Low-Power Distinct Sum for Wireless Sensor Networks

Ebram Kamal William
National University of Singapore

Mun Choon Chan
National University of Singapore

***Abstract*—Continuous monitoring is a major component of many applications in wireless sensor network (WSN). In these applications, to reduce the communication overhead, some form of data summary or aggregation can be performed. However, performing non-trivial in-network data processing such as finding frequent items, Top-K monitoring, and clustering efficiently are challenging in practice.**

In this paper, we present Low-Power Distinct Sum (LDS), a distributed in-network data aggregation primitive that performs the sum of unique items in WSN. LDS serves as the underlying primitive that can be used to implement many distributed data processing efficiently. To demonstrate LDS's capabilities, we design and implement a distributed data streaming application with LDS running on Contiki OS. Compared to the baseline algorithm, LDS can reduce the completion time by up to 66%.

***Index Terms*—wireless sensor network, synchronous transmission, distributed edge processing, data aggregation**

I. Introduction

Streaming applications (e.g. continuous monitoring) are a major component of many applications in wireless sensor network (WSN) such as the detection of malfunctions or outliers [1]–[3]. The naive approach of collecting all the data consumes a large amount of energy and bandwidth [4]. As is typical in many monitoring applications, the sensed data change infrequently and continuous data collection often results in lots of unnecessarily overhead if all the sensed data are sent to the gateway. However, determining whether there is a need for updates is data and application dependent. This motivates a research direction to reduce the amount of communication.

One approach to reduce network communication and processing at the gateway node is to perform in-network computation. Several in-network computation primitives have been proposed for WSN to support different applications. Some of the earliest applications focus on data aggregation on a tree towards the root to perform Min/Max/count etc., e.g. TAG [5], Cougar [6], TinyDB [7]. However, aggregation along a tree is prone to reliability issues. If a loss occurs at a particular link, information from nodes within the sub-tree will be lost.

To address the challenge of route maintenance and improve reliability through multi-path forwarding, there is much recent research that proposes to disseminate data using synchronous transmissions. There are approaches for in-network aggregation that are based on synchronous transmissions. Examples include Chaos [8] and A^2 [9] for agreement, and WPaxos [10] for consensus. However, a key limitation of these approaches is that they rely on duplicate insensitive operations such as MIN and/or MAX. Aggregations like MAX and MIN are *duplicate insensitive* since duplicate readings will not change the result [4]. However, there is a large class of applications that require *duplicate sensitive* aggregations, such as COUNT, SUM, AVERAGE, and MEDIAN where the result is affected by duplicate readings of the same value.

In this paper, we present Low-Power Distinct Sum (LDS). LDS is a primitive that computes the sum of unique items in a distributed manner. With LDS, many applications that require duplicate sensitive operations such as threshold function monitoring [11], finding recent frequent items [12], largest K monitoring [13], number of distinct values estimation [14], etc., can be implemented efficiently on a WSN.

LDS is designed to run over a synchronous transmission layer. To support duplicate-sensitive aggregation efficiently, LDS incorporates features that include (1) segregated data structures for efficient storing and processing of different state information, and (2) heuristics for data decoding/encoding and transmission priority to speed up the completion time of the in-network data aggregation.

We implement the Geometric Monitoring (GM) [11] application using LDS to illustrate its usage and benefits. GM is a general approach that allows monitoring of global threshold functions (e.g. monitoring if the average temperature of the network is above a certain threshold). In GM, nodes locally monitor received data and suppress data propagation to reduce data exchange if the node determines, using local constraints, that the changes detected will not affect the global monitoring function. LDS provides the basic primitive for updating the arithmetic constraints in GM efficiently using in-network duplicate sensitive aggregation.

