DAO^2: Overcoming Overall Storage Overflow in Intermittently Connected Sensor Networks

Bin Tang
Computer Science Department, California State University Dominguez Hills
btang@csudh.edu

Abstract*—Many emerging sensor network applications operate in challenging environments wherein sensor nodes do not always have connected paths to the base station. Data generated from such *intermittently connected sensor networks* therefore must be stored inside the network for some unpredictable period of time before uploading opportunities become available. Consequently, sensory data could overflow limited storage capacity available in the entire network, making discarding valuable data inevitable. To overcome such *overall storage overflow* in intermittently connected sensor networks, we propose and study a new algorithmic problem called *data aggregation for overall storage overflow (DAO^2)*. Utilizing spatial data correlation that commonly exists among sensory data, DAO^2 employs data aggregation techniques to reduce the overflow data size while minimizing the total energy consumption. To solve DAO^2, we uncover a new graph theoretic problem called *multiple traveling salesman walks (MTSW)*, and show that with proper graph transformation, the DAO^2 is equivalent to the MTSW. We prove that MTSW is NP-hard and design a $(2 - \frac{1}{q})$-approximation algorithm, where q is the number of nodes to visit (i.e., the number of sensor nodes that aggregate their overflow data). The approximation algorithm is based on a novel routing structure called *minimum q-edge forest* that accurately captures information needed for energy-efficient data aggregation. We further put forward a heuristic algorithm and empirically show that it constantly outperforms the approximation algorithm by $15\% - 30\%$ in energy consumption. Finally, we propose a distributed data aggregation algorithm that can achieve the same approximation ratio as the centralized algorithm under some condition, while incurring comparable energy consumption.

Keywords – Intermittently Connected Sensor Networks, Data Aggregation, Approximation Algorithms, Graph Theory

I. Introduction

In recent years sensor networks have been adopted to tackle some of the most fundamental problems facing human beings, such as disaster warning, climate change, and renewable energy. These emerging scientific applications include underwater or ocean sensor networks [19, 35], wind and solar harvesting [25], and seismic sensor networks [33]. One common characteristic of these applications is that they are all deployed in challenging environments such as in remote or inhospitable regions, or under extreme weather, to continuously collect large volumes of data for a long period of time.

In those challenging environments, it is usually not possible to deploy high-power, high storage data-collecting base stations in the field. Consequently, sensory data generated are stored inside the network for some unpredictable period of time and then collected by periodic visits of data mules [29], or by low rate satellite link [27]. We refer to such sensor networks wherein sensor nodes do not always have connected paths to the base station as *intermittently connected sensor networks*. Due to inadequate human intervention in the inhospitable environments, intermittently connected sensor networks must operate more resiliently than traditional sensor networks (with base stations and in friendly environments).

In this paper we tackle data resilience in intermittently connected sensor networks. *Data resilience* refers to the ability of long-term viability and availability of data despite insufficiencies of (or disruptions to) the physical infrastructure that stores the data. In intermittently connected sensor networks, one such disruption and major obstacle is data storage overflow. On one side, sensing a wide range of physical properties in real world, above scientific applications generate massive amounts of data, such as videos or high resolution images. On the other side, storage is still a serious resource constraint of sensor nodes despite the advances in energy-efficient flash storage [19, 26]. As a consequence, the massive sensory data could soon overflow data storage of sensor nodes and cause data loss. Such storage overflow problem is further exacerbated in intermittently connected sensor networks, wherein most of the time the high-storage base stations are not available to collect and store the data.

To avoid data loss, our previous work has designed a suite of techniques to *offload* overflow data from storage-depleted sensor nodes to nearby sensor nodes with available storages [14, 30, 34]. However, if these offloaded data cannot be collected and uploaded timely by data mules or satellite links, they could soon overflow the available storage in the entire network. This unfortunately can not be alleviated by aforesaid data offloading techniques. We refer to this more severe obstacle in the intermittently connected sensor networks as *overall storage overflow*.

Consider a recent application of underwater exploration and monitoring [3], wherein camera sensors take pictures of the underwater scenes while an autonomous underwater vehicle (AUV) is dispatched periodically to collect the pictures from the camera sensors. Suppose there are 100 underwater camera sensors, 10 of which are generating one 640×480 JPEG color image per second. Even using the latest parallel NAND flash technology with 16GB for sensor storage [16], it takes less than one day to exhaust the storages of all the 100 camera sensors, causing overall storage overflow. If the AUV cannot be dispatched timely due to inclement and stormy weather, discarding valuable data becomes inevitable. In this paper, we attempt to answer following question: *How to preserve the data in intermittently connected sensor networks despite the overall storage overflow?*

Fortunately, we can take advantage of spatial correlation that commonly exists among sensory data, and employ data aggregation techniques to reduce the overflow data size in order to overcome overall storage overflow. We formulate a new algorithmic problem called *data aggregation for overall storage overflow (DAO^2)*. At the core of DAO^2 is a new

graph-theoretic problem called *multiple traveling salesman walks (MTSW)*, which has not been studied in any existing literature. To solve DAO^2, we design a suite of energy-efficient optimal, approximation, heuristic, and distributed data aggregation algorithms with detailed analytical analysis of their performances. One novelty of our aggregation techniques is a routing structure called *minimum q-edge forest*, where q is the number of sensor nodes that aggregate their overflow data. The minimum q-edge forest generalizes minimum spanning tree, one of the most fundamental graph structures, and accurately captures information needed for energy-efficient data aggregation.

After being aggregated to the size accommodable by the network, the overflow data can then be stored into sensor nodes with available storage using data offloading techniques proposed in [14, 30, 34] (we further illustrate this using Example 1 in Section II). Note that in this paper we do not consider how to upload data from sensor nodes to base station, which has been studied extensively by using data mules or mobile data collectors [10, 29].

The rest of the paper is organized as follows. Section II formulates the DAO^2 with an illustrative example. In Section III we study the MTSW problem and design a suite of optimal, approximation and heuristic algorithms. We show that with proper graph transformation the DAO^2 is equivalent to the MTSW, therefore the algorithms for MTSW can be applied to solve DAO^2. In Section IV, we design a distributed data aggregation algorithm for the DAO^2 with time and message analyses. In Section V, we compare all the algorithms under different network dynamics and discuss the results. (Since overall storage overflow has not been addressed at all in the existing literature, and to the best of our knowledge this is the first work to solve it, we could not find any existing work suitable for comparison.) Section VI and VII review related work and conclude the paper with possible future research.

II. Problem Formulation of DAO^2

Problem Statement. Fig. 1 shows an intermittently connected sensor network. In our model, some sensor nodes are close to the events of interest thus are constantly generating sensory data and have depleted their own storages. We refer to sensor nodes with depleted storage spaces while still generating data as **data nodes**. The newly generated data that can no longer be stored at data nodes is called **overflow data**. To avoid data loss, overflow data is offloaded to sensor nodes with available storages (referred to as **storage nodes**). Note that sensor nodes that have generated data but have not depleted their storage spaces are considered as storage nodes, as they can store overflow data from data nodes.

To start the aggregation process, one or more data nodes (called **initiators**) send their overflow data to visit other data nodes in multi-hop manner. When a data node receives the data, it aggregates its own overflow data and becomes an **aggregator**, and then forwards the initiator's entire overflow data to another data node. This data node also becomes an aggregator and aggregates its overflow data. This continues until enough aggregators are visited such that total size of the overflow data after aggregation equals to or is slightly less than total available storage in the network. Note that during the aggregation process, it does not store overflow data to the storage nodes since other aggregators need the entire data in order to aggregate their own data.

