EDR^2: A Sink Failure Resilient Approach for WSNs

Marco Valero*†, Mingsen Xu*, Nick Mancuso*, Wen-Zhan Song* and Raheem Beyah†
*Department of Computer Science
Georgia State University
Atlanta, Georgia 30303,USA
{mvalero,mxu4,nmancuso,songwz, }@cs.gsu.edu
†GT CAP Group, The School of ECE
Georgia Institute of Technology
Atlanta, GA 30332, USA
{rbeyah}@ece.gatech.edu

***Abstract*—Data collection, redistribution and retrieval are essential components of monitoring wireless sensor networks (WSNs). In dense WSN deployments, generated and sensed data are usually sent to a sink that can be reached through one or multiple hops. In the case where communications with the sink are disrupted due to different reasons, the data must be stored in the network for later retrieval. When we consider in-network storage, we must redistribute the data among an energy-constrained network with sensors that have a low storage capacity. In this paper, we combine both data redistribution and retrieval into a single problem and propose an algorithm named EDR^2: energy-efficient data redistribution and retrieval. EDR^2 is a distributed energy-efficient algorithm for in-network storage and later retrieval in WSNs. We analyze different scenarios including multiple data producers and multiple sinks. We formulate the problem as an optimization problem and use linear programming (LP) to find the optimal solution. We implemented our technique using real sensors to demonstrate the feasibility of our algorithm. We show that our approach is an energy-efficient solution for node selection when redistributing data in a WSN for eventual retrieval.**

I. Introduction

Over the past decade, wireless sensor networks (WSNs) have gained popularity due to the fact that they are potentially low cost solutions to a variety of real-world problems [1]. WSNs have been used in both military and civilian domains where they usually generate large amounts of data over their lifetime.

The goal of large-scale WSN deployments is to collect, process, store and forward data to remotely monitor and control the activities in specific areas of interest. Usually, these networks are unattended and sometimes some nodes including sink nodes might fail due to different reasons (e.g., battery depletion and tampering). In the case where sink nodes are unreachable, sensor nodes are required to store generated and collected data in the network to preserve it while sink nodes are replaced or reassigned. When considering in-network storage, we must redistribute the data among an energy constrained network with low storage capacity sensors. In order to provide low cost in-network storage solution, we are the first to combine data redistribution and retrieval into a single problem to propose a sink failure resilient approach. We provide EDR^2, an energy-efficient algorithm for data redistribution and retrieval, and the linear programming formulation to find the optimal solution to this problem. The ultimate goal of EDR^2 is to extend the network lifetime by reducing communication costs of redistribution and data retrieval when sink nodes fail.

During normal operation of WSNs, the data storage follows a centralized approach where data is collected by individual nodes and sent back to the sink or base station for storage and processing. Whenever the sink fails, a distributed storage approach must be used and all nodes should participate in the data storage by committing some local space for the data collected at the data producers. To accomplish data redistribution, nodes can learn some network topology information by communicating with their neighbors and use it to make decisions about how and where the data should be stored according to the location of the sink.

The goal of this research is to provide an energy-efficient algorithm for data redistribution and retrieval in WSNs. We formulate the data redistribution and retrieval problem using linear programming (LP) and find the most energy-efficient way to distribute data items in the network given a set of Data Producers (DPs) and sink nodes. The contribution of this work is threefold. First, the introduction of a graph transformation to generate a new flow network is proposed in order to use it as the input of the minimum cost problem. Second, we combine both data redistribution and retrieval into a single problem and present the optimal solution. Third, a new distributed algorithm for data redistribution and retrieval is proposed and implemented.

The rest of this paper is organized as follows. The related work is presented in Section II. Section III describes the network model and the problem definition. Section IV presents our graph transformation, the linear programming formulation and analyzes the results of different scenarios. In Section V, we introduce and explain our distributed algorithm for data redistribution and retrieval. In Section VII, we explain the details of our implementation in TinyOS and the experimental evaluation. Section VIII concludes the paper.

