DAO-R: Integrating Data Aggregation and Offloading in Sensor Networks Via Data Replication

Basil Alhakami, Bin Tang, Jianchao Han, and Mohsen Beheshti
Computer Science Department
California State University Dominguez Hills, Carson, CA
balhakami1@toromail.csudh.edu, {btang,jhan,mbeheshti}@csudh.edu

***Abstract*—We study *overall storage overflow problem* in sensor networks, wherein data-collecting base station is not available while more data is generated than available storage spaces in the entire network. Existing research designs a two-stage solution to solve this problem. It first aggregates overflow data to the size that can be accommodated by the available storage capacity in the network, and then offloads the aggregated data into the network to be stored. We refer to this naive two-stage solution as DAO-N. In this paper, we demonstrate that this approach does not necessarily achieve good performance. We propose a more unified method that is based upon data replication techniques, referred to as DAO-R, in order to improve the performance of DAO-N. Specifically, we design two energy-efficient data replication algorithms to integrate data aggregation and data offloading in DAO-N. We show via extensive simulations that DAO-R outperforms DAO-N by around 30% in terms of energy consumption under different network parameters.**

Keywords – Data Aggregation, Data Offloading, Overall Storage Overflow, Sensor Networks, Energy-Efficiency

I. Introduction

In this paper, we focus on some emerging sensor networks such as underwater sensor networks [6] and wind and solar harvesting [10, 15]. A common characteristic of such networks is that they are all deployed in inaccessible or inhospitable regions, or under extreme weather, to constantly collect large amounts of data from the physical environments for a long period of time. Due to the inaccessible and hostile environments, it is not viable to deploy base stations (with power outlets) to collect data in or near the sensor fields. Therefore, data generated have to be stored inside the sensor network for some period of time and then be collected by periodic visits of robots or data mules [9], or by low rate satellite links [14].

Meanwhile, storage is still a serious resource constraint of sensor nodes despite the advances in energy-efficient flash storage [16]. As a consequence of this resource constraint and the absence of base stations, the massive sensory data could soon overflow data storage of sensor nodes and cause data loss. Below we outline two levels of data overflow and their corresponding solutions.

- Node Storage Overflow. The first level of data overflow is *node storage overflow*, wherein some data-generating sensor nodes deplete their own storage spaces, causing data loss. These sensor nodes with depleted storage spaces while still generating data are referred to as *data nodes*. The newly generated data that can no longer be stored at data nodes is called *overflow data*. The solution to avoid such data loss is simple: the overflow data is offloaded to other nodes with available storages (referred to as *storage nodes*).[1] Different data offloading techniques have been proposed with the goals of either minimizing the total energy consumption during data offloading [17], or maximizing the minimum remaining energy of storage nodes to prolong network lifetime [8], or offloading the most useful information considering data could have different priorities [21]. However, these techniques did not address the second level of data overflow, which is *overall storage overflow* explained below.
- Overall Storage Overflow. This happens when the total size of the overflow data is larger than the total size of the available storage in the network. Data offloading solely can not solve this problem and discarding data becomes inevitable if no actions taken. This is a more severe problem compared to the node storage overflow.

Fortunately, the spatially redundant or correlated sensory data provides us an opportunity to solve aforesaid overall storage overflow problem, by allowing us to aggregate and reduce the size of overflow data without sacrificing information loss. To solve overall storage overflow, existing research [17, 18] presents a naive data aggregation and offloading solution called DAO-N, which aggregates the overflow data and then offloads them into the network. In particular, they present an approximation algorithm [18] to solve the data aggregation part and an optimal algorithm for data offloading part [17].

In this paper, we show that solving each stage independently and combining the results does not necessarily achieve best performance. We present an unified approach, called DAO-R, to leverage the synergies existing between data aggregation and data offloading. Specifically, we design two centralized data replication algorithms viz. Global Replication Algorithm and Localized Replication Algorithm, wherein data is replicated on the way of data aggregation. The novelty of DAO-R is that replicating data along aggregation paths achieves the effect of "offloading" overflow data to storage nodes, without introducing extra energy cost. We show via extensive simulations that DAO-R outperforms DAO-N by around 30% in terms of energy consumption under different network parameters.

Paper Organization. The rest of the paper is organized as

[1]Sensor nodes that generate data but have not depleted their storage spaces are considered as storage nodes.

follows. Section II discusses related work. In Section III, we introduce the overall storage overall problem with its network, data correlation, and energy models. Section IV reviews DAO-N and its algorithmic solutions. In Section V, we present DAO-R and design two data replication algorithms to integrate data aggregation and offloading. In Section VI, we compare DAO-R with DAO-N and discuss the simulation results. Section VII concludes the paper with possible future research.

