Using Mobile Mules for Collecting Data from an Isolated Wireless Sensor Network

Yu-Chee Tseng¹, Wan-Ting Lai¹, Chi-Fu Huang², and Fang-Jing Wu¹
¹Department of Computer Science, National Chiao Tung University, Taiwan
²Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan
Email: {yctseng, wtlai, cfhuang, fangjing}@cs.nctu.edu.tw

Abstract—This paper considers storage management in an isolated WSN, under the constraint that the storage space per node is limited. We formulate the memory spaces of these sensor nodes as a distributed storage system. Assuming that there is a sink in the WSN that will be visited by mobile mules intentionally (e.g., pre-arranged buses) or occasionally (e.g., non-pre-arranged taxis), we address three issues: (1) how to buffer sensory data to reduce data loss due to shortage of storage spaces, (2) if dropping of data is inevitable, how to avoid higher priority data from being dropped, and (3) how to keep higher priority data closer to the sink, such that the mobile mules can download more important data first when the downloading time is limited. We propose a Distributed Storage Management Strategy (DSMS) based on a novel shuffling mechanism similar to heap sort. It allows nodes to exchange sensory data with neighbors based on only local information. To the best of our knowledge, this is the first work addressing distributed and prioritized storing strategies for isolated WSNs.

Index Terms—distributed computing, distributed storage, mule, protocol, wireless communication, wireless sensor network.

I. INTRODUCTION

Wireless sensor networks (WSNs) have gained much attention recently [1]. A WSN is composed of a large number of nodes, each of which is a microprocessor with multiple sensors onboard. Nodes can communicate with one another through their wireless interfaces. WSNs have many applications, such as military safety, health care, and environment surveillance [2][3][4][5].

In this paper, we consider an isolated WSN [6] that is disconnected from outside world for the most of the time. It thus relies on mobile mules [7] to visit it and carry its sensory data to the outside world. A WSN may be isolated for many reasons, such as node failure owing to destructive events or distance and cost constraints. In particular, this work is motivated by the project in [8], which is targeted at using mobile wireless sensors for in-situ tracking of debris flows in wild mountain areas which are hard to reach by vehicles or human. Collecting data from such isolated WSNs may rely on rangers or hiker (when they reach those areas) to relay the sensing data back, as shown in Fig. 1. As another example, a WSN could be deployed under water to monitor undersea oilfields [9]. Such applications do not need real-time information, so data can be collected once in several months.

Since mules may not visit an isolated WSN frequently, how to buffer data is a challenging issue. We thus address the corresponding storage management problem in such an environment, under the constraint that the storage space per node is limited. (For example, MPR300CB has only 4KB RAM and 4KB ROM [10].) We formulate the memory spaces of sensor nodes as a distributed storage system. Assuming that there is a sink in the WSN that will be visited by mobile mules intentionally (e.g., pre-arranged buses) or occasionally (e.g., non-pre-arranged taxis), we address three issues: (1) how to buffer sensory data to reduce data loss due to shortage of storage spaces, (2) if dropping of data is inevitable, how to avoid higher priority data from being dropped, and (3) how to keep higher priority data closer to the sink, such that the mobile mules can download more important data first when the downloading time or mule storage is limited. In (3), the remaining data may be downloaded next time. We propose a Distributed Storage Management Strategy (DSMS) for data buffering in an isolated WSN. DSMS is designed based on a novel shuffling mechanism similar to heap sort [11] to keep higher priority data closer to the sink. However, unlike heap sort, which is based on a tree structure, DSMS uses a mesh-like structure to facilitate data exchange and thus to keep higher-priority data closer to the sink.¹ Also note that the location of the sink could be any node in the WSN. In the above mud flood example, the node that is most frequently visited by rangers can be the sink.

¹Note that heap sort must be conducted in a complete binary tree. Insertion begins at a leaf and moves up toward the root, while deletion begins by removing the root element, moving the rightmost leaf element to the root, and then adjusting the heap. These operations are basically centralized operations and can not be applied directly to a realistic distributed WSN environment.
Whenever a mule arrives at the sink, the sink can communicate with the mule. However, we assume that the connection time and the available storage space of the mule are both unpredictable. Therefore, packets generated by nodes should be prioritized to reflect their importance. This can be done by a pre-agreed function, by an aging process, or by importance of readings. Higher priority means more importance. We also assume that there is a predefined region nearby the sink called Buffer Area (BA). The set of nodes in BA are designated to store sensing data of the WSN which is to be forwarded to the mule whenever there is one arriving. Therefore, all static sensor nodes will try to forward their data to BA. As an example, in Fig. 2, the BA of network C contains nodes within 3 hops from the sink. (In an extreme case, one may designated all WSN as the BA.) We regard the storage spaces of nodes in BA as a distributed storage system. Our goal is to design a distributed protocol to achieve three goals.