II. Related Works

Data redistribution and retrieval are often considered and studied as two independent problems. In this paper we combine them into a single problem to propose a sink failure resilient approach for WSNs. In the literature, the most similar research to our work was proposed by Tang et al. in [2] where the authors formulated the data redistribution problem as a minimum cost flow problem to find the optimal solution. We based our work on the idea of using minimum cost flow to provide a sink failure resilient approach for more

realistic scenarios. Our approach is different than [2] in that we included the transformation of the original graph into a flow network and use it to find the minimum data redistribution and retrieval cost. In [2], the authors analyze data redistribution and their goal is to assign sink nodes to offload data. In our case, the sink nodes, gateways usually connected to external networks, are already defined but still can be reassigned. In [2], the authors briefly mentioned that data retrieval can be accomplished using data mules or manually. In the case of large scale WSN deployments, the use of data mules has many disadvantages since they must have a notion of the global storage location, they have to visit most nodes in the network to retrieve data, and they are vulnerable to attacks putting in risk all the collected data.

EnviroStore [3] is another work that focus on data redistribution. In [3], the authors focused on maximizing storage capacity of disconnected sensor network to accommodate the most data. They also use the idea of data mules to share data across the partitioned network and increase the storage capacity. Our work differs from [3] since we do not consider disconnected networks and our main goal is to minimize communication costs while redistributing and retrieving the data.

In [4], the authors consider data retrieval as an individual problem. SolarStore [4] studies the tradeoffs between storage reliability and energy consumption in solar-powered sensor networks. SolarStore also adopts a disconnected network model, where the stable connection is not always available. They focus on replicating data in the network and dynamically adapting the degree of data replication depending on solar energy and storage availability. Their goal is to maximize the amount of data that can eventually be retrieved subject to energy and storage constraints, while our goal is to minimize the total energy consumption incurred in this procedure.

III. Network Model and Problem Definition

A. Network Model

In our model, we have a network represented as a weighted graph $G = (V, E, w)$, where each edge $(u, v) \in E$ has a transmission cost $w_{u,v}$, and each sensor $v \in V$ has a storage capacity c_v. We also have a set of data producers also called source nodes $P_i \in V$ that generate data and transmit the data to sink nodes $S_i \in V$ through one or multiple hops. In our model, we consider that all data elements have the same size and the amount of data is much larger than the local storage capacity of a single sensor. In the case where sink nodes fail, all data must be stored in the network. Therefore, all the nodes should commit some local storage to save some data elements for later retrieval. The objective of our work is to $minimize\ the\ combined\ cost$, that includes the redistribution cost when storing data in the network and the retrieval cost when collecting the stored data.

B. Problem Definition

In this section, we formally define the data redistribution and retrieval problem.

Assumption 1. Fixed Packet Size: we assume without loss of generality that the size of the packets generated at the data producers have the same size and we consider each packet as a data element.

Definition 1. Transmission Energy Function and Cost: the energy required to transmit a signal is approximately proportional to d^{α}, where d is the distance and $\alpha \geq 2$ is the attenuation factor that depends on the transmission medium. For each data transmission there is packet overhead θ which is the cost for sending one packet. In order to minimize the energy consumption, our objective is to minimize redistribution and retrieval costs when storing data in network. The problem studied in this work can be formally defined as follows:

Definition 2. Data redistribution and retrieval problem: Given a sensor network, a set of data producers P_i, and a set of sink nodes S_i, the data redistribution and retrieval problem is to find the minimum energy cost paths for both: distributing data elements $d_i's$ from P_i to different nodes M_i in the network and retrieving data from M_i nodes to the sink nodes once the sink nodes are ready to collect the stored data. Nodes M_i are storage nodes with heterogeneous storage capacities.

IV. Graph Transformation and Linear Programming Formulation

In this section we formulate the problem. Similar to [2], we formulate the data redistribution and retrieval problem as a minimum cost flow problem [5] and use linear programming to find the optimal solution.

The first step to solving our problem is to perform a graph transformation to generate a flow network that can be used as the input for our LP formulation.