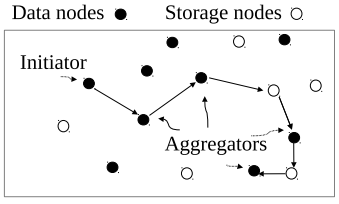


Fig. 1. Illustrating DAO^2.

Network Model. The sensor network is represented as an undirected connected graph $G(V, E)$, where $V = \{1, 2, ..., |V|\}$ is the set of $|V|$ sensor nodes and E is the set of $|E|$ edges. There are p data nodes, denoted as V_d (the other $|V| - p$ nodes are storage nodes). Let R denote the size of generated overflow data in bits at each data node, and let m be the available storage space in bits at each storage node. (We leave that data nodes have different sizes of overflow data and storage nodes have different sizes of storage spaces as future work.) Due to the overall storage overflow, $p \times R > (|V| - p) \times m$, giving that $p > \frac{|V|m}{m+R}$.

Spatial Correlation Data Model. The model is as follows. Let $H(X)$ denote the entropy of a discrete random variable X, and $H(X|Y)$ denote the conditional entropy of a random variable X given that random variable Y is known. If data node i receives no side information (i.e., overflow data) from other data nodes, its overflow data is entropy coded with $H(i|j_1, ..., j_p) = R$ bits, $j_k \in V_d \wedge j_k \neq i, 1 \leq k \leq p$. If data node i receives side information from at least one data node, the size of its overflow data is $H(i|j_1, ..., j_p) = r \leq R$. This correlation model has two advantages. First, it captures the uniform data spatial correlation scenario, wherein data generated at different data nodes have similar correlation with each other (we leave the more challenging and realistic model that different nodes have different data correlation as future work). Second, it is an effective distributed coding strategy, which works well in large scale sensor network applications. We are aware of other distributed coding techniques such as Slepian-Wolf coding [36]. However, they need global correlation structure, which is impractical in large networks.

Our model is based on a well-known entropy-based model proposed in [9], with one difference. The model in [9] assumes that each sensor node generates a R-bit packet and transmits it back to the base station. In our model, R is the amount of overflow data generated only at data nodes. We have two observations about the correlation model.

Observation 1: 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 data serves as side information for other nodes to aggregate. An aggregator cannot be an initiator since its aggregated data loses the side information needed for others nodes' aggregation. □

Observation 2: Each aggregator can be visited multiple times by the same or different initiators (if that is more energy-efficient). However, the data of an aggregator can only be aggregated once, with size reduced from R to r. □

Energy Model. We adopt first order radio model [12] for battery power consumption. When node u sends R-bit data to its one-hop neighbor v over distance $l_{u,v}$, transmission energy cost at u is $E_t(R, l_{u,v}) = E_{elec} \times R + \epsilon_{amp} \times R \times l_{u,v}^2$, receiving energy cost at v is $E_r(R) = E_{elec} \times R$. Here, $E_{elec} = 100nJ/bit$ is energy consumption per bit on transmitter and receiver circuits, and $\epsilon_{amp} = 100pJ/bit/m^2$ is energy consumption per bit on transmit amplifier. Let $W = \{v_1, v_2, ..., v_n\}$ be a *walk*, a sequence of n nodes with

$(v_i, v_{i+1}) \in E$ and $v_1 \neq v_n$ (if all nodes in W are distinct, W is a *path*). Let $w(R, u, v) = E_t(R, l_{u,v}) + E_r(R)$, and $c(R, W) = \sum_{i=1}^{n-1} w(R, v_i, v_{i+1})$ denote the *aggregation cost* on W, the energy consumption of sending R-bit from v_1 to v_n along W. We assume that there exists a contention-free MAC protocol to avoid overhearing and collision (e.g. [5]).

Number of Aggregators q and Valid Range of p in Overall Storage Overflow. We first calculate the number of aggregators, denoted as q, that need to be visited for the data reduction. Since each aggregator reduces its overflow data size by $(R-r)$, and the total anticipated data size reduction in the network is $p \times R - (|V| - p) \times m = p \times (R+m) - |V| \times m$,

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

To guarantee that the overflow data after aggregation can fit in the available storage in the network, next we compute the upper bound of number of data nodes p. As at least one data node needs to be the initiator to start the aggregation process, there can only be maximum of $p-1$ aggregators (Observation 1). We therefore have $q = \lceil \frac{p \times (R+m) - |V| \times m}{R-r} \rceil \leq p - 1$, which gives $p \leq \lfloor \frac{|V|m - R + r}{m + r} \rfloor$. As we have calculated the lower bound of p in network model above, the valid range of p for overall storage overflow to occur is therefore

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

Problem Formulation of DAO^2. Given a valid p value and its corresponding q value, meaning q out of the p data nodes need to be aggregators and the rest $p-q$ data nodes can serve as initiators (Observation 1), DAO^2 determines:

- set of a ($1 \leq a \leq (p-q)$) initiators, denoted as $\mathcal{I}$, and
- corresponding set of a *aggregation walks*: W_1, W_2, ..., W_a, where W_j ($1 \leq j \leq a$) starts from a distinct initiator $I_j \in \mathcal{I}$, such that $|\bigcup_{j=1}^{a} \{W_j - \{I_j\} - G_j\}| = q$. Here, G_j is the set of storage nodes in W_j thus $W_j - \{I_j\} - G_j$ is the set of aggregators in W_j. Since an aggregator can appear multiple times in the same or different aggregation walks (Observation 2), $\bigcup_{j=1}^{a} \{W_j - \{I_j\} - G_j\}$ signifies a set of q *distinct* aggregators in the network.

TABLE I
NOTATION SUMMARY

Notation	Explanation
V, $\|V\|$	Set and number of sensor nodes
V_d, p	Set and number of data 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, $r < R$	Overflow data size at each data node after aggregation
$\mathcal{I}$, a	Set and number of initiators, $1 \leq a \leq (p-q)$
I_j	j^{th} initiator, $1 \leq j \leq a$
W_j	Aggregation walk starting with I_j
$w(R, u, v)$	Aggregation cost of sending R bits from u to v
$c(R, W_j)$	Aggregation cost of sending R bits along W_j

The goal DAO^2 is to minimize the *total aggregation cost* $\sum_{1 \leq j \leq a} c(R, W_j)$, the total energy consumption incurred in the aggregation process. Table I lists all the notations.

EXAMPLE 1: Fig. 2 gives an example of DAO^2 in a grid sensor network of 9 nodes (we use grid only for illustration purpose – the DAO^2 and its solutions are designed for general graph topologies). Nodes B, D, E, G, and I are data nodes, while nodes A, C, F and H are storage nodes. Assume that $R = m = 1$, $r = 3/4$, and energy consumption along any edge is 1 for one unit of data. In this scenario overall storage overflow exists, as there are 4 units of storage space while there are 5 units of overflow data. Using Equation 1, the number of aggregators q is 4, leaving one data node to be initiator. One of the optimal solutions could be selecting B as initiator and setting its aggregation walk as: B, E, D, G, H, I, with total aggregation cost of 5. After aggregation, the sizes of overflow data at B, E, D, G, and I are 0, 3/4, 3/4, 3/4, and 7/4, respectively, totaling 4 units. Note that 7/4 units of data at I now include 3/4 unit of I's own aggregated overflow data and one unit of initiator B's overflow data. □

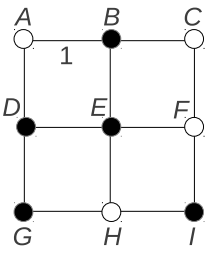


Fig. 2. A DAO^2 example.

