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—**When sensor network applications are deployed in an inaccessible or inhospitable region, or under extreme weather, it is usually not viable to install a long-term base station in the field to collect data. The generated sensory data is therefore stored inside the network first, waiting to be uploaded. However, when more data is generated than available storage spaces in the entire network can possibly store, and uploading opportunities have not arrived, data loss becomes inevitable. We refer to this problem as *overall storage overflow* in sensor networks. To overcome overall storage overflow, existing research designs a two-stage approach as below. First, by taking advantages of spatial correlation that commonly exists among sensory data, it aggregates overflow data to the size that can be accommodated by the available storage capacity in the network. Then, it offloads the aggregated data into the network to be stored. We refer to this naive two-stage approach as DAO-N. DAO-N is NP-hard. In this paper, we demonstrate that this approach does not necessarily achieve good performance in terms of energy consumption. We propose a more unified framework that is based upon data replication techniques in order to solve overall storage overflow and improve the performance of DAO-N. We refer to our approach as DAO-R. Specifically, we design two energy-efficient data replication algorithms to integrate data aggregation and data offloading seamlessly. We also give a sufficient condition under which DAO-R can be solved optimally. Via extensive simulations, we show 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 network applications such as underwater sensor networks [12] and wind and solar harvesting [16, 29]. 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 uploaded by periodic visits of robots or data mules [15], or by low rate satellite links [27].

Despite the advances in energy-efficient flash storage [30] with good compression algorithms (data is compressed before stored) and good aging algorithms (fidelity of older data is reduced to make space for newer data), storage is still a serious resource constraint of sensor nodes. For example, in an acoustic sensor network that monitors bird vocalizations in a forest [24, 25], an acoustic sensor that has a 1GB flash memory and is designed to sample the entire audible spectrum will run out of its storage in just seven hours. 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.

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*. To avoid data loss in node storage overflow, the technique of *data offloading* is usually employed: the overflow data is offloaded from its data node to other nodes with available storages (referred to as *storage nodes*) before uploading opportunities arrive.[1] In particular, different data offloading techniques have been proposed in existing research with the goals of either minimizing the total energy consumption during data offloading [34], or maximizing the minimum remaining energy of storage nodes to prolong network lifetime [14], or offloading the most useful information under battery energy constraint considering data could have different priorities [39]. However, these techniques did not address the second level of data overflow viz. *overall storage overflow* introduced below, which is what this paper focuses.

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, therefore not all the overflow data can be stored and kept in the network using aforesaid data offloading techniques. Here we use an underwater sensor network application that monitors coral reefs and fisheries [3, 35] to further motivate overall storage overflow. In this application, each underwater sensor node is equipped with a camera and can take pictures of the surrounding environment. An autonomous underwater vehicle (AUV) is then dispatched

[1]Sensor nodes that generate data but have not depleted their storage spaces are considered as storage nodes, as their storage spaces can be used to store overflow data from data nodes.

periodically to upload the generated images from sensor nodes. Each sensor node has 512KB of flash storage and the resolution of the images is 255×143 pixels. Consider that each pixel is one byte (each byte is either red, green, or blue), each image thus requires around 36KB of data storage. Suppose there are 100 sensor nodes deployed in this application, 10 of which are generating one image per minute. It will then take less than three hours to exhaust the available storage in the network and to reach overall storage overflow. In the same scenario, even using the latest parallel NAND flash technology with 16GB storage [20] and taking typical 640×480 JPEG color images, it will take less than one day to reach overall storage overflow. If the AUV cannot be dispatched timely due to inclement and stormy weather, and no other appropriate actions are taken, discarding valuable data becomes inevitable.

Therefore overall storage overflow is a more severe problem compared to the node storage overflow. How to overcome overall storage overflow and prevent data loss becomes a new challenge. Below we introduce two main techniques used in this paper viz. data aggregation and data replication.

- **Data Aggregation.** In order to achieve fine-grain monitoring, it usually requires dense sensor nodes deployment in wireless sensor networks. Due to high density of nodes, spatially proximal sensor observations are highly correlated [17]. Therefore, 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 [33, 34] implies a two-stage approach. First, it aggregates and reduces the size of overflow data such that it can be accommodated by the available storage in the network [33]. Then, it offloads the data into the network [34] to be stored. That is, it treats data aggregation and data offloading as two independent stages that take place sequentially and solves each separately. We refer to this naive and straightforward approach as naive data aggregation and offloading (DAO-N). The existing research present an approximation algorithm [33] to achieve energy-efficient data aggregation and an minimum cost flow based optimal algorithm [34] for energy-efficient data offloading.

	MSP-430 instruction	Toshiba NAND Read	Toshiba NAND Write	CC2420 Radio Tx	CC2420 Radio Rx
Energy (μ J/byte)	0.0008	0.004	0.009	1.8	2.1
Ratio	1	5	11	2250	2600

TABLE I
PER-BYTE ENERGY USAGE: COMMUNICATION, STORAGE, AND COMPUTATION (TABLE 5, [28]).

- **Data Replication.** In this paper, we show that even by solving each stage independently with good performance, DAO-N does not necessarily achieve best overall performance. The key observation is that instead of taking place strictly one after the other, data offloading and data aggregation can actually take place simultaneously to achieve better energy efficiency overall. We propose a unified approach that integrates data aggregation and data offloading via data replication techniques. 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. We refer to our integrated approach as DAO-R.

The novelty of DAO-R is that replicating data in the process of data aggregation achieves the effect of data offloading. Mathur et al. [28] did extensive investigation of the energy cost of NAND-flash storage, in comparison to computation and communication in sensor network systems, as shown in Table I. They found that radio transmission and reception represent a more than 200-fold increase in energy usage over writing to NAND flash. As data replication is simply writing a copy of the data into the storage, its energy cost can be considered as negligible compared to wireless communication cost. Therefore in this paper we assume that data replication does not incur extra energy cost. We also give a sufficient condition that solves DAO-R optimally. Finally, we show via extensive simulations that DAO-R outperforms DAO-N by around 30% in terms of energy consumption under different network parameters. Note that in this paper we do not consider how to upload data from the network to base station when uploading opportunities are available. Data mules or mobile data collectors can be used to upload data using techniques in [26] and [22].