II. Related Work

How to preserve data in sensor networks in the absence of the base station has become an active research in recent years. In particular, Tang et al. [17] addressed it as an energy-efficient data redistribution problem. Tang and Ma [18] recently solved overall storage overflow problem by aggregating overflow data and reducing their sizes so that they can be accommodated by the available storage. However, important research problem such as how to integrate data aggregation and data offloading to further save energies is not addressed.

Ganesan et al. [3] adopted data aggregation techniques to tackle storage constraint of sensor networks. They proposed wavelet compression techniques to construct summaries for data at different spatial resolutions, and designed a progressive aging scheme wherein older data gets more aggressively summarized to save storage space for newer data. The summarization is based on a hierarchical grid-based overlay, in which summaries at each higher level of the hierarchy encompass larger spatial scales but are more lossy. In contrast, our approach does not rely upon any hierarchy of overlays.

Traditional data aggregation in sensor networks is to collect sensor data by combining the data from different sensor nodes on the way to the base station, in order to eliminate redundancy and to reduce energy consumption. As a result, the underlying routing structures for data aggregation are usually trees rooted at the base station. Data aggregation techniques have been designed for different purposes. Some are used to maximize the network lifetime (the time until the first node depletes its energy) [13, 19], some are used to minimize the total energy consumption or communication cost [11, 12], and some to reduce the delay of data gathering [20]. In contrast, data aggregation in this paper has very different goal – it is to aggregate the overflow data so that their size can be reduced and accommodated by the storage spaces available in the network, in order to prevent data loss caused by overall storage overflow.

III. Overall Storage Overflow Problem [18]

The sensor network consists of *data nodes* (with overflow data) and *storage nodes* (with available storage spaces), as shown in Fig. 1. The total size of the overflow data from the data nodes exceeds the total size of the storage spaces from the storage nodes, resulting in overall storage overflow. To aggregate data, one or multiple data nodes (called *initiators*) send their entire overflow data to other data nodes. When a data node (called an *aggregator*) receives the data, it aggregates its own overflow data (each aggregator can aggregate its data only once). After that, the aggregator forwards initiators' overflow data to another data node, which then becomes an aggregator and aggregates its own overflow data, and so on and so forth. This continues until enough aggregators are visited such that the total size of the overflow data after aggregation equals to or is slightly less than the total available storages in the network. At this point there is zero amount of overflow data on each initiator, the last aggregator being visited by each initiator has both its own aggregated data and the entire data from the initiator, and all other aggregators have their own aggregated overflow data. If a data node is not involved in data aggregation (i.e., not an initiator and not an aggregator), its overflow data is not aggregated. After aggregation, all the overflow data (aggregated or not) are then offloaded to storage nodes. The goal is to minimize the total energy consumption in the entire process. Fig. 1 shows both stages of data aggregation and data offloading.

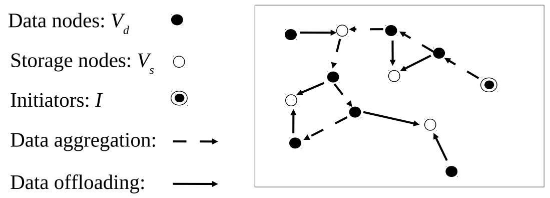


Fig. 1. An illustration of data aggregation and offloading.

Network Model and Data Correlation Model. We represent the sensor network as a graph $G(V, E)$, where $V = \{1, 2, ..., |V|\}$ is the set of $|V|$ sensor nodes uniformly distributed inside the network, and E is the set of $|E|$ edges. There are p data nodes (denoted as V_d), each has R bits of overflow data. The other $|V| - p$ nodes are storage nodes, denoted as V_s, each of which has m bits of available storage spaces. We adopt data correlation model proposed in [2]. Let $H(X|Y)$ denote the conditional entropy of a random variable X given that random variable Y is known. Overflow data at data node i is represented as an entropy $H(i) = R$ bits if no side information is received from other data nodes; and $H(i|j_1, ..., j_p) = r \leq R$ bits, $j_k \in V_d \wedge j_k \neq i, 1 \leq k \leq p$, if data node i received side information from at least another data node. That is, a data node aggregates its overflow data from R to r if it receives data from at least one initiator.

First Order Radio Energy Model [7]. For node u sending R-bit data to its one-hop neighbor v over their distance $l_{u,v}$, the transmission energy cost at u is $E_t(R, l_{u,v}) = E_{elec} \times R + \epsilon_{amp} \times R \times l^2_{u,v}$, the receiving energy cost at v is $E_r(R) = E_{elec} \times R$. Here, $E_{elec} = 100nJ/bit$ and $\epsilon_{amp} = 100pJ/bit/m^2$. Let $w(R, u, v) = E_t(R, l_{u,v}) + E_r(R)$. Let $W = \{v_1, v_2, ..., v_n\}$ be a *walk*, i.e., a sequence of n nodes with $(v_i, v_{i+1}) \in E$, $1 \leq i \leq n-1$ and $v_1 \neq v_n$. If all the nodes in W are distinct, W is a *path*. Let $c(d, W) = \sum_{i=1}^{n-1} w(d, v_i, v_{i+1})$ denote the energy consumption of sending d-bit of data along W.

