section II. The algorithms and protocol specification for Tethys is presented in Section IV. Section V presents the evaluation of Tethys via simulation. Section VI presents conclusions and future directions.

II. Aggregation Function Metrics

The characteristics of the aggregation function being used determine the optimal aggregation tree for a given topology. Two data types in the same network with different aggregation functions may have very different trees. For example, a simple aggregation function (*e.g.*, finding the maximum) will produce an optimal aggregation tree with the aggregation points very close to the sources of the data [6]. Conversely, an aggregation function such as audio mixing, with high costs and low efficiency, will produce an optimal aggregation tree with the aggregation points pushed downstream towards the sink [6]. These two cases represent opposing corners of a space with two dimensions: aggregation efficiency, and aggregation cost.

Aggregation efficiency is a measure of the achieved data reduction. Assume that a node has n upstream neighbors, each sending D-byte packets. A node that performed no aggregation would forward all $n \times D$ bytes. Let it be the case that an aggregation function will result in an aggregate with size in the range $[D, n \times D]$ (*i.e.*, aggregation will never increase the total size). Then, let aggregation efficiency be defined as follows: if a given aggregation function produces an aggregate of size

$$D_{agg} = D + (1 - \varepsilon) \times (n - 1) \times D, \quad (1)$$

from n D-byte inputs, that function has aggregation efficiency ε. Thus an aggregation function with efficiency 1 corresponds to *perfect* aggregation, where n packets are compressed to a single packet of the same size, and an aggregation function with efficiency 0 corresponds to *simple concatenation* where the only savings comes from not sending header information multiple times.

Aggregation cost is a function of the number of CPU cycles used in the computation of the aggregate and can be measured in terms of the cost to aggregate a single byte relative to the cost of transmitting that byte a single hop. Therefore, if a given aggregation function required $C \times D$ mJ to aggregate D bytes and $T \times D$ mJ to transmit D bytes, the aggregation cost would be C/T. In this fashion, the differing computational and transmission costs of various platforms can be taken into account without loss of generality, normalized to the transmission costs. Under this model, aggregation functions that require little or no computation are approximated by an aggregation cost of 0, regardless of the actual CPU power requirements for a particular platform. The next section examines how these metrics are used to create energy-efficient aggregation trees.

III. When To Aggregate

The question of *when* to aggregate depends on the aggregation cost and efficiency. An easy upper bound on the cost of communicating a packet to the sink H hops away exists: since aggregation cost is normalized to the transmission cost (see the previous section), the cost to transmit a byte is simply 1. Thus there is an upper bound of $D \times H$ to transmit a D-byte packet to the sink. However, if there are multiple packets in the network, aggregation may be able to improve this bound. If two D-byte packets are H_1 and H_2 hops away from the sink respectively, not aggregating at all would give a cost of $D \times (H_1 + H_2)$; however, if the sources are H_1' and H_2' away from an aggregation point which is, in turn, H_3' hops away from the sink, then the cost to aggregate and send the data is as follows:

$$D \times (H_1' + H_2') + D \times (1 + \varepsilon) \times H_3' + D \times C_A, \quad (2)$$

where C_A is the cost to aggregate a byte and ε is the aggregation efficiency. Thus we can see that the decision as to when and where to aggregate depends on H_1 and H_2 relative to H_1', H_2' and H_3', as well as the aggregation efficiency ε and the aggregation cost C_A.

IV. Tethys: Intelligent Aggregation Tree Creation

To demonstrate the use of this model in supporting energy-efficient sensor network communication, this section presents our distributed aggregation tree creation protocol, which we call Tethys[1].