Paper Organization. The rest of the paper is organized as follows. Section II discusses state-of-the-art and related work. In Section III, we introduce the overall storage overall problem, present the 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. We also give an efficient condition under which DAO-R can be solved optimally. 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

When sensor networks are deployed in challenging environments such as inaccessible regions or extreme weather, it is not viable to deploy a base station with power outlets near the sensor network to collect data. How to preserve data in sensor networks in the absence of the base station has become a active research in recent years. There are active system research that focused on disconnection-tolerant operations in the absence of the base station [24, 25, 36, 40]. he authors in these papers design acoustic solar-powered sensor networks, which monitor the social behavior of animals in the wild. Since no base station is available, they design cooperative distributed storage systems specifically for disconnected operations of sensor networks, to improve the utilization of the networks data storage capacity. Other research instead took an algorithmic approach by focusing on the optimality of the solutions [14, 34, 39]. Tang et al. [34] addressed the energy-efficient data redistribution problem in data-intensive sensor networks. Hou et al. [14] studied how to maximize the minimum remaining energy of the nodes that finally store the data, in order to store the data for long period of time. Xue et al. [39] considered that sensory data from different source nodes have different importance, and study how to preserve data with highest importance. Tang and Ma [33] 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, they either does not address the overall storage overflow problem or important research problem such as how to integrate data aggregation and data offloading to further save energies is not addressed.

Ganesan et al. [8] 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. Moreover, we study the hardness of the problems and solve it optimally or sub-optimally in terms of total energy consumption.

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 during data collection. 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) [23, 37], some are used to minimize the total energy consumption or communication cost [19, 21], and some to reduce the delay of data gathering [38]. In contrast, the overall storage overflow problem studied in this paper takes place when the base station does not exist and there is not enough storage to store all the overflow data. Consequently, data aggregation in has very different goal compared to traditional data aggregation – 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.

Our work addresses data resiliency issue to overcome overall storage overflow in sensor networks. Data resiliency refers to the ability of long-term viability and availability of data despite of insufficiencies of (or disruptions to) the physical infrastructure that stores the data. Many data resilience techniques have been proposed to overcome against different causes of data loss in sensor networks. Ghose et al. [9] are among the first to propose Resilient Data-Centric Storage (R-DCS) to achieve resilience by replicating data at strategic locations in the sensor network. Ganesan [7] consider constructing partially disjoint multipaths to enable energy efficient recovery from failure of the shortest path between source and sink. Recently, network coding techniques are used to recover data from failure-prone sensor networks. Albano et al. [2] propose in-network erasure coding to improve data resilience to node failures. Kamra et al. [18] propose to replicating data compactly at neighboring nodes using growth codes that increase in efficiency as data accumulates at the sink. However, all these data resilience measures adopt the traditional sensor network model wherein base stations are always available near or inside the networks. We consider a more challenging scenario wherein base stations are not available due to the harsh environments therefore data must be stored and preserved inside the network. Consequently, overall storage overflow and its resulted data resiliency becomes new challenges.

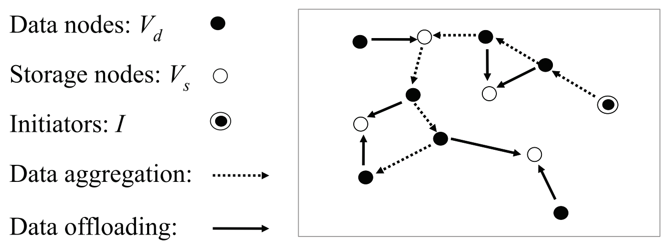


Fig. 1. An illustration of overall storage overflow.

III. **Overall Storage Overflow Problem [33]**

In this section, we introduce overall storage overflow problem, including its network model, data spatial correlation model, energy model. We also discuss its feasible condition, under which overall storage overflow occurs in the network.

Network 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. As shown in Fig. 1, the sensor network consists of *data nodes* (with overflow data) and *storage nodes* (with available storage spaces). There are p data nodes (denoted as V_d), each has generated R bits of overflow data over some period of time. The other $|V| - p$ nodes are storage nodes, denoted as V_s, each of which has m bits of available storage spaces. The total size of the overflow data from the data nodes exceeds the total size of the storage spaces of the storage nodes, resulting in overall storage overflow. Therefore we have $p \times R > (|V| - p) \times m$, which gives

$$p > \frac{|V|m}{m + R}. \tag{1}$$

To aggregate data and to reduce data size, 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 can aggregate its own overflow data due to spatial correlation existing between its own overflow data and the one from initiators (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 overflow 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. As shown in Fig. 1, 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.

Data Correlation Model. We adopt data correlation model proposed in [6], in which the correlation function is derived as an approximation of the conditional entropy. Following this model, 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 $H(i) = r \leq R$ bits if data node i receives side information from at least another data node $j_1, ..., j_p$, $j_k \in V_d \wedge j_k \neq i, 1 \leq k \leq p$. That is, a data node aggregates its overflow data from R to r if it receives overflow data from at least one initiator. We are aware several entropy-based correlation models such as the ones proposed in [5, 31]. We adopt this correlation model because a) it is a simple and distributed coding strategy, making it easy to implement in large-scale sensor network and b) it is a realistic model that approximates the case where the correlation function between two nodes decreases with their distance [6].

In this data correlation model, we assume that each data node can be either an initiator, or an aggregator, or none of them, but not both of them. An initiator cannot be an aggregator because its overflow data has served as side information for other data nodes (i.e., aggregators) to aggregate. An aggregator cannot be an initiator since its aggregated data loses the side information needed for others aggregators' aggregation. We also assume that the overflow data of an aggregator can be aggregated only once, with size reduced from R to r, even though it can be visited by the same or different initiators' overflow data multiple times (if that is more energy efficient).

Feasible Overall Storage Overflow. Inequality 1 only gives the sufficient condition for overall storage overflow. However, not all the overall storage overflow can be solvable using data aggregation and offloading. One extreme example is that all the nodes in the network are data nodes and there are no storage nodes at all. In this case there is obviously no way it can aggregate the overflow data and offload them to storage nodes. Below we introduce *feasible overall storage overflow*, wherein not only overall storage overflow occurs, but also it is possible to solve it. It must satisfy three conditions.

a). There is an overall storage overflow, which gives Inequality 1.

b). There are enough number of aggregators to visit in order to reduce the data size.

c). The data after aggregation can fit in the available storage in the network.

TABLE II
NOTATION SUMMARY

Notation	Explanation
V, $\|V\|$	Set and number of sensor nodes
V_d, p	Set and number of data nodes
V_s	set of storage nodes
q	Number of aggregators needed
m	Storage capacity of a storage node in $V - V_d$
R	Overflow data size at each data node before aggregation
r	Overflow data size at each data node after aggregation
I, l	Set and number of initiators, $1 \leq l \leq (p - q)$
I_j	j^{th} initiator, $1 \leq j \leq l$
D	The set of data packets after aggregation
$D^{init}(D_j^{int})$	Overflow data of all initiators (j^{th} initiator)
D^{aggr}	Overflow data of all aggregators
$D^{non-aggr}$	Overflow data that are not aggregated
$D^r(D_j^r)$	Replicated overflow data of all initiators (j^{th} initiator)

From b) and c), since the size of overflow data that needs to be reduced is $p \times R - (|V| - p) \times m = p \times (R + m) - |V| \times m$, and visiting one aggregator reduces its overflow data size by $(R - r)$, the number of needed aggregators, denoted as q, is

$$q = \lceil \frac{p \times R - (|V| - p) \times m}{R - r} \rceil = \lceil \frac{p \times (R + m) - |V| \times m}{R - r} \rceil. \quad (2)$$

Given a specific p value and its corresponding q value calculated from Equation 2, at most $p - q$ data nodes can then be selected as initiators. Finally, we find the valid range of p in a feasible overall storage overflow.