A. Flow Network

Before we can use minimum cost flow to find the optimal solution to our problem, we need to transform our original graph into a flow network. We propose two transformations that can be used to calculate this solution. The steps to accomplish the graph transformation are the following:

Transformation 1 (T1):

1) Take the original graph and use any single-source shortest path algorithm such as Dijkstra's algorithm [6] to calculate the minimum cost from the source and the sink to all the nodes. Note that the source and sink nodes should be defined before doing this transformation
2) Connect the source to every node with an edge that represents the minimum cost calculated in the previous step and the link capacity is the storage capacity of each node.
3) Connect all the nodes to the sink. The edges also represent the minimum cost.
4) In the case where we have multiple data producers (sources) and/or multiple sinks, we can use an auxiliary node connected to all sources or sinks to represent a single source or sink. The cost of these edges from and to the auxiliary node are zero and the link capacities

are infinity for an auxiliary sink and the total amount of data for the auxiliary source.

Transformation 2 (T2):

1) Take the output graph from T1 and for each node v that is not a source, sink, or auxiliary node, create another node and re-label the two nodes as v' and v''.
2) Connect the two nodes v' and v'' using an edge. The capacity of this new edge is equal to the storage capacity of the original node v.
3) Connect the sources to all v' nodes with the original cost and edge capacity.
4) Connect v'' nodes to the sink with the original cost and edge capacity.

Figure 1 shows an example of how to do transformations T1 and T2 to generate the flow network for different scenarios.

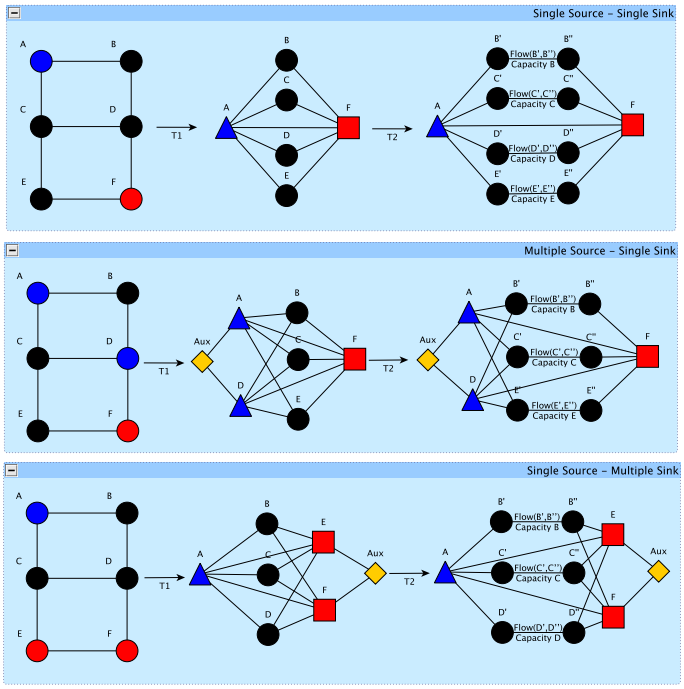


Fig. 1: Flow Network Generation for Different Scenarios.

B. Minimum Cost Flow Problem

In our case, the minimum cost flow problem consists in finding the cheapest way in terms of energy consumption for sending a certain amount of flow or data items through the network from the source nodes to the sink nodes. Each communication link between any two nodes has a maximum capacity.

Given a weighted flow network $G = (V, E, w)$ with a source and sink, $s, t \in V$ and amount of flow, p, the LP formulation including our parameters for communication cost and data size is as follows:

Variables:

$$w_{u,v} \leftarrow real \qquad \forall (u,v) \in E$$

$$f_{u,v} \leftarrow real \qquad \forall (u,v) \in E$$

Objective Funtion:

$$\sum_{(u,v)\in E} w_{u,v} \cdot f_{u,v} \rightarrow Min$$

Subject to:

$$\sum_{w\in V} f_{u,w} = 0 \ \forall u \neq s,t \qquad (flow\ conservation)$$

$$\sum_{w\in V} f_{s,w} = \sum_{w\in V} f_{w,t} = p \qquad (required\ flow)$$

$$0 \leq f_{u,v} \leq l_{u,v} \qquad (link\ capacity)$$

If we use the T1 graph:

$$\sum_{w\in V} f_{s,w} \leq c_w \qquad (storage\ capacity)$$

If we use the T2 graph:

$$\sum_{w\in V} f_{s,w'} = f_{w',w''} \qquad (already\ met)$$

Where:

$$w_{u,v} = EnerCost(u,v) + \theta$$

$$EnerCost(u,v) = d^{\alpha}$$

We can now explain the differences between the graphs generated with Transformation T1 and T2. In the case were we use the graph T1, we need to guarantee that the storage capacity of each node is not exceeded. Therefore, we need to add one more constraint (fourth constraint) that will change the classical minimum cost flow formulation. In the case where we use the graph T2, we have an edge between the nodes w' and w'' whose capacity is equal to the storage capacity of the original node w. In this way, we guarantee that the capacity of each node will not be exceeded and we can relax the assumption that all the nodes have the same capacity or the capacity of each node is fully utilized. If we use the T2 graph we do not really need the fourth constraint since it is already covered with the flow conservation constraint.

By solving this LP formulation, we can obtain the minimum data redistribution and retrieval cost for any given set of data producers and sinks in a sensor network.

V. Distributed Algorithm

We now present EDR^2, a distributed energy-efficient algorithm for data redistribution. It is inspired by the distributed "Push-Relabel" algorithm for solving max-flow problem [7].

Algorithm 1 consists of a network initialization stage and a distributed flow scheduling stage. In the initialization stage, the sink node broadcasts a probing message which propagates through the network to determine the total communication cost from each individual node towards the sink (CtS). The initial local CtS is a number larger than $local\ \ communication\ cost\ *\ |V|$. We minimize the total message cost during initialization stage by exploring the *Dominant Pruning* based broadcasting [8]. First, every node broadcasts once to get the neighbor list. A second broadcast is applied to exchange the one-hop neighbor list in order to obtain the knowledge of two-hop neighbors. Then, sink calculates the CtS for each of its one-hop neighbors, determining the forwarding list with minimum $\sum CtS$ which covers all the two-hop neighbors of sink. The computation of CtS value is based on: CtS = $MIN(local\ CtS,\ Received\ CtS\ +\ local\ communication\ cost)$. The CtS is also updated in recipient nodes, and the nodes set the gateway node information as well.

During the minimum cost flow distribution stage, the data producer also sends a message to collect information about the path to redistribute the data. We calculate the path capacity based on the total amount of storage capacity in its downstream nodes to the sink. For example, if a node is 3 hops away from the sink and each node has a storage capacity of 10, the total capacity of the path is 30. We use push-relabel to send the data using the minimum cost path until the capacities are fully utilized. The algorithm basically perform push-relabel operations using node capacity, and minimum cost path to sink. Thus, the message complexity for the termination of the algorithm is bounded with $O(2|V|^2 + |V|)$.

Algorithm 1 EDR^2 Algorithm

procedure: *Initialization*

1: $status_u$ = $inactive$
2: CtS_u = $comcost_u * |V|$
3: each node broadcast to get neighbor list
4: **if** $identity_u$ = $SINK$ **then**
5: broadcast the message
6: **end if**
7: Upon reception of (INIT, rCtS) msg from v
8: update CtS_u = $rCtS + comcost_u$
9: **for** all $i \in nbrlist_u$ **do**
10: update CtS_i = $rCtS + comcost_i$
11: **end for**
12: **if** $u \in forwarding\ list\ of\ v$ **then**
13: choose forwarding list with minimum $\sum_{i \in nbrlist_u} CtS_i$ covering all two-hop neighbors
14: broadcast the message
15: **else**
16: drop the message
17: **end if**

