Data Preservation in Intermittently Connected Sensor Networks With Data Priority

Xinyu Xue[1], Xiang Hou[1], Bin Tang[2], Rajiv Bagai[1]
[1]Department of Electrical Engineering and Computer Science, Wichita State University, USA
[2]Department of Computer Science, Azusa Pacific University, USA
Email: {xxxue2, xxhou}@wichita.edu, btang@apu.edu, rajiv.bagai@wichita.edu

***Abstract*—Data generated in sensor networks may have different importance and priority. Different types of data contribute differently for scientists to analyze the physical environment. In a challenging environment, wherein sensor nodes do not always have connected paths to the base station, and not all the data can be preserved inside the network due to severe energy constraints and storage constraints at sensor nodes, how to preserve data with maximum priority is a new and challenging problem. In this paper, we study how to preserve data that yield maximum total priorities, under the constraints that each sensor node has limited energy level and storage capacity. We design an efficient optimal algorithm and prove its optimality. The core of the problem is a *maximum weighted flow problem*, which is to maximize the total weight of flow in the network considering different flows have different weights. Maximum weighted flow is a generalization of the classic maximum flow problem, wherein each unit of flow has the same weight. To the best of our knowledge, our work is the first to study and solve the maximum weighted flow problem. We propose a more time efficient heuristic algorithm. Via simulation, we show that it performs comparably to the optimal algorithm and performs better than the classic maximum flow algorithm, which does not consider data priority. Finally we design a distributed data preservation algorithm based on push-relabel algorithm, analyze its time and message complexities, and empirically show that it outperforms the push-relabel distributed maximum flow algorithm in terms of the total preserved priorities.**

Keywords – Data Preservation, Data Priority, Intermittently Connected Sensor Networks, Energy-Efficiency

I. Background and Motivation

Many of the emerging sensor network applications, such as underwater or ocean sensor networks [24], volcano eruption monitoring and glacial melting monitoring [21, 25], are deployed in challenging environments. In those remote or unattended regions, it is not feasible to have long-term deployment of high power data collection base stations (with power outlets) in the field. Consequently, sensory data generated is first stored inside the network and then uploaded to the faraway base station via different means. These uploading opportunities could be periodic visits by human operators or data mules [14], or transmission to the base station through wireless communication such as a low rate satellite link [22]. Due to the fact that the base station does not coexist with other sensor nodes in the field, and that the communication between sensor nodes and base station becomes possible only when such uploading opportunities arise, we refer to such sensor networks as *intermittently connected sensor networks*. The main function of the intermittently connected sensor networks is to collect and store the generated sensory data inside the network before the next uploading opportunity arises.

Meanwhile, sensor networks are evolving from dedicated application-specific platforms measuring single type of data to integrated infrastructure measuring multiple types of data simultaneously. Even within the same application, different types of data could be collected, each of which contributes differently to the sensor network application. For example, in a sensor network that was deployed at Reventador, an active volcano in Ecuador, to monitor the volcano activities, the data collected includes seismic, infrasonic (low-frequency acoustic), and temperature [25]. While all the collected data are helpful for scientists to understand and analyze the volcano activities, seismic and infrasonic data are more crucial than temperature data to interpret the key features of a volcano such as its scale and magnitude. In general, in modern sensor network applications, different types of data has different importance. We refer to the importance of data as data priorities. In this paper, we use the terms weights and priorities interchangeably.

In intermittently connected sensor networks, when events of interest take place, sensors close to them may collect data more frequently than nodes far away, therefore run out of their storage space more quickly than others and cannot store newly generated data. To avoid data loss, the overflow data (that is, the newly generated data that can no longer be stored at data generating nodes due to their storage depletion) must be distributed from such storage-depleted data generating nodes (referred to as *data generators*) to other sensor nodes with available storage spaces (referred to as *destination nodes*), so that they can be uploaded when uploading opportunities become available. We call this process *data preservation in intermittently connected sensor networks*. In this paper, we do not consider the cost for data retrieval, which is done by data mules, human operators, or low-rate satellite link, using techniques such as those proposed in [20] and [18]. There could be data generating nodes whose storage is not yet depleted and can store more data - they are not considered as data generators.

However, preserving all the overflow data in intermittently connected sensor network is not always possible, for the following two reasons. First, data preservation itself incurs energy-expensive wireless communication; if not managed well, it could expedite the energy depletion of sensor nodes and thus exacerbate the data preservation problem. Second, in a more challenging environment wherein sensor nodes

have severe energy constraints, energy depletion of sensor nodes and network partition between data generators and destination nodes will inevitably occur, blocking any further data preservation. Under such severe energy constraints and considering different data have different priorities, how to preserve data with maximum total priorities so that it can be most useful for the scientists is a new and challenging problem. In this paper, we ask the following question: *in a challenging intermittently connected sensor network environment wherein not all the overflow data can be preserved, how to ensure data preservation of maximum total priorities?*

More specifically, we aim to preserve the overflow data so that the total priorities of the preserved data is maximized, under the constraint that each sensor node has limited battery power and storage capacity. We refer to this problem as *data preservation with data priority (DPP)*. We formulate this problem as a graph theoretic problem and study from a network-flow perspective. The main contributions of this paper include the following:

1). We identify, formulate, and solve the data preservation problem in sensor networks by considering data priorities. (Section II and Section IV)

2). We formulate and solve optimally maximum weighted flow problem (MWF), which is a generalization of the classic maximum flow problem [7]. To the best of our knowledge, maximum weighted flow problem has not been studied. (Section III)

3). We design a distributed data preservation algorithm based on classic push-relabel maximum flow algorithm [11]. The time and message complexities of our distributed algorithm are $O(kn^2)$ and $O(n^2m)$ respectively, where n, m, and k are number of nodes, number of edges, and number of data generators respectively. (Section V)

4). We design an efficient heuristic that performs competitively to the optimal algorithm. We also show that the distributed data preservation algorithm performs better than push-relabel algorithm in terms of solution quality. (Section VI)

II. Data Preservation Problem With Data Priority (DPP)

In this section, we formally define the DPP, and discuss related work.