Among all the p data nodes, since at least one data node needs to be initiator (therefore can not be an aggregator), we can have at most $p - 1$ aggregators; i.e., $q \leq p - 1$. Combining it with Equation 2, we have

$$\lceil \frac{p \times (R + m) - |V| \times m}{R - r} \rceil \leq p - 1, \quad (3)$$

which gives:

$$p \leq \lfloor \frac{|V|m - R + r}{m + r} \rfloor. \quad (4)$$

Combining Inequalities 1 and 4 , the valid range of p for feasible overall storage overflow is therefore

$$\frac{|V|m}{m + R} < p \leq \lfloor \frac{|V|m - R + r}{m + r} \rfloor. \quad (5)$$

Energy Model. We adopt the well-known first order radio model [13] for energy consumption in wireless communication. 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 $E_t(R, l_{u,v}) = E_{elec} \times R + \epsilon_{amp} \times R \times l_{u,v}^2$, 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. We consider two energy costs.

Aggregation Cost. In data aggregation, the route that the R-bit overflow data from each initiator traverses could be either a path or a walk. This is because for enough number of aggregators to be visited in an energy-efficient way in order to reduce overflow data size, an aggregator could be visited multiple times. We refer to such path or walk as *aggregation path/walk*, and $c(R, W)$ as the *aggregation cost* of R-bit data traversing along W starting from its initiator.

Offloading Cost. In data offloading, however, the route that the overflow data (either aggregated, with size of r, or not aggregated, with size of R) traverses is always a path that minimizes the energy consumption for offloading the overflow data. 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, we assume that the overflow data at each data node is splittable. That is, each overflow data can be splitted into small units, referred to as *data packets*, each of which is then offloaded to storage nodes. Let the size of each data packet be x-bit, where $x \ll r \leq R$, then $c(x, W)$ is the *offloading cost* of offloading this x-bit data packet from its data node to a storage node along path W. Any overflow data thus has $\frac{R}{x}$ data packets before aggregation and has $\frac{r}{x}$ data packets after aggregation.

Our goal is to overcome overall storage overflow by finding a data aggregation and offloading scheme that minimizes the *total energy cost*, the sum of aggregation cost and offloading cost. In Section IV, we formulate and present a naive, two-stage approach, referred to as DAO-N, that treats data aggregation and data offloading as two independent stages. We design time-efficient approximation algorithm and optimal algorithm to solve each stage respectively. In Section V, we propose and formulate a more unified as well as more energy-efficient approach, referred to as DAO-R, that employs data replication techniques to integrate data aggregation and offloading.

EXAMPLE 1: Fig. 2 illustrates the overall storage problem with a linear sensor network of seven nodes. Nodes A, D, F, and G are data nodes, while B, C, and E are storage nodes. Each data node has one unit of overflow data (i.e., $R = 1$). Each storage node has one unit of storage space (i.e., $m = 1$). As there are total 4 units of overflow data while only 3 units of available storage, overall storage overflow occurs. $r = 1$. Number of aggregators q is calculated as 2 using Equation 2. Thus at most two data nodes can be selected as initiators. Note this scenario is also a feasible overall storage overflow, as the valid range of p given by Inequality 5 is a single value of 4. The energy consumption on any edge is 1 for one unit of data. □

In Section IV and V, we use Example 1 to show the limitations of DAO-N and demonstrate how DAO-R improves upon it. In particular, we show how DAO-N and DAO-R select the initiators and corresponding aggregation paths to solve overall storage overflow problem differently.

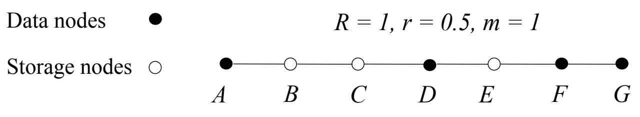


Fig. 2. An example of overall storage overflow.

IV. DAO-N: A Naive Two-Stage (Data Aggregation and Data Offloading) Approach

Problem Formulation of DAO-N. Given an instance of feasible overall storage overflow, the DAO-N first decides a set of l $(1 \leq l \leq (p-q))$ initiators $I \subset V_d$, and a corresponding set of l *aggregation paths/walks*:[2] $W_1^a, W_2^a, ..., W_l^a$, where W_j^a $(1 \leq j \leq l)$ starts from a distinct initiator $I_j \in I$, and $|\bigcup_{j=1}^{l}\{W_j^a - \{I_j\} - G_j\}| = q$ (G_j is the set of storage nodes in W_j^a).

Let $D = \{D_1, ..., D_{|D|}\}$ denote the set of $|D|$ overflow data packets *after* above aggregation, each is x-bit. Let $s(i)$, where $1 \leq i \leq |D|$, denote the data node of data packet D_i. DAO-N then decides an offloading function $o : D \rightarrow V_s$, to offload data packet $D_i \in D$ from its data node $s(i) \in V_d$ to storage node $o(i) \in V_s$; or equivalently, a set of $|D|$ offloading paths: $W_1^o, W_2^o, ..., W_{|D|}^o$, where W_i^o $(1 \leq i \leq |D|)$ starts from $s(i)$ and ends with $o(i)$.

The goal of DAO-N is to minimize the total energy cost in aggregation and offloading:

$$C_{total} = \sum_{1 \leq j \leq l} c(R, W_j^a) + \sum_{1 \leq j \leq |D|} c(x, W_j^o), \qquad (6)$$

under the constraint that the size of overflow data offloaded to any storage node can not exceed its available storage capacity $|\{j | 1 \leq j \leq |D|, o(j) = i\}| \cdot x \leq m, \forall i \in V_s$.

NP-Hardness of DAO-N. DAO-N treats data aggregation and data offloading as two separate and independent stages, therefore 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 [33].

Data Aggregation Algorithm. Below we present an approximation algorithm, which yields aggregation cost that is at most $(2 - \frac{1}{q})$ times of the optimal [33].

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. We refer to $G'(V', E')$,

[2]For ease of presentation, we use aggregation path instead of aggregation path/walk for the rest of paper.

which will be used in next few steps to find initiators and their aggregation paths, as the *aggregation graph*.