Data Offloading After Data Aggregation. In the example of Fig. 2, as the total size of the overflow data after aggregation is 4, they can be offloaded and stored into the 4 units of storage space available in the network. In our previous work [30], we have shown that offloading those data (some are aggregated and some not) from data nodes to storage nodes with minimum energy consumption is a minimum cost flow problem, which can be solved optimally and efficiently. One optimal minimum cost flow solution is offloading E's 1/4 unit of data to A (cost 1/2), E's 2/4 unit of data to C (cost 1), D's 3/4 unit of data to A (cost 3/4), G's 3/4 unit of data to H (cost 3/4), I's 1/4 unit of data to H (cost 1/4), I's 2/4 unit data to C (cost 1), and B's one unit of data, now located at I, to F (cost 1), totaling 5.25 offloading cost. In this paper data aggregation and data offloading are separated stages, an integrated, more energy-efficient solution is proposed in [1].

Since data offloading can be achieved optimally, we only focus on data aggregation in this paper. We find that DAO^2 gives rise to a new graph-theoretic problem, which we refer to as *multiple traveling salesman walks problem (MTSW)*. In Section III we formulate MTSW, prove its NP-hardness, and solve it by an efficient $(2 - \frac{1}{q})$-approximation algorithm. We then show that the DAO^2 is equivalent to the MTSW therefore the algorithms for MTSW can be applied to solve DAO^2.

III. Multiple Traveling Salesman Walks (MTSW)

A. Problem Formulation and NP-Hardness.

Given an undirected weighted graph $G = (V, E)$ with $|V| = n$ nodes and $|E|$ edges, a cost metric (which represents the distance or traveling time between two nodes), and that the number of nodes that must be visited is q. The objective of the MTSW is to determine a subset of *at most* $b = n - q$ starting nodes (i.e., the initiators in DAO^2), from each of which a salesman is dispatched to visit some nodes following a walk, such that a) all together q nodes (i.e., the aggregators in DAO^2) are visited, and b) total cost of the walks is minimized.

Let $w(u, v)$ denote weight of edge $(u, v) \in E$. We assume that triangle inequality holds: for edges $(x, y), (y, z), (z, x) \in E$, $w(x, y) + w(y, z) \geq w(z, x)$. Given a walk $W = \{v_1, v_2, ..., v_n\}$, let $c(W) = \sum_{i=1}^{n-1} w(v_i, v_{i+1})$ denote its cost. The objective of MTSW is to decide:

- the set of a ($1 \leq a \leq b$) starting nodes $\mathcal{I} \subset V$, and
- the set of a *walks* $W_1, W_2, ..., W_a$: W_j ($1 \leq j \leq a$) starts from a distinct node $I_j \in \mathcal{I}$, and $|\bigcup_{j=1}^{a} \{W_j - \{I_j\}\}| = q$,

such that *total cost* $\sum_{1\le j\in a} c(W_j)$ is minimized.

Theorem 1: The MTSW is NP-hard.

Proof: Given an undirected graph $G(V, E)$, its *metric completion*, denoted as $G^{mc}(V, E^{mc})$, is a complete graph with the same set of nodes V, while for any pair of nodes $u, v \in V$, the cost of $(u, v) \in E^{mc}$ is the cost of the shortest path connecting u and v in $G(V, E)$. Recall that *traveling salesman path problem (TSPP)* [13] is to find a minimum-cost *hamiltonian path* that visits each node exactly once in a complete graph. MTSW in $G(V, E)$ is thus a *multiple traveling salesman path (MTSPP)* problem in $G^{mc}(V, E^{mc})$. MTSPP selects at most b starting nodes, from each of which a salesman is dispatched to visit a distinct subset of nodes following one of its hamiltonian paths, such that *exactly* q other nodes are visited with minimum total cost. We therefore prove MTSPP in $G^{mc}(V, E^{mc})$ is NP-hard. In particular, we prove TSPP, a special case of MTSPP, is NP-hard. Below we reduce the well-known *traveling salesman problem (TSP)* [8] to TSPP. Recall that TSP is to find a minimum-cost *hamiltonian cycle* in a complete graph that visits each node exactly once.

As shown in Fig. 3, let complete graph G be an instance of TSP, we construct an instance of TSPP, G^*, as follows. We choose an arbitrary node A in G and add a copy of it, $A^{'}$. We connect A' to all other nodes except A, and assign the same cost on each edge as the corresponding edge in G (that is, $(A^{'}, B)$ has the same cost as (A, B), $(A^{'}, C)$ has the same cost as (A, C), etc.). Then we introduce nodes s and t and add edges (s, A) and $(t, A^{'})$ with any *finite* edge costs. Finally, as G^* must be a complete graph, we add the rest edges (not shown in Fig. 3) and assign their costs to be *infinite*. We show that G contains a minimum-cost Hamiltonian cycle, say, A, C, B, D, A, if and only if G^* contains a minimum-cost Hamiltonian path s, A, C, B, D, $A^{'}$, t.

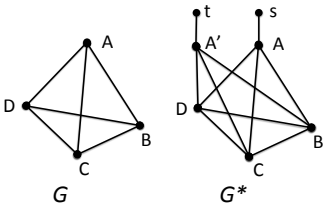


Fig. 3. Proving TSPP is NP-hard.

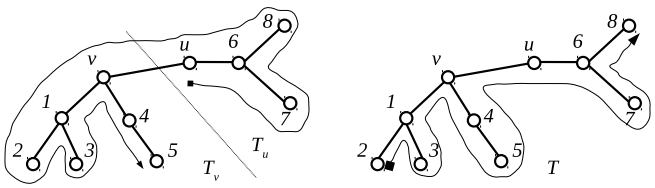
Suppose that G contains a minimum-cost Hamiltonian cycle A, C, B, D, A. Then we get a minimum-cost Hamiltonian path in G^* when we start from s, follow the cycle back to $A^{'}$ instead of A, and finally end in t. Conversely, suppose G^* contains a minimum-cost Hamiltonian path. This path (with finite cost) must end in s and t. We transform it to a cycle in G by a) deleting s and t, which results in a path that end in A and $A^{'}$, and b) removing $A^{'}$. The resulted path, instead of going back to $A^{'}$, goes back to A, forming a minimum-cost Hamiltonian cycle in G. ∎

B. Approximation Algorithm for MTSW.

We first introduce some definitions.

Definition 1: (**Binary Walk (B-Walk).**) Given a tree T with a maximum-weight edge (u, v) (ties are broken randomly), T is divided into (u, v) and subtrees T_u and T_v. The B-walk on T, denoted as $W_B(T)$, starts from u and visits all the nodes in T_u following depth-first-search (DFS) and comes back, then visits v, from where it visits all the nodes in T_v following DFS and stops when all the nodes are visited. □

Fig. 4(a) shows a tree T with $w(u, v) = 2$ and weights of other edges being 1, and a B-walk of cost 16. In B-walk, each edge in T_u is traversed twice, and each edge in T_v is traversed once or twice. B-walk saves cost traversing a tree since the maximum-weight (u, v) is traversed only once.



(a) B-Walk. (b) LP-Walk.