Network Model. The sensor network is represented as an undirected connected graph $G(V, E)$, where $V = \{1, 2, ..., n\}$ is n uniformly deployed sensor nodes, and E is the set of m edges. Two sensor nodes are connected by an edge if they are within transmission range of each other and thus can communicate directly. There are k data generators, denoted as $V_s = \{1, 2, ..., k\}$ without loss of generality. According to our definition, data generators are storage-depleted. Therefore sensor nodes whose storage is not depleted are not considered as data generators. Data generator i is referred to as DG_i. The sensory data are modeled as a sequence of raw data items, each of which is of unit size (our work can be easily extended to the case where data items have different sizes as well). Data items of DG_i all have priority v_i, indicating their importance level in a specific sensor network application. Let d_i denote the number of data items DG_i needs to distribute (that is, the amount of overflow data at DG_i). We do not consider data preservation with temporal and spatial data correlation in this paper, which is left as future work. Let $q = \sum_{i=1}^{k} d_i$ be the total number of data items to be distributed in the network. Let m_i be the available free storage space (in terms of number of data items) at sensor node $i \in V$. If $i \in V_s$, then $m_i = 0$, implying that a DG node is storage-depleted and thus has zero available storage space. If $i \in V - V_s$, then $m_i \geq 0$, implying that non-DG node i can store another m_i data items. We do not assume either $\sum_{i=k+1}^{n} m_i \geq q$ or $\sum_{i=k+1}^{n} m_i < q$, since the techniques proposed in Section III and IV are applicable to both cases.

Energy Model. Sensor node i (including DGs) has a finite and unreplenishable initial energy E_i, which is an integer number. We adopt a unicast communication model, and data items from a DG are distributed one by one. We do not consider energy spent on idle listening for each node. In our energy model, if a node is on the data distribution path of a data item, it costs 1 unit of energy. That is, for each node, sending, receiving, or relaying (receiving and immediately sending) a data item each costs 1 unit of energy. Therefore, the energy consumption of distributing a data item from its DG to a destination node equals to the number of nodes involved in the data distribution. For uniformly deployed sensor nodes, we believe this is a good approximation of the energy consumption. We assume that there exists a contention-free MAC protocol (e.g. [6]) that provides channel access to the nodes. Note that in this paper we do not consider how to retrieve data from destination nodes, which can be solved using techniques in [20] and [18].

General Energy Model. The first order radio model [12] is a more general energy model for wireless communication. In this model, for k-bit data over distance l, the transmission energy $E_{Tx}(k, l) = E_{elec} \times k + \epsilon_{amp} \times k \times l^2$, and the receiving energy $E_{Rx}(k) = E_{elec} \times k$, where $E_{elec} = 100nJ/bit$ is the energy consumption per bit on the transmitter circuit and receiver circuit, and $\epsilon_{amp} = 100pJ/bit/m^2$ calculates the energy consumption per bit on the transmit amplifier. Even though we cannot prove the optimality of our algorithm under this general model, we do implement it using this general model in Section VI and show that it outperforms other algorithms.

Problem Formulation. Let $D = \{D_1, D_2, ..., D_q\}$ denote the set of q overflow data items in the entire network. Let $s(j) \in V_s$, where $1 \leq j \leq q$, denote D_j's DG. The DPP decides:

- the set of data items $\mathcal{D} \subseteq D$ to distribute, and
- *distribution function* $r : \mathcal{D} \rightarrow V - V_s$, indicating that data item $D_j \in \mathcal{D}$ is distributed from $s(j)$ to its *destination node* $r(j) \in V - V_s$, and
- *distribution path* of $D_j \in \mathcal{D}$, referred to as $\mathcal{P}_j : s(j), ..., r(j)$, a simple path (i.e., a set of distinct sensor nodes) along which D_j is distributed from $s(j)$ to $r(j)$. Note that an intermediate node on the path could be any node, including a DG.

Let V_d denote the set of destination nodes, i.e., $V_d =$

$\{r(j)|1 \leq j \leq q\} \subseteq V - V_s$. Let x_{ij} be the energy cost incurred by sensor node i in distributing D_j from $s(j)$ to $r(j)$, and let E'_i denote i's energy level after all the q data items are distributed. Then, $E'_i = E_i - \sum_{D_j \in \mathcal{D}} x_{ij}, \forall\ i \in V$, where $x_{ij} = 1$ if $i \in \mathcal{P}_j$, and $x_{ij} = 0$ otherwise. Here, i could be the DG, the destination node of D_j, or an intermediate relaying node of D_j (each with energy cost one), or not involved with the distribution of D_j at all (with energy cost zero). We assume that a node can still relay data items even though it has a full storage. Table I lists all the notations.

TABLE I
NOTATION SUMMARY

Notation	Explanation
V	The set of sensor nodes
E_i	The initial energy of node i
m_i	The storage capacity of node i
E'_i	The remaining energy of node i after data preservation
V_s	The set of data generators (DGs)
V_d	The set of destination nodes
D	The entire set of data items in the network
$\mathcal{D}$	The set of data items selected to be distributed
DG_i	The i^{th} DG
D_j	The j^{th} data item
d_i	The number of overflow data items at DG_i
v_i	The priority of each data item at DG_i
x_{ij}	The energy cost of node i in distributing D_j
$s(j)$	The DG node of data item D_j
$r(j)$	The destination node of D_j
$\mathcal{P}_j$	The distribution path of D_j

The objective of DPP is to select data items $\mathcal{D} \subseteq D$ to distribute, and find corresponding distribution path $\mathcal{P}_j$ of $D_j \in \mathcal{D}$, to distribute them to their destination nodes, such that the total priorities of the distributed data is maximized, i.e. $\max_{\mathcal{D}} \sum_{D_i \in \mathcal{D}} v_i$, under the energy constraint that each node can not spend more energy than its initial energy level, $E'_i \geq 0, \forall\ i \in V$, and the storage capacity constraint that the number of data items distributed to node i is less than or equal to node i's storage capacity, $|\{j \mid r(j) = i, D_j \in \mathcal{D}\}| \leq m_i, \forall\ i \in V$.

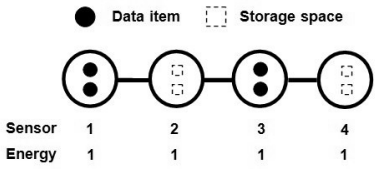


Fig. 1. Illustration of the DPP problem.

EXAMPLE 1: Fig. 1 gives an example of the DPP in a small linear sensor network with four nodes. The initial energy levels of all nodes are 1. Nodes 1 and 3 are DGs, with 2 and 2 overflow data items to distribute respectively. Nodes 2 and 4 are non-DGs, with 2 and 2 available storage spaces respectively. The priority of each data in node 1 is 2, and 1 in node 3. Here, the higher the priority, the more important the data. The optimal solution is that one data item of node 1 is distributed to node 2, while one data item of node 3 is distributed to node 4, resulting in total preserved priority of 3. The other solution that one data item of node 3 is distributed to node 2, which results in preserved priority of 1, is not optimal. □

A. Related Work

Intermittently connected sensor networks bear some resemblance with delay tolerant networks (DTN) [5, 9]. However, these two networks are fundamentally different in both mobility models and objectives. First, in DTNs, mobile nodes are intermittently connected with each other due to their mobility and low density, and data is opportunistically forwarded by relay nodes to destination nodes; in an intermittently connected sensor network, 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 are available. Second, the research objectives in DTNs are mainly to increase data delivery rate or reduce data delivery delay, whereas in intermittently connected sensor networks, the goal is to preserve the most valuable data for maximum amount of time.