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. We refer to the resulted graph as *q-edge forest*.

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). We refer to the resulted graph as *shortest-path q-edge forest*.

5). For each connected component of the resulted shortest-path q-edge forest, if it is linear, it randomly selects one end as the initiator, starts from it 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 selects u as the initiator, starts from it 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.

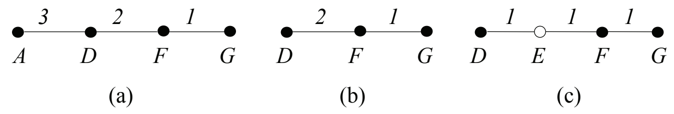


Fig. 3. Execution of Algorithm 1 on the sensor network graph $G(V, E)$ in Fig. 2. (a) Aggregation graph $G'(V', E')$. (b) 2-edge forest. (c) shortest-path 2-edge forest. The number on each edge indicates its weight.

The running time of Algorithm 1 is $O(|E|\log|E|)$ with disjoint-set data structure [4]. Fig. 3 and Fig. 4(a) show the execution of Algorithm 1 on the linear sensor network in Fig. 2. Fig. 4(a) shows the aggregation graph, which is a linear topology for this simple example. The numbers above edges indicate their weights. Fig. 3(b) shows the 2-edge forest. And Fig. 3(c) shows the shortest-path 2-edge forest, which has only one connected component thus needs to select one initiator.

Fig. 4(a) shows one of the two data aggregation solutions following Step 5 in Algorithm 1 (to illustrate data offloading algorithm below, we include nodes A, B, and C as well). Data node D is selected as the initiator and the aggregation path is D, E, F, and G. The aggregation cost is 3, which happens to be optimal in this small network. Numbers in parentheses are the sizes of overflow data at each data node after aggregation (but before offloading). In particular, there is no overflow data left at initiator D, and there are 0.5 and 1.5 units of data at data nodes F and G respectively. Plus 1 unit of data at data node A, which does not participate in the data aggregation process, the total size of the overflow data is now 3. The other solution, wherein data node E is selected as the initiator, will be discussed in Section V to show how DAO-R improves upon DAO-N.

Data Offloading Algorithm. As there are enough storage spaces available to store the overflow data after data aggregation, data offloading is to offload all the overflow data, whether it is aggregated or not, from its data node to storage node. The goal of data offloading is to minimize the energy cost incurred in the offloading process. Tang et al. [34] 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 [10, 11], with time complexity of $O(|V|^2 \cdot |E| \cdot log(|V| \cdot C))$. Here $C = \max\{\frac{R+r}{x}, \frac{m}{x}\}$ is the maximum capacity of an edge in the transformed graph.

Fig. 4(b) shows the data offloading solution that follows data aggregation in Fig. 4(a). It shows that 0.5 unit of data at F and 0.5 unit of data at G are offloaded to storage node E, 1 unit of data at G is offloaded to storage node C, and 1 unit of data at A is offloaded to storage node B. The offloading cost is 6.5. The total cost in this two-stage solution is therefore 9.5.

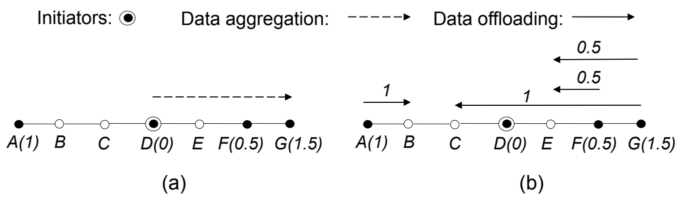


Fig. 4. One naive two-stage solution with D being the initiator. (a) Data aggregation stage: values in parentheses are sizes of overflow data after aggregation. Data aggregation path is shown in dashed arrow line. (b) Data offloading stage: values on the arrowed lines are sizes of overflow data that is offloaded from its data node to a storage node. Data offloading path is shown in solid arrow line.

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, which integrates data aggregation and offloading. Finally we solve DAO-R by designing two time-efficient data replication algorithms.

A. Limitations of DAO-N.

Selection of Initiators. Another naive two-stage solution for Example 1 is shown in Fig. 5. Fig. 5(a) shows that data node G is selected as initiator and the aggregation path is G, F, E, and D, with the aggregation cost as 3. Fig. 5(b) shows the data offloading: A offloads its 1 unit of data to B, D offloads 1 unit of data to C and the rest 0.5 to E, while F offloads its 0.5 unit of data to E as well. The offloading cost of 3. The total cost in this two-stage solution is thus 6. Compared to total cost of 9.5 in Fig. 4, it is an 37% improvement.

Therefore, even though the solution in Fig. 4 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. In DAO-R, we will propose a scheme that finds an initiator for each aggregation path that yields minimum data offloading cost.

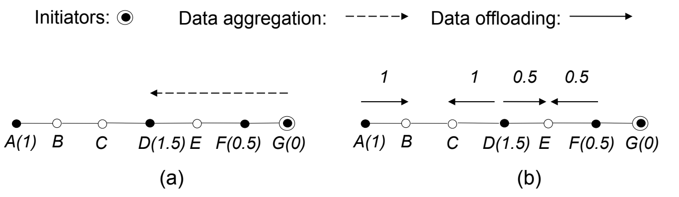


Fig. 5. Another naive two-stage solution solution with G being the initiator.

Effect of Data Replication. Even though Fig. 5 gives optimal combined total energy cost of data aggregation and offloading, its performance can be further improved using data replication. Our key observation is that when the initiator's overflow data traverses along the aggregation path, part or entirety of it can be replicated on the storage nodes along the aggregation path. We assume that replicating data does not incur any extra energy consumption. As such, this amount of replicated data does not needs to be offloaded from the last aggregator in data offloading stage, saving energy cost.

Fig. 6 shows that when initiator G sends its one unit data passing storage node E, it replicates 0.5 unit of data and stores it at E. Next, in data offloading stage, node D only needs to offload 1 unit of data (including the other 0.5 unit of G's overflow data as well as D's own 0.5 unit of overflow data) to C, costing one unit of energy. Plus 1 unit of A's offloading cost and 0.5 unit of F' offloading cost, the total offloading cost is 2.5. Even though D still has 1.5 units of overflow data after aggregation, only one unit of them is offloaded due to replication. The total cost is thus 5.5. This is a 8% of improvement upon the total cost of 6 in Fig. 5, wherein no data replication is employed.