**G1** : Dropping of packets in BA should be minimized.

**G2** : If dropping of packets is unavoidable, the lower-priority ones should be dropped first.

**G3** : To facilitate mobile mules to collect data, higher priority packets should be stored closer to the sink.

**Definition 1.** Given a graph $G = (V, E)$, a sink $v \in V$, a subset $BA \subseteq V$, and a priority assignment function $F$ for packets, the Distributed Storage Management (DSM) problem is to develop a packet exchange protocol to maintain packets being generated by the WSN within BA such that $G1$-$G3$ are met and $\Omega(BA) = \sum_{v \in BA, p \rightarrow v} F(p)$ is maximized, where $p \rightarrow v$ means that a packet $p$ is stored at the storage of $v$.

### III. Properties and Protocols of DSMS

Intuitively, the objective function $\Omega(BA)$ in Definition 1 means that the total value of the sensing data inside BA should be as large as possible. We propose a Distributed Storage Management Strategy (DSMS) based on a shuffling mechanism. DSMS is a distributed solution. Nodes not in BA will forward their packets to BA, while nodes in BA will observe their neighbors’ states and exchange packets with each other as necessary. We assume that after the WSN is deployed, each node $u$ has calculated its distance $D(u)$ to the sink and its neighbor set $N(u)$. Without loss of generality, we assume that each node $u$ has only one buffer space (i.e., $S_{sn} = 1$). So the (only) packet in $u$ is written as $P(u)$ and its priority is $F(P(u))$ (if $u$ has no packet, $F(P(u)) = -1$). Our scheme can be easily extended to $S_{sn} > 1$.

DSMS tries to maintain the following properties for each node $u \in BA$

**P1** : For each node $v \in N(u)$ such that $D(v) > D(u)$, $F(P(v)) \leq F(P(u))$.

**P2** : For each node $v \in N(u)$ such that $D(v) < D(u)$, $F(P(v)) \geq F(P(u))$.

**P3** : For each node $v \in N(u)$ such that $D(v) = D(u)$, $\max\{F(P(w)) | w \in N(u), D(w) > D(u)\} \leq F(P(v)) \leq \min\{F(P(w)) | w \in N(u), D(w) < D(u)\}$. 

There have been many works related to mules. Data collection using mules is addressed in [7][12][13]. Using mules to connect sparse sensor networks at the cost of higher latencies is explored in [7]. Reference [12] analyzes the upper bound of the optimal data transfer with mules. In [13], it shows that using mules with predictable mobility can significantly reduce communication power in WSNs. The routing problem in a highly disconnected ad hoc network using mobile ferries is discussed in [14][15][16]. A comprehensive survey of mobile sensor networks can be found in [17]. However, how to buffer packets generated by an isolated WSN remains an obscure problem. To the best of our knowledge, this is the first work addressing distributed prioritized storing strategies for isolated WSNs using mobile mules.

The rest of the paper is organized as follows. Section II presents our system model. DSMS is given in Section III. Some extensions of DSMS are in Section IV. Section V contains our simulation results. Our implementation results are shown in Section VI. Section VII concludes this paper.

### II. System Model

We consider a heterogeneous WSN consisting of some static sensors and mobile mules. Static sensors, or simply nodes, can continuously monitor the environment and periodically generate reporting packets, or simply packets. Each node has the same storage space of $S_{sn}$ (in unit of packet) and communication range of $R_{sn}$. Two nodes $u$ and $v$ can communicate with each other if their distance $\text{dist}(u, v) \leq R_{sn}$. These static sensors form a connected WSN through multi-hop routing. The WSN is deployed in a remote field and is isolated from the outside world. Mobile mules are thus designed to collect sensory data from them. They can stop by a specific node, called sink, for a period of time to collect sensory data. During this period, the sink can relay its own and others’ packets to the mule. The stop-by period can be a fixed or a random length. The movement of mules can be by intention (pre-arranged, such as a bus) or by opportunity (not pre-arranged, such as a taxi). These mules can later deliver the collected packets to an external base station. Fig. 2 shows an example.

![Fig. 2. System model of the DSMS system.](image-url)
P1 (resp., P2) implies that nodes that are farther from (resp., closer to) the sink than $u$ should have lower-priority (resp., higher-priority) packets than $u$. P3 enforces that nodes that have the same distance to the sink as $u$ should have the same property as $u$. When a node has the above properties, we say that it is in-order. In Fig. 3 (a), every node is in-order except node $m$ and $j$.