The collection of data in Tethys follows a standard sensor network pattern as it is broken down into two phases, query dissemination and data communication. In the query dissemination phase, a shortest-path tree to the sink is established using the usual technique of reverse-path forwarding. However, Tethys then takes advantage of the continuous transmission of

[1]In Greek mythology, Tethys was the Titan associated with the fertile sea, and the mother of all the rivers of the world.

```
PROCESS_UPDATE(pkt)
 1  if src = shortest_path_next_hop
 2     then sp_next_hop_updated ← true
 3  if !sp_next_hop_updated
 4     then DROP_UPDATE()
 5          return
 6  if pkt.cost + link_cost < rtable.cost
 7     then if pkt.src = rtable.next_hop
 8             then rtable.cost ← (pkt.cost + link_cost)
 9                  return
10          if pkt.dest = my_addr||
11             pkt.flowID = rtable.flowID
12             then return
13          rtable.next_hop ← pkt.src;
14          rtable.cost ← (pkt.cost + link_cost)
```

Fig. 1. Tethys: Process an update packet

```
FORWARD(pkt)
 1  if rtable.flowID = NIL
 2     then rtable.flowID ← pkt.flowID
 3  if rtable.flowID ≠ pkt.flowID
 4     then do_aggregation ← true
 5  if !do_aggregation||!IS_AGG_CHEAPER()
 6     then SEND(pkt)
 7          return
 8  if !data_waiting
 9     then START_AGG_TIMER()
10          data_waiting ← true
11     else AGGREGATE_DATA(pkt)
```

Fig. 2. Tethys: Forward data

data that typifies sensor networks to evolve this tree into one that is optimized for energy-efficiency.

A. Interest Dissemination

Many algorithms for efficient interest dissemination have been proposed [1], [2], [5], [13], [14], [17], [18]. The Interests in Tethys work much like the interests in Directed Diffusion [9]: when a user or process at a node requests data, an Interest is flooded throughout the network. The Interests contain a field containing the current estimated cost to send one byte of data back to the sink.

When a node receives an Interest, it compares the cost included in the packet with the cost in its routing table. If the node already has a path with a lower cost than that advertised by the Interest, it silently discards the packet. However, if the node does not contain a cheaper route to the sink, it updates its routing table accordingly and adds its estimated cost to reach the next hop into the cost in the Interest and forwards it.

After the Interests have been flooded throughout the network, each source node will contain the next hop in a cheapest path tree to the sink assuming there is no aggregation. The next phase of the protocol uses event dissemination to alter these paths to use aggregation, depending on the aggregation cost and efficiency, to minimize the cost of transmitting the data through the sensor network.

```
IS_AGG_CHEAPER()
1  agg_savings ← rtable.cost × agg_efficiency
2  if agg_cost < agg_savings
3     then return true
4     else return false
```

Fig. 3. Tethys: Is aggregation cheaper

B. Processing Events

When an event is detected by a source, it sends data along its minimum-energy route to the sink. Each node which detects the event labels it with a locally unique flow ID. Also included in the packet is the marginal cost to send an additional byte back to the sink; this cost will be a function of the node's current estimated cost, the efficiency of aggregation for that data type, and the cost of performing the aggregation. If another node has detected the same event and overhears this flow, it compares its minimum cost to send to the source along its current route and the combined cost of sending the un-aggregated data to the advertising node and that node's advertised cost to forward the aggregated data to the sink along its route (see Figure 1 line 6). If the latter cost is less, it will update its routing table to reflect this, and forward the data to be aggregated.

Since this cost is included in all subsequent packets the overhearing node sends, any node which is forwarding packets to that node will hear this and update its cost accordingly, with the new costs propagating back to the sources. Therefore, the initial, shortest-path-based routes are converted to an efficient aggregation tree if and only if it saves energy to do so.

Each node listens to all events, even if the Events are not addressed for themselves. In this way, each node will learn about flows in their neighborhood that could be aggregated. Specifically, when a node hears an Event, it uses that event to update its routing tables (See Figure 1). It is required that the first change made to a nodes routing table is to update the cost for its original next hop; updates that arrive before this point are not processed. This prevents costs for aggregated data from being compared with costs for un-aggregated data, potentially causing routing loops. Once this initial cost update occurs (during the first round of transmissions for each flow), for each overheard event, if the cost to send data to be aggregated and sent along the path at the next hop hop in the Event message is cheaper than sending along the current path, the routing table is updated (see Figure 1 line 6).