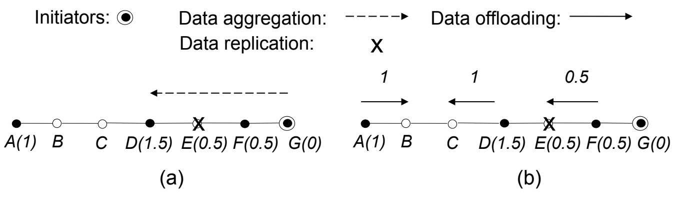


Fig. 6. Illustrating DAO-R. 0.5 unit of initiator G's data is replicated and stored at E in data aggregation stage. Thereafter in the data offloading stage, node D does not need to offload this part of data.

B. Problem Formulation of DAO-R.

We assume that l data aggregation paths $W^a_1, W^a_2, ..., W^a_l$ have already been found using Algorithm 1 in Section IV. Like in Section III, we assume that the overflow data consists of small data packets, each is x-bit and different data packets can be offloaded to different storage nodes. As all the aggregation paths are given, the total aggregation cost is the same whether replication takes place or not in DAO-R. Therefore we only need to minimize offloading cost while considering replication.

The overflow data that needs to be offloaded after data aggregation falls into one of the three categories:

- $D^{init} = \bigcup_{j=1}^{l} D^{init}_j$: the overflow data of all the initiators. D^{init}_j is the R amount of overflow data of I_j, initiator of W^a_j $(1 \le j \le l)$. Note that after data aggregation, each initiator's overflow data has been transmitted to the last aggregator on each aggregation path.
- D^{aggr}: the overflow data of all the aggregators, each having r amount of overflow data.
- $D^{non-aggr}$: the overflow data of data nodes that are not on any aggregation path therefore are not aggregated. $D^{non-aggr}$ 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^a_j:

- the data node, denoted as I_j, that serves as initiator,
- a subset of D^{init}_j, denoted as D^r_j, of data packets to replicate,
- a *replication function* $r : D^r_j \to V_s \cap W^a_j$, to replicate and store a data packet in D^r_j at a storage node in W^a_j, when R amount of overflow data from I_j traverses along W^a_j,

under the constraint that the total size of replicated data on any storage node can not exceed this node's available storage capacity: $\sum_{1 \le j \le l} |\{k | k \in D_j, r(k) = i\}| \cdot x \le m, \forall i \in V_s \cap W^a_j$.

Recall that $D = \{D_1, ..., D_{|D|}\}$ is the entire set of data packets to be offloaded after data aggregation. For each aggregation path W^a_j, with D^r_j replicated and stored on W^a_j, the rest $D^{init}_j - D^r_j$ amount of overflow data from I_j still needs to be offloaded from the last aggregator of W^a_j in data offloading stage. Therefore,

$$\begin{aligned} D &= D^{aggr} \cup D^{non-aggr} \cup \bigcup_{j=1}^{l} (D^{init}_j - D^r_j) \\ &= D^{aggr} \cup D^{non-aggr} \cup (D^{init} - \bigcup_{j=1}^{l} D^r_j). \end{aligned} \quad (7)$$

Let $s(j)$ denote the data node of any data packet $D_j \in D$. The data offloading algorithm is to decide an offloading function $o : D \to V_s$, to offload $D_j \in D$ from its data node $s(j) \in V_d$ to storage node $o(j) \in V_s$. Or equivalently, the data offloading algorithm is to decide a set of $|D|$ offloading paths: $W^o_1, W^o_2, ..., W^o_{|D|}$, where W^o_j $(1 \le j \le |D|)$ starts from $s(j)$ and ends with $o(j)$, to minimize the offloading cost $\sum_{1 \le j \le |D|} c(x, W^o_j)$, under the constraint that the size of overflow data offloaded to any storage node in the network can not exceed its available storage capacity: $|\{j | 1 \le j \le |D|, o(j) = i\}| \cdot x \le m, \forall i \in V_s$. Minimizing data offloading cost can be solved by minimum cost flow algorithm in polynomial time [10, 11].

C. Data Replication Algorithms for DAO-R.

Selecting initiator for each aggregation path. We have below observations in order to decide which data node gets picked to be the initiator for each aggregation path. After aggregation, among all data nodes on a particular aggregation path, the initiator has least amount of overflow data, which is zero; the last aggregator visited has most amount, which is $R + r$; and other aggregators have r amount of overflow data. Having more available storage nodes close to overflow data will certainly make data offloading more energy efficient, as overflow data can be offloaded to storage spaces close by. Therefore it is preferred that more storage spaces are around the last aggregator for energy-efficient data offloading. For example, in Fig. 2, since D has two neighboring storage nodes while G has zero, G is selected as the initiator. Once the initiator is selected for each aggregation path, it begins the data aggregation and replication process simultaneously. Below we present two data replication algorithms for this process.

Global Replication Algorithm. Algorithm 2 works as follows. First, it offloads all the overflow data on the aggregators D^{aggr} as well as all the overflow data that are not aggregated $D^{non-aggr}$ using minimum cost flow algorithm (line 1). Then for each aggregation path, it finds its available storage spaces left (line 3-7). The amount to be replicated on each aggregation path is the minimum 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 on the storage nodes it visits (line 9-17). Finally, it offloads each initiator's overflow data that has not been offloaded from the last aggregator of each path (line 19). This is can be done by having a light-weighted data structure such a bitmap included in the message. Each bit corresponds to a data packet in the initiator's overflow data and is initially set as 0. It is set as 1 whenever the corresponding data packet is replicated along the path. By doing this, the last aggregator on each aggregation path knows exactly which data packets have not bee replicated therefore need to be offloaded. 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, $v = mcf(O, G)$ means running the minimum cost flow algorithm on $G(V, E)$ to offload a set of data packets 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 $(1 \le j \le l)$
Output: Data offloading cost C_{off}

```
0.  Notations:
      $u$: a node in $W_j^a$;
      $u.next$: the successor 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 path $W_j^a$;
1.  $C_{off} = mcf(D^{aggr} \cup D^{non-aggr}, G)$;
2.  for each $W_j^a$ $(1 \le j \le l)$
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.      $z = \min\{avail_j, R\}$;
9.      $u = I_j.next$, $D_j^r = \phi$ (empty set);
10.     while ($z > 0$)
11.         if ($u \in V_s$)
12.             Replicate $avail(u)$ amount of data $re$ at $u$;
13.             $D_j^r = D_j^r \cup re$;
14.             $z = z - avail(u)$;
15.         end if;
16.         $u = u.next$;
17.     end while;
18. end for;
19. $C_{off} = C_{off} + mcf(D^{init} - \bigcup_{j=1}^{l} D_j^r, G)$;
20. RETURN $C_{off}$.
```

Executing Algorithm 2 on the example in Fig 2, we can either get the result shown in Fig 5 with offloading cost of 2.5, or the result shown in Fig. 6 with offloading cost of 2.

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 $O(|V|^2|E|log(|V|C)))$.