For each node $u$, we let $\text{maxPost}(u)$ be the packet with the highest priority of all neighbors $v$ of $u$ such that $D(v) > D(u)$, $\text{minPre}(u)$ be the packet with the lowest priority of all neighbors $v$ of $u$ such that $D(v) < D(u)$, $\text{maxEqual}(u)$ be the packet with the highest priority of all neighbors $v$ of $u$ such that $D(v) = D(u)$, and $\text{minEqual}(u)$ be the packet with the lowest priority of all neighbors $v$ of $u$ such that $D(v) = D(u)$. Based on the above properties, we design our packet exchange rules for node $u \in BA$ as follows:

- **E1**: When $F(\text{maxPost}(u)) > F(P(u))$, node $u$ tries to exchange packet with $\text{maxPost}(u)$.
- **E2**: When $F(P(u)) > F(\text{minPre}(u))$, node $u$ tries to exchange packet with $\text{minPre}(u)$.
- **E3.1**: When $F(\text{maxEqual}(u)) > F(\text{minPre}(u))$, these two packets are exchanged.
- **E3.2**: When $F(\text{maxPost}(u)) > F(\text{minEqual}(u))$, these two packets are exchanged.

The above rules are event-triggered ones. These events are triggered when a node changes its packet, including exchanging with others or generating a new one, or when its neighbors change their packets. When multiple events are triggered, a node should prioritize rules **E1**, **E2**, **E3.1**, and **E3.2** in that order because we prefer nodes exchanging with those at different distance first. For $u$ to exchange packet with $v$, it can send a Request To Exchange (RTE) to node $v$. Node $v$, if agrees, replies a Clear To Exchange (CTE). Then the exchange can be conducted. These operations should be atomic. If an exchange happens, a node should broadcast the priority of its new packet to its neighbors.

A node $u \notin BA$, when it has a packet, it will try to send it to any neighbor $v$ such that $D(v) < D(u)$. When a node $w \in BA$ receives the packet, it will accept it if $F(P(w)) = -1$, drop it if $F(P(w)) \geq F(P(u))$, and replace $P(w)$ by $P(u)$ if $F(P(u)) > F(P(w))$.

Fig. 3 gives an example. Node $a$ is the sink and there is a new packet with priority 12 arriving at node $m$ in Fig. 3(a). Node $m$ will realize that it violates P2 and will exchange with node $j$ by E2 as shown in Fig. 3(b). The same will happen to nodes $j$, $f$, and $b$, resulting in the scenario in Fig. 3(c). Now $j$ finds that it violates P3 because $F(P(i))$ is not between 10 and 4. So $j$ will notify $i$ and $m$ to exchange their packets by E3.2. Similarly, $g$ will find that it violates P3 after receiving $f$’s broadcast and notify $c$ and $f$ to exchange their packets by E3.1. The final result is in Fig. 3(d), where every node is in-order. Note that DSMS does not guarantee an optimal arrangement of packets since it is a distributed protocol and relies only on neighboring information.

Below, we formally prove that DSMS will ultimately stop in an in-order status. We say a packet is stable if this packet is stored in a certain node and will not exchange to other nodes, until it is collected by a mule or new packets with higher priority are generated. Next, we first show each packet will become stable in finite time, which means DSMS will eventually stop. Then we show each node is in-order while DSMS stops.

**Theorem 1.** Given any arrangement of packets in BA, if no packets are generated during exchange, the packet exchange following **E1**, **E2**, **E3.1** and **E3.2** will eventually stop in finite time.

**Proof:** It is obvious that the packet with the highest priority eventually migrates to the sink and becomes stable once it reaches the sink. Once the highest prioritized packet becomes stable, the packet with the second-high priority can become stable once it reaches one of sinks neighbors. Similarly, each packet can become stable if all the packets with higher priorities than it become stable and it reaches the place as close to the sink as possible. Note that, it is not necessary that packets become stable in the order of their priorities. But once higher prioritized packets become stable, a packet can become stable without doubt. Since the BA is limited, the packet exchange can terminate in finite steps.

Next, we try to prove that all nodes are in-order when the packet exchange stops.