Fig. 4. (a) A binary walk (B-walk): u, 6, 7, 6, 8, 6, u, v, 1, 2, 1, 3, 1, v, 4, 5, with cost of 16. (b) A longest-path walk (LP-Walk): $2, 1, 3, 1, v, 4, 5, 4, v, u, 6, 7, 6, 8$, with cost of 14. ■ and ◄– indicate the first and last node in a walk, respectively. Here, $w(u, v) = 2$ and weights of other edges are 1.

Lemma 1: $c(W_B(T)) \le (2 - \frac{1}{|T|}) \times c(T)$. Here $c(T) = \sum_{e\in T} w(e)$ and $|T|$ is the number of edges in T.

Proof: Since (u, v) is the edge in T with maximum weight, $w(u, v) \ge \frac{1}{|T|} \times c(T)$. In $W_B(T)$, since (u, v) is traversed exactly once and other edges are traversed *at most* twice, $c(W_B(T)) \le \big(2 \times c(T) - w(u, v)\big)$. Therefore $c(W_B(T)) \le \big(2 \times c(T) - \frac{1}{|T|} \times c(T)\big) = (2 - \frac{1}{|T|}) \times c(T)$. ∎

Definition 2: (**Forest and q-Edge Forest**) A forest F of G is a subgraph of G that is acyclic (and possibly disconnected). A q-edge forest F_q is a forest with q edges. □

Approximation Algorithm. Algorithm 1 works as follows. Line 1 and 2 sort all the edges in E in non-descending order of their weights, and initialize an edge set E_q to be empty. The while loop in lines 3-9 finds the first q edges in E that do not cause a cycle and store them in E_q. It then obtains a q-edge forest $G[E_q]$ (line 10). Each connected component of $G[E_q]$ is either linear or a tree as no cycles are introduced. If it is linear, it starts from one end and visits the rest nodes exactly once; if it is a tree, it does a B-walk to visit all the nodes (lines 11-15).

Algorithm 1: Approximation Algorithm for MTSW.
Input: $G(V, E)$ and number of nodes to visit q;
Output: a walks: $W_1, W_2, ..., W_a$, and $\sum_{1\le j\in a} c(W_j)$;

0. **Notations**:
   E_q: set of q cycleless edges;
   $G[E_q]$: a q-edge forest;
   $C(G[E_q])$: set of connected components in $G[E_q]$;
   C_j: the j^{th} connected component in $C(G[E_q])$;
1. Let $w(e_1) \le w(e_2) \le ... \le w(e_{|E|})$;
2. $E_q = \phi$ (empty set), $i = j = k = 1$;
3. **while** $(k \le q)$
4. **if** (e_i is a cycleless edge w.r.t. E_q)
5. $E_q = E_q \cup \{e_i\}$;
6. $k++$;
7. **end if;**
8. $i++$;
9. **end while;**
10. Let $|C(G[E_q])| = a$; /*a connected components*/
11. **for** $(1 \le j \le a)$
12. **if** (C_j is linear) Start from one end node of C_j and visit the rest nodes in C_j once;
13. **if** (C_j is a tree) Do a B-walk on C_j;
14. Let the resulted walk (or path) be W_j;
15. **end for;**
16. **RETURN** $W_1, W_2, ..., W_a$, and $\sum_{1\le j\in a} c(W_j)$.

Discussions. Algorithm 1 takes $O(|E|\log|E|)$ and works alike the well-known Kruskal's minimum spanning tree (MST) algorithm [7], except that instead of finding $|V|-1$ edges to connect all the nodes in V, it finds $q \le |V|-1$ edges, to "connect" *some* nodes in V. Algorithm 1 therefore generalizes Kruskal's MST algorithm, as MST is a special case of $G[E_q]$, a *minimum q-edge forest* formally defined below.

Definition 3: (**Minimum q-Edge Forest**) Let $c(F_q) = \sum_{e\in F_q} w_e$ denote the cost of a q-edge forest F_q in G. Let $\mathcal{F}_q$ be the set of all q-edge forests in G. A q-edge forest F_q^m is minimum iff $c(F_q^m) \le c(F_q), \forall F_q \in \mathcal{F}_q$. □

Lemma 2: $G[E_q]$ is a minimum q-edge forest.

Proof: Let $E = \{e_1, e_2, ..., e_{|E|}\}$, with $w(e_1) \le w(e_2) \le ... \le w(e_{|E|})$. Let $E_q = \{e_1^g, e_2^g, ..., e_q^g\}$, with $w(e_1^g) \le w(e_2^g) \le ... \le w(e_q^g)$. By way of contradiction, assume that another q-edge forest, O_q, is a minimum q-edge forest with cost smaller than that of $G[E_q]$. Let $O_q = \{e_1^o, e_2^o, ..., e_q^o\}$ with $w(e_1^o) \le w(e_2^o) \le ... \le w(e_q^o)$. Assume that $e_l^g \in E_q$ and $e_l^o \in O_q$, $1 \le l \le q$, are the first pair of edges that differ in E_q and O_q: $e_l^g \ne e_l^o$ and $e_i^g = e_i^o$, $\forall\ 1 \le i \le l-1$. According to Algorithm 1, $w(e_l^g) \le w(e_l^o)$. Now consider subgraph $O_q \cup \{e_l^g\}$.

Case 1: $O_q \cup \{e_l^g\}$ is a forest. Then $c(O_q \cup \{e_l^g\} - \{e_l^o\}) \le c(O_q)$, contradicting that O_q is a minimum q-edge forest.

Case 2: $O_q \cup \{e_l^g\}$ is not a forest, i.e., there is a cycle in it. e_l^g must be in this cycle since there is no cycle in O_q. Besides, among all the edges in this cycle that is not e_l^g, at least one of them is not in E_q; otherwise there will not be any cycle (as they all belong to E_q, which is cycleless). Denote this edge as e'. Let e_l^g be the n^{th} edge in $E = \{e_1, e_2, ..., e_{|E|}\}$, that is, $e_l^g = e_n, 1 \le n \le |E|$.

Case 2.1: $e' \in \{e_1, e_2, ..., e_{n-1}\}$. Thus $w(e') \le w(e_{n-1}) \le w(e_n) = w(e_l^g) \le w(e_l^o)$, contradicting that e_l^g and e_l^o are the first pair of edges that differ in E_q and O_q.

Case 2.2: $e' \in \{e_{n+1}, e_{n+2}, ..., e_{|E|}\}$. Thus $w(e') \ge w(e_{n+1}) \ge w(e_n) = w(e_l^g)$. $c(O_q \cup \{e_l^g\} - \{e'\}) \le c(O_q)$, contradicting that O_q is a minimum q-edge forest.

Reaching contradiction in all the cases, it concludes that $c(G[E_q]) \le c(F_q), \forall F_q \in \mathcal{F}_q$. ∎

Let O be an optimal algorithm of MTSW with minimum cost of $\mathcal{O}$. Next we show $c(G[E_q])$ is a lower bound of $\mathcal{O}$.

Lemma 3: $c(G[E_q]) \le \mathcal{O}$.

Proof: Assume that all the edges selected in O induce λ connected components, denoted as O_j $(1 \le j \le \lambda)$. Assume that there are l_j nodes in O_j, and s_j $(l_j > s_j \ge 1)$ of them are starting nodes (therefore there are s_j walks in O_j visiting altogether $l_j - s_j$ nodes). Denote the s_j walks in O_j as W_j^o and let $c(W_j^o)$ be its cost. We have $\sum_{j=1}^{\lambda} c(W_j^o) = \mathcal{O}$.