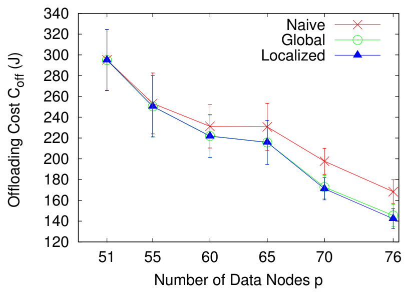
Theorem 1: In Algorithm 2, if $\forall\ 1 \le j \le l$, $avail_j \ge R$, then C_{off} is minimum.

Proof: Recall $D = D^{aggr} \cup D^{non-aggr} \cup \bigcup_{j=1}^{l}(D_j^{init} - D_j)$. Now if $\forall\ 1 \le j \le l$, $avail_j \ge R$, then $|D_j| = \min\{avail_j, R\} = R$. Recall that D_j^{init} is exactly this R amount of overflow data of I_j. Therefore $D_j^{init} = D_j$, $\forall\ 1 \le j \le l$. $D = D^{aggr} \cup D^{non-aggr}$, which is the minimum amount of data that must be offloaded. Therefore $C_{off} = mc(D^{aggr} \cup D^{non-aggr}, G)$ is minimum. ∎

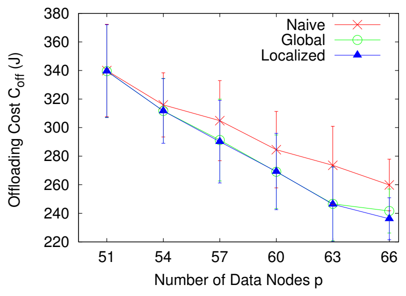
Localized Replication Algorithm. In Global Algorithm, when the initiator replicates its overflow data along its aggregation path, it does not take into account of other aggregation paths. However, it is possible that a storage node belongs to multiple aggregation paths; thus multiple initiators need to coordinate to replicate their overflow data onto this shared storage node in order to achieve efficient data offloading. Below we design a localized replication algorithm that addresses this issue. We first 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. Let $d(u) = \sum_{v \in N(u) \cap V_d} \frac{1}{s(v)}$. □

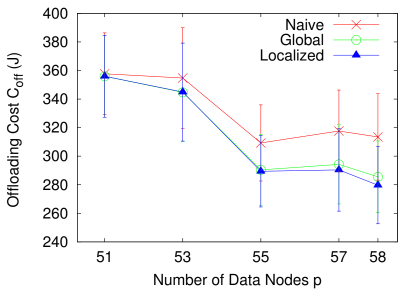
Note that $s(v) \neq 0$ since v has at least one neighboring storage node u. The idea behind $d(u)$ is that the more data



(a) $r = 0.3R$.

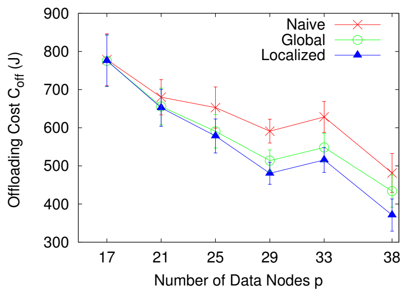


(b) $r = 0.5R$.

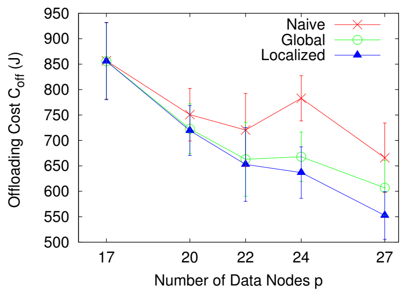


(c) $r = 0.7R$.

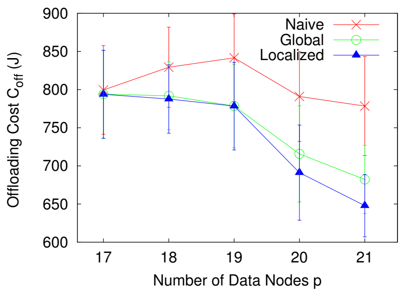
Fig. 7. Performance comparison when $R = 1m$.



(a) $r = 0.3R$.



(b) $r = 0.5R$.



(c) $r = 0.7R$.

Fig. 8. Performance comparison when $R = 5m$.

nodes surrounding u and the less storage nodes surrounding these data nodes, the more likely u will be used and shared by such data nodes to store their overflow data (that is, u has a higher demand from surrounding data nodes for its storage spaces). $d(u)$ is designed to decide how much each initiator should replicate its data 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 (line 2-4). Next it calculates the amount of data that 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 $(1 \le j \le l)$
Output: C_{off}

```
0.   Notations:
         $S_j = \{S_1^j, S_2^j, ..., S_{k_j}^j\}$: the set of $k_j$ $(k_j \ge 0)$ 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 $W_j^a$ $(1 \le j \le l)$
2.       for $u \in S_j$, calculate $d(u)$;
3.       Sort $S_j$ in non-descending order of $d(u)$;
4.       Let $d(S_1^j) \le d(S_2^j) \le ... \le d(S_{k_j}^j)$, WLOG;
5.       $z = R$, $i = 1$, $D_j^r = \phi$ (empty set);
6.       while $(z > 0 \wedge i \le k_j)$
7.           Replicate $\min(R/d(S_i^j), R)$ amount data $re$ on $S_i^j$;
8.           $z = z - \min(R/d(S_i^j), R)$;
9.           $D_j^r = D_j^r \cup re$;
10.          $i++$;
11.      end while;
12.  end for;
13.  $D = D^{aggr} \cup D^{non-aggr} \cup \bigcup_{j=1}^{l} (D_j^{init} - D_j^r)$;
14.  $C_{off} = mcf(D, G)$;
15.  RETURN $C_{off}$.
```

For the example in Fig 2, as only storage node E is on the aggregation path, we calculate its demand number as $\frac{3}{2}$. Therefore G replicates $\frac{2R}{3}$ of its overflow data on E. The offloading cost following Algorithm 3 is $\frac{8}{3}$. As there is only one aggregation path in this example, the advantage of demand numbers to help to share storages among multiple aggregation paths is not particularly demonstrated. However, we show that in simulations that Algorithm 3 performs better than Algorithm 2 does.

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 $O(|E|^3)$.

Distributed implementation of the algorithms. As there are three kinds of algorithms presented in the paper viz. data aggregation, data replication, and data offloading, we discuss their distributed implementations one by one. For data aggregation, we can use the minimum spanning tree-based distributed algorithm designed by Tang [33]. It can find optimal data aggregation when there is only one initiator. Both of its time and message complexities are $O(p|E|)$. For the data offloading algorithm, which is mainly a minimum cost flow algorithm, it can be implemented using the technique proposed by Quadrianto et al. [32]. They designed scalable and distributed algorithms for both maximum flow problem and minimum cost flow problem. Their distributed algorithms are based on convex-concave saddle point reformulation and take $|E|$ iterations, where $|E|$ is the number of edges.

