InDP: In-Network Data Processing for Wireless Sensor Networks

Ebram Kamal William
National University of Singapore
Singapore
ebramkw@comp.nus.edu.sg

Mun Choon Chan
National University of Singapore
Singapore
chanmc@comp.nus.edu.sg

***Abstract*—Wireless sensor networks have emerged as an important information collection and monitoring tool. The data collected is typically uploaded to a central gateway for processing and analysis. However, the approach of forwarding all the sensed data to a sink for processing is not always practical due to the high communication cost.**

In this paper, we present InDP, a framework that is designed to support data dissemination and processing in the edge. InDP has a communication component and a computation component. The communication component supports a low duty cycle mode for an infrequent status update and a high throughput mode to support distributed computation. The computation component implements a distributed version of Principal Component Analysis (PCA). As an application, we have implemented an outlier detection component over InDP.

InDP and the outlier detection application have been implemented on Contiki using a modified version of Codecast as the underlying many-to-many communication protocol. Our evaluations show that InDP can terminate as fast as 100ms and 1.3s on the average running on a testbed with more than 70 nodes. In terms of PCA computation, InDP's computation uses 91.6% less data than a centralized approach and is able to detect anomalies using a fraction of the sensed data.

***Index Terms*—wireless sensor networks, principal component analysis, machine learning, distributed algorithms**

I. Introduction

Wireless Sensor Networks (WSNs) have been used in a large number of monitoring applications involving sensing and data communication. These applications cover a variety of domains such as (i) monitoring of active volcano, structural health, and wildlife, (ii) acoustic/image based sensing, (iii) soil monitoring studies and energy auditing [1].

Existing WSN protocols typically focus on communication, in particular, data collection. With the emergence of Internet-of-Things (IoT), there will be many more applications that require the collection of a huge amount of data to a fusion center for processing. The cost of these approaches can be prohibitive due to the limited bandwidth of the WSN and the limited energy available on the sensors. With these challenges in mind, there is a strong incentive to move more processing to the edge in order to address issues related to latency, energy, and privacy [2]. In this work, we are motivated to address the following questions. What kind of network communication protocol will be required to support in-network edge processing? How can network-wide distributed computation based on edge processing be exploited and implemented efficiently with low overhead?

In this paper, we present InDP, a framework that is designed to support data dissemination and processing in the edge. InDP has a communication component and a computation component. The communication layer is designed to support efficient many-to-many communication over a multi-hop wireless sensor network. The computation component implements distributed PCA. PCA is a statistical technique that has been applied in many domains and is a popular technique for finding patterns in data of high dimension [3].

The contributions of this paper are:

- The design of InDP that supports efficient communication by switching between two operating modes, a low duty cycle mode to support infrequent status update and a high throughput mode to support distributed computation within the same framework depending on the application requirements.
- The separation of control and data dissemination in InDP so that early termination is made possible.
- The implementation of a distributed version of PCA based on disPCA [4] in InDP. To the best of our knowledge, this is the first work that implements distributed PCA over a multi-hop wireless sensor network based on many-to-many communication.
- Demonstrates the potential of edge processing in two ways. First, by using InDP , the global PCA can be computed by each node efficiently using only a small number of the largest eigenvectors and their corresponding eigenvalues. Second, by sharing only the singular matrices, each node can obtain the underlying structure of the data and perform outlier detection locally with minimum data exchange.

InDP can be used to support different applications on the edge that is enabled by PCA such as data reduction, outlier detection, clustering, etc. In this paper, only the outlier detection and compression applications are shown.

We have implemented InDP on the Contiki OS using a modified version of Codecast [5]. In the evaluation, with 16 nodes serving as sinks/nodes, when there is only one source, InDP can terminate as fast as 100ms running on a testbed with more than 70 nodes. With 16 sources updating their data, InDP terminates in 1.3 sec on the average. Compare

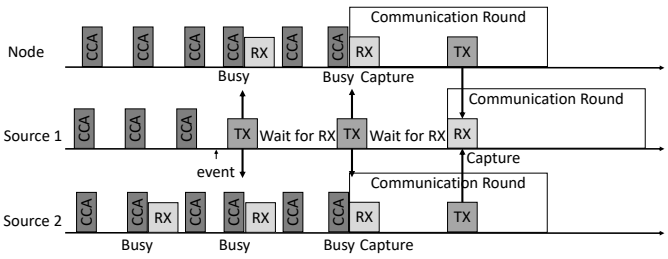

Fig. 1. Overview of InDP in action, *duty-cycling mode*

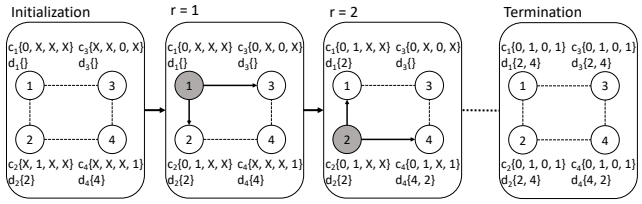

Fig. 2. InDP overview. (1 - "Needed", 0 - "Not Needed", X - "Unknown")

to the baseline Collection Tree Protocol (CTP), InDP reduces completion time and energy consumption by more than 85% and 76% respectively. In terms of PCA computation, InDP's computation uses 91.6% less data than a centralized approach. Finally, InDP is able to detect anomalies using $\frac{1}{4}$ to $\frac{1}{3}$ of the sensed data.