The contributions of this paper are:

- The design of LDS allows nodes to compute the sum of unique items in a distributed manner running on top of a synchronous transmissions protocol efficiently. LDS incorporates processing of two different data structures and collision management to speed up in-network aggregation convergence.
- We design a distributed data streaming application, GM, that uses LDS as the underlying primitive. The design of GM also incorporates additional application logic to enhance its performance.
- We have implemented LDS and the distributed data streaming application using it on Contiki OS. LDS has

been ported to run on TelosB and CC2650 SensorTag.

In the evaluation performed on the Indriya2 testbed [15], we show that LDS can reduce the completion times by up to 66% using Arctium [16] as the baseline.

II. Related Work

Many approaches have been presented to reduce the traffic by incorporating processing on the data streams [11]–[14]. These approaches look at monitoring based on a threshold function, looking for the largest set or the most frequent items, etc., While these approaches present many interesting and useful algorithms, they do not touch on the networking aspect of practical WSN deployments.

Recently, researchers have presented some practical designs and implementations of network aggregation approaches in WSN based on synchronous transmission (Glossy [17]). Chaos [8] and A^2 [9] are many-to-many data aggregation protocols that can be used to address agreement problems. The implementation of Paxos, a consensus protocol, on a wireless network is presented in WPaxos [10]. Further, a general monitoring threshold function approach which is based on geometric monitoring [11] is presented in [16]. However, these protocols are limited in terms of the type of in-network data aggregation utilized and supported.

Inspired by the previous work, we design the sum of unique items on top of our implementation of a many-to-many communication protocol that is based on capture effect synchronous transmissions and incorporates the decoupling of the communication and computation that is also implemented in Mixer [18]. As the sum of unique items can serve as an underlying primitive for many data streaming distributed computation approaches, we design and implement Geometric Monitoring [11] using LDS to demonstrate its benefits.

III. LDS

We describe the challenges of computing the distinct sum (Duplicate Sensitive Aggregation) using a baseline approach in § III-A followed by our approach (LDS) in § III-B.

A. Baseline Approach

In this section, we first present a baseline approach whereby partial sums can be merged only if the partial sums have no overlapping components.

When a node receives an aggregated value, there are two possible scenarios based on the properties of the local and received aggregated values, namely merge and keep-maximum.

In the merge case, the received aggregated value has no common component with the local aggregated value. In this case, the two aggregated values can be merged and we sum up both aggregated values (received and local).

If the received and local aggregated values are not disjoint, the values cannot be merged. We simply select the aggregated value with the most number of components and discard the other aggregated value. This is the keep-maximum scenario.

Each node will always merge its local value to the result aggregated value (merged or keep-maximum) if the aggregated value does not include its local value.

There are four core components to the baseline approach:

- **State Stored:** Nodes only keep one aggregated value and its corresponding flag bit vector.
- **What to Transmit:** There is no need to decide on what to transmit as nodes have only one aggregated value (merged or keep-maximum).
- **When to Transmit:** Nodes decide to transmit in the next slot if they successfully receive or compute a new aggregate value. In the case of no new information, nodes transmit with a low probability.
- **Termination:** Nodes terminate when the distinct sum is computed for itself and all its neighbors for predefined subsequent slots. In the implementation, this is done by checking the status of a bit vector.

The baseline approach requires minimum processing and storage. However, with limited aggregation opportunity and stored state, many communication rounds are wasted as the local and received aggregate values cannot be merged, and either the newly received or currently stored aggregate value has to be discarded.

B. LDS Approach

Several challenges need to be addressed to make the aggregation process more efficient. First, storing and processing up to 2^N entries for a network of size N in order to not ”throwaway” any information received is clearly not scalable. Second, the amount of processing capability available on the nodes is limited especially when we run over a communication layer based on synchronous transmissions. Hence, there is a need to limit the number of states and the amount of processing that needs to be done.