For the data replication algorithm, above localized replication algorithm (Algorithm 3) can be implemented in a distributed manner as follows. As each aggregation path and its initiator have already been computed, the initiator can serve as the controller of its belonged path by collecting information from all other storage nodes and calculating each of their demand number. Specifically, each storage node on each path collects the information of their data node neighbors, from which it collects information of their storage node neighbors, and sends such information to the initiator. Once received from all the storage nodes on the path, the initiator calculates each storage node's demand number, sorts them in non-descending order and begins to traverse and replicate on them according to the calculated demand number. This is done until either the last aggregator on the path is reached or all the R amount of overflow data of this initiator is replicated. Finally, aforesaid distributed minimum-cost flow algorithm [32] is executed to offload the overflow data that has not been replicated. As this executes on the aggregation paths in a sequential manner, synchronization is needed.

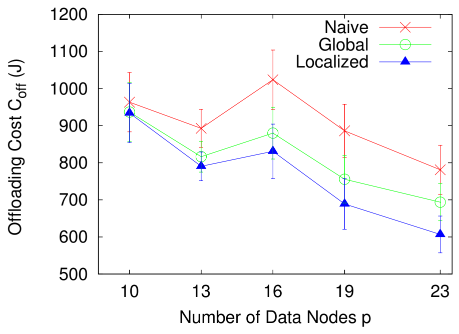
VI. Performance Evaluation

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 × 1000m. Transmission range of sensor nodes is 250m. Unless otherwise mentioned, R=m=512KB. We also vary R/m to 5 and 10, with m=512KB while R=2560KB and 5120KB, respectively. Each data point in all plots is an average over 20 runs, and the error bars indicate 95% confidence interval.

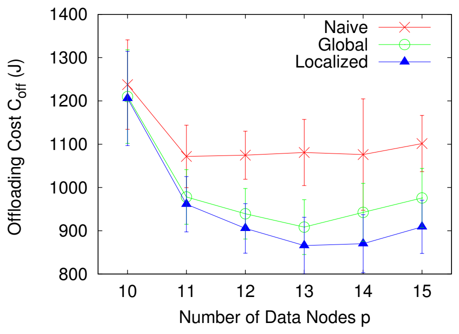
Effect of Varying p. Fig. 7 shows the offloading cost of three algorithms when $R = 1m$, with r varied from $0.3R$, $0.5R$, to $0.7R$. In each case, we find the valid range of p (using Inequality 5) and increase p from its smallest to largest valid values. It clearly shows that Localized performs better than Global, which performs better than Naive. In most cases, performance improvement of Localized upon Naive is around 10%. 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. We notice a few cases wherein the offloading cost increases a bit with the increase of p (such as $p = 57$ in Fig. 7(c)), this is because offloading paths could possibly become longer, even though the total size of offloaded data is smaller with increase of p. Localized performs better than Global does. This is because for Global, after offloading all the overflow data of the aggregators and all the overflow that are not aggregated using 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 replicating the overflow data from the initiators. For Localized, however, it always replicates based upon the calculated demand number of each storage node therefore replicating more wisely.

Effect of $R = 5m$ and $R = 10m$. Next, we study and compare the algorithms for $R = 5m$ and $R = 10m$. First it shows when inFig. 8 and 9 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%. With the increase of the ratio R/m, more data needs to be aggregated in order to fit into the available storages, making the aggregation process more challenging. In partially, with the increase of R/m, both the range and the number of valid p decrease (according to Inequality 5), therefore there are less number of storage nodes on each aggregation path. Compared to Fig. 7, Fig. 8 and 9 show that the performance differences between Localized and Naive further increases. This demonstrates that our replication algorithms works even better in these challenging scenarios.

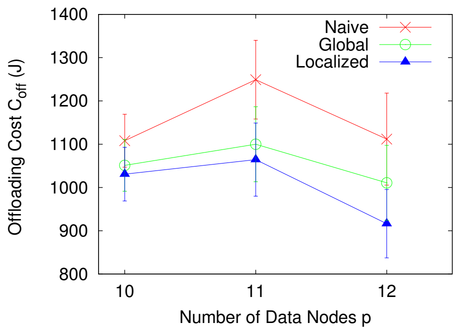
Effect of Varying r. Fig. 10 shows the performance comparison by varying r with $R = 5m$ and $p = 20$. It shows that the offloading cost of all three algorithms generally increase with the increase of r. This can be explained by the non-uniform distribution of overflow data after data aggregation when increasing r, whereas uniformly distributed data can be offloaded more easily by costing less energy. Initially, overflow data are uniformly spread inside the network (as data nodes are uniformly distributed). Now by increasing r, q increases (Equation 2) thus the aggregation paths get longer in order to visit more aggregators. This results in that more data nodes get their overflow data reduced. Therefore after data aggregation, overflow data are no longer uniformly distributed with increase of q. Such non-uniform distribution of overflow data incurs more energy cost for data offloading. Fig. 10 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 effect of data replication. As aggregation paths get longer due



(a) $r = 0.3R$.



(b) $r = 0.5R$.



(c) $r = 0.7R$.

Fig. 9. Performance comparison when $R = 10m$.

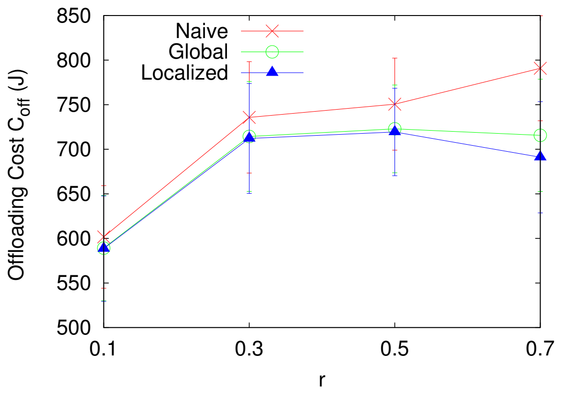


Fig. 10. Performance comparison by varying r. Here, $R = 5m$ and $p = 20$.

to the fact that more aggregators are visited when increasing r, there are potentially more storage nodes on each aggregation path. Data replication thus can benefit from this by having more of the initiators' overflow data replicated on aggregation path, reducing the amount of initiators' overflow data that need to be offloading during data offloading stage.