The rest of the paper is organized as follow. In section II, we present the communication component of InDP and in section III the computation component. Details of the implementation of InDP is presented in section IV. Section V presents the evaluation results and related work in section VI. Finally, we conclude in Section VII.

II. InDP: Communication

We will first present a typical usage scenario of InDP. Sensor nodes collect sensor data over some period of time (say from a few minutes to a few hours depending on the application requirement). Based on the sensed data collected, a node may choose not to perform any update or may decide to share detected changes with the rest of the network. The number of nodes that want to perform a status update can thus vary from none to many (or all) nodes. As a result, InDP needs to be able to support periods with very low duty cycles with no update as well as periods with high data rate exchanges with many nodes sharing their data with other nodes at the same time.

A. Two Modes Operation

Duty Cycling Mode By default, every node runs in the duty-cycling mode. In this mode, a node performs periodically clear channel assessment (CCA) say every 62.5ms (16Hz). If no channel activity is detected, a node goes back to sleep. If a node finds a busy channel, it starts a reception slot.

When one of the sources generates a transmission event, it switches to run InDP as an initiator and start dissemination of the data. In order to wake up the source node's neighbors, the source node transmits the packet multiple times and wait for receiving after each transmission. Whenever the source node receives a packet, it switches back to the default transmit/receive dissemination phase. When other nodes detect these transmissions, they join the dissemination process and eventually all nodes participate in the dissemination. Figure 1 shows an overview of InDP in action.

This duty cycling mode works well when there is only a single data source. However, a problem arises when additional transmissions are triggered on other nodes after the first source node initiates the dissemination but the other source nodes have not detected the transmission yet. In such scenarios, the other source nodes trigger separate dissemination processes, resulting in data loss due to separate, concurrent disseminations. Such scenarios can be detected by all the nodes and the protocol switches to operate in the periodic mode to handle these higher data rate events.

Periodic Mode The Periodic mode is triggered to handle high traffic scenarios when events can be detected by two or more sources within a single dissemination cycle. The periodic mode is basically similar to the basic Codecast with a single node as the initiator. In such a mode, only one dissemination cycle can occur and many nodes can contribute data. At the end of each round, the nodes will check if there is more data to be disseminated. If there is no more data to be shared, nodes switch back to the duty cycling mode.

B. Control and Data Dissemination

InDP defines two kinds of information, namely control and data. Let there be k nodes. For $node_i$, $1 \leq i \leq k$, let $c_{ij}(r)$ and $d_{ij}(r)$ be the values of the control and data items of node j known by node i respectively at round r.

$c_{ij}(r)$ can take three values, namely 0, 1 and X. Depending on the values, $c_{ij}(r)$ can be interpreted as *"Needed"* ($c_{ij}(r) = 1$), *"Not Needed"* ($c_{ij}(r) = 0$), and *"Unknown"* ($c_{ij}(r) = X$).

Figure 2 shows a simple example of four nodes running InDP. In this example, nodes 1 and 3 have no data to share while nodes 2 and 4 have data to contribute to the computation.

The overall flow is as follow. At the start of the computation $r = 0$, $d_{ii}(0)$ has the most recent locally sensed/available data at node i in round 0. $d_{ij}(0)$ is set to Unknown $\forall i \neq j$. Node i set $c_{ii}(0)$ to 1 if it wants to contribute its data to the computation or 0 otherwise. $c_{ij}(0)$ is set to *"Unknown"* $\forall i \neq j$.

Data Sharing: For ease of exposition, let the execution be performed in rounds. In a given round r, node i shares with its neighbor $c_{ij}(r)$, $\forall j$. Let this be denoted as $c_i(r)$. For data item, the value shared is application dependent, and let it be denoted as $D_i(r)$. Note the size of the control item shared is small, while the size of $D_i(r)$ depends on the application and can be as large as the maximum packet size.

Update: If node i successfully receives $c_j(r)$ from node j, node i updates $c_{ij}(r)$ and $d_{ij}(r)$. These updated values

becomes the value for $c_{ij}(r+1)$ and $d_{ij}(r+1)$ in the next round. Update is performed as follow:

- if ($c_{jm}(r)$ is set to 0 OR 1) set $c_{im}(r) = c_{jm}(r)$.
- Update d_{ij} according to the application.

There is no change in the control and data item if there is no successful packet reception. Set $c_{ij}(r+1) = c_{ij}(r)$ and $d_{ij}(r+1) = d_{ij}(r)$.

Termination: Update terminates for node i when $\forall j$, either ($c_{ij} = 0$) or ($c_{ij} = 1$ and d_{ij} is not Unknown). Processing terminates when all nodes satisfy the termination condition.

In the following sections, we will provide a concrete example of how the framework can be utilized in practice by showing how distributed PCA can be implemented as an application using the framework presented in this section. For completeness, we will first provide a brief overview of PCA and distributed PCA, followed by details of the implementation.

III. InDP: Computation

A. PCA Overview

PCA is a technique that can be used to reduce the dimension of the original data with little loss of information and without losing the distribution of the data. In a WSN context, data is initially sensed and available only on the individual nodes. For the purpose of this exposition, the following terms are defined:

- There are S sensor nodes.
- Each sensor node collects q records (over time) of n features (different sensor modalities, e.g, temperature, voltage, air quality, etc.).
- Let $m = S * q$. Data from all sensors can be represented as a data matrix X with m records of n features ($X \in R^{m \times n}$).

In order to perform PCA, we can perform the computations either in the centralized or distributed manner.