Data preservation in intermittently connected sensor networks is a relatively new research topic. Tang et al. [23] study how to minimize the total energy consumption in data preservation process, and formulate it as a minimum cost flow problem. Hou et al. [13] study how to maximize the minimum remaining energy of the nodes that finally store the data, such that the data can be preserved for maximum amount of time. Both works, however, assume that the data can all be successfully distributed from DGs to non-DGs thus being preserved. In more challenging scenarios when nodes reach severely low energy levels and not all the data items (with different priorities) can be preserved, preserving the data with the total maximum priorities is a new problem.

At the core of the data preservation with priority is the *maximum weighted flow* problem: given a source and a destination of a flow network, and that each unit of flow has different weight, the objective is to find a flow of *maximum total weight* from source to destination. The classic maximum flow problem [7] is a special case of the maximum weighted flow problem, in which each unit of flow has the same weight and the goal is to find *maximum amount of flow* from source to destination. To the best of our knowledge, this work is the first one to formulate and study maximum weighted flow problem, and design a polynomial algorithm to solve it optimally.

In theory community, researchers have considered edge priorities in maximum flow problem. Kozen [16] studies a lexicographic flow problem, wherein edges are assigned priorities and the goal is to find a lexicographically maximum flow with respect to such priority assignment. As stated in [16], a lexicographically max flow is not necessarily a max flow. Another related problem, called maximum priority flow, is studied in [2, 4]. In this problem, each node specifies a priority ordering for all the edges leaving it, and the flows leaving this node always go through the edges with higher priority before going through the edges with lower priority. Both work do not consider assigning priority to each individual flow. The most related work to ours is by Fatourou et al. [10]. They study priority-based max-min fairness, wherein each individual session bears a priority and the goal is to assign to each session the maximum possible rate corresponding to

its priority. However, their focus is classic theory of max-min fairness. In the maximum weighted flow problem we study, flows are assigned priorities and the goal is to maximize the total priorities of flows. We show maximum weight flow is indeed a maximum flow but not vice versa.

In sensor network community, priority-based data dissemination has not been studied extensively. We are only aware the following two works. Kumar et al. [17] address how to deliver data with different importance (priority) in the presence of congestion in sensor networks. Kim [15] proposes a quality-of-service MAC protocol based on data priority levels among data transmissions. In contrast, the energy constraints of sensor nodes and their intermittent connectivity with the base stations (which are not considered in above two works), coupled with data priorities, result in the study of our work.

There are extensive works, called data persistence in sensor networks, wherein various network coding techniques are introduced to provide reliable data access in the event of node failure [3, 19]. Data persistence is mainly due to node failure and therefore redundant fragments are spread into the network so that the data can be recovered with maximum reliability. In data preservation, a single copy of overflow data is moved from DG nodes to non-DG nodes, thus no redundancy is introduced.

III. Maximum Weighted Flow Problem (MWF)

In this section we formulate the maximum weighted flow problem and design an efficient and optimal algorithm for it.

Problem Formulation of MWF. Let $G = (V, E)$ be a directed graph. The capacity of an edge $(u, v) \in E$, denoted by $c(u, v)$, is a mapping $c : E \to \mathbb{R}^+$. It represents the maximum amount of flow that can pass through an edge. There are k source nodes $S = \{s_1, s_2, ..., s_k\} \subset V$ and one sink node $t \in V$. A flow on an edge $(u, v) \in E$, denoted by $f(u, v)$, is a mapping $f : E \to \mathbb{R}^+$, subject to the following two constraints:

1). Capacity constraint: $f(u, v) \leq c(u, v), \forall (u, v) \in E$. That is, the flow of an edge cannot exceed its capacity.

2). Flow conservation constraint: $\sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u)$, for each $v \in V \setminus S \cup \{t\}$. That is, the sum of the flows entering a node must equal the sum of the flows exiting a node, except for the source and the sink nodes. For node $s_i \in S$, the net flow $\sum_{u \in V}(f(s_i, u) - f(u, s_i)) > 0$; for node t, the net flow $\sum_{u \in V}(f(t, u) - f(u, t)) < 0$.

Each of the net flow out of $s_i \in S$ has a weight of v_i. We define the total weight of a flow as follows.

Definition 1: (**Total Weight of A Flow.**) Given a flow f in the network, its total weight, denoted as $\mathcal{V}_f$, is the sum of weights of all the net flow out of source nodes. That is, $\mathcal{V}_f = \sum_{s_i \in S} \sum_{u \in V} v_i \times (f(s_i, u) - f(u, s_i))$. It represents the total weight of flow passing from source nodes to the sink node, instead of the total amount of flow desired in the classic maximum flow problem. □

The objective of MWF is to find a flow f from S to t such that $\mathcal{V}_f$ is maximized. Next we present an optimal algorithm for MWF, which is also a greedy algorithm.

Greedy Optimal Algorithm (GOA) for MWF. First, we transform G into G' by adding a super source node s and a directed edge (s, s_i) with capacity $c(s, s_i) = \infty$ for each $i = 1, 2, ..., k$. Then we apply GOA on G'. GOA (Algorithm 1) is essentially a classic maximum flow augmenting path algorithm, such as Edmonds-Karp algorithm [7], executed in non-ascending order of source nodes' priorities. It first maximizes the amount of flow from s to the source node with the highest priority, from where flow goes to t, then the source node with the second highest priority. And so on and so forth until no more augmenting path can be found from s to t. Since the time complexity of Edmonds-Karp algorithm is $O(nm^2)$, the running time of the GOA is therefore $O(knm^2)$. There are more efficient maximum flow algorithms, such as Dinitz's blocking flow algorithm [8] with dynamic tree with $O(mn\lg n)$ running time. However, such algorithms usually rely upon complicated data structures to speed up the efficiency, and can not be easily implemented.

Algorithm 1: Greedy Optimal Algorithm (GOA) on G'.
Input: G', S, t and v_i, where $s_i \in S$;
Output: flow f and its total weight $\mathcal{V}_f$;

0. **Notations**:
   f: current flow from s to t
   G'_f: residual graph of G' with flow f
1. $f = 0$, $G'_f = G'$;
2. Sort source nodes in non-ascending order of their weights: $v_1 \geq v_2 \geq ... \geq v_k$;
3. **for** $(1 \leq i \leq k)$
4. **while** (G'_f contains an augmenting path *s*-s_i-*t*)
5. Augment flow f along such path;
6. **end while;**
7. **end for;**
8. **RETURN** f and $\mathcal{V}_f$.