In data aggregation, the route that the R-bit overflow data at each initiator traverses could be either a path or a walk, because enough number of aggregators needs to be visited in order to reduce overflow data size. $c(R, W)$ is the *aggregation cost* of R-bit data traversing along W starting from its initiator. In data offloading, however, the route that the overflow data traverses is always a path, in order to minimize the energy consumption for offloading. Besides, in data offloading, each offloaded overflow data unit is not necessarily in sizes of R or r. Instead, for the purpose of energy efficiency, the overflow data at each data node can be splitted into small units, each of which can be offloaded to different storage nodes. Let the size of each small unit be x-bit, then $c(x, W)$ is the *offloading cost* of offloading this x-bit data from its data node v_1 to a storage node v_n, along path W.

Feasible Overall Storage Overflow. It refers to the conditions that a) there is an overall storage overflow, b) enough number of aggregators are visited such that the data after aggregation can fit in the available storages. Let q denote the number of needed aggregators. It derives [18] that for feasible overall storage overflow,

$$q = \lceil \frac{p \times R - (|V| - p) \times m}{R - r} \rceil = \lceil \frac{p \times (R + m) - |V| \times m}{R - r} \rceil, \tag{1}$$

and the valid range of p is

$$\frac{|V|m}{m + R} < p \leq \lfloor \frac{|V|m - R + r}{m + r} \rfloor. \tag{2}$$

Given p and q, at most $(p - q)$ data nodes can be selected as initiators.

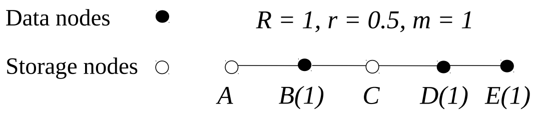


Fig. 2. A sensor network with overall storage overflow problem.

EXAMPLE 1: Fig. 2 illustrates the overall storage problem with a linear sensor network of five nodes. Nodes B, D, and E are data nodes, while A and C are storage nodes. Numbers in parentheses are the overflow data sizes R. The energy consumption along any edge is one per unit of data. There are total 3 units of overflow data while there are only 2 units of available storage, causing overall storage overflow. Number of aggregators q is calculated as 2 using Equation 1. This leaves one data node to be initiator. In Section IV, we show how to select the initiator and corresponding aggregation path to solve this overall storage overflow problem. □

IV. Review of DAO-N: A Naive Two-Stage Approach

DAO-N solves data aggregation and data offloading as two separate and independent problems, and then combine the solutions (i.e., the energy cost of DAO-N is the sum of aggregation cost and offloading cost). The DAO-N is NP-hard, since its constituent data aggregation problem itself is NP-hard [18]. Below we present algorithms solving data aggregation and data offloading, respectively.

Data Aggregation Approximation Algorithm. Below we present an approximation algorithm, which yields energy cost that is at most $(2 - \frac{1}{q})$ times of the optimal [18].

Algorithm 1: Data Aggregation Approximation Algorithm.

1). Transform sensor network graph $G(V, E)$ to $G'(V', E')$ as follows. V' is set of p data nodes in V, i.e. $V' = V_d$. For any two data nodes $u, v \in V_d$ in G, there exists an edge $(u, v) \in E'$ in G' if and only if all the shortest paths between u and v in G do not contain other data nodes. For each edge $(u, v) \in E'$, its weight $w(u, v)$ is the cost of the shortest path between u and v in G.

2). Create a set S containing all the edges in E' in nondecreasing order of their weights. Create a forest F of $|V'|$ trees, each is one of the $|V'|$ nodes initially.

3). Starting from the first edge in S, if that edge connects two different trees, add it to F and combines two trees into a single tree. This repeats for q times.

4). Replace each edge (u, v) in F with a shortest path between u and v in G (choose one randomly if there are multiple).

5). For each connected component of the resulted F, if it is linear, it starts from one end (the initiator) and visits the rest nodes exactly once; if it is a tree, it does the following. Find an edge (u, v) with maximum weight in the tree (tie is broken randomly), which has three parts: T_u, (u, v), and T_v. It starts from u (the initiator) and visits all the nodes in T_u in a sequence following depth-first-search (DFS) and comes back, then visits v, from where it visits all the nodes in T_v in a sequence following DFS.

Applying above approximation algorithm to Example 1, it obtains two data aggregation solutions. One is that data node B is initiator and the aggregation path is B, C, D, and E, as shown in Fig. 3(a). It also shows the sizes of overflow data at each data node after aggregation (but before offloading). In particular, there is no overflow data left at initiator B, and there are 0.5 and 1.5 units of data at data nodes D and E respectively. The aggregation cost is 3, which happens to be optimal in this small network.