**Theorem 2.** After all nodes in BA stop exchanging packets, they are in-order.

**Proof:** We prove this theorem by contradiction. If node $u$ is not in-order, then there are only three possible cases:

1. **Case 1**: Node $u$ violates P1. That is there is a neighbor $v \in N(u)$ such that $D(v) > D(u)$ and $F(P(v)) > F(P(u))$. Since $F(\text{maxPost}(u)) \geq F(P(v)) > F(P(u))$. It will not stop exchanging according to **E1**.
Case 2: Node $u$ violates $\mathbf{P2}$ but it follows $\mathbf{P1}$. That is there is a neighbor $v \in N(u)$ such that $D(v) < D(u)$ and $F(P(v)) < F(P(u))$. Since $F(minPre(u)) \leq F(P(v)) < F(P(u))$, it will not stop exchanging according to $\mathbf{E2}$.

Case 3: Node $u$ violates $\mathbf{P3}$ but it follows $\mathbf{P1}$ and $\mathbf{P2}$. That is there is a node $v \in N(u)$ such that $D(v) = D(u)$ and $F(P(v))$ is not between $F(maxPost(u))$ and $F(minPre(u))$. Since node $u$ follows $\mathbf{P1}$ and $\mathbf{P2}$, we can get $F(minPre(u)) \geq F(P(v)) > F(minPre(u))$. The value of $F(P(v))$ is either bigger than $F(minPre(u))$ or smaller than $F(maxPost(u))$.

1. $F(P(v)) > F(minPre(u))$. Since $F(minPre(u)) \geq F(P(v)) > F(minPre(u))$, it will not stop exchanging according to $\mathbf{E3.1}$.

or

2. $F(P(v)) < F(maxPost(u))$. Since $F(minPre(u)) \leq F(P(v)) < F(maxPost(u))$, it will not stop exchanging according to $\mathbf{E3.2}$.

Case 1, 2 and 3 all contradict our assumption of stopping exchanging, so it is proved.

Because DSMS can utilize many mesh-like communication links to exchange packets, higher-priority packets have chance to stay closer to the sink by rules $\mathbf{E3.1}$ and $\mathbf{E3.2}$. One question is: how many packet exchanges may be incurred when a new packet is generated given a stabilized network. We will investigate this issue via simulations.

Finally, we comment that a mule arrives at the sink, the sink can broadcast an UPLOAD message. Then every node in BA simply tries to transmit its packet toward the sink in a greedy way.

IV. SOME EXTENSIONS

We discuss two extensions below. We first extend DSMS to $S_{sn} > 1$. We define $maxMine(u)$ (resp., $minMine(u)$) to the packet of $u$ with the highest (resp., lowest) priority. Since node may have multiple packets, the exchange rules for node $u$ are modified as follows:

$\mathbf{E1'}$: When $F(maxPost(u)) > F(minMine(u))$, node $u$ tries to exchange its packet $minMine(u)$ with packet $maxPost(u)$.

$\mathbf{E2'}$: When $F(maxMine(u)) > F(minPre(u))$, node $u$ tries to exchange its packet $maxMine(u)$ with packet $minPre(u)$.

$\mathbf{E3.1'}$, $\mathbf{E3.2'}$ are the same as previous as $\mathbf{E3.1}$ and $\mathbf{E3.2}$.

The definition of “in-order” can be directly extended to $S_{sn} > 1$. Note that nodes only need to broadcast the highest and the lowest priorities of its packets. It is not hard to prove that previous properties still hold when $S_{sn} > 1$.

The second extension is to add a few transmission buffers to each node to handle packet overflow. A packet waiting to be transmitted should be put in a transmission buffer. When a node $u \in BA$ whose storage space is full generates a new packet, it will keep packets with higher priorities and move the lowest-priority one to its transmission buffer. We assume that BA is more crowded, so such packets will be forwarded to node $v$, where $v \in N(u), D(v) > D(u)$ and $F(minMine(v))$ is minimum. However, this decision will not affect the correctness of our protocol.

Fig. 4. A snapshot of priority distribution.

Fig. 5. DSMS using communication graph and tree structure.

V. SIMULATION RESULTS

We have conducted some simulations to verify our results. Unless otherwise indicated, the simulation environment contains 400 sensor nodes randomly deployed in $200 \times 200$ field, each with a transmission range of 25. The packet arrival rate is $1/50$ per node and each packet has a random priority between 0 and 1000. The BA is within 10 hops from the sink. All results are from the average of 50 test runs. Fig. 4 shows a snapshot of priority distribution in a network with the sink at $(0, 0)$ after applying DSMS.