Lemma 1: Both GOA and an optimal algorithm of MWF yield maximum flow.

Proof: GOA is essentially a maximum flow algorithm with fixed order of choosing augmenting paths according to source nodes' weights. When no more augmenting paths can be found between source and sink in the residual graph [7], it yields maximum amount of flow following maximum flow-minimum cut theorem [7]. By way of contradiction, if an optimal algorithm is not maximum flow, we can further augment the flow and thus yield a solution with larger weight of flow. ∎

Optimality of GOA. Before proving that GOA is optimal, we first give some notations.

Notations. Let $P = \{P_1, P_2, ..., P_k\}$ be the solution yielded by GOA (Algorithm 1), indicating that s_i has P_i amount of net flow in GOA. Let $O = \{O_1, O_2, ..., O_k\}$ be the optimal solution, indicating that s_i has O_i amount of net flow in the optimal solution. We assume that $v_1 \geq v_2 \geq ... \geq v_k$.

Let $\Lambda = \{s_{\lambda(1)}, s_{\lambda(2)}, ..., s_{\lambda(a)}\}$, with $1 \leq \lambda(1) < \lambda(2) < ... < \lambda(a) \leq k$, be the set of a source nodes with $P_{\lambda(i)} > O_{\lambda(i)}$. We refer to $P_{\lambda(i)} - O_{\lambda(i)}$ as the *surplus* of $s_{\lambda(i)}$. The total surplus of Λ is $\sum_{i=1}^{a}(P_{\lambda(i)} - O_{\lambda(i)})$.

Let $\Xi = \{s_{\xi(1)}, s_{\xi(2)}, ..., s_{\xi(b)}\}$, with $1 \leq \xi(1) < \xi(2) < ... < \xi(b) \leq k$, be the set of b source nodes with $P_{\xi(j)} < O_{\xi(j)}$. We refer to $O_{\xi(j)} - P_{\xi(j)}$ as the *deficit* of $s_{\xi(j)}$. The total deficit

of Ξ is $\sum_{j=1}^{b}(O_{\xi(j)} - P_{\xi(j)})$.

Obviously Λ and Ξ are mutually disjoint, and $a + b \leq k$. According to Lemma 1, the total surplus of Λ equals the total deficit of Ξ, that is, $\sum_{i=1}^{a}(P_{\lambda(i)} - O_{\lambda(i)}) = \sum_{j=1}^{b}(O_{\xi(j)} - P_{\xi(j)})$.

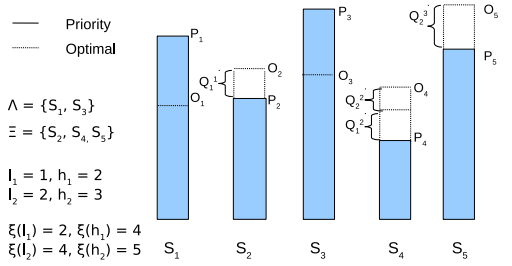


Fig. 2. An example with five source nodes: $s_1, ..., s_5$, illustrating constructive algorithm mapping $\Xi = \{s_2, s_4, s_5\}$ to $\Lambda = \{s_1, s_3\}$. The matching set of s_1 is $\{s_2, s_4\}$, with $P_1 - O_1 = Q_1^1 + Q_1^2$. The matching set of s_3 is $\{s_4, s_5\}$, with $P_3 - O_3 = Q_2^2 + Q_2^3$.

Rewriting $\sum_{j=1}^{b}(O_{\xi(j)} - P_{\xi(j)})$. Next, we are going to rewrite $\sum_{j=1}^{b}(O_{\xi(j)} - P_{\xi(j)})$, a sum of b terms, into a sum of a terms, with i^{th} term having value of $(P_{\lambda(i)} - O_{\lambda(i)})$.

Definition 2: (Matching Set & Matching Amount.) The matching set of $s_{\lambda(i)} \in \Lambda$, denoted as $\{s_{\xi(l_i)}, s_{\xi(l_i+1)}, ..., s_{\xi(h_i)}\} \subseteq \Xi$, where $1 \leq l_i \leq h_i \leq b$, is a subset of Ξ whose total matching amount is $(P_{\lambda(i)} - O_{\lambda(i)})$. Here, the matching amount of $s_{\xi(j)}$ to $s_{\lambda(i)}$, denoted by Q_i^j, is the amount of $s_{\xi(j)}$'s deficit that is allocated for $s_{\lambda(i)}$; the total matching amount of $s_{\lambda(i)}$'s matching set is $\sum_{j=l_i}^{h_i} Q_i^j = (P_{\lambda(i)} - O_{\lambda(i)})$. □

In other words, we will rearrange the total deficit of Ξ and map Ξ to Λ, by finding $s_{\lambda(i)}$'s matching set and calculating the corresponding set of matching amount, such that this matching set's total matching amount equals $s_{\lambda(i)}$'s surplus. Algorithm 2 below decides $s_{\lambda(i)}$'s matching set and calculates the matching amount of each element in this matching set.

Algorithm 2: A Constructive Algorithm Mapping Ξ to Λ.
Input: O_i and P_i, $1 \leq i \leq k$;
Output: l_i, h_i, $\{Q_i^{l_i}, Q_i^{l_i+1}, ..., Q_i^{h_i}\}$, $1 \leq i \leq a$.
0 **Notations**:
$alloc$: total matching amount that is allocated to $s_{\lambda(i)}$;
$flag = true$ if $l_i = h_i$; $false$ if $l_i < h_i$;
1 $l_1 = 1$;
2 $Q_1^1 = alloc = O_{\xi(1)} - P_{\xi(1)}$;
3 **for** $(1 \leq i \leq a)$
4 $j = l_i$; $flag = true$;
6 **while** $\big(alloc < (P_{\lambda(i)} - O_{\lambda(i)})\big)$
7 $j++$;
8 $Q_i^j = O_{\xi(j)} - P_{\xi(j)}$; $alloc = alloc + Q_i^j$;
10 $flag = false$;
11 **end while;**
12 $h_i = j$;
13 **if** $(flag == true)$ $Q_i^j = P_{\lambda(i)} - O_{\lambda(i)}$;
15 **else**
16 $Q_i^j = (O_{\xi(j)} - P_{\xi(j)}) - \big(alloc - (P_{\lambda(i)} - O_{\lambda(i)})\big)$;
17 **if** $\big(alloc == (P_{\lambda(i)} - O_{\lambda(i)})\big)$
18 $l_{i+1} = j + 1$;
19 $alloc = O_{\xi(l_{i+1})} - P_{\xi(l_{j+1})}$;
20 **else**
21 $l_{i+1} = j$;
22 $alloc = alloc - (P_{\lambda(i)} - O_{\lambda(i)})$;
23 **end for;**
24 **RETURN** $[l_1, l_2, ..., l_a]$, $[h_1, h_2, ..., h_a]$, and Q_i^j.