In the centralized approach, a fusion center collects data from all sensors. Each sensor i sends the data matrix x_i with q observations of n features ($x_i \in R^{q \times n}$) to the gateway. The fusion center then builds the data matrix X with m records of n features ($X \in R^{m \times n}$). There are two common approaches to perform PCA (1) by deriving the data covariance matrix and (2) by performing Singular Value Decomposition (SVD).

B. Distributed PCA

The distributed PCA computation in InDP is based on the disPCA presented in [4]. Comparing to other introduced distributed PCA techniques [6]–[10], disPCA incurs the least communication overhead as it requires only one round of (many-to-many) communication. Table I summarizes notations used.

Figure 3 shows an overview of how InDP utilizes disPCA by introducing two modes, namely update mode and approximation mode. The update mode is to check if the local PCA needs to be updated and for nodes to check for changes in the global PCA (if any). The approximation mode is for nodes to get the approximated data (if required). At the start, each node i has a locally sensed data matrix x_i with q observations of n features ($x_i \in R^{q \times n}$). Each node computes local PCA over its data matrix, share it, and calculate the global PCA.

TABLE I
Table of Notations

Symbol	Meaning
S	Number of sensors
x_i	Data matrix of node i
R	Observation value
q	Number of observations
n	Number of features
U_i	Left singular vectors matrix of node i
D_i	Singular values matrix of node i
E_i	Right singular vectors matrix of node i
z_i	Projected data matrix of node i
t_i	Dimension of PCA used at node i
$\widetilde{x_i}$	Approximated data matrix of node i

After the initialization and in the next round with a new data matrix, in the update mode, each node computes the local PCA on its data matrix (x_i) and then check if the local PCA is changed comparing to the previous round and if the new local PCA may change the last computed global PCA. If a node finds a change, it starts a communication round and share the matrix D_i and E_i to update the global PCA. In order to reduce the amount of communication needed later, each node only transmits the t largest eigenvectors $E^{(t)}$ $t < n$ to all the sensor nodes. In the approximation mode, if a node i is required to share its data (e.g, it's predicted to be an outlier), it shares its projected data matrix z_i and the number of global PCA used to generate it t_i which can be used along with the global PCA to compute its approximated data matrix $\widetilde{x_i}$.

The data reconstruction error can be computed using Eq. 1,

$$error = ||x_i - \widetilde{x_i}|| \tag{1}$$

Reduction in Communication In the centralized approach, there are S nodes and each node has $q \times n$ amount of data. The total amount of data that needs to be communicated from all nodes to the fusion center is $S \times q \times n$. The fusion center will calculate the global PCA which of size $n + (n \times n)$ and transmits it to all the nodes. Each node needs to transmit $(q \times n) + (n + (n \times n))$ amount of data plus the amount of data it needs to relay as part of the routing processing from various nodes to the fusion center.

In the distributed approach, each node sends or receives local PCA of size $S \times (q \times t)$. With this information, each node calculates the (same) global PCA locally. Typically, the number of data records (q) on each node in a data exchange cycle is much larger than both the number of features (n) and the dimensions used in the reduced PCA matrix (t). Hence, $q >> n > t$. Communication cost in the distributed PCA is $O(Snt)$, compared to $O(Snq)$ for the centralized form. Since $q >> t$, the reduction is on the order of $\frac{t}{q}$.

As an example, to calculate the global PCA in the centralized approach for a network of size 16 nodes each with 10 observations of 4 features. The amount of sensed data