procedure: *Distributed Min-cost Flow*

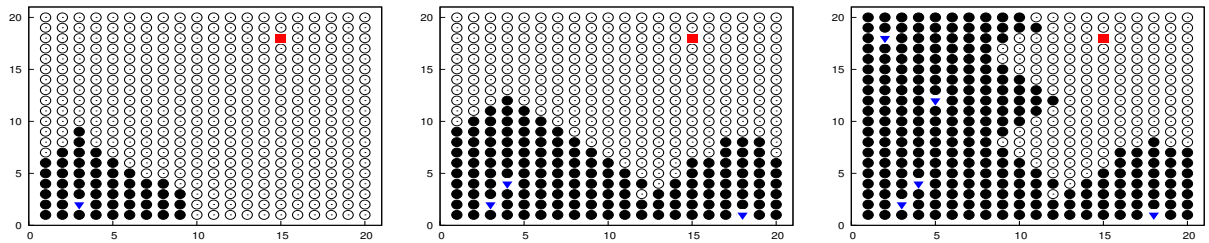
if $identity_u$ == DP **then**
2: $excess_u$ = data demand;
end if
4: upon receive flow push msg with $f_{(w,u)}$
if ($excess_u$ = $f_{(w,u)}$)> 0 **then**
6: find the neighbor v with shortest path to sink
δ = min $\{cap_v, excess_u\}$
8: **if** δ >0 **then**
push δ to node v
10: **else**
if there is remaining storage space **then**
12: store the δ data item in node u
$excess_u$ = 0
14: **else**
relabel node $height_u$ and go to 5
16: **end if**
end if
18: **end if**

VI. Analysis and Performance

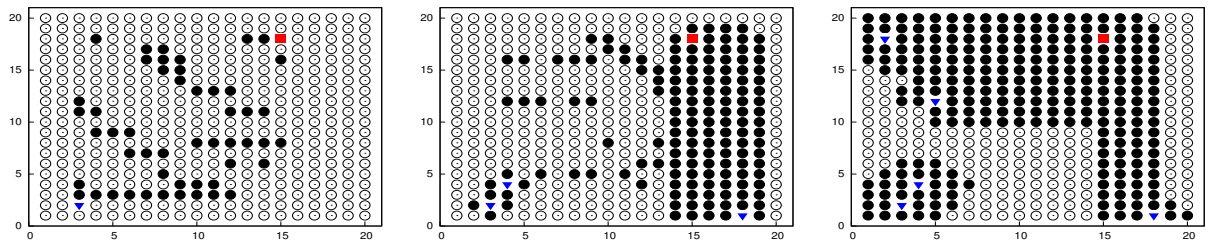
In order to compare the results of our formulation with the related work [2], where only redistribution was considered, we modeled different network scenarios, implemented our formulation using the GNU Linear Programming Kit (GLPK) and obtained the optimal solutions. One scenario we modeled is a network of 400 nodes placed on a grid of 20x20. We selected one through five data producers and one sink (node 355 = (18,15)). Each data producer has 500 data elements to redistribute and the storage capacity of each node is 10 data elements. Figures 2a-2c show the node selection when considering only data redistribution [2] and Figures 2d-2f when considering both redistribution and retrieval. As expected, the data is distributed close to the data producers when data retrieval is not considered and in between data producers and a sink otherwise. Figure 3 shows the cost for three cases: only redistribution [2], redistribution and retrieval (our approach), and only redistribution plus only retrieval. In Figure 3 we can observe that local optimal (only redistribution) plus local optimal (only retrieval) is not equal to global optimal (redistribution and retrieval). Note that the gap between our approach and only redistribution plus only retrieval increases when we have more data producers and/or data.

VII. Implementation in TinyOS and Evaluation

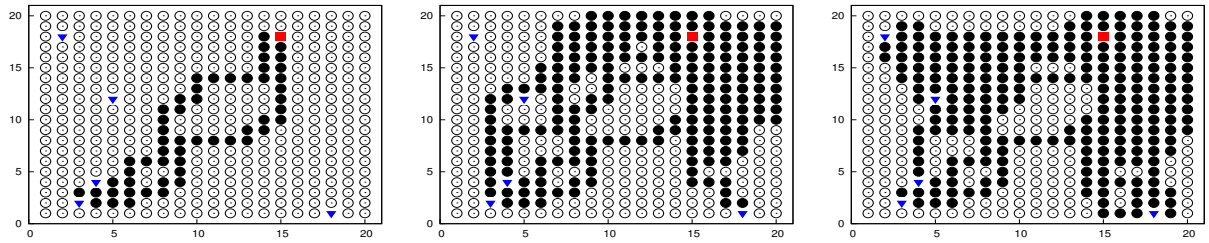
We implemented our EDR^2 algorithm, a distance vector-based (DV) algorithm (for redistribution only), and two heuristic algorithms in TinyOS-2.x to compare their performances in a WSN with heterogeneous data storage space and different migration costs for different links. Various parameters including total communication cost and data persistence have been examined to evaluate the proposed algorithm. For our experiments, we set up a network using 16 sensor nodes placed in a 4x4 grid and set the transmission power to minimum in