It is not difficult to check that $(P_{\lambda(i)} - O_{\lambda(i)}) = \sum_{j=l_i}^{h_i} Q_i^j$. Note that $s_{\xi(j)}$ could be in multiple matching sets. That is, the deficit of $s_{\xi(j)}$, $O_{\xi(j)} - P_{\xi(j)}$, may be divided into multiple parts, each is allocated to the matching set of a different $s_{\lambda(i)}$. Fig. 2 is an example to illustrate this algorithm. Algorithm 2 immediately gives us the following result.

Lemma 2: $\lambda(i) < \xi(l_i)$, $1 \leq i \leq a$.

Proof: By way of contradiction, assume that there exists j such that $\lambda(j) > \xi(l_j)$, and $\lambda(i) < \xi(l_i)$ for all $1 \leq i \leq j-1$. Since $(P_{\lambda(i)} - O_{\lambda(i)}) = \sum_{j=l_i}^{h_i} Q_i^j$, for all $1 \leq i \leq j-1$, Algorithm 1 found a way to $equalize$ number of flows obtained in O to number of flows obtained in P, for all s_i with $i \leq \xi(l_j)$, by moving flows from source nodes in Ξ to source nodes in Λ. Next when both GOA and optimal algorithm try to find the number of flows for $s_{\xi(l_j)}$, it obtains that $P_{\xi(l_j)} < O_{\xi(l_j)}$, because $s_{\xi(l_j)} \in \Xi$. This contradicts the fact that GOA is a greedy algorithm that finds the maximum number of data to distribute from $s_{\xi(l_j)}$ at this stage. ∎

Theorem 1: GOA is an optimal algorithm. That is, it finds flow with total maximum weight.

Proof: By way of contradiction, assume that GOA is not optimal, therefore $\sum_{i=1}^{k} O_i \times v_i > \sum_{i=1}^{k} P_i \times v_i$. It can be shown that

$$\begin{aligned}
& \sum_{i=1}^{k} P_i \times v_i - \sum_{i=1}^{k} O_i \times v_i \\
= & \sum_{i=1}^{a}(P_{\lambda(i)} - O_{\lambda(i)}) \times v_{\lambda(i)} - \sum_{j=1}^{b}(O_{\xi(j)} - P_{\xi(j)}) \times v_{\xi(j)} \\
= & \sum_{i=1}^{a}(P_{\lambda(i)} - O_{\lambda(i)}) \times v_{\lambda(i)} - \sum_{i=1}^{a}\sum_{j=l_i}^{h_i} Q_i^j \times v_{\xi(j)} \\
= & \sum_{i=1}^{a}\left((P_{\lambda(i)} - O_{\lambda(i)}) \times v_{\lambda(i)} - \sum_{j=l_i}^{h_i} Q_i^j \times v_{\xi(j)}\right).
\end{aligned}$$

According to Lemma 2, for any $1 \leq i \leq a$, $v_{\lambda(i)} \geq v_{\xi(j)}$, the priority of $s_{\xi(j)}$ in $D_{\lambda(i)}$'s matching set. And we have $(P_{\lambda(i)} - O_{\lambda(i)}) = \sum_{j=l_i}^{h_i} Q_i^j$. Each of the above a difference terms is therefore non-negative. Thus we have $\sum_{i=1}^{k} P_i \times v_i \geq \sum_{i=1}^{k} O_i \times v_i$, a contradiction. Note that h_a equals ξ_b due to the fact that $\sum_{i=1}^{k} O_i = \sum_{i=1}^{k} P_i$ (Lemma 1). ∎

IV. Optimal Algorithm for DPP

In this section we first show that the GOA algorithm in Section III is an optimal algorithm for DPP. We then present another heuristic algorithm, which is both efficient (in terms of running time) and effective (in terms of data preservation as

shown in Section VI). We first transform the sensor network $G(V, E)$ to a flow network $G'(V', E')$ as follows:

1). Replace each undirected edge $(i, j) \in E$ with two directed edges (i, j) and (j, i). Set the capacities of all the directed edges as infinity.
2). Split node $i \in V$ into two nodes: *in-node* i' and *out-node* i''. Add a directed edge (i', i'') with capacity of E_i, the initial energy level of node i. All the incoming directed edges of node i are incident on i' and all the outgoing directed edges of node i emanate from i''. Therefore the two directed edges (i, j) and (j, i) in above are now (i'', j') and (j'', i').
3). Add a source node s, and connect s to the in-node i' of the DG node $i \in V_s$ with an edge of capacity d_i.
4). Add a sink node t, and connect out-node j'' of the non-DG node $j \in V - V_s$ to t with an edge of capacity m_j.

Therefore, $V' = \{s\} \cup \{t\} \cup \{i' : i \in V\} \cup \{i'' : i \in V\}$ and $E' = \{(i'', j') : (i, j) \in E\} \cup \{(j'', i') : (i, j) \in E\} \cup \{(i', i'') : i \in V\} \cup \{(s, i') : i \in V_s\} \cup \{(j'', t) : j \in V - V_s\}$. We have $|V'| = 2n + 2$ and $|E'| = 2m + 2n$. Fig. 3 shows the transformed network for the linear network in Fig. 1.

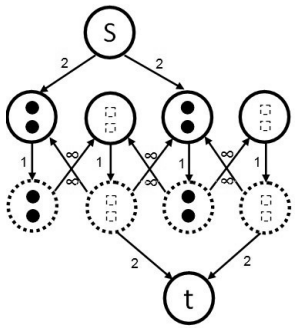


Fig. 3. Transformed network $G'(V', E')$ for linear network in Fig. 1.

Theorem 2: GOA algorithm on $G'(V', E')$ is an optimal algorithm for DPP on $G(V, E)$.

Proof: Note that with the above transformation from $G(V, E)$ to $G'(V', E')$, the energy constraints of sensor nodes and storage capacities of non-DG nodes are specified as edge capacities in $G'(V', E')$ that need to conform with. Next, we need to show that the maximum weighted flow resulted from GOA on $G'(V', E')$ correspond to the data preservation on $G(V, E)$ that preserves maximum amount of data priorities.