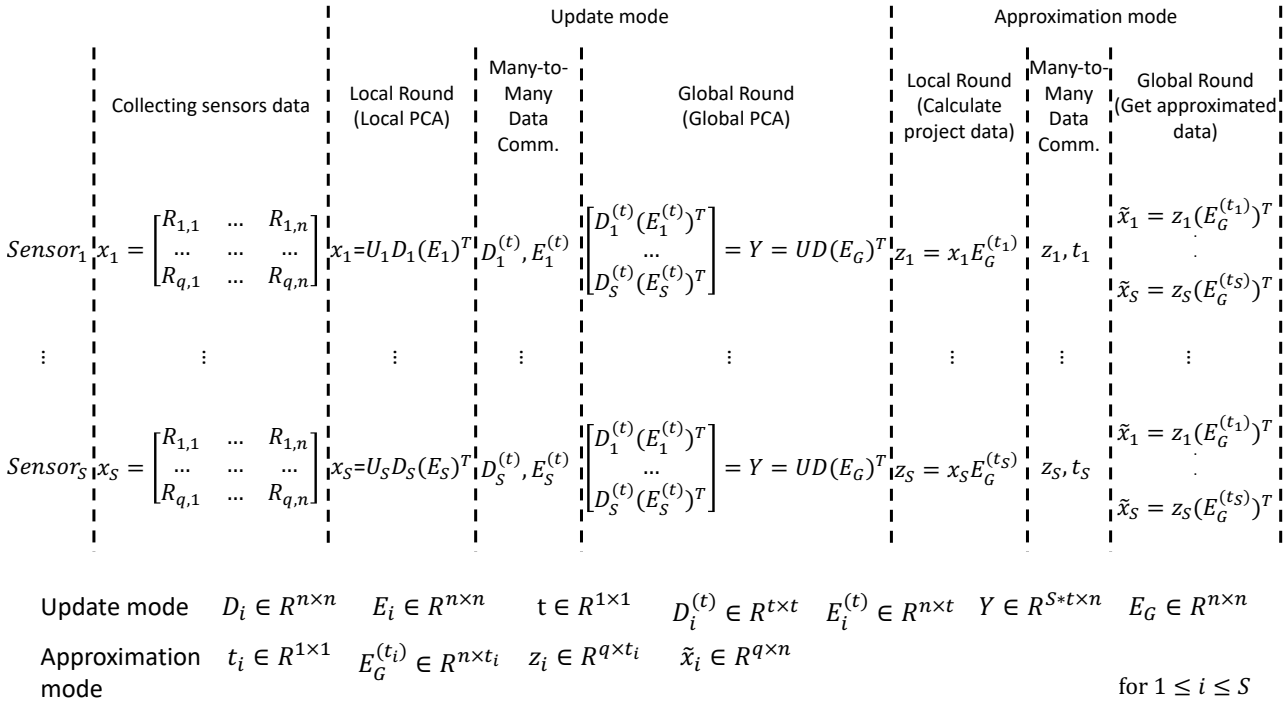

Fig. 3. Overview of data update and approximation using disPCA.

to be communicated is $16 \times 10 \times 4 = 640$. For the fusion center to send back the global PCA, the amount of data is $16 \times (4 + (4 \times 4)) = 320$. The total amount of data to be communicated is 960.

In disPCA, the amount of data to be communicated depends on how many local PCA is required to calculate the global PCA. In the case where only 1 local PCA needs to be shared by each node, the amount of data is $16 \times (1 + (1 \times 4)) = 80$. In the case where 4 local PCA needs to be shared by each node, the amount of data is $16 \times (4 + (4 \times 4)) = 320$. The amount of data reduction using disPCA is thus 66.6% to 91.6%.

To get the approximated data, each node calculates the projected data on the t largest global PCA. Using the projected data, nodes can compute the approximated data. The cost of the approximated mode is the same for the centralized and distributed PCA which is $S \times (t + q \times t)$. However in InDP, each node can locally determine if its local computation will impact the global PCA in the update mode and if it's required to share its sensed data in the approximation mode, so the number of nodes contributing in both the modes is typically much smaller.

IV. Implementation

We have implemented InDP on the Contiki OS using a modified version of Codecast. InDP utilize the radio-off time between communication rounds in Codecast to perform its computation. Figure 4 shows an overview of how InDP works on top of Codecast. Codecast supports efficient many-to-many communication using a combination of capture effect and network coding. More details can be found in [5]

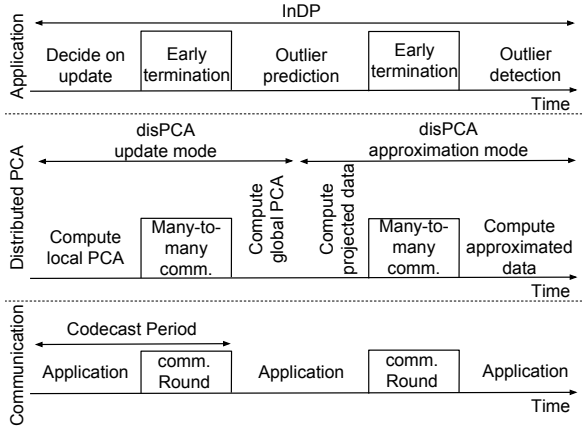

Fig. 4. InDP : Layering for communication, disPCA and application.

Next, we will describe how the distributed PCA computation is supported by the underlying communication layer.

Initialization: At the start of the computation r, in the update mode, $d_{ii}(r)$ is set to the local PCA $(D_i^{(t)}, E_i^{(t)})$ of round r, if node i needs to update its local PCA, and set it to the projected data (z_i, t_i) in the approximation mode if its data is required to be shared. $d_{ij}(r)$ is set to Unknown $\forall i \neq j$. Node i set $c_{ii}(r)$ to 1 for nodes with data update and 0 otherwise. $c_{ij}(r)$ is set to Unknown $\forall i \neq j$.

Data Sharing: For data value, node i shares $D_i^{(t)}, E_i^{(t)}$ in update mode and shares z_i, t_i in appriximation mode if it wants to contribute its data to the computation in this round. For

control information, node i shares with its neighbor $c_i(r)$, as mentioned previously.

Update: In the update mode, node i updates its Y matrix, for later use to compute global PCA, with the received local PCA for nodes with updates and use the previous values in its matrix for nodes which have no update. In the approximation mode, node i computes the approximated data matrix for nodes with data to share.

Termination: As described before, node i will terminate when c_i is updated $\forall j$ and d_{ij} is known $\forall j$ if $c_{ij} = 1$.

A. Outlier Detection

As an illustration on how distributed PCA can be utilized, we consider an outlier detection application by showing how data from the sensors can be processed to detect outliers in the network (nodes which have readings different than all others in the same network).

Note that the local eigenvalues capture the variance of each node's observations. Hence, outliers will have an eigenvalue that is different from most of the others. One way to utilize InDP more efficiently is to use an outlier prediction mechanism to check for anomalies instead of executing both modes and disseminate projected sensor data to all nodes. If an outlier is detected, then the node which has been detected that it is the outlier broadcasts its projected data (z_i and t_i). This will significantly reduce the amount of traffic to be shared if the number of records q is large.

InDP uses Eq: 2 to measure the distance between each sensor's projected data and the others to detect outliers.

$$d_{i,j} = d(z_i, z_j) = \sqrt{\sum_{k=1}^{q} (z_{i_k} - z_{j_k})^2} \quad \forall i \in N, \forall j \in N, j \neq i \tag{2}$$

where $d_{i,j}$ is the distance between sensor i and sensor j, z_i is the projected data matrix for node i, and N is the set of S sensor nodes.