We tackle these challenges to implement the distinct sum by keeping a small number of states. In addition, we implement a state maintenance logic to merge and discard these states. The details of the state stored are as follows:

State Stored: We observe that the smallest unit of the aggregated distinct sum is the single values (values with a single node’s contribution). Single values are often more useful than combination values. Thus, we design the data structure to keep track of two kinds of data. The first part stores only single values. The size of this part of the state is up to N. However, note that it is unnecessary, in fact highly unlikely, to need to store all N values before completing the network-wide aggregation. The other part stores the different combinations of aggregated values. In principle, the size of these combined values can vary from 1 to 2^N. However, we will show in the evaluation that a (very) small number suffice. The logic for state maintenance is presented in Section III-B1.

Other components, when to transmit and termination, are the same in LDS and the baseline approach. For what to transmit, nodes send their value without aggregation at the start of the communication round. This helps to disseminate useful information in the network and to have a good distribution of the data which can help in building more different aggregated values. After that, nodes will send the aggregated value with the maximum number of nodes contributed to it.

1) State Maintenance Logic: During the update phase and for each newly received packet, the processing is separated into two parts. The first part decodes and stores more single values. The second part merges new and stored aggregated values.

- **Decode single values:** Each node checks if the received packet has a single contributor or if a node can decode a single value out of the stored single values and the received aggregated value. In the case whereby a node receives or calculates a new single value, this value is stored in the single values data structure.
- **Merge different combinations:** The received aggregated value is merged with all the values in the single values data structure, without replication. The result is stored in the combined values data structure. After that, nodes go through the items in the combined values data structure and try to merge them (merge of mutual disjoint items). If a node can merge two entries, it stores the result in the combined values data structure. If the new item does not exist in the combined values data structure, the new aggregated value is stored if storage space is available or it replaces an aggregated value with fewer components. In the case of a tie, the oldest value is replaced.

As LDS needs to store some amount of states, we investigate how the size of the states stored affects the performance. To study the performance running on a large network, we run an experiment of LDS on the Cooja simulator using 100 sky motes randomly placed in an area of 250,000 m^2. The median completion times of storing 1 and 25 combined values are similar at around 30 slots. Hence, the storage size of the combined values data structure can be set to a very small value. Finally, we observe that the average number of single values solved before completion is 14.72. This is much less than the 100 single values available in the network. The result indicates that in-network aggregation is a much more dominant factor than accumulating sufficient single values.

IV. EVALUATION

The performance of LDS is evaluated on the Indriya2 testbed [15]. Indriya2 has two types of motes namely, TelosB and CC2650 SensorTag deployed indoors over three floors. During the evaluation, Indriya2 has 40 TelosB and 17 CC2650 SensorTag. There are two parts to the evaluation. We first evaluate the performance of different features of LDS. Then, we compare the performance of running GM using LDS to Arctium [16].

A. Impact of LDS's Components

In this section, we evaluate the impact of different components of LDS on performance. We run five different versions, each removing a single feature from the original version (LDS), starting with LDS:

- **LDS** The original version with all the features running.
- **No single value extraction (ext_ones)** Nodes maintain the single bit structure but they stop trying to extract single values from the received combinations.

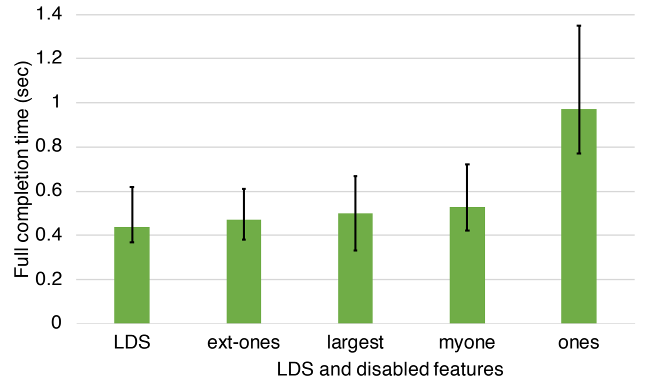

Fig. 1. LDS's different design elements completion time on Indriya2 testbed.