Suppose that GOA gives net flow of P_i amount from s_i to t. Because GOA is optimal (Theorem 1), $\sum_{i=1}^{k} P_i \times v_i$ is maximum. With the above transformation from $G(V, E)$ to $G'(V', E')$, P_i is actually the amount of data items distributed from source node s_i to sink node t, v_i is actually the priority of data items generated by s_i. Therefore, the corresponding data distributed from all s_i to t have the maximum amount of priorities. Solving the maximum weighted flow problem on $G'(V', E')$ provides the optimal solution for the data preservation problem with priority in $G(V, E)$. ∎

We have the following observation regarding the optimal solution for DPP:

Observation 1 (Cascading Effect): A non-DG stores data until its storage capacity is full before relaying data. □

A consequence of Observation 1 is that in the optimal algorithm, data is always distributed to the nearest non-DG with available storage before being distributed further. We call this the *cascading effect* of data preservation.

A Heuristic Algorithm on $G(V, E)$. We present an efficient heuristic algorithm (Algorithm 3) for DPP. Like the optimal algorithm, this algorithm distributes data in the descending order of their priorities. However, it is not a flow algorithm and thus residual graph and flow undo are not used, which makes its implementation much simpler. In each iteration, a DG distribute a data to its closest non-DG node with available storage, if all the nodes along the path from this DG to this non-DG have at least one unit of energy. The algorithm stops when no more data can be distributed.

Algorithm 3: Heuristic Algorithm on $G(V, E)$.
Input: $G(V, E)$
Output: flow f and its total weight $\mathcal{V}_f$

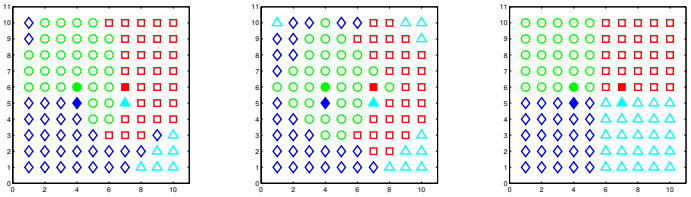
```
1.  Sort all the DGs in descending order of their priorities:
    v_1 ≥ v_2 ≥ ... ≥ v_k
2.  for (1 ≤ i ≤ k)
3.    while (It can still distribute a data item from DG i
4.      to a non-DG node)
5.      Distribute it to the closest non-DG node;
6.    end while;
7.  end for;
8.  RETURN f and V_f.
```

Time Complexity. Due to space constraint, we omit detailed analysis here. Its time complexity is $O(km + k\bar{d}n)$, where $\bar{d}$ is the average number of data items of each DG. This is more efficient than that of GOA, which is $O(knm^2)$.

V. Distributed Data Preservation With Data Priority

We design a distributed algorithm by combining the idea behind push-relabel maximum flow algorithm [11] and data priority-based data preservation. In push-relabel algorithm, each node only needs to know its neighbors and the capacities of its incident edges in the residual graph [7]. Therefore it is desirable for a distributed sensor network environment.

Overview of Push-Relabel Algorithm [11]. Given a flow network $G(V, E)$ with $|V| = n$ and $|E| = m$, a source node s and a sink node t, the algorithm works as follows. It begins by sending as much flow out of s as allowed by the capacities of the edges coming out of s. Then, at each iteration, it considers a node that has more incoming flow than outgoing flow (the difference between them is called *excess flow* of the node). The node then routes the excess flow to their neighbors, and so on. This is called $push$. To make progress, the algorithm defines a height function $h : V \to N$ and initially, $h(s) = n$; $h(t) = 0$; and $h(u) = 0$ for all $u \notin V \setminus \{s, t\}$. This is based on the intuition that the flow always goes "downhill", node u sends extra flow to a node v only if $h(u) > h(v)$. When a node can no longer sends out its excess flow, it increases its height and pushes the flow to the node with lower height than it. This is



(a) Optimal. (b) Heuristic. (c) Edmonds-Karp.

Fig. 4. Data Preservation Blocked by Storage Constraint.

called $relabel$. A maximum flow is obtained when no more overflowing nodes left except the sink and possibly the source node. The running time of push-relabel algorithm is $O(n^2m)$. The detailed operations of push-relabel algorithm on a node u is presented in Algorithm 4.

Algorithm 4: Push-Relabel (u)

0. **Notations:**
   $e(u)$: node u's excess flow
   $h(u)$: node u's height
   $cap(u, w)$: residual capacity of (u, w)
1. **if** $e(u) > 0$
2: **while** $\big(e(u) > 0$, there exists (u, w) s.t.
3: $h(u) = h(w) + 1$, and $cap(u, w) > 0\big)$
4: Push $y = \min\{e(u), cap(u, w)\}$ through
5: (u, w) by sending a message to w;
6: $e(u) = e(u) - y$; $e(w) = e(w) + y$;
7: update $cap(u, w)$;
8: **end while;**
9: **if** $e(u) > 0$
10: $h(u) = 1 + \min\{h(w) : cap(u, w) > 0\}$;
11: Broadcast $h(u)$ to neighboring nodes;
12: **end if;**
13: **end if;**
15. **RETURN**

Distributed Data Preservation with Data Priority. The flow in push-relabel algorithm bears intrinsic resemblance with the cascading effects exhibited by the data flow in our data preservation problem (see Observation 1). In push-relabel, the flow goes through the network as water flows through downhills and in data preservation, data is always distributed to the nearest non-DG with storage before being distributed farther away. Therefore, push-relabel algorithm is particularly suitable for our data preservation scheme.

However, there are a few modifications needed on push-relabel algorithm to make the distributed data preservation work. First, since push-relabel algorithm is to find maximum flow while data preservation is to find maximum preserved priorities, the DGs need to coordinate with each other so that DGs with higher priorities push data into the network before DGs with lower priority do. Second, energy constraint of each node should be represented in the flow network, and energy consumption of sending and receiving packets by each node should be taken into account. To handle energy constraints and energy consumptions of nodes, nodes are splitted according to Section IV. Third, push-relabel algorithm determines the maximum flow from a single source to a single sink, whereas in a sensor network, there are multiple data generating sensor nodes and multiple sensor nodes collecting and storing sensed data. Therefore, as specified in Section IV, a virtual source node s is connected with the in-node of each DG, with the edge capacity as the number of data items of each DG, and a virtual sink node t is connected with the out-node of each non-DG, with the non-DG's storage as edge capacity.

The distributed data preservation begins by this virtual source pushing maximum allowable number of flow to the in-node of the DG with highest priority, which continues the push-relabel process, until no nodes with excess flow (except virtual source and sink) exists. Then the virtual source pushes to the in-node of the DG with the second highest priority. The algorithm works in rounds, in each round a node with positive excess performs push-relabel. Such synchronization prevents multiple nodes from sending their packets to their neighbors simultaneously. When there are multiple neighbors with the same number of heights, one of them is randomly picked. The distributed algorithm stops when the DG with the lowest priority finishes the push-relabel process. The detailed operations of the distributed data preservation is presented in Algorithm 5.