InDP decides that sensor i is an outlier if Eq: 3 is satisfied for a predefined percentage of S.

$$d_{i,j} > predefined\_threshold, \quad j \neq i \tag{3}$$

V. Evaluation

A. Overview

We evaluate the performance of InDP by executing the protocol on the Flocklab testbed and Indriya testbed. The Flocklab testbed has 27 TelosB nodes deployed inside offices and in hallways in the same building at the Swiss Federal Institute of Technology Zurich in Switzerland [11]. On the other hand, the Indriya testbed is denser as it has 73 TelosB nodes deployed in the National University of Singapore over three floors [12].

Two datasets are used:

- Intel Lab Dataset [1]: a dataset from an indoor WSN deployment at Intel Berkeley Research Lab. Readings from the temperature, humidity, light, and voltage sensors are used. Data is sampled once every 31 seconds and 18 motes are used. Readings from some motes (5, 14, and 15) are excluded as a lot of the readings are missing or faulty.
- Indriya Dataset: a 24 hours temperature, humidity, and light sensors data collected from Indriya testbed. Data is sampled once every 10 seconds and 18 motes are used.

We presented four sets of evaluation. The set of experiments cover execution time (Section V-B), power consumption (Section V-C), outlier detection (Section V-D) and reconstruction error vs data reduction (Section V-E).

B. Completion Time

To evaluate the average time required for InDP to complete, we performed experiments on the Flocklab testbed using up to 27 nodes, as well as the Indriya testbed using up to 73 nodes. We use the maximum transmission power and IEEE 802.15.4 channel 26.

In each experiment, there are always 16 destination nodes. Different destination nodes are selected in each experimental run. The remaining nodes in the testbed serve as relays. The number of data sources is either 1 or 16 and are chosen from the set of destination nodes. Unless stated otherwise, each experiment runs for one hour on Flocklab and three hours on Indriya. In the evaluation, we measure the time taken for the destination nodes to receive data from all the source nodes that have data to share. If there is no data to be shared, then completion time is taken for all the destination nodes to receive control information from all source nodes. As a baseline, we compare the completion time of InDP to CTP.

The CDF of the completion time of InDP are reported in Figures 5 and 6 on the Indriya and the Flocklab testbeds respectively. We can observe that the completion times can vary significantly depending on the choice of data sources and ambient interference.

The completion times with only a single source vary from 100ms to about 800ms in both testbeds. The median is 296ms and 396ms for FlockLab and Indriya respectively.

As expected, when the number of source nodes increases to 16, completion time increases. Considering only control information with zero or 1 data update, the completion times vary from 500ms to 1.4s on Indriya and 700ms to 1s on FlockLab, with the corresponding median completion of 945ms and 891ms respectively. When there are data from all 16 nodes, the completion times vary from 0.5s to 1.4s on Indriya and 0.8s to 1.5s on FlockLab, with the corresponding median completion times of 1.34s and 1.2s respectively.

The results also show the benefit of separating control and data dissemination when the number of sources increased. When there is no (or 1) data update, the protocol can terminate much faster. In the case of 16 sources, the median completion times reduce by 26% and 29% on FlockLab and Indriya respectively when the number of sources reduces from 16 to 0.

[1] http://db.csail.mit.edu/labdata/labdata.html

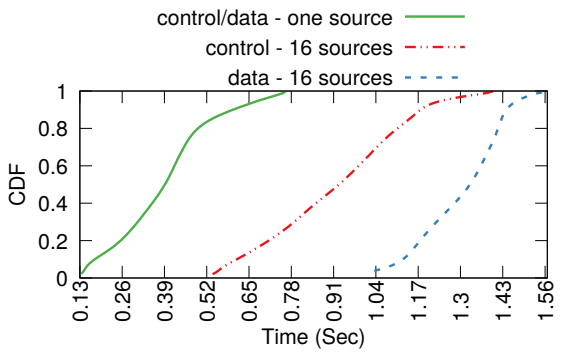

Fig. 5. InDP completion time CDF on Indriya testbed.

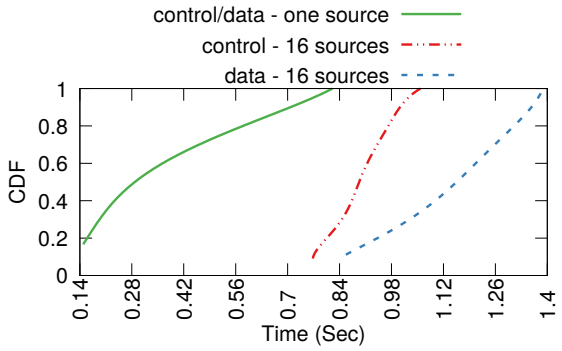

Fig. 6. InDP completion time CDF on Flocklab testbed.

The completion times of the CTP baselines are shown in Figures 7 and 8. Comparing the median completion times, the reduction is 85% and 98% for 1 and 16 sources respectively. Given that InDP is designed to support many-to-many communication, the significant improvement over CTP is expected.