Data Offloading Algorithm. Data offloading is to offload the overflow data from its data node to storage node, since there are enough storage spaces available to store the overflow data after data aggregation. The goal of data offloading is to minimize the energy cost incurred in the offloading process. Tang et al. [17] show that by transforming sensor network graph $G(V, E)$ into a flow network, data offloading problem is equivalent to minimum cost flow problem [1], which is solvable optimally in polynomial. In this paper we adopt the scaling push-relabel algorithm proposed and implemented in [4, 5], with time complexity of $O(|V|^2|E|\log(|V|C))$. Here C is the maximum capacity of an edge in the transformed graph.

Fig. 3(b) shows the data offloading solution that follows data aggregation in Fig. 3(a). It shows that 0.5 unit of data at D and 0.5 unit of data at E are offloaded to storage node C, while 1 unit of data at E is offloaded to storage node A, with offloading cost of 5.5.

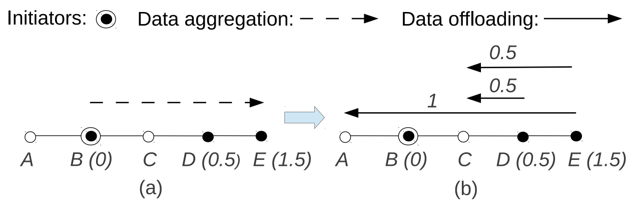


Fig. 3. One naive two-stage solution with B being the initiator. (a) Data aggregation stage: values in parentheses are sizes of overflow data after aggregation. (b) Data offloading stage: values on the arrowed lines are sizes of overflow data that is offloaded from its data node to a storage node.

V. DAO-R: Integrating Data Aggregation and Data Offloading via Replication

In this section we first demonstrate the limitations of DAO-N using Example 1. We then formulate DAO-R that integrates data aggregation and offloading. Finally we solve DAO-R by designing two data replication algorithms.

A. Limitations of DAO-N.

Another naive two-stage solution for Example 1 is shown in Fig. 4, wherein data node E is initiator and the aggregation path is E, D, C, and B. In this case, the aggregation cost is again 3. However, the offloading cost is 2, a 64% improvement compared to 5.5 in Fig. 3(b). Therefore, even though the solution in Fig. 3 independently solves each of the data aggregation and data offloading nicely (one with approximation algorithm and the other optimal algorithm), the combined solution may not give the best result.

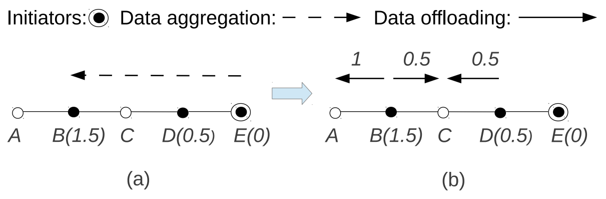


Fig. 4. Another naive two-stage solution solution with E being the initiator.

Furthermore, even though Fig. 4 gives optimal combined total energy cost of data aggregation and offloading, its performance can be further improved. Our key observation is that while aggregating data, it can also replicate data along the aggregation paths, since replicating data does not introduce extra energy consumption. As result of replicating, less data needs to be offloaded in data offloading stage. Fig. 5 shows that when initiator E sends its one unit data passing storage node C, it replicates half of the data and stores it at C. Therefore, next in data offloading stage, node B only needs to offload the other half unit of E to A (combined with its own 0.5 unit after aggregation, B actually offloads one unit to A).[2] The offloading cost is 1.5, a 25% of improvement compared to offloading cost of 2 in Fig. 4(b).

[2]Note that B still has 1.5 units of overflow data after aggregation, but only one unit of them is offloaded.

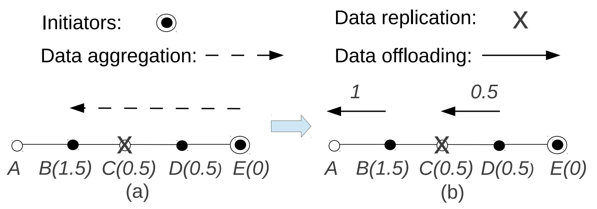


Fig. 5. Illustrating DAO-R. 0.5 unit of initiator E's data is replicated and stored at C in data aggregation stage. Thereafter in the data offloading stage, node B does not need to offload this part of data.

B. Problem Formulation of DAO-R.

We assume that l data aggregation paths $W_1^a, W_2^a, ..., W_l^a$ have already been found using Algorithm 1 in Section IV. Like in Section III, we assume that the overflow data consists of small units, each of which is x-bit and different units be offloaded to different storage nodes. The overflow data that needs to be offloaded after data aggregation falls into one of the three categories:

- $D^{'} = \bigcup_{j=1}^{l} D_j^{'}$: the overflow data of all the initiators. $D_j^{'}$ is the R amount of overflow data of I_j, initiator of W_j^a ($1 \leq j \leq l$). Note that each initiator's overflow data has been transmitted to the last aggregator of each aggregation path after data aggregation.
- $D^{''}$: the overflow data of all the aggregators, each having r amount of overflow data.
- $D^{'''}$: the overflow data that are not aggregated and are not on any aggregation path. $D^{'''}$ is empty if all the data nodes are on some aggregation paths.