(a) Only Redistribution - 1 Source (b) Only Redistribution - 3 Sources (c) Only Redistribution - 5 Sources



(d) Redistribution and Retrieval - 1 Source (e) Redistribution and Retrieval - 3 Sources (f) Redistribution and Retrieval - 5 Sources



(g) EDR^2 - 1 Source (h) EDR^2 - 3 Sources (i) EDR^2 - 5 Sources

Fig. 2: Node Selection for Data Redistribution and Retrieval

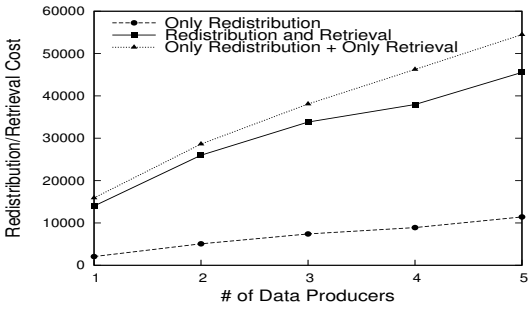


Fig. 3: Redistribution and Retrieval Costs.

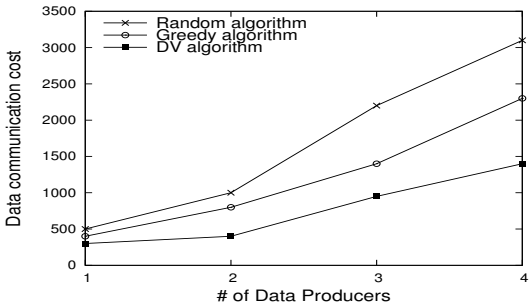


Fig. 4: Data Migration Cost for Different Algorithms.

order to have a multihop network. The transmission cost is defined as the function of residual energy in the node, with the range from 10 to 50. The number of packets for each data producer to send is 20, and each node is assigned with the same initial storage capacity of 10 packets.

Figure 4 presents experimental results for redistribution only. In this scenario, we compare the performance of a distance vector-based algorithm, a greedy algorithm and a random algorithm. We implemented the distance vector algorithm which created a local table on each node with global cost in-

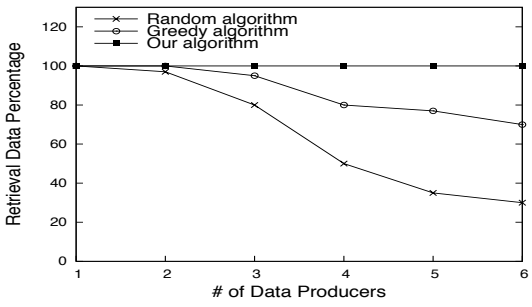
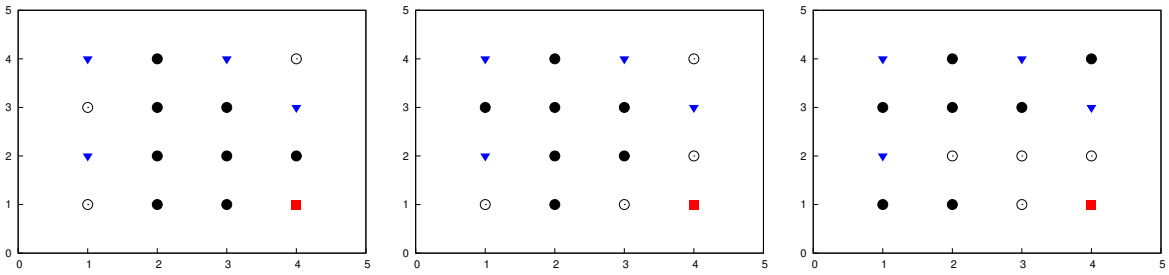


Fig. 5: Retrieval Data Persistency.