Algorithm 5: Distributed Data Preservation on $G'(V', E')$

1 Each DG broadcasts its priority to the network;
2. **for** $\big($Each DG in the descending order of its priority$\big)$
3. s pushes maximum allowable data to this DG;
4. **while** $\big($there exists a node u with positive excess$\big)$
5. Push-Relabel(u);
6. **end while;**
7. **end for;**
8. **RETURN**

Theorem 3: The distributed data preservation algorithm preserves maximum total priority. It runs in $O(kn^2)$ time and uses $O(n^2m)$ messages.

Proof: The optimality of the distributed data preservation algorithm is due to the optimality of the distributed push-relabel algorithm [11]. It is shown in [11] that the distributed push-relabel algorithm in a graph $G'(V', E')$ runs $O(|V'|^2)$ in time and uses $O(|V'|^2|E'|)$ messages. Since $|V'| = 2n + 2$ and $|E'| = 2m + 2n$ (Section IV), its time and message complexities are $O(n^2)$ and $O(n^2m)$ respectively. The distributed data preservation differs from distributed push-relabel algorithm in lines 1 and 2 in Algorithm 5. Line 1 incurs $O(kn) = O(n^2)$ broadcast messages and Line 2 increases the running time by at most k times. Therefore the distributed algorithm runs in $O(kn^2)$ and uses $O(n^2m + n^2) = O(n^2m)$ messages. ∎

VI. Performance Evaluation

We compare the performance of the optimal GOA algorithm (referred to as **Optimal**), the heuristic (referred to as **Heuristic**), and the distributed algorithm (referred to as **Distributed**). We also implement Edmonds-Karp maximum flow algorithm

[7] (referred to as **Edmonds-Karp**), which does not consider flow priorities when preserving data.

Visual Performance Comparison in Grid Network. There are four DGs in the grid network, located at (4, 6) (solid circle), (7,6) (solid square), (4, 5) (solid diamond), and (7, 5) (solid triangle), with data priority 8, 6, 4, and 2, respectively. Each non-DG node has one storage and can store one data item.

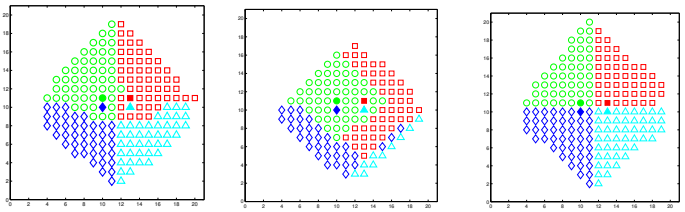
Data Preservation Blocked by Storage Constraint. We first investigate the data preservation when there is not enough space in the network to store all the overflow data, as shown in Fig. 4. We set the grid network size as 10×10. Each node has initial energy level of 30 units. Each DG has 30 data items to distribute. The number of data items (from highest priority to lowest) distributed by Optimal is $[30, 30, 30, 6]$, distributed by Heuristic is $[30, 28, 28, 10]$, and distributed by Edmonds-Karp is $[24, 24, 24, 24]$. It shows that both Optimal and Heuristic prefer to preserve data with higher priority, while Optimal does much better to "filter" out the low priority data. Edmonds-Karp, however, distributes the same amount of data of different priority as Optimal does. Table II presents the comparison results summarized from Fig. 4.

TABLE II
RESULTS OF VISUAL COMPARISON IN FIG. 4.

	Optimal	Heuristic	Edmonds-Karp
Number of Preserved Data	96	96	96
Total Preserved Priority	552	540	480

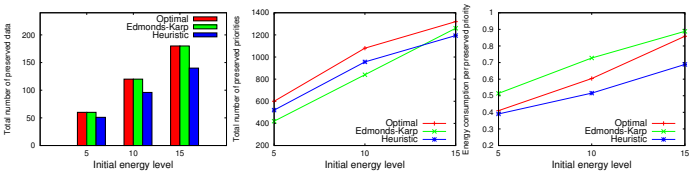
Data Preservation Blocked by Energy Constraint. We also investigate the data preservation when there is not enough energy in the network to store all the overflow data. We set the network size as 20×20. Each of the four DGs has 50 data items to offload. The initial energy level of sensor nodes are increased from 5 to 10 to 15. Due to space constraint, we only show the visual comparison for energy level 15 (Fig. 5). Fig. 6 shows the performance comparison of the three algorithms, in terms of the total number of preserved data, total number of preserved priorities, and energy consumption per preserved data priority, in varying energy levels. It shows that Optimal performs best constantly in terms of total amount of preserved priorities. However, in terms of energy consumption per preserved priority, Heuristics performs better than Optimal, which performs better than Edmonds-Karp. This can be explained by that Heuristics offloaded least amount of priorities, as indicated by Fig. 5 (a).

Performance Comparison Under General Energy Model and General Topology. We randomly generate 100 sensor nodes in a field of $2000m \times 2000m$, and set the transmission range as $100m$. Each time a new node is generated, we check if its closest distance to the existing nodes is within the transmission range (otherwise, this node is discarded). This way, we can guarantee that all the 100 nodes are connected. The initial energy of nodes are $500mJ$. Each DG has 500 data items, each data item is 400 bytes. The storage capacity of each non-DG is 51.2 Kbytes. The priority of each DG is a random number in [1, 100]. Next we compare three algorithm



(a) Optimal. (b) Heuristic. (c) Edmonds-Karp.

Fig. 5. Data Preservation Blocked by Energy Constraint (Initial Energy Level = 15).



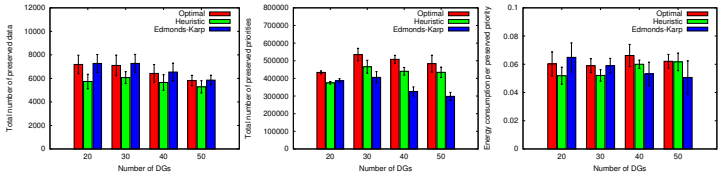
(a) Total number of preserved data. (b) Total number of preserved priorities. (c) Energy consumption per preserved priority.

Fig. 6. Performance Comparison With Varying Initial Energy Level.

by varying number of DGs. In all plots, each data point is an average over five runs. and the error bars indicate 95% confidence interval.