Finally, having finished processing the update, if the Event is addressed to the node, it forwards the packet (See Figure 2). If packets are only being forwarded for a single flow, the packet is sent immediately. if packets have been forwarded for multiple flows and aggregation causes energy savings (see Figure 3), the packet is delayed for some interval to see if more data arrives which can be aggregated with it. When that interval expires, the packet or aggregate is forwarded to the next hop.

Tethys does not incorporate mechanisms to ensure aggrega-

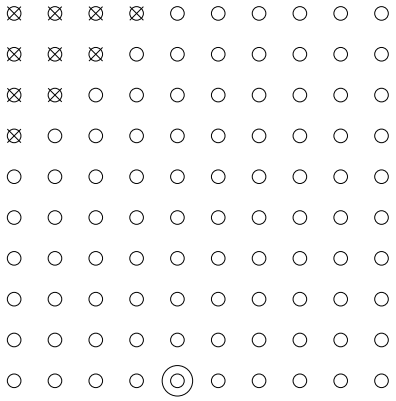

Fig. 4. A scenario representing a widespread event entering the network at one corner. Nodes marked with an X represent sources and the circled node represents the sink.

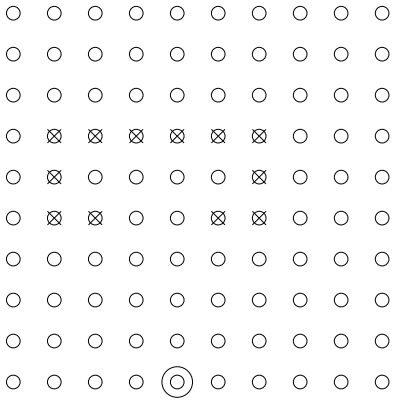

Fig. 6. A scenario representing a query to find the boundary of a region; the sources are nodes which lie on that boundary. Nodes marked with an X represent sources and the circled node represents the sink.

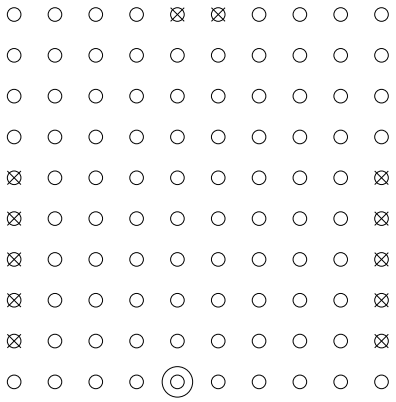

Fig. 5. A scenario representing multiple events entering the network at different edges. Nodes marked with an X represent sources and the circled node represents the sink.

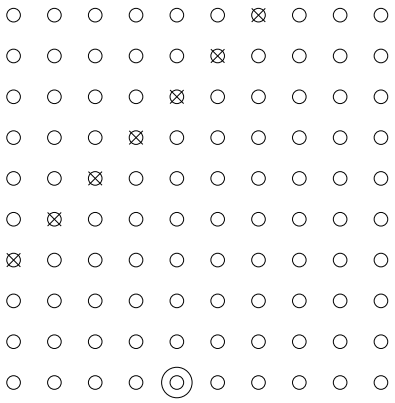

Fig. 7. A scenario representing a target moving through the network. Nodes marked with an X represent sources and the circled node represents the sink.

tion points are shifted so no one node bears the aggregation burden for too long of a time period as the present work is primarily concerned with locating optimal aggregation points in the network. Many methods of shifting aggregation points to increase network lifetime have been previously studied [3], [7], [12], [16] and their integration is out of the scope of this paper.