When the R amount overflow data traverses each aggregation path starting from its initiator, it can replicate part or all R on storage nodes along the path. The data replication algorithms decide for each aggregation path W_j^a:

- the end node I_j that serves as initiator,
- a subset of $D_j^{'}$, denoted as D_j, of data units to replicate,
- a *replication function* $r : D_j \rightarrow V_s \cap W_j^a$, to replicate and store a data unit in D_j at a storage node in W_j^a, when R amount from I_j traverses along W_j^a,

under the constraint that the total size of replicated data on any storage node along any aggregation path can not exceed this node's available storage capacity: $|\{k|k \in D_j, r(k) = i\}| \cdot x \leq m, \forall i \in V_s \cap W_j^a$.

Let $D = \{D_1, ..., D_{|D|}\}$ denote the set of $|D|$ overflow data *after* aggregation, each is x-bit. With D_j replicated and stored on W_j^a, the rest $D_j^{'} - D_j$ amount still needs to be offloaded from the last aggregator of W_j^a in data offloading stage. Therefore,

$$D = \bigcup_{j=1}^{l} (D_j^{'} - D_j) \cup D^{''} \cup D^{'''} = (D^{'} - \bigcup_{j=1}^{l} D_j) \cup D^{''} \cup D^{'''}.$$

Let $s(j)$ denote the data node of any data unit $D_j \in D$. The data offloading algorithm is to decide an offloading function $o : D \rightarrow V_s$, to offload $D_i \in D$ from its data node $s(i) \in V_d$ to storage node $o(i) \in V_s$. Or equivalently, the data offloading algorithm is to decide a set of $|D|$ offloading paths:

$W_1^o, W_2^o, ..., W_{|D|}^o$, where W_j^o $(1 \leq j \leq |D|)$ starts from $s(j)$ and ends with $o(j)$, to minimize the offloading cost:

$$C_{off} = \sum_{1 \leq j \leq |D|} c(x, W_j^o), \quad (3)$$

under the constraint that the size of overflow data offloaded to any storage node in the network can not exceed its available storage capacity: $|\{j|1 \leq j \leq |D|, o(j) = i\}| \cdot x \leq m, \forall i \in V_s$. In DAO-R, since all the aggregation paths are given, the total aggregation cost is the same whether replication takes place or not. Therefore we need to minimize offloading cost, which is done by solving minimum cost flow algorithm [4, 5].

C. Data Replication Algorithms for DAO-R.

Selecting initiator for each aggregation path. After aggregation, among all data nodes on a particular aggregation path, the initiator has least amount of overflow data (zero), the last aggregator has most amount $(R + r)$, while others having the same r amount of overflow data. Therefore, having more available storage spaces around the last aggregator would make the data offloading next more energy-efficient. For example, in Fig. 2, since B has two neighboring storage nodes while E has zero, E is selected as the initiator.

Global Replication Algorithm. Once the initiator is selected for each aggregation path, it begins the data aggregation and replication process. Our Global replication algorithm works as follows. First, it offloads the r amount of data at all the aggregators and the overflow data that are not aggregated (line 1). Then for each aggregation path, it finds its available storage spaces left (line 3-7). The amount to be replicated is the smaller of R and the size of the available spaces (line 8). Next, while the R amount of data from initiator is traversing along the path performing data aggregation, it replicates this amount (line 9-16). Finally, it offloads each initiator's overflow data that has not been offloaded from the last aggregator of each path (line 18). Since it uses minimum cost flow algorithm to find the available spaces to replicate, Algorithm 2 takes a global perspective and is therefore referred to as *Global*. For ease of presentation, in algorithms below, $v = mc(O, G)$ means running the minimum cost flow algorithm on $G(V, E)$ to offload a set of data units O from its belonged aggregators, yielding a minimum energy cost v.