Fig. 7 compares the performances of Optimal, Heuristic, and Edmonds-Karp, by varying number of DGs in [20, 30, 40, 50]. Fig. 7 (a) shows that Edmonds-Karp preserves more amount of data than Optimal and Heuristic (it leaves as future work to investigate theoretically if Edmonds-Karp yields maximum flow under this general energy model). Even so, Fig. 7 (b) shows that Optimal preserved more priorities than Heuristic, which preserves more than Edmonds-Karp does. This seems to be more prominent when number of DGs is large (at 30, 40, and 50). However, Edmonds-Karp does yield less energy consumption per preserved priority than Optimal and Heuristic, since it always finds the globally minimum energy path when offloading a data item.

Comparing Priority-Based Distributed Algorithm and Distributed Push-Relabel Algorithm. Fig. 8 compares distributed data preservation with data priorities (referred to as Distributed) and without considering data priorities (referred to as PushRelabel). We use and modify the implementation of distributed push-relabel algorithm that is available at [1]. It shows that both algorithms achieve the same amount of distributed data. However, the Distributed yields higher total preserved priority than that of PushRelabel. Due to the broadcast messages that coordinate the data distributing of different DGs, distributed data preservation does incur a bit more energy consumption.

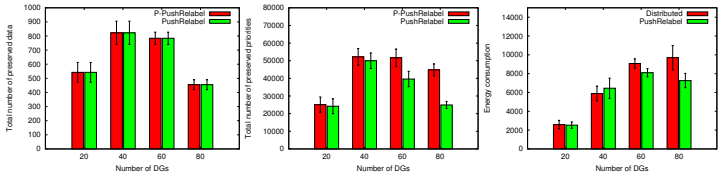


(a) Total number of preserved data.

(b) Total number of preserved priorities.

(c) Energy consumption per preserved priority.

Fig. 7. Performance Comparison Under General Energy Model.



(a) Total number of preserved data.

(b) Total number of preserved priorities.

(c) Total energy consumption.

Fig. 8. Comparing Distributed and PushRelabel.

VII. Conclusion and Future Work

The maximum weighted flow problem studied in this paper is uniquely derived from the data preservation problem with data priority (DPP) we identified in sensor networks. We believe that it is a theoretically fundamental problem since it generalizes the classic maximum flow problem. Because of this theoretical root, the techniques proposed in this paper could be applicable not only in sensor networks, but also in any applications in which data priorities and resource constraints coexist, such as peer-to-peer networks, data centers, and smartphone communication. Currently the DPP is a static problem, in which the data to be preserved is generated at the beginning and only once, and the set of storage-depleted nodes may vary over time due to the depletion of nodes' storage and the consumption of data in nodes' storage. As future work, we will address a real-time problem where data is generated and transmitted dynamically and periodically, and storage-depleted nodes vary over time. Second, the paper assumes that the overflow data of DGs can be distributed to only the non-DGs. To maximize the total priorities preserved, it would be interesting to explore if certain DGs can discard their locally generated data of low priority and make room to store the data from other DGs having high priority. Third, currently, under general energy model, it is not clear if Edmonds-Karp gives maximum flow, and it is not clear if GOA algorithm gives total optimal priorities. We will study these two topics theoretically.

Acknowledgment

This work was supported in part by NSF Grants CNS-1116849 and EPS-0903806.

References

[1] http://avglab.com/andrew/soft.html.

[2] Maximum priority flow. http://www.nada.kth.se/ viggo/wwwcompendium/node120.html.

[3] Salah A. Aly, Zhenning Kong, and Emina Soljanin. Fountain codes based distributed storage algorithms for large-scale wireless sensor networks. In *Proc. of IPSN*, 2008.

[4] Mihir Bellare. Interactive proofs and approximation: Reductions from two provers in one round. In *Proceedings of the Second Israel Symposium on Theory and Computing Systems*, pages 266–274, 1992.

[5] Muhammad Mukarram Bin Tariq, Mostafa Ammar, and Ellen Zegura. Message ferry route design for sparse ad hoc networks with mobile nodes. In *Proc. of MobiHoc*, 2006.

[6] Costas Busch, Malik Magdon-Ismail, Fikret Sivrikaya, and Bulent Yener. Contention-free mac protocols for wireless sensor networks. In *Proc. of DISC*, pages 245–259, 2004.

[7] Thomas Corman, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2009.

[8] Yefim Dinitz. Algorithm for solution of a problem of maximum flow in a network with power estimation. page 12771280, 1970.

[9] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proc. of SIGCOMM*, 2003.

[10] Panagiota Fatourou, Marios Mavronicolas, and Paul Spirakis. Max-min fair flow control sensitive to priorities. In *Proceedings of the 2nd International Conference on Principles of Distributed Systems*, pages 45–59, 1999.

[11] A V Goldberg and R E Tarjan. A new approach to the maximum flow problem. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, pages 136–146, 1986.

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

[13] Xiang Hou, Zane Sumpter, Lucas Burson, Xinyu Xue, and Bin Tang. Maximizing data preservation in intermittently connected sensor networks. In *Proc. of IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2012)*. Short Paper.

[14] 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.

[15] Hoon Kim. Priority-based qos mac protocol for wireless sensor networks. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS 2009)*.

[16] Dexter Kozen. Lexicographic flow. Technical report, Computing and Information Science, Cornell University, June 2009.

[17] Raju Kumar, Riccardo Crepaldi, Hosam Rowaihy, Albert F. Harris III, Guohong Cao, Michele Zorzi, and Thomas F. La Porta. Mitigating performance degradation in congested sensor networks. *IEEE Transactions on Mobile Computing*, 7(6):682–697, June 2008.

[18] Ke Li, Chien-Chung Shen, and Guaning Chen. Energy-constrained bi-objective data muling in underwater wireless sensor networks. In *Proc. of the 7th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2010)*, pages 332–341, 2010.

[19] Yunfeng Lin, Ben Liang, and Baochun Li. Data persistence in large-scale sensor networks with decentralized fountain codes. In *Proc. of INFOCOM*, 2007.

[20] Ming Ma and Yuanyuan Yang. Data gathering in wireless sensor networks with mobile collectors. In *Proc. of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008)*, pages 1–9, 2008.

[21] K. Martinez, R. Ong, and J.K. Hart. Glacsweb: a sensor network for hostile environments. In *Proc. of SECON 2004*.

[22] Ioannis Mathioudakis, Neil M. White, and Nick R. Harris. Wireless sensor networks: Applications utilizing satellite links. In *Proc. of the IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2007)*, pages 1–5, 2007.

[23] Bin Tang, Neeraj Jaggi, Haijie Wu, and Rohini Kurkal. Energy efficient data redistribution in sensor networks. *ACM Transactions on Sensor Networks*, 9(2), May 2013.

[24] 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*.

[25] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proc. of OSDI 2006*.