(a) EDR^2 Node Selection (b) Greedy Node Selection (c) Random Node Selection

Fig. 6: Node Selection for Experimental Data Redistribution and Retrieval

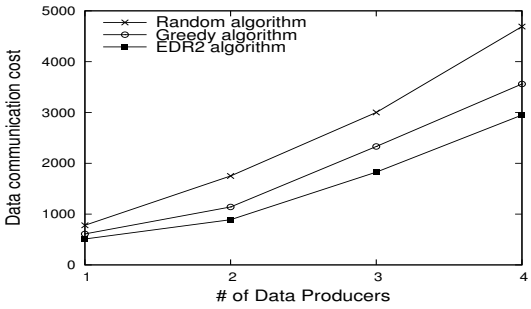


Fig. 7: Redistribution and Retrieval Costs.

formation. Data producers take advantage of this table to select the neighbors to offload their data. In the greedy algorithm, the data producer selects the neighbor with the smallest cost to offload all its data. The selected node stores data in flash until it is full and then starts forwarding the packets to the next minimum cost hop. In the random algorithm, the producer randomly selects one node to offload all its data. When the selected node's flash is full, it starts forwarding the packets to the next randomly selected hop.

Figure 5 shows the retrieval data persistency of the three different algorithms. Random and greedy algorithms will cause data to be lost when the number of data producer increase to 3, while our DV algorithm can maintain 100% data persistency as long as the total network storage is larger than the data generated. Since, the random and greedy algorithms push the excessive data from data producers to the selected neighbor, and pass the selection token to the next neighbor, the data flow can be directed to a "dead end" and therefore it can be lost. Compared to the other algorithms, our DV algorithm had a good data offloading plan based on the minimum cost path. Also, every data element was successfully stored in the network and retrieved from the sink.

Figure 6 shows the node selection for our data redistribution and retrieval experiments when using different algorithms. Figure 7 compares EDR^2's total communication costs with the random and greedy algorithms. EDR^2 always selects the lowest cost path to offload data and retrieve the data, while respecting each node's storage capacity. The results show that EDR^2 outperforms the other two algorithms for single and multiple data producers.

VIII. Conclusion

In this paper, we formulated in-network data storage and retrieval as an optimization problem, which can be optimally solved using a linear programming model. We combined both data redistribution and retrieval into a single problem and propose an energy efficient approach to preserve the data in cases where communications with the sink are disrupted. We designed EDR^2, an efficient distributed algorithm for data redistribution and retrieval that achieves the minimum cost, and implemented the algorithm in TinyOS-2.x to show its feasibility and efficiency through different experimental scenarios.

IX. Acknowledgements

This work is partially supported by NSF Grant CNS-1052769

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, Mar. 2002.

[2] B. Tang, N. Jaggi, H. Wu, and R. Kurkal, "Energy-Efficient Data Redistribution in Sensor Networks," in *MASS*, Aug. 2010.

[3] L. Luo, C. Huang, T. Abdelzaher, and J. Stankovic, "EnviroStore: A cooperative storage system for disconnected operation in sensor networks," in *26th Annual IEEE Conference on Computer Communications (INFOCOM)*, May 2007.

[4] Y. Yang, L. Wang, D. K. Noh, H. K. Le, and T. F. Abdelzaher, "SolarStore: Enhancing Data Reliability in Solar-powered Storage-centric Sensor Networks," Jun. 2009.

[5] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," *Journal of Algorithms*, no. 22, pp. 1–29, 1997.

[6] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959. [Online]. Available: http://jmvidal.cse.sc.edu/library/dijkstra59a.pdf

[7] T. L. Pham, I. Lavallee, M. Bui, and S. H. Do, "A Distributed Algorithm for the Maximum Flow Problem," in *Proceedings of the 4th International Symposium on Parallel and Distributed Computing*, Jul. 2005.

[8] H. Lim and C. Kim, "Flooding in wireless ad-hoc networks," *Computer Communications Journal*, vol. 24, pp. 353–363, 2001.