Algorithm 2: Global Data Replication Algorithm.
Input: All aggregation paths in $G(V, E)$: W_j^a $(1 \leq j \leq l)$
Output: C_{off}

```
0.   Notations:
     $u$: a node in $W_j^a$;
     $u.next$: the next node of $u$ in $W_j^a$;
     $z$: total size of data in $D_j$ that are not yet replicated;
     $avail(u)$: amount of available storage at node $u$;
     $avail_j$: amount of available storage at node $W_j^a$;
1.   $C_{off} = mc(D'' \cup D''', G)$;
2.   for each $W_j^a$ $(1 \leq j \leq l)$
3.       $avail_j = 0$, $u = I_j.next$;
4.       while ($u$ is not the last aggregator on $W_j^a$)
5            $avail_j = avail_j + avail(u)$;
6.           $u = u.next$;
7.       end while;
8.       $|D_j| = \min\{avail_j, R\}$;
9.       $u = I_j.next$, $z = |D_j|$;
10.      while ($z > 0$)
11.          if ($u \in V_s$)
12.              Replicate $avail(u)$ amount at $u$;
13.              $z = z - avail(u)$;
14.          end if;
15.          $u = u.next$;
16.      end while;
17.  end for;
18.  $C_{off} = C_{off} + mc(D' - \bigcup_{j=1}^{l} D_j, G)$;
19.  RETURN $C_{off}$.
```

Time complexity. The minimum cost flow algorithm takes $O(|V|^2|E|\log(|V|C))$, with $C = \max\{\frac{R+r}{x}, \frac{m}{x}\}$. Since each of the l $(l = O(|E|))$ aggregation paths can not have more than $|V|$ nodes, finding available storages and replicating data along each aggregation path takes $O(|V|)$. It takes $O(|E| \times |V|)$ for all the aggregation paths. Therefore, the time complexity of Algorithm 2 is $O(|V|^2|E|\log(|V|C)))$.

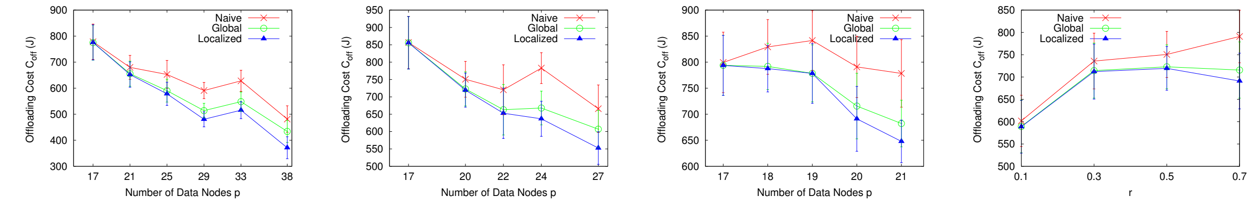
Localized Replication Algorithm. We give below definition.

Definition 1: (**Demand Number** $d(u)$ **of Storage Node** u) For any storage node u on any aggregation path, let $N(u)$ be all its one-hop neighbors. For each data node $v \in N(u) \cap V_d$, let $s(v)$ denote number of v's one-hop neighbors that are storage nodes. Then $d(u) = \sum_{v \in N(u) \cap V_d} \frac{1}{s(v)}$. □

Note that $s(v) \neq 0$ since v has at least one neighboring storage node u. The idea behind $d(u)$ is that the more number of data nodes surrounding u and the less number of storage nodes surrounding such data nodes, then more likely u will be used to store the overflow data from those data nodes. Therefore, we should replicate less amount of initiator's data D_j' on u. It is a localized algorithm since it works on each aggregation path one by one, and figures out the replicating amount for each storage node based on demand number. The algorithm works as follows. It first calculates demand number of each storage node on the aggregation path, and then sorts the storage nodes in non-descending order of their demand numbers. Next it calculates the amount of data to be replicated at storage node with smallest demand number, say u, as $\min(R/d(u), R)$. If not the whole part of R are replicated as well as not all the storage nodes on this path have been considered, it then calculates the amount to be replicated on the storage node with second-smallest demand numbers, and so on. It stops either the whole part of R is replicated or all storage nodes on the path are considered for replication.

Algorithm 3: Localized Data Replication Algorithm.
Input: All aggregation paths in $G(V, E)$: W_j^a $(1 \leq j \leq l)$
Output: C_{off}

```
0.   Notations:
```



(a) $r = 0.3R$.
(b) $r = 0.5R$.
(c) $r = 0.7R$.
(d) Varying r with $p = 20$.

Fig. 6. Performance comparison when $R = 5m$.

```
    $S_j = \{S_1^j, S_2^j, ..., S_{k_j}^j\}$: the set of $k_j$ ($k_j \geq 0$) storage
    nodes on $W_j^a$;
    $i$: index for storage nodes;
    $z$: total size of data in $D_j$ that are not yet replicated;
1.  for each $W_j^a$ ($1 \leq j \leq l$)
2.    for $u \in S_j$, calculate $d(u)$;
3.    Sort $S_j$ in non-descending order of $d(u)$;
4.    Let $d(S_1^j) \leq d(S_2^j) \leq ... \leq (S_k^j)$, WLOG;
5.    $z = R$, $i = 1$;
6.    while ($z > 0 \wedge i \leq k$)
7.      Replicate min($R/d(S_i^j), R$) amount data $D_j$ on $S_i^j$;
8.      $z = z - R/d(S_i^j)$;
9.      $D_j^{'} = D_j^{'} - D_j$;
10.     $i + +$;
11.   end while;
12. end for;
13. $D = \bigcup_{j=1}^{l}(D_j^{'} - D_j) \cup D^{''} \cup D^{'''}$;
14. $C_{off} = mc(D, G)$;
15. RETURN $C_{off}$.
```