Let $c(O_j) = \sum_{e\in O_j} w(e)$. Denote any spanning tree of O_j as T_j^o, and let $c(T_j^o) = \sum_{e\in T_j^o} w(e)$. We have $c(T_j^o) \le c(O_j) \le c(W_j^o)$. The first inequality is because all the edges in T_j^o are in O_j (but not vice versa); the second inequality is because each edge in O_j is traversed at least once in O. Therefore $\sum_{j=1}^{\lambda} c(T_j^o) \le \sum_{j=1}^{\lambda} c(W_j^o) = \mathcal{O}$.

Let $q' = \sum_{j=1}^{\lambda} |T_j^o|$, where $|T_j^o|$ is the number of edges in T_j^o. We have $q' = \sum_{j=1}^{\lambda}(l_j - 1)$. The subgraph induced by all T_j^o $(1 \le j \le \lambda)$ is therefore a q'-edge forest. Since all together q nodes are visited, $\sum_{j=1}^{\lambda}(l_j - s_j) = q$. Since $s_j \ge 1$, we have $q \le \sum_{j=1}^{\lambda}(l_j - 1) = q'$. Therefore, $c(G[E_q]) \le c(G[E_{q'}]) \overset{\text{Lemma 2}}{\le} \sum_{j=1}^{\lambda} c(T_j^o) \le \mathcal{O}$. ∎

Theorem 2: Algorithm 1 is a $(2-\frac{1}{q})$-approximation algorithm.

Proof: In Algorithm 1, each of the a connected components C_j $(1 \le j \le a)$ is either linear or a tree. Let q_j and $c(C_j)$ denote the number of edges in C_j and the sum of weights of edges in C_j, respectively. We have $q = \sum_{j=1}^{a} q_j$ and $c(G[E_q]) = \sum_{j=1}^{a} c(C_j)$. Let W_j be a B-DFS walk of C_j.

$$\begin{aligned}\sum_{j=1}^{a} c(W_j) &\overset{\text{Lemma 1}}{\le} \sum_{j=1}^{a}\left((2-\frac{1}{q_j})\times c(C_j)\right)\\ &< \sum_{j=1}^{a}\left((2-\frac{1}{q})\times c(C_j)\right)\\ &= (2-\frac{1}{q})\times c(G[E_q])\\ &\overset{\text{Lemma 3}}{\le} (2-\frac{1}{q})\times\mathcal{O}.\end{aligned}$$

∎

Corollary 1: If C_j $(1 \le j \le a)$ resulted from Algorithm 1 are all linear, Algorithm 1 is optimal.

Proof: In this case, $\sum_{1\le j\in a} c(W_j) = \sum_{1\le j\in a} c(C_j) = c(G[E_q]) \overset{\text{Lemma 3}}{\le} \mathcal{O}$. Since $\sum_{1\le j\in a} c(W_j) \ge \mathcal{O}$, we have $\sum_{1\le j\in a} c(W_j) = \mathcal{O}$. ∎

Smaller-Tree-First-Walk (STF-Walk). When a B-Walk traverses T_u first and then T_v, each edge in T_u is traversed twice while each edge in T_v is traversed once or twice. A simple improvement is to traverse, between T_u and T_v, the one with smaller cost first. We refer to this as *smaller-tree-first-walk (STF-walk)*. The walk in Fig. 4(a) is indeed an STF-walk.

A Heuristic Algorithm. Next we present another heuristic algorithm. It differs with Algorithm 1 only in line 13: Instead of a B-walk along each tree, it does a *longest-path walk*.

Definition 4: (**Longest-Path Walk (LP-Walk).**) Let $P = \{v_1, v_2, ..., v_n\}$ be a longest path in tree T. A *LP-walk* starts from v_1, visiting all the nodes in T following DFS, and ends at v_n, such that every edge in P is traversed once. □

In LP-walk, since more edges are traversed only once, the cost of a walk can be further reduced. Finding longest path in a tree is to find the shortest path among all pair of leaf nodes and choose the longest one, which takes $O(|V|^3)$. Fig 4(b) shows a LP-walk with cost of 14. Because the maximum-weight edge (u, v) is not necessarily on the longest path P, we are not able to obtain performance guarantee for LP-walk. However, we show empirically in Section V that it outperforms Algorithm 1 by $15\% - 30\%$ in terms of energy consumption under different network parameters.

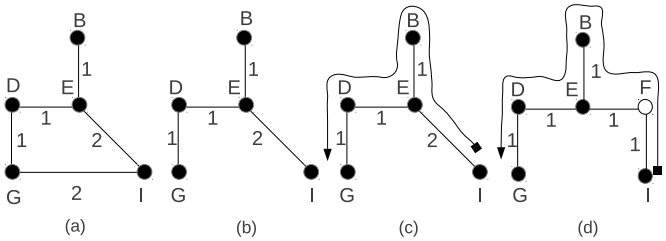


Fig. 5. (a) Aggregation network G' of sensor network G in Fig. 2. (b) 4-edge forest F_q. (c) B-walk on F_q. (d) Aggregation walk with total aggregation cost of 6. The numbers on edges are their weights.

C. Equivalency Between MTSW and DAO^2

Now we transform the original sensor network $G(V, E)$ into an *aggregation network* $G'(V', E')$, and prove that solving DAO^2 in G is equivalent to solving MTSW in G'.

Definition 5: (Aggregation Network $G'(V', E')$) V' is the set of p data nodes in V, $V' = V_d$. For any two data nodes $u, v \in V'$, there exists an edge $(u, v) \in E'$ if and only if all the shortest paths between u and v in G do not contain any other data nodes. For edge $(u, v) \in E'$, its weight $w(u, v)$ is the cost of the shortest path between u and v in G. □

Fig. 5(a) shows the aggregation network G' of sensor network G in Fig. 2. Fig. 5(b) shows a 4-edge forest F_q of G' obtained from Algorithm 1. Fig. 5(c) shows the B-walk on F_q. Finally, we obtain the aggregation walk in G, shown in Fig. 5(d), by replacing each edge (u, v) in F_q with a shortest path between u and v in G. The total aggregation cost following this walk is 6, one more than the optimal cost shown in Example 1. The B-walk in this example happens to be a LP-walk. Note that aggregation network is not necessarily a complete graph, therefore is different from the metric completion of a graph defined in Section III-A.

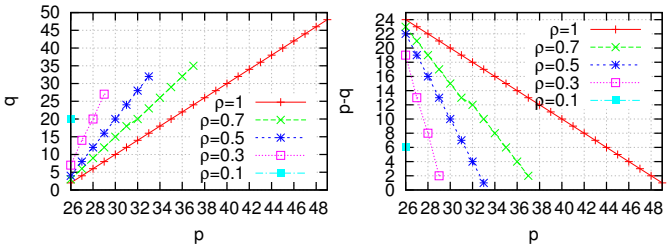
Theorem 3: DAO^2 in G is equivalent to MTSW in G'.