C. Energy Consumption

To evaluate the energy consumption of InDP and CTP, we measure the power consumption on the TelosB mote. In this experiment, the payload size is set to 20 bytes and the network size is set to 16 nodes. Power measurement is obtained by connecting one of the 16 TelosB motes running InDP and connecting the sink node running CTP to a Monsoon power

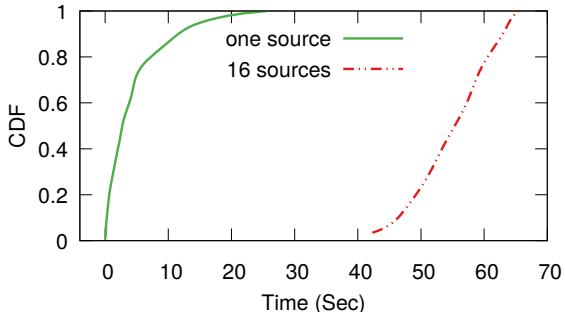

Fig. 7. CTP completion time CDF on Indriya testbed.

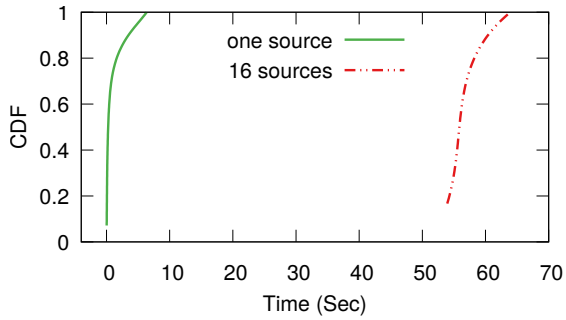

Fig. 8. CTP completion time CDF on Flocklab testbed.

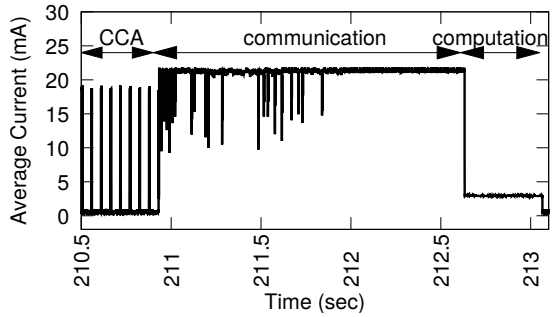

Fig. 9. InDP Energy consumption of 16 sources with data.

meter [2]. Table II shows the average energy consumed for InDP and CTP under scenarios with a different number of source nodes.

We can make the following observations. The energy consumed by InDP increases from 17mJ to 140mJ when the network size increases from 1 to 16. As a baseline, the energy consumed for CTP for the corresponding scenarios varies from 47mJ to 590mJ.

Figure 9 shows the detailed power consumption of InDP in the case of network size of 16 nodes and all sharing data. Note that computation energy is much less than communication energy and vary from 95.55% to 96.57% of the total energy consumption depending on the number of nodes with a data update.

D. Outlier Detection

In this section, we perform the following experiments to evaluate the outlier detection mechanism.

In the first experiment, 10 nodes are placed in an indoor environment over different rooms and along the corridor in the lab. 2 of the nodes (nodes 2 and 3) were attached to a laptop. The flashlight from a smartphone was used to increase the light sensor readings on these two nodes at different times. Nodes sampled the temperature, humidity, and light sensors at a frequency of 0.1Hz.

We execute InDP over these 10 nodes for 1000s. In each 100s cycle, the flashlight was used to increase the light

[2]https://www.msoon.com/

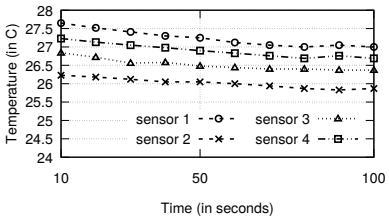

Fig. 10. Temperature readings.

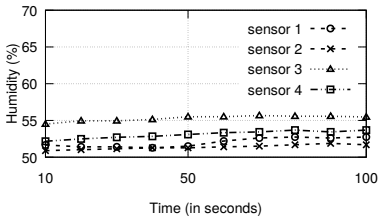

Fig. 11. Humidity readings.

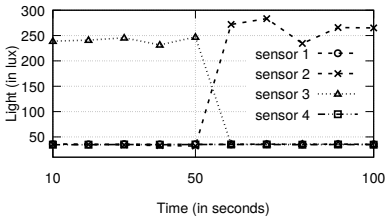

Fig. 12. Light readings.

TABLE II
TABLE OF INDP AND CTP: AVERAGE ENERGY CONSUMED (MJ)

Number of sources		Average energy consumed (mJ)		
w/ No Update	w/ Update	InDP total	InDP computation	CTP
Total sources of 1				
1	0	17	0.75	47
0	1	17	0.75	47
Total sources of 16				
15	1	108	4.8	590
0	16	140	4.8	590
16	0	1.5	-	590

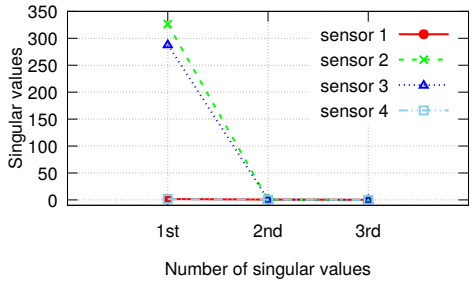

Fig. 13. Singular values of local PCA.

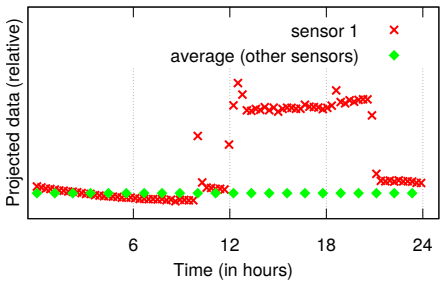

Fig. 14. Indriya traces: projected data.