Time complexity. For each aggregation path, finding demand number for a storage node takes $O(|E|^2)$ (assuming an adjacency list graph data structure), sorting storage nodes takes $|V|\log|V|$, and traversing each aggregation path takes $O(|V|)$. Each of the l aggregation path could have at most $|E|$ edges. Therefore it takes $O(|E|^3 + |V|\log|V| + V) = O(|E|^3)$ for all the aggregation paths. The minimum cost flow algorithm take $O(|V|^2|E|\log(|V|C))$, with $C = \max\{\frac{R+r}{x}, \frac{m}{x}\}$. Therefore the total time complexity of Algorithm 3 is $O(|E|^3)$.

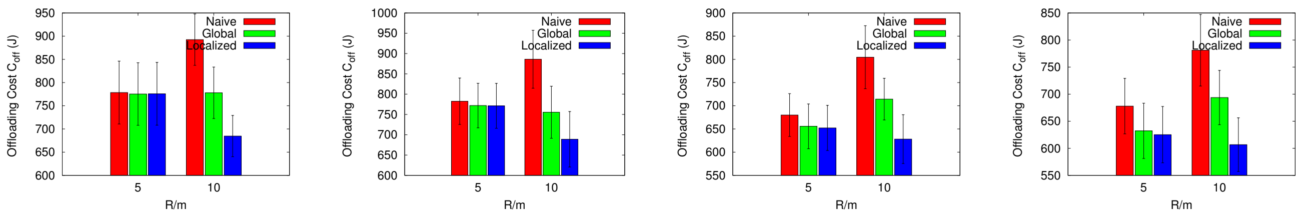
VI. Performance Evaluation

We compare the performances of DAO-N algorithm (**Naive**), Global Replication Algorithm (**Global**), and the Localized Replication Algorithm (**Localized**). 100 sensors are uniformly distributed in a region of 1000m $\times$ 1000m. Transmission range of sensor nodes is 250m. Unless otherwise mentioned, R=5m where m=512KB and R=2560KB. We also vary R/m to 10, with m=512KB while R=5120KB. In all plots, each data point is an average over 20 runs, and the error bars indicate 95% confidence interval.

Effect of Varying p. Fig. 6 shows the offloading cost of three algorithms when $R = 5m$, with r varied from $0.3R$, $0.5R$, to $0.7R$. In each case, we find the valid range of p (using Equation 2) and increase p from its smallest to largest valid values. It clearly show that Localized performs better than Global, which performs better than Naive. In most cases, performance improvement of Localized upon Naive is around 30%. In general, the offloading costs decrease with increase of p in each case. This is because the total size of data to be offloaded after aggregation, which is $(|V| - p) \cdot m$, decreases with increase of p. There are a few cases, however, show that offloading cost increase with increase of p. This is because offloading paths could possibly become longer, even though the total size of offloaded data gets smaller with increase of p. Localized performs better than Global is because for Global, after offloading according to the minimum cost flow algorithm (line 1 of Algorithm 2), most of the storage nodes on the aggregation paths are filled with offloaded data, leaving not much space for data replication. For Localized, however, it always replicates based upon the calculated demand number of each storage node therefore replicating more wisely. We notice a few cases wherein the offloading cost increases with the increase of p (such as $p = 33$ in Fig. 6(a)), this is again because offloading paths could possibly become longer, even though the total size of offloaded data is smaller.

Effect of Varying r. Fig. 6(d) shows the performance comparison by varying r, with $R = 5m$ and $p = 20$. It shows that the offloading cost of all three algorithms generally increase with the increase of r. This can be explained by the non-uniform distribution of overflow data when r increases, whereas uniformly distributed data can be offloaded more easily. Initially, overflow data are uniformly spread inside the network. Now by increasing r, q increases (Equation 1) and the aggregation paths get longer. This means that more data nodes get their overflow data reduced. Therefore overflow data are no longer uniformly distributed with increase of q. Fig. 6(d) also shows that when r is increased from 0.5 to 0.7, the offloading costs for both Global and Localized slightly decrease. This is due to the same reason above as well as the effect of data replication – there are potentially more storage nodes on the aggregation paths when they get longer, therefore benefiting the data replication.

Effect of Varying R/m. Fig. 7 investigates the effect of R/m. Since $R/m = 5$ and $R/m = 10$ have common valid range



(a) $p = 17$. (b) $p = 19$. (c) $p = 21$. (d) $p = 23$.