Proof: To prove their equivalence, it is sufficient to show that the data, energy consumption, and topology information that are used for computing energy-efficient aggregation in sensor network $G(V, E)$ are all preserved in aggregation network $G'(V', E')$. Below we show that this is achieved during the construction of G' from G. First, as all the data nodes in G are now nodes in G', data information is preserved. Second, if all the shortest paths between a pair of data nodes X and Y do not contain any other data nodes, then in G' all those shortest paths are replaced by one single edge (X, Y), whose weight is the cost of any of such shortest paths. Therefore the energy consumption information is preserved. Third, if there exists at least one shortest path between data nodes X and Y in G that includes at least another data node as its intermediate node, there is no edge (X, Y) in G'. This mandates that if X and Y participate in the same aggregation walk, they should take a shortest path between them with data nodes as intermediate nodes as part of the aggregation walk. Therefore the topological requirement of DAO^2 to "visit as many data nodes (aggregators) as possible while using as least amount of energy as possible" is preserved (Otherwise, less number of aggregators are visited with the same amount of energy consumption, which is against the goal of DAO^2). Therefore, solving MTSW in G' is equivalent to solving DAO^2 in G. ■

IV. Distributed Data Aggregation Algorithm

We design a distributed data aggregation algorithm, referred to as *Distributed DAO^2*. It is based on the classic GHS algorithm, a distributed, asynchronous algorithm by Gallager, and Humblet, and Spira [11] that finds the optimal minimum spanning tree (MST) of a graph. There is work [6] that constructs distributed MST in wireless ad hoc networks with less energy consumption. However, it is a $O(|V|\log|V|)$-approximation instead of optimal.

Distributed DAO^2. Our distributed algorithm has two stages. First, it finds the aggregation network $G'(V', E')$ using distributed Bellman-Ford algorithm [24]. In particular, each data node sends to its one hop neighbor about its cost to reach all other data nodes. This takes place iteratively and asynchronously until all the data nodes update accurately such information. Second, it finds the minimum q-edge forest in $G'(V', E')$ distributedly. Most steps in second stage are similar to those in original GHS algorithm. We therefore review GHS algorithm first.



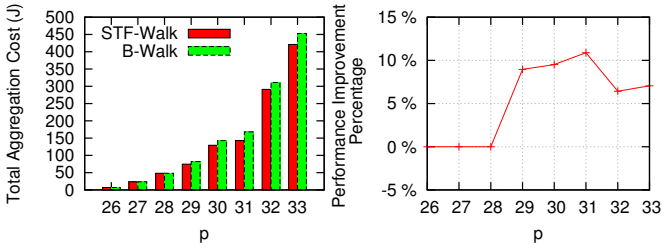
(a) q vs. p. (b) $p - q$ vs. p.

Fig. 6. Valid range of p while varying ρ ($R = m$).

The algorithm maintains a spanning forest of trees. It starts with each node being considered as a fragment, which has level 0 initially. In each "round" of the algorithm, each fragment independently finds its minimum weight outgoing edge (MWOE) and uses this edge to combine with other fragments. Specifically, each fragment has its leader to manage the combining operations, which are either "merge" or "absorb" operations. Absorb operation doesn't change the maximum level among all fragments while merge operation may increase the maximum level by 1.

To find the MWOE, the leaders of two fragments, which are adjacent to the edge added immediately in the previous step, send initiate message to the members of the fragment. Upon receipt of the initiate message, each member node finds its outgoing edge and reports it to the leaders. Upon receipt of reports, the leaders select a new leader, the node that is adjacent to the MWOE for the entire fragment, and then begins a new round. During the execution of the algorithm, an edge that becomes part of the MST is a *branch* edge. The only difference between Distributed DAO^2 and GHS is the termination condition: for GHS, it is when a fragment is unable to find a MWOE; for Distributed DAO^2, it is when the number of branch edges reaches q.

Time and Message Complexity. For the first stage of Distributed DAO^2, both its time and message complexities are $O(p|E|)$. For the second stage, its time complexity is $O(p \cdot \log p)$ while its message complexity is $O((p+|E|) \cdot \log p)$. Therefore, both the time and message complexities of Distributed DAO^2 are $O(p|E|)$.

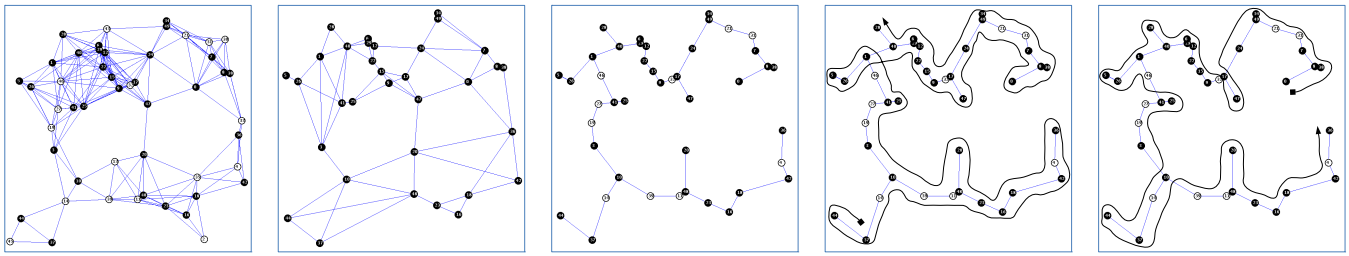


(a) Total aggregation cost. (b) Performance improvement.

Fig. 7. Performance improvement of STF-Walk over B-Walk.

Theorem 4: When there is one initiator allowed, Distributed DAO^2 finds an optimal aggregation cost.

Proof: When there is only one initiator, finding minimum q-edge forest in data aggregation is equivalent to finding a MST in the aggregation network. Consequently, Distributed



(a) Sensor network of 50 nodes. (b) Aggregation network. (c) 32-edge forest. (d) B-Walk (381.2KJ). (e) LP-Walk (290.6KJ).

Fig. 8. Visually comparing B-Walk with LP-Walk with one initiator. Black nodes are data nodes and white nodes are storage nodes, with node ID shown inside. Here, $\rho = 0.5$, $p = 33$, and $q = 32$. ■ and ◄– indicate the first and last node in a walk, respectively.

DAO^2 is equivalent to GHS algorithm. Due to the optimality of finding MST by the GHS algorithm, the optimality of Distributed DAO^2 also sustains. ■

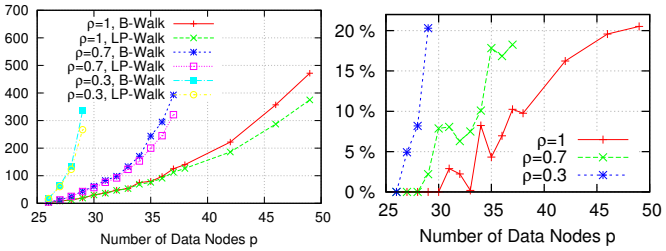
V. Performance Evaluation

A. Centralized Algorithms.

We implemented a Java-based simulator in which 50 and 100 sensors are uniformly distributed in a region of 1000m $\times$ 1000m. Transmission range is 250m. Unless otherwise mentioned, $R = m = 512$MB. We define *correlation coefficient* as $\rho = 1 - r/R$. $\rho = 0$ means no spatial correlation at all, while $\rho = 1$ means perfect correlation (i.e., data at aggregators are duplicate copies of data at initiators thus can be completely removed). Each data point is an average over 10 runs and the error bars indicate 95% confidence interval, wherever applicable. Since the comparison results are similar for 50 and 100 nodes, we only present the results for 50 nodes, because they can be more clearly visualized.