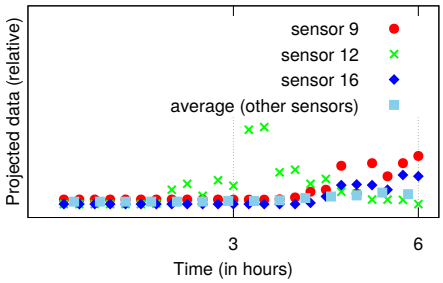

Fig. 15. Intel lab traces: projected data.

readings on nodes 2 or 3 at different times for a duration of 50s as shown in Figure 12.

Figures 10, 11, and 12 show the sensed data observations on 4 selected nodes with nodes 2 and 3 being the outlier nodes caused by the changes in light readings. The duration shown is 100s. As shown in Figures 10, 11, and 12, the sensor readings of all nodes for temperature and humidity are similar and do not vary much throughout the experiment. The only change comes from the sudden change in light readings in nodes 2 and 3.

Recall that after running the update mode of InDP, each node has the singular values from the matrix D. Figure 13 shows the singular values obtained from the PCA computed for 4 nodes based on data from a 100s interval (10 samples). It is clear that the singular values (and thus PCA) computed can efficiently be used to predict the outliers as the singular values from the outlier nodes are relatively different than other nodes in the network.

In order to evaluate InDP over a longer period, we utilize data traces collected over 24 hours from the Indriya testbed and the Intel Lab dataset. In this evaluation, we do not run InDP on the actual sensor nodes due to long data collection periods. Instead, we execute only the distributed PCA part of InDP offline using a C program.

For the dataset from the Indriya testbed, instead of using the singular values (D) for outlier detection, we used the project data ($\widetilde{x_i}$) available after approximation mode. Note that while we are able to detect the presence of an outlier in the update mode using the differences in the eigenvalues, we obtain the projected data after the approximation mode to confirm its status as an outlier.

The results are shown in Figure 14. One node (sensor 1) out of the 18 nodes shows a much higher projected data and can be easily observed. Going back to the original sensing data for validation, we see that sensor 1 sensed a lower temperature because it was located in a room whereby the air conditioning was set to a lower temperature during part of the experiment causing its temperature to be much lower for an 8hr period over the entire experiment duration.

Finally, we apply InDP to the Intel lab data. The data used is six hours of observations. The results using the projected data as outlier detector is shown in Figure 15. We observe that sensors 9, 12, and 16 exhibit different behaviors compared to

the other nodes, with sensor 12 showing the largest difference or the most anomaly. On further inspection, we note that nodes 9, 12, and 16 are close to the windows which may explain why the light readings are different (higher) than the other nodes.

E. Reconstruction Error and Compression

In this evaluation, we look into the accuracy, measured in reconstruction error of InDP. We will also investigate the trade-off between reconstruction error and compression achievable. The evaluation is done using the distributed PCA part of InDP using a C program. The parameters varied in the experiment are the following:

- **Number of Local PCA**: the number of the largest principal components of the SVD shared in the update mode of InDP.
- **Number of Global PCA**: the number of the largest principal components of the SVD used in the approximation mode of InDP to project and reconstruct data.

The reconstruction error is computed using Eq: 1. The results are shown in Table III using the 24 hours dataset from the Indriya testbed and 6 hours from the Intel lab dataset. From the table, we can make two observations. First, it is clear that the effect of the number of local PCA is low in that the reconstruction error is dominated by the number of global PCA. In fact, the number of local PCA almost has no impact on the reconstruction error for these two datasets. This is positive since outlier detection can be done only using the local PCA in the update mode. Second, the reconstruction error decreases as more data is used in the computation of the global PCA, which is expected.

Table IV shows the reconstructed error for the centralized PCA for 24 hours and 6 hours data from the Intel lab and the Indriya testbed datasets, respectively. In tables III and IV, the compressed data ratio can be determined by the number of global PCA that is used to project the data (e.g, 1 global PCA in the Intel lab data means that the projected data has one dimension that is projected on one PCA out of the four original dimensions and the compressed data ratio is $1/4$).

Comparing the reconstruction error of disPCA when only one local PCA is used to derive the global PCA, we can make the following observations. First, the reconstruction error of the distributed PCA is only slightly higher than the centralized version, whereby all data are sent to a single fusion center. Second, the reconstruction error decreases rapidly when the number of global PCA is increased. Third, the reconstruction error versus the compressed data ratio in both of the centralized and distributed PCA versions follow the same pattern which shows that disPCA can capture the distribution of the data the same way as the centralized PCA.

F. Discussions

The scalability of the protocol depends on how much computation can be done within the fairly small time frame of a single time slot. The current TelosB hardware places fairly tight constraints on the amount of the computation that can be done and thus limits the processing that is feasible and size of the network that can be supported. More complex processing can be performed, more data can be shared or a much larger network can be supported if either the processing power available on the node increases or the size of the time slot is increased. However, given that the TelosB hardware is fairly old and many newer and more powerful sensor hardware are now available, we expect the scalability and applicability of InDP to improve over time.