We first compares DSMS using a mesh-like communication graph with a tree structure. To construct a tree structure, we reduce the communication graph into a short-path spanning tree rooted at the sink. Each node only allow to exchange packets with its parent or children. The results are shown in Fig. 5. Using communication graph can collect much higher priority packets than using tree structure.
Fig. 6. Comparison of average priorities of packets collected by mobile mules under various stop-by intervals of mules and collected sizes (in terms of numbers of nodes) of mules. We compare DSMS against Greedy Forward (GF),

Fig. 6 compares the average priorities of the packets collected by mobile mules under various stop-by intervals of mules and collected sizes (in terms of numbers of nodes) of mules. We compare DSMS against Greedy Forward (GF),

Fig. 6 compares the average priorities of the packets collected by mobile mules under various stop-by intervals of mules and collected sizes (in terms of numbers of nodes) of mules. We compare DSMS against Greedy Forward (GF),

Fig. 7. Effect of BA size on (a) average priority and (b) traffic overhead and packet loss.

Fig. 7. Effect of BA size on (a) average priority and (b) traffic overhead and packet loss.

Fig. 8. Traffic overheads (a) when packets arrive at arrival rates and (b) when one packet arrives at a random node in a stabilized network.

Fig. 8. Traffic overheads (a) when packets arrive at arrival rates and (b) when one packet arrives at a random node in a stabilized network.
where a node always tries to send its packets to any node closer to the sink until the latter has no storage space. OPT represents the ideal solution if global optimization is possible. Fig. 6(a) shows that as the stop-by interval increases, the average priority also increases. Fig. 6(b) shows that the average priority decreases slightly as the collecting sizes increases. However, the impact is insignificant.

Fig. 7 shows the effect of the BA size. In Fig. 7(a), we vary the BA size but fix the stop-by interval such that $\frac{1}{3}$ of the data in the BA can be collected. In terms of the average priority of collected packets, DSMS outperforms GF and is close to OPT when the hop count is larger than 5. Fig. 7(b) shows that as the BA size gets larger, the data loss will decrease since the storage space is larger. On the other hand, traffic overhead will decrease first and then increase as the BA size gets larger. Each packet transmission counts for one. When the BA size is very small (say, 4 hops), packets have to travel long to reach the BA. For example, a packet with a small priority travel long before being dropped. That causes the traffic overhead keep on decreasing before the hop count reaches 7. However, as the hop count is larger than 7, there are more exchanges in BA, causing the overhead to increase.

Fig. 8 shows our DSMS overheads. Fig. 8(a) compares the overhead by varying the number of nodes and the packet arrival rate. So DSMS gets packets with higher priorities at the cost of more packet exchanges. The overhead of DSMS is about a constant higher than to that of GF. Fig. 8(b) shows the number of packet transmissions incurred when a new packet with a random priority is inserted into a stabilized network. The transmission increases while the number of nodes increases, but the effect is insignificant.

VI. IMPLEMENTATION

We have implemented DSMS in a simplified hardware platform. A toy train is designed to repeatedly circle around a toy rail. The train serves as a mule, and we deploy a wireless node on it. A number of isolated grid WSNs are deployed around the rail. Whenever the train has a connection with a sink, it will pull as much data from the sink (and thus its BA) as possible. Fig. 9(a) shows our implementation structure. Our sensor hardware platform is a low-power and low-cost wireless microcontroller, JN5139 [18], with ZigBee-compliant wireless interface. The WSN in Fig. 9(b) is a $4 \times 4$ grid plus a sink. We implement our DSMS on these Jennic microcontroller boards. We use a light sensor to generate sensory packets with priority ranging from 0 to 9, where a higher light intensity means a higher priority. To view the priority of a piece of sensing data, we display the value on an on-board 7-segment display. To facilitate data exchange, we design a COLLECT_DATA message that can be initiated by the mule to the sink. After collecting a piece of data, the mule will transmit an ACK message to the sink.

Our implementation result shows that the DSMS can be easily implemented in a real sensor platform with a very small image in each microcontroller. Fig. 10 shows a snapshot where after applying our exchange rules, data in all nodes are in-order. The implementation verifies that our DSMS protocol is quite simple and only needs local information. So it is suitable for distributed WSN environment.

VII. CONCLUSIONS

We have proposed a distributed storage management strategy (DSMS) for data buffering in an isolated WSN. DSMS can reduce data loss while keep higher-priority packets closer to the sink area. Properties of DSMS are proved and its efficiency is verified by simulations as well as real implementation.

ACKNOWLEDGMENT

Y.-C. Tseng’s research is co-sponsored by MoE ATU Plan, by NSC grants 97-3114-E-009-001, 97-2221-E-009-142-MY3,
Fig. 10. A snapshot of DSMS’s behavior in an isolated $4 \times 4$ grid WSN.

98-2219-E-009-019, and 98-2219-E-009-005, by MOEA
C.-F. Huang’s research is co-sponsored by NSC grants 98-2218-E-194-010-MY2.

REFERENCES