Fig. 7. Performance comparison by varying R/m. Here, $r = 0.3$.

of p, $17 \leq p \leq 23$, when $r = 0.3R$, we adopt these set of parameters. We vary p from 17, 19, 21, to 23. First we observe that the offloading cost for Naive is always higher in $R/m = 10$ than in $R/m = 5$. This can be explained as below. R is doubled from $R/m = 5$ to $R/m = 10$. However, since the total available storage, being $(|V| - p) \times m$ is fixed, the same amount of overflow data after aggregation are offloaded. Therefore, more amount of overflow data needs to be reduced for $R/m = 10$, resulting in increased q (q equals 6 and 18 in $R/m = 10$ and $R/m = 5$, respectively). Therefore it causes more non-uniformity of data distribution for $R/m = 10$, causing increased data offloading cost. However, when increasing from $R = 5m$ to $R = 10m$, the offloading cost for Localized always decreases while could be either way for Global. This again demonstrates the effectiveness of our replication algorithms – with increased q, longer aggregation paths exist, allowing more storage nodes to store replicated data.

VII. Conclusion and Future Work

We propose DAO-R to solve overall storage overflow problem in sensor networks. DAO-R employs data replication to integrate data aggregation and data offloading to achieve energy-efficiency. We show via simulations that DAO-R outperforms the existing approach DAO-N by around 30% in terms of energy consumption. As future work, we will consider that different data nodes could have different amount of overflow data as well as different storage nodes could have different storage capacity. We will also consider more dynamic scenarios, for example, some nodes could deplete their battery power, and design distributed algorithm.

Acknowledgment

This work was supported in part by NSF Grant CNS-1116849.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[2] R. Cristescu, B. Beferull-Lozano, M. Vetterli, and R. Wattenhofer. Network correlated data gathering with explicit communication: Np-completeness and algorithms. *IEEE/ACM Transactions on Networking*, 14:41–54, 2006.

[3] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. In *Proc. the 1st international conference on Embedded networked sensor systems (SenSys)*, 2003.

[4] A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22(1):1–29, 1997.

[5] A. V. Goldberg. Andrew goldberg's network optimization library, 2008. http://www.avglab.com/andrew/soft.html.

[6] J. Heidemann, M. Stojanovic, and M. Zorzi. Underwater sensor networks: applications, advances and challenges. *Phil. Trans. R. Soc. A*, 370:158 – 175, 2012.

[7] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proc. of HICSS 2000*.

[8] X. Hou, Z. Sumpter, L. Burson, X. Xue, and B. Tang. Maximizing data preservation in intermittently connected sensor networks. In *Proc. of MASS 2012*, pages 448–452.

[9] S. Jain, R. Shah, W. Brunette, G. Borriello, and S. Roy. Exploiting mobility for energy efficient data collection in wireless sensor networks. *MONET*, 11(3):327–339, 2006.

[10] J. Jeong, X. Jiang, and D. Culler. Design and analysis of micro-solar power systems for wireless sensor networks. In *Proc. of INSS 2008*.

[11] T. Kuo and M. Tsai. On the construction of data aggregation tree with minimum energy cost in wireless sensor networks: Np-completeness and approximation algorithms. In *Proc. of INFOCOM 2012*.

[12] J. Li, A. Deshpande, and S. Khuller. On computing compression trees for data collection in wireless sensor networks. In *Proc. of INFOCOM 2010*, pages 2115–2123.

[13] D. Luo, X. Zhu, X. Wu, and G. Chen. Maximizing lifetime for the shortest path aggregation tree in wireless sensor networks. In *Proc. of INFOCOM 2011*, pages 1566 – 1574.

[14] Ioannis Mathioudakis, Neil M. White, and Nick R. Harris. Wireless sensor networks: Applications utilizing satellite links. In *Proc. of PIMRC 2007*.

[15] D. Mosse and G. Gadola. Controlling wind harvesting with wireless sensor networks. In *Proc. of IGCC 2012*.

[16] L. Mottola. Programming storage-centric sensor networks with squirrel. In *Proc. of IPSN 2010*, pages 1–12.

[17] B. Tang, N. Jaggi, H. Wu, and R. Kurkal. Energy efficient data redistribution in sensor networks. *ACM Transactions on Sensor Networks*, 9(2), May 2013.

[18] B. Tang and Y. Ma. Dao: Overcoming overall storage overflow in base station-less sensor networks via energy efficient data aggregation. Technical report.

[19] Y. Wu, S. Fahmy, and N. B. Shroff. On the construction of a maximum-lifetime data gathering tree in sensor networks: Np-completeness and approximation algorithms. In *Proc. of INFOCOM 2008*.

[20] X. Xu, X. Li, X. Mao, S. Tang, and S. Wang. A delay-efficient algorithm for data aggregation in multihop wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 22:163 – 175, 2011.

[21] X. Xue, X. Hou, B. Tang, and R. Bagai. Data preservation in intermittently connected sensor networks with data priorities. In *Proc. of SECON 2013*, pages 65–73.