Valid Range of p. Fig. 6(a) shows the valid range of p for different correlation coefficient ρ. When $\rho = 0.1$, the valid range of p is a single value of 26, with corresponding value of q, the number of aggregators, as 20. When increasing ρ, the valid range of p expands, from $26 - 29$ for $\rho = 0.3$, to $26 - 33$ for $\rho = 0.5$, to $26 - 37$ for $\rho = 0.7$, to $26 - 49$ for $\rho = 1$. This is because strong data correlation leads to more data being aggregated, thus allowing more data nodes to exist under overall storage overflow. It also shows that for each ρ, q increases when increasing p. This is because more data nodes means more overflow data and less available storage, therefore more aggregators are needed to achieve enough data size reduction. Finally it shows that for the same p, q decreases when increasing ρ. This is implied by Equation 1, which can be rewritten as: $q = \lceil \frac{p \times (1+m/R) - |V| \times m/R}{\rho} \rceil$. Fig. 6(b) shows the maximum number of initiators $p-q$ for each valid p value. There are two cases in which one initiator is allowed: $\rho = 0.5$ and $p = 33$, and $\rho = 1$ and $p = 49$, while multiple initiators are allowed for other cases.

Performance Improvement of STF-Walk Over B-Walk. We first study the performance improvement of STF-Walk over B-Walk. We choose $\rho = 0.5$, which is a representative correlation coefficient, and vary p from 26 to 33. Fig. 7(a) shows that when p is 26, 27, or 28, both STF-Walk and B-Walk yield the same total aggregation costs. This is because when the number of data nodes p is small, the number of aggregators q is small, causing that the connected components of the resulted q-edge forests are all linear. In linear topologies, aggregation takes place by simply traversing from one end of the linear topology to the other end, resulting the same performances for both STF-Walk and B-Walk. However, when p gets larger, STF-Walk yields less cost and performs better than B-Walk does, because STF-Walk always traverses the smaller subtree twice while B-Walk could possibly traverse the bigger subtree twice. Fig. 7(b) shows that the performance improvement of STF-Walk over B-Walk is around $5\% - 10\%$. Therefore, for the rest of the simulations we choose STF-Walk instead of B-Walk, but still refer to it as B-Walk.



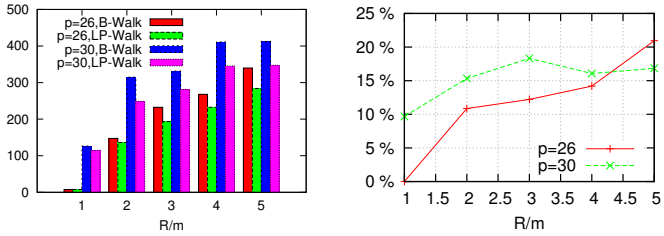
(a) Total aggregation cost (KJ). (b) Performance improvement.

Fig. 9. Comparing B-Walk with LP-Walk by varying p and ρ.

Comparing B-Walk with LP-Walk Visually. Before we perform a comprehensive comparison between B-Walk and LP-Walk, we first compare them visually to gain some insights. We consider $\rho = 0.5$ and $p = 33$, which has 32 aggregators and one initiator. Fig. 8(a) and (b) show such a sensor network and its corresponding aggregation network, respectively. Fig. 8(c) shows the corresponding 32-edge forest. Fig. 8(d) and (e) show the aggregation walks from B-Walk and LP-Walk, respectively. B-Walk visits 32 edges twice, resulting in a total aggregation cost of 381.2 kilojoules (KJ); while LP-Walk only visits 12 edges twice, with a total cost of 290.6KJ, a 23.8% of improvement upon B-Walk.

Comparing B-Walk with LP-Walk by Varying p and ρ. Next we compare B-Walk and LP-Walk while considering the whole ranges of $p \in [26, 49]$ and $\rho = 0.1, 0.3, 0.5, 0.7, 1.0$. Fig. 9(a) shows that for each ρ, with the increase of p, the total aggregation costs of both B-Walk and LP-Walk increase. However, LP-Walk constantly performs better than B-Walk. It also shows that for the same p, with the increase of ρ, the aggregation costs for both B-Walk and LP-Walk decrease. This is because more correlation means that less number of aggregators are visited, thus reducing aggregation costs.

Fig. 9(b) shows the performance improvement percentage of LP-Walk over B-Walk is generally $10\% - 20\%$. Combining the $5\% - 10\%$ performance improvement of STF-Walk over B-Walk, the performance improvement of LP-Walk over B-Walk is therefore around $15\% - 30\%$. Furthermore, we observe the smaller the ρ, the larger the performance improvement percentage is. For example, when $p = 26$ (the only valid value for $\rho = 0.1$), the performance improvement percentage



(a) Total aggregation cost (KJ). (b) Performance improvement.

Fig. 10. Comparing B-Walk with LP-Walk by varying R/m.

for $\rho = 0.1$ is 14% while zero for $\rho = 0.3, 0.5, 0.7, 1.0$. When $\rho = 0.5$, in its valid p range $(26 - 33)$, it almost always has a larger performance improvement percentage compared to $\rho = 0.7, 1$. When less data correlation exists, more aggregators are visited, making the sizes of the resulted q-edge forest as well as its constituent trees larger. By traversing the longest paths of larger trees once, LP-Walk can thus save more aggregation cost compared to traversing smaller trees.

Comparing B-Walk with LP-Walk by Varying R/m. We compare B-Walk with LP-Walk on different R/m. When increasing R/m, the overall storage overflow situation gets more challenging since there are relatively more overflow data compared to available storage spaces. We choose $\rho = 0.5$ and vary R/m from 1 to 5, under which the common valid range of p is $[26, 30]$. Therefore we pick $p = 26$ and $p = 30$ for comparison. Fig. 10(a) shows again that LP-Walk yields less total aggregation cost under different R/m. Fig. 10(b) further shows that the performance improvement percentage of LP-Walk upon B-Walk generally increases when increasing R/m. This shows LP-Walk performs even better in more challenging overall storage overflow scenarios. When increasing R/m, the resulted q-edge forests get larger. This favors LP-Walk, which travels large amount of edges only once.

B. Distributed Algorithm.

Distributed DAO^2 is implemented in DistAlgo [20], a Python-based high-level language for programming distributed algorithms. 100 nodes are randomly placed in a 2000m $\times$ 2000m region. The transmission range is 250m and $m = R = 512$MB. We set the size of each overhead message as 20B. We adopt $\rho = 0.6$ and vary p from 51 to 71, at which point it allows for one initiator.

p	55	60	65	70	71
q	17	34	50	67	70
Number of Initiators	38	26	15	3	1
Centralized (KJ)	78.79	251.76	494.12	787.07	876.29
Distributed (KJ)	209.52	479.12	680.93	827.76	876.29

TABLE II
AGGREGATION COSTS IN CENTRALIZED AND DISTRIBUTED ALGORITHMS.

To show the solution quality, Table II first compares the aggregation costs on the q-edge forests resulted from the Distributed DAO^2 as well as the centralized approximation algorithm (Algorithm 1), while varying number of data nodes p in its valid range [55,71]. It shows that when p is small, the centralized algorithm performs much better than Distributed DAO^2. However, Distributed DAO^2 performs closer to the centralized algorithm when increasing p. When it gets to the case of one initiator ($p = 71$), Distributed DAO^2 yields the same aggregation cost as the centralized algorithm does, since they both result in a minimum spanning tree.

Finally, Fig. 11 compares the total energy costs of the two algorithms by varying number of data nodes. The energy cost for Distributed DAO^2 includes the energy consumption for the aggregation network construction using distributed Bellman-Ford algorithm, the energy consumption of all the overhead messages, as well as the aggregation cost. We observe that while centralized algorithm obviously costs less energy than Distributed DAO^2, their energy consumptions are generally comparable.