- **Send random entries instead of the largest (largest)** Nodes send random entries from their buffer instead of always sending the sum with the largest number of values.
- **Remove sending node's value (myone)** Nodes do not always send their values in the first few slots.
- **Remove states separation (ones)** Nodes do not explicitly maintain separate data structures for single values.

We run using 16 TelosB acting as sources in a network of size 40 motes on the Indriya2 testbed. We measure the full completion time for each setting. The reported result for each set is computed based on an average of at least 110 rounds.

Figure 1 shows the median latency of each set with the error bars showing the minimum and maximum completions time in seconds. As expected, the results in the figure show that removing any of the features increases the latency. Based on the increase in latency, the feature that has the largest impact is the separate storage for the single and combination values. This can be explained by the understanding that single values, being the smallest component of the distinct sum, are significantly useful for completing the sum.

B. Geometric Monitoring Application

In this section, we evaluate the performance of the GM algorithm introduced in [11] running using LDS. GM uses local constraints to suppress communication. Our objective is to show that the performance of such an application can be improved due to the ability to perform in-network aggregation.

We use Arctium [16] as the baseline protocol in this evaluation. The high-level design of Arctium is to run GM on top of Crystal [19]. In Crystal, a fixed initiator (sink) starts the communication with a synchronous flood. After that, all the source nodes start Glossy rounds competing to deliver their values to the sink. The sink receives one of them with a high probability which in turn initiates a Glossy round to acknowledge the value received and the corresponding sender will stop sending its value in the subsequent rounds.

Arctium reduces Crystal communication by coalescing values from multiple sources into a single data packet when these values meet at a common node. Arctium does not perform additional computation for in-network aggregation. On the other hand, in LDS, nodes aggregate different values in the

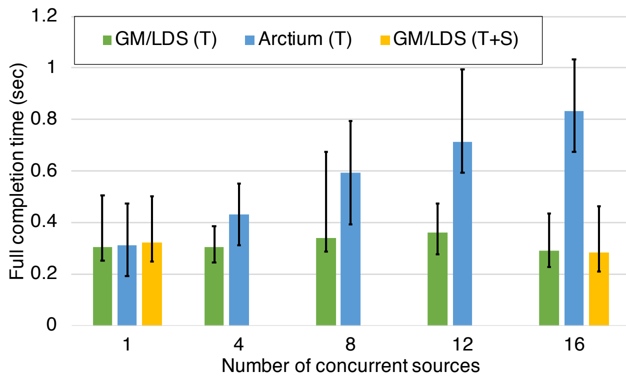

Fig. 2. GM/LDS and Arctium completion time against the number of concurrent sources on Indriya2 testbed. T - TelosB mote; S - CC2650 SensorTag.

network. In addition, in Arctium, nodes deliver their data to a fixed gateway which will send back the calculated sum to all the nodes. In LDS, each node computes and shares the sum based on received data in a many-to-many fashion.

In the experiment, we set Arctium's coalescing buffer size to two (two messages can be combined into one packet) as recommended [16] and all other settings are set to the suggested default values. The default slot length in Arctium is 5ms for the transmission slot and 7ms for the acknowledgment slot. In LDS, we set the combined values storage size to 5 and we set the slot length to 5ms. The slot length in LDS is defined by a single packet transmission (1.5ms), a time gap (2ms), and a single packet reception (1.5ms). We run the experiments for at least 110 rounds. We vary the number of sources to be 1, 4, 8, 12, and 16.

Figure 2 shows the median completion time of executing GM using LDS and Arctium running on TelosB (T) nodes on the Indriya2 testbed. For GM/LDS, the network has a total of 40 TelosB nodes and up to 16 monitoring nodes. Non-monitoring nodes simply act as relays. For Arctium, one node serves as the sink and there are up to 16 monitoring nodes. Again, non-monitoring nodes act as relays. The error bars in the figure represent the minimum and maximum completion times.