V. Evaluations

The metric we chose to evaluate Tethys is total energy consumed per event delivered to the sink. This metric captures the two key components of data-centric communication in sensor networks: energy consumption and data delivery rate. Additionally, by including the energy used to perform aggregation and accounting for the imperfection of routing algorithms, we are able to achieve a deeper insight into the dynamics of the tradeoff between network energy and CPU energy.

We implemented Tethys in the *ns2* simulator [15] and compared its performance to that of a non-aggregating, shortest-path protocol and the calculated performance of a Steiner-Tree-based, always-aggregating protocol; this algorithm is centralized and therefore impractical to actually implement, but represents the ideal case when aggregation is perfect and free.

Four scenarios on a grid topology were used for the simulations, as shown in Figures 4 to 7. Each of these test cases represents a plausible scenario in a sensor network: widespread point detection at the edge of the network, multiple events at the edges of the network, edge detection of a region, and tracking a moving event through the network, respectively.

For each run, 32-byte reports with 32-byte headers were used. To measure performance under varying conditions, the aggregation efficiency was varied from 0 (where two 32-byte reports combine to make a 64-byte report) to 1 (where two 32-byte reports combine to make a 32-byte report). Since headers can always be combined, this gives a worst case of two 64-byte packets being combined into a single 96-byte packet.

The energy cost to perform aggregation was also varied, testing the case where aggregating two bytes is free, the case where aggregating two bytes costs the same as transmitting a byte, and the case where aggregating a byte costs the same as transmitting two bytes. These aggregation costs, however, are

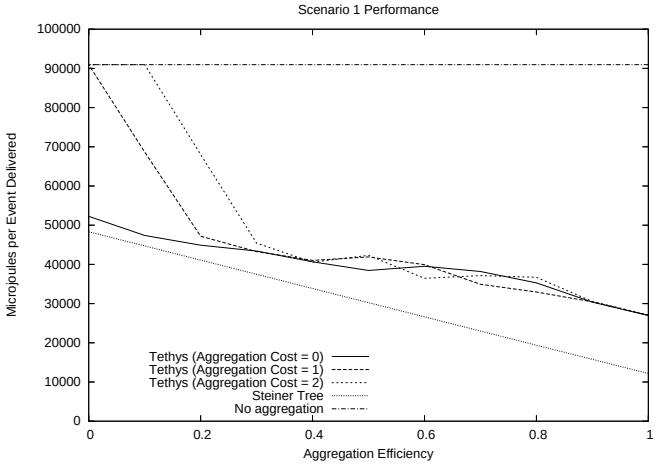

Fig. 8. Energy expended per event delivered to the sink as aggregation efficiency varies from 0 (simple concatenation) to 1 (perfect aggregation) for scenario 1.

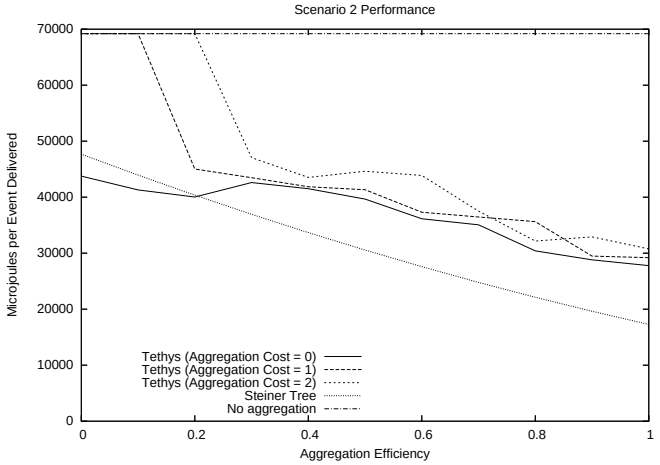

Fig. 9. Energy expended per event delivered to the sink as aggregation efficiency varies from 0 (simple concatenation) to 1 (perfect aggregation) for scenario 2.

only incurred once, at the point of aggregation.