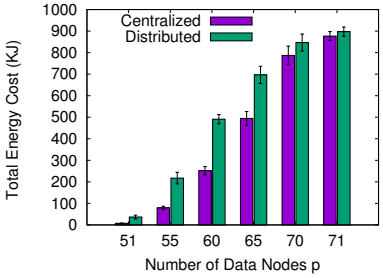


Fig. 11. Total energy costs in centralized and distributed algorithms.

VI. Related Work

MSTW is different from well-known multiple traveling salesman problem (mTSP) [4] and vehicle routing problem (VRP) [32] studied in theory community. In mTSP, a group of traveling salesmen are given, and it needs to decide a tour for each salesman such that the total tour cost is minimized and that each city is visited exactly once. MSTW, however, needs to first decide how many salesmen can be dispatched (and from which cities), then to find the tour for each. Besides, not necessarily all the nodes will be visited in MSTW. In VRP, the set of vehicle nodes and the set of customer nodes are usually disjoint. There is no such distinction in MTSW – each node can either dispatch a salesman or be visited.

In sensor network community, there are extensive research that focused on disconnection-tolerant operations in the absence of the base station. Some system research were conducted to design cooperative distributed storage systems and to improve the utilization of the network's data storage capacity [22, 23]. Other research instead took an algorithmic approach by focusing on the optimality of the solutions [14, 30, 34]. However, all above works assumed that there is enough storage space available to store the overflow data, thus not addressing the overall storage overflow problem.

Intermittently connected sensor networks are different from delay tolerant sensor networks (DTSN) [18]. In DTSNs, mobile nodes are intermittently connected with each other due to their mobility and low density, and data is opportunistically forwarded to destination nodes. In intermittently connected sensor networks, however, all the static sensors are connected with each other while being disconnected from the base station, and data is uploaded to the base station only when uploading opportunities such as data mules become available.

There is vast amount of literature of data aggregation in sensor networks [2, 15, 17, 21, 31]. Tree-based routing structures were often proposed to either maximize network lifetime (the time until the first node depletes its energy) [21], or minimize total energy consumption or communication cost [15, 17], or reduce delay of data gathering [2]. Some other works were based on non-tree routing structures, using mobile base stations to collect aggregated data in order to maximize the network lifetime [28, 31]. Data aggregation in DAO^2,

however, significantly differs from existing data aggregation techniques in both goals and techniques. First, existing data aggregation is to reduce number of transmissions by combining data from different sensors en route to base station, thus saving energy. The goal of data aggregation in DAO^2, however, is to aggregate the overflow data so that they can fit into storage available in the network, thus preventing data loss caused by overall storage overflow. Second, the underlying routing structures in most of the existing data aggregation techniques are trees rooted at the base station covering all sensor nodes. In DAO^2, however, since the base station is not available, those routing structures are no longer suitable. Instead, DAO^2 introduces minimum q-edge forest, a routing structure that serves as the building block of our techniques.

VII. Conclusion and Future Work

This paper introduced DAO^2, an architectural and algorithmic framework that tackles the overall storage overflow problem in intermittently connected sensor networks. We modeled the DAO^2 as a new graph-theoretic problem called multiple traveling salesman walks problem, and designed a suite of energy-efficient optimal, approximation, heuristic, and distributed data aggregation algorithms to solve it. The building block of our algorithmic techniques is minimum q-edge forest, a routing structure that generalizes minimum spanning tree and achieves energy-efficient data aggregation with performance guarantees. Because of this general and theoretical root, the techniques proposed in this paper could possibly be applicable to any applications wherein data correlation and resource constraints coexist.

As it is an architectural framework, there are a few future extensions. We will consider that overflow data generated from different data nodes could have different sizes, and that different storage nodes could have different storage capacities. We will also consider that different data nodes could have different correlation coefficients. Currently data aggregation and data offloading are treated as two separate stages – as ongoing [1] and future work, we will integrate these two stages and explore a more unified energy-efficient solution for the overall storage overflow problem.

Acknowledgment

This work was supported in part by NSF Grant CNS-1419952. We thank Yan Ma and Basil Alhakami for many discussions and simulations.

References

[1] B. Alhakami, B. Tang, J. Han, and M. Beheshti. Dao-r: Integrating data aggregation and offloading in sensor networks via data replication. In *Proc. of GLOBECOM*, 2015.

[2] B. Alinia, M. Hajiesmaili, and A. Khonsari. On the construction of maximum-quality aggregation trees in deadline-constrained wsns. In *Proc. of INFOCOM*, 2015.

[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. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Elsevier Omega*, 34:209–219, 2006.

[5] C. Busch, M. Magdon-Ismail, F. Sivrikaya, and B. Yener. Contention-free mac protocols for wireless sensor networks. In *Proc. of DISC*, 2004.

[6] Y. Choi, M. Khan, V. A. Kumar, and G. Pandurangan. Energy-optimal distributed algorithms for minimum spanning trees. *IEEE Journal on Selected Areas in Communications*, 27(7):1297–1304, 2009.

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

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009.

[9] 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.

[10] M. Di Francesco, S. K. Das, and G. Anastasi. Data collection in wireless sensor networks with mobile elements: A survey. *ACM Trans. Sen. Netw.*, 8(1):7:1–7:31, 2011.

[11] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

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

[13] J.A. Hoogeveen. Analysis of christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10:291 – 295, 1991.

[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] 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.

[16] 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.

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

[18] Y. Li and R. Bartos. A survey of protocols for intermittently connected delay-tolerant wireless sensor networks. *Journal of Network and Computer Applications*, 41:411–423, 2014.

[19] L. Liu, R. Wang, D. Guo, and X. Fan. Message dissemination for throughput optimization in storage-limited opportunistic underwater sensor networks. In *Proc. of SECON*, 2016.

[20] Y. Liu, S. Stoller, B. Lin, and M. Gorbovitski. From clarity to efficiency for distributed algorithms. In *Proc. of OOPSLA*, 2012.

[21] 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.

[22] 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.

[23] 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.

[24] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

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

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

[27] F. Shahzad. Satellite monitoring of wireless sensor networks. *Procedia Computer Science*, 21:479 – 484, 2013.

[28] Y. Shi and Y.T. Hou. Theoretical results on base station movement problem for sensor network. In *Proc. of INFOCOM*, 2008.

[29] R. Sugihara and R. K. Gupta. Path planning of data mules in sensor networks. *ACM Trans. Sen. Netw.*, 8(1):1:1–1:27, 2011.

[30] 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.

[31] S. Tang, J. Yuan, X. Li, Y. Liu, G. Chen, M. Gu, J. Zhao, and G. Dai. Dawn: Energy efficient data aggregation in wsn with mobile sinks. In *Proc. of IWQoS*, 2010.

[32] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2001.

[33] B. Weiss, , H.L. Truong, W. Schott, A. Munari, C. Lombriser, U. Hunkeler, and P. Chevillat. A power-efficient wireless sensor network for continuously monitoring seismic vibrations. In *Proc. of SECON*, 2011.

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

[35] H. Zheng and J. Wu. Data collection and event detection in the deep sea with delay minimization. In *Proc. of SECON*, 2015.

[36] J. Zheng and P. Wang C. Li. Distributed data aggregation using slepianwolf coding in cluster-based wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 59:2564 – 2574, 2010.