As shown in the figure, for the case with 1 source, the completion times for GM/LDS and Arctium are similar. This is expected as there is no aggregation with a single source. As the number of sources increases, the gain of in-network aggregation becomes more obvious. For 16 sources, the median completion time of GM/LDS is up to 64% smaller than Arctium. It is expected for Arctium's completion time to increase with increasing the number of sources as more and more nodes compete to deliver their data to the sink. On the other hand, a higher number of sources provide more opportunities for data aggregation in LDS.

As we have ported LDS to run on CC2650 SensorTag, we experimented to show that GM/LDS can run over a network of heterogeneous nodes. We run GM/LDS on the Indriya2 testbed using 16 monitoring nodes with half of the nodes being TelosB (T) and the other half being CC2650 SensorTag (S) nodes. In this experiment, the network size increased from 40 TelosB nodes to also include 17 CC2650 SensorTag for a total of 57 nodes. We performed evaluations with either 1 monitoring node or 16 monitoring nodes. As shown in Figure 2, the performance of running LDS on a network of heterogeneous devices is similar to running it on a single device type.

V. Conclusions

We have designed LDS, a technique based on synchronous transmissions to compute distinct sum efficiently. LDS can serve as a primitive for many distributed approaches to perform continuous monitoring. We demonstrate the benefits of LDS using a general threshold monitoring framework (GM), showing that continuous monitoring can be achieved efficiently through in-network aggregation. We believe that LDS opens up the potential to achieve practical implementation of diverse classes of applications on WSN.

Acknowledgment

This research was supported by the Singapore Ministry of Education Academic Research Fund Tier 1 (T1 251RES1910).

References

[1] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing letters*, 2015.
[2] E. K. William and M. C. Chan, "Indp: In-network data processing for wireless sensor networks," in *SECON*, 2019.
[3] S. Burdakis and A. Deligiannakis, "Detecting outliers in sensor networks using the geometric approach," in *ICDE*, 2012.
[4] P. Jesus, C. Baquero, and P. S. Almeida, "A survey of distributed data aggregation algorithms," *IEEE Communications Surveys & Tutorials*, 2014.
[5] S. Madden et al., "Tag: A tiny aggregation service for ad-hoc sensor networks," *ACM SIGOPS Operating Systems Review*, 2002.
[6] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *ACM Sigmod record*, 2002.
[7] S. R. Madden et al., "Tinydb: an acquisitional query processing system for sensor networks," *TODS*, 2005.
[8] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale," in *SenSys*, 2013.
[9] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "Network-wide consensus utilizing the capture effect in low-power wireless networks," in *SynSys*, 2017.
[10] V. Poirot, B. Al Nahas, and O. Landsiedel, "Paxos made wireless: Consensus in the air." in *EWSN*, 2019.
[11] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," *TODS*, 2007.
[12] A. Manjhi et al., "Finding (recently) frequent items in distributed data streams," in *ICDE*, 2005.
[13] B. Babcock and C. Olston, "Distributed top-k monitoring," in *SIGMOD*, 2003.
[14] P. B. Gibbons and S. Tirthapura, "Distributed streams algorithms for sliding windows," in *ACM SPAA*, 2002.
[15] P. Appavoo et al., "Indriya2: A heterogeneous wireless sensor network (wsn) testbed," in *TridentCom*, 2018.
[16] C. Stylianopoulos et al., "Continuous monitoring meets synchronous transmissions and in-network aggregation," in *DCOSS*, 2019.
[17] F. Ferrari et al., "Efficient network flooding and time synchronization with glossy," in *IPSN*, 2011.
[18] C. Herrmann, F. Mager, and M. Zimmerling, "Mixer: Efficient many-to-all broadcast in dynamic wireless mesh networks," in *SenSys*, 2018.
[19] T. Istomin et al., "Data prediction+ synchronous transmissions= ultra-low power wireless sensor networks," in *SenSys*, 2016.