Figure 8 shows the performance of the three protocols under the conditions of the first scenario (Figure 4) with three different aggregation costs. This scenario favors the aggregation strategy, since the sources are clumped close to each other and far from the sink. It can be seen that when aggregation is free or relatively inefficient, Tethys approximates the performance of the Steiner-tree algorithm, but even when aggregation is expensive, it never performs worse than the shortest-path algorithm.

Scenario two (Figure 5) presents more of a challenge for aggregation, as Figure 9 shows, since aggregation leads to longer paths that increase the network costs. By aggregating locally then forwarding via the shortest path for each of the clusters, Tethys actually outperforms the Steiner-tree algorithm when aggregation is inefficient and cheap. Again, it can be seen that as aggregation costs increase and aggregation efficiency decreases, Tethys gracefully degrades to the shortest-path case.

Scenario three (Figure 6) is designed to stress the Steiner-tree algorithm. Because the Steiner tree is much longer than the shortest paths, Tethys is able to outperform it in 40% of the tests, as can be seen in Figure 10, and this is true even when aggregation is not free. When the aggregation costs increase sufficiently or aggregation is insufficiently efficient, however, Tethys again falls back to the shortest-path strategy.

A complement to scenario three, scenario four (Figure 7) is designed to favor the Steiner-tree algorithm (see Figure 11). Its performance relative to Tethys is noticeably better than in the other scenarios. However, Tethys still performs respectably, splitting the difference between the Steiner-tree algorithm and the shortest-path algorithm in most of the tests.

These evaluations show that Tethys can perform considerably better than current protocols, which use either fixed aggregation trees (which can perform extremely poorly when aggregations is inefficient and expensive), or only opportunistic aggregation (which fails to take advantage of the

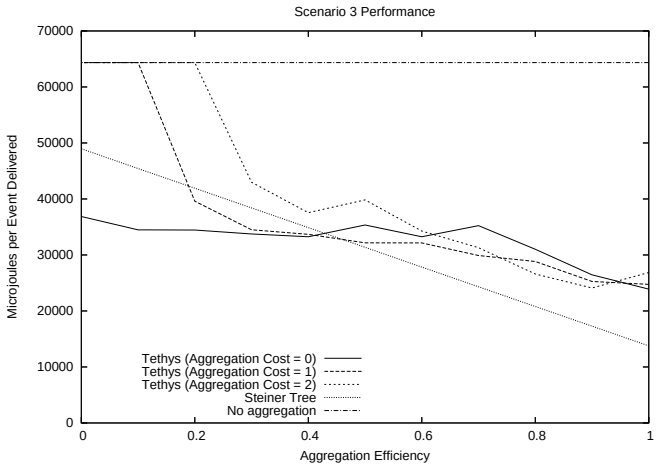

Fig. 10. Energy expended per event delivered to the sink as aggregation efficiency varies from 0 (simple concatenation) to 1 (perfect aggregation) for scenario 3.

significant energy savings available when aggregation is cheap and efficient).

VI. Conclusion and Future Work

This paper has described the problem of aggregating different types of data in a wireless sensor network, and the challenges associated with doing so efficiently. It then presented Tethys, a dynamic, distributed algorithm for creating aggregation and routing trees, in an energy-efficient manner.

It has been shown that a routing and aggregation protocol which chooses routes and aggregation points based on the data and prevailing network conditions can significantly outperform a protocol which statically chooses them. Furthermore, since a single network may support multiple types of query, and network conditions may change drastically over the course of a deployment, it is not feasible to choose optimal values in advance.