Effect of Varying R/m**.** Fig. 11 investigates the effect upon the algorithm performances by varying R/m. For fair comparison, we first need to find a common valid range of p for $R/m = 5$ and $R/m = 10$. It turns out that when $r = 0.3R$, they have the same valid range of p, $17 \leq p \leq 23$. We thus vary p from 17, 19, 21, to 23, and compare the offloading cost of the algorithms by varying R/m from 5 to 10. 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 after data aggregation, it causes more non-uniformity of data distribution for $R/m = 10$, increasing 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.

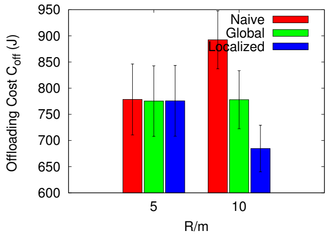
VII. Conclusion and Future Work

In this paper we tackled the overall storage overflow problem in sensor networks that are deployed in challenging environments, wherein the generated sensory data overflows the storage capacity available in the entire network due to the absence of base stations. This is an important problem since many of the emerging sensor network applications fall into this scenario. To solve this problem, previous research treated data aggregation and data offloading as two independent and separate stages, and did not explore the interactions and synergies between them. In this paper, we propose DAO-R, which instead employed data replication to integrate data aggregation and data offloading, and achieved energy-efficiency while solving the overall storage overflow problem. We designed two energy efficient data replication algorithms for DAO-R. We also gave a sufficient condition to solve DAO-R optimally. We show via simulations that DAO-R data replication algorithms outperform the existing approach DAO-N by around 30% in terms of energy consumption. As future work, we will first consider that different data nodes could have different amount of overflow data as well as different storage nodes could have different storage capacity. Second, we will extend the uniform data size reduction by considering that different sensor nodes could have different correlation coefficients. Finally, we will consider more dynamic scenarios, for example, some nodes could deplete their battery power, and design energy-efficient distributed algorithm.

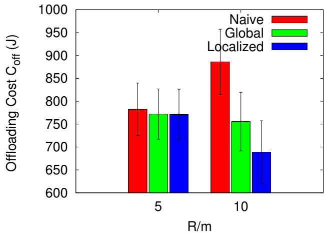
References

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

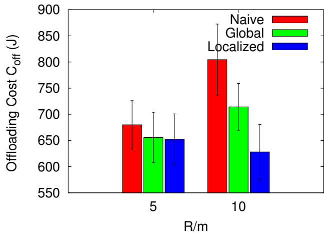
[2] M. Albano and J. Gao. In-network coding for resilient sensor data storage and efficient data mule collection. In *Proc. of ALGOSENSOR*, 2010.



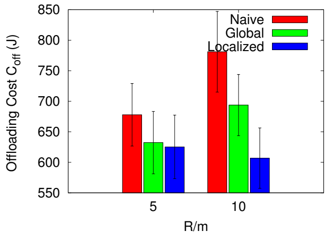
(a) $p = 17$.



(b) $p = 19$.



(c) $p = 21$.



(d) $p = 23$.

Fig. 11. Performance comparison by varying R/m. Here, $r = 0.3R$.

[3] S. Basagni, L. Boloni, P. Gjanci, C. Petrioli, C. A. Phillips, and D. Turgut. Maximizing the value of sensed information in underwater wireless sensor networks via an autonomous underwater vehicle. In *Proc. of INFOCOM*, 2014.

[4] T. Corman, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

[5] R. Cristescu, B. Beferull-lozano, and M. Vetterli. On network correlated data gathering. In *Proc. of INFOCOM*, 2004.

[6] 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.

[7] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(4):11–25, October 2001.

[8] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. In *Proc. of SenSys*, 2003.

[9] A. Ghose, J. Grossklags, and J. Chuang. Resilient data-centric storage in wireless ad-hoc sensor networks. In *Proc. of MDM*, 2003.

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

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

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

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

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

[15] 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.

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

[17] A. Jindal and K. Psounis. Modeling spatially correlated data in sensor networks. *ACM Tran. on Sensor Net*, 2:466–499, 2006.

[18] A. Kamra, J. Feldman, V. Misra, and D. Rubenstein. Growth codes: Maximizing sensor network data persistence. In *Proc. of SIGCOMM*, 2006.

[19] 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.

[20] H. Li, D. Liang, L. Xie, G. Zhang, and K. Ramamritham. Flash-optimized temporal indexing for time-series data storage on sensor platforms. *ACM Trans. Sen. Netw.*, 10(4):1565–1572, 2012.

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

[22] K. Li, C. Shen, and G. Chen. Energy-constrained bi-objective data muling in underwater wireless sensor networks. In *Proc. of MASS*, 2010.

[23] 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.

[24] L. Luo, Q. Cao, C. Huang, L. Wang, T. Abdelzaher, and J. Stankovic. Design, implementation, and evaluation of enviromic: A storage-centric audio sensor network. *ACM Transactions on Sensor Networks*, 5(3):1–35, 2009.

[25] L. Luo, C. Huang, T. Abdelzaher, and J. Stankovic. Envirostore: A cooperative storage system for disconnected operation in sensor networks. In *Proc. of INFOCOM*, 2007.

[26] M. Ma and Y. Yang. Data gathering in wireless sensor networks with mobile collectors. In *Proc. of IPDPS*, 2008.

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

[28] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Ultra-low power data storage for sensor networks. In *Proc. of IPSN*, 2006.

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

[30] L. Mottola. Programming storage-centric sensor networks with squirrel. In *Proc. of IPSN*, 2010.

[31] S. Pattem, B. Krishnamachari, and R. Govindan. The impact of spatial correlation on routing with compression in wireless sensor networks. *ACM Trans. Sen. Netw.*, 4(4):1–33, September 2008.

[32] N. Quadrianto, D. Schuurmans, and A. J. Smola. Distributed flow algorithms for scalable similarity visualization. In *2010 IEEE International Conference on Data Mining Workshops*, 2010.

[33] B. Tang. Dao: Overcoming overall storage overflow in intermittently connected sensor networks. In *Proc. of Infocom*, 2018.

[34] B. Tang, N. Jaggi, H. Wu, and R. Kurkal. Energy efficient data redistribution in sensor networks. *ACM Transactions on Sensor Networks*, 9(2):11:1–11:28, May 2013.

[35] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *Proc. of SenSys*, 2005.

[36] Lili Wang, Yong Yang, Dong Kun Noh, Hieu Le, Tarek Abdelzaher, Michael Ward, and Jie Liu. Adaptsens: An adaptive data collection and storage service for solar-powered sensor networks. In *Proc. of the 30th IEEE Real-Time Systems Symposium (RTSS 2009)*.

[37] 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.

[38] 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.

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

[40] Yong Yang, Lili Wang, Dong Kun Noh, Hieu Khac Le, and Tarek F. Abdelzaher. Solarstore: enhancing data reliability in solar-powered storage-centric sensor networks. In *Proc. of MobiSys*, 2009.