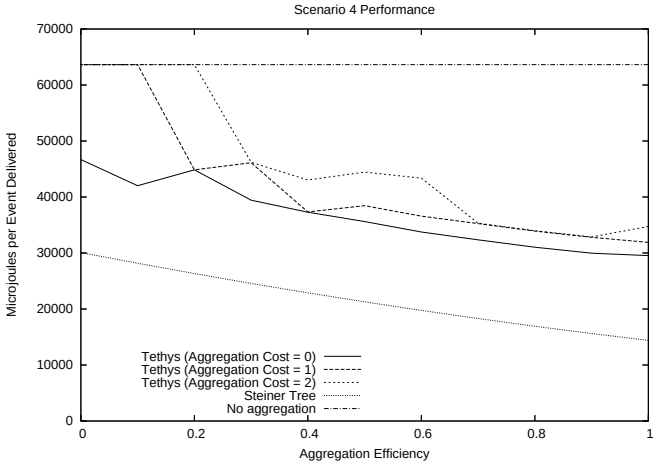

Fig. 11. Energy expended per event delivered to the sink as aggregation efficiency varies from 0 (simple concatenation) to 1 (perfect aggregation) for scenario 4.

This work has also demonstrated that it is possible to have a non-trivial distributed algorithm which creates low-cost routing trees in a network. Since existing algorithms such as [8] rely on global information to create their trees, this is a significant advancement.

Tethys admits numerous directions for extension and future work. Tethys currently makes no effort to bridge flows that are traveling two hops apart. Another refinement would be to allow nodes that overhear two separate flows but are not part of either to inform the flow with the higher-cost path of the existence of the lower-cost path via Cost Advertisement Messages. This presents several challenges: distinguishing flows from their subsequent aggregated forms to prevent routing loops, and ensuring that the cost of transmitting these messages does not outweigh the savings they present.

Another extension to Tethys would be adding support for transmit power control, bringing forward an interesting tradeoff: transmitting with high power uses more energy but increases the set of nodes that can overhear transmissions, thus increasing the potential for aggregation. While this tradeoff has been studied in multi-hop wireless networks, including the possibility for aggregation changes the problem fundamentally.

REFERENCES

[1] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Mobile Data Management*, 2001.

[2] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad-hoc heterogeneous sensor networks. In *Intl J. High Performance Computing Applications*, 2002.

[3] Qing Fang, Feng Zhao, and Leonidas Guibas. Lightweight sensing and communication protocols for target enumeration and aggregation. In *MobiHoc*, 2003.

[4] M. Garey and D. Johnson. *Computers and Intractability*. Freeman, San Francisco, CA, 1979.

[5] R. Govindan, J. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker. The sensor network as a database. Technical Report 02-771, Computer Science Department, University of Southern California, September 2002.

[6] A. Harris, R. Kravets, and I. Gupta. Building trees based on aggregation efficiency in sensor networks. In *The IFIP Fifth Annual Mediterranean Ad Hoc Networking Workshop (Med Hoc Net)*, 2006.

[7] Wendi Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the Hawaii International Conference on System Sciences*, 2000.

[8] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *ICDCS*, 2002.

[9] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCom*, 2000.

[10] B. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. In *DEBS*, 2002.

[11] Rajnish Kumar, Matthew Wolenetz, Bikash Agarwalla, JunSuk Shin, Phillip Hutto, Arnab Paul, and Umakishore Ramachandran. Dfuse: A framework for distributed data fusion. In *Proceeding of the First International Conference on Embedded Network Sensor Systems*, 2003.

[12] Stephanie Lindsey, Cauligi Raghavendra, and Krishna Sivalingam. Data gathering algorithms in sensor networks using energy metrics. *IEEE Transactions on Parallel and Distributed Systems*, 13(9), 2002.

[13] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[14] Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. The design fo an acquisitional query processor for sensor networks. In *SIGMOD*, 2003.

[15] ns2 Network Simulator. http://www.isi.edu/nsnam/ns/.

[16] Younis Ossama and Sonia Gahmy. Heed: A hybrid, energy-efficient, distributed clustering apporach for ad-hoc sensor networks. *IEEE Transactions on Mobile Computing*, 3(4), 2004.

[17] Narayanan Sadagopan, Bhaskar Krishnamachari, and Ahmed Helmy. The aquire mechanism for efficient querying in sensor networks. In *SNPA*, 2003.

[18] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. In *SIGMOD*, 2002.