VI. Related Work

Data Compression A lot of research techniques have been presented to reduce the amount of data obtained from the sensors as reducing the communication will extend the lifetime of the network [13]. Switching to lower sampling rate is one way to reduce the number of transmissions but it may cause missing high-frequency events. Another way is to aggregate the sensor readings, a survey of the aggregation approaches is introduced in [14]. Data prediction based protocols are presented to reduce communication in WSN. In [15], the authors used a data prediction model built on top of synchronous transmissions. In [15], each sensor builds its model in the training phase and sends the model to the gateway. After that, each sensor will only send the reading if the predicted value, using the model, is not within a predefined threshold. Such a technique does not scale as the gateway is required to maintain a different model for each sensor. Another research interest in WSN is to use principal component analysis (PCA) to compress data while preserving the data structure. However, the current techniques require moving $O(mn)$ amount of data, where m is the number of data samples and n is the number of features. These approaches cannot scale with the number of nodes in the network [6], [7], [9], [10]. InDP shows how an algorithm such as disPCA [4] can be implemented.

Communication Protocols Collection Tree Protocol (CTP) [16] is a distance vector routing protocol designed for data collection in sensor networks. CTP uses a contention-based CSMA/CA layer as an underlying protocol which makes it extremely slow in a multi-hop setting. Ferrari et al. in [17] presented glossy, which exploits the concept of constructive interference to improve dissemination time for one-to-many communication. Authors in [18] introduced a data dissemination protocol (LWB) based on glossy which builds a shared bus in the wireless network. Splash is presented in [19], which is suitable for bulk data dissemination. In [20], the authors improve upon Splash by introducing better use of network coding. p^3 [1] is a high throughput bulk data transfer between two points in the network. p^3 uses synchronous transmissions over multi-channel along with packets pipeline to solve a problem of stalls in packets pipeline due to channel assignment in PIP [21]. For many-to-many communication protocols, Chaos [22] is an all-to-all aggregation protocol. In A^2 [23], the authors schedule multiple rounds of chaos to support network-wide agreement. However, these techniques cannot be used for sharing large data size between many nodes. Finally, Codecast [5] uses synchronous transmissions and a

TABLE III
DISPCA AVERAGE RECONSTRUCTION ERROR AND COMPRESSED DATA RATIO VS. THE NUMBER OF LOCAL AND GLOBAL PCA.

		Average Reconstruction Error															
Num. of local-global PCA		1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4	4-1	4-2	4-3	4-4
Compressed data ratio		1/4	2/4	3/4	1	1/4	2/4	3/4	1	1/4	2/4	3/4	1	1/4	2/4	3/4	1
Intel lab	Temp. (C)	0.69	1.21	0	0	0.69	1.21	0	0	0.69	1.21	0	0	0.69	1.21	0	0
	Humidity (%)	1.62	0.64	0	0	1.62	0.64	0	0	1.62	0.64	0	0	1.62	0.64	0	0
	Light (lux)	0.01	0.001	0	0	0.01	0.001	0	0	0.01	0.001	0	0	0.01	0.001	0	0
	Voltage (volts)	0.09	0.1	0.1	0	0.09	0.1	0.1	0	0.09	0.1	0.99	0	0.09	0.1	0.99	0
Compressed data ratio		1/3	2/3	1	-	1/3	2/3	1	-	1/3	2/3	1	-			-	
Indriya	Temp. (C)	0.46	0.33	0		0.46	0.02	0		0.46	0.02	0					
	Humidity (%)	0.21	0.16	0	-	0.21	0.006	0	-	0.21	0.006	0	-			-	
	Light (lux)	0.29	0.42	0		0.29	0.23	0		0.29	0.23	0					

TABLE IV
CENTRALIZED PCA AVERAGE RECONSTRUCTION ERROR AND COMPRESSED DATA RATIO VS. THE NUMBER OF GLOBAL PCA

		Average reconstruction error			
Number of global PCA		1	2	3	4
Compressed data ratio		1/4	2/4	3/4	1
Intel lab	Temp. (C)	0.69	1.2	0	0
	Humidity (%)	1.62	0.61	0	0
	Light (lux)	0.01	0	0	0
	Voltage (volts)	0.09	0.1	0.1	0
Compressed data ratio		1/3	2/3	1	-
Indriya	Temp. (C)	0.25	0.41	0	
	Humidity (%)	0.69	0.1	0	-
	Light (lux)	0.26	0	0	

network-assisted network coding (NANC) to better support all-to-all communication.

VII. CONCLUSIONS

We have presented a new technique for edge processing on constraint devices. InDP exploits distributed principal component analysis to reduce data transmissions without losing data structure nor high-frequency events. We believe the use of PCA is just one example. Our work has demonstrated the promise of edge computing on small devices and many more interesting applications can be introduced, pushing the limits on what edge computing can do.

ACKNOWLEDGMENT

This work is supported in part by the Ministry of Education, Singapore, under Tier One Grant R-252-000-621-114.

REFERENCES

[1] M. Doddavenkatappa and M. Choon, "P 3: a practical packet pipeline using synchronous transmissions for wireless sensor networks," in *IPSN*, 2014.

[2] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, 2016.

[3] J. Shlens, "A tutorial on principal component analysis," *arXiv preprint arXiv:1404.1100*, 2014.

[4] Y. Liang, M.-F. F. Balcan, V. Kanchanapally, and D. Woodruff, "Improved distributed principal component analysis," in *NIPS*, 2014.

[5] M. Mohammad and M. C. Chan, "Codecast: Supporting data driven in-network processing for low-power wireless sensor networks," in *IPSN*, 2018.

[6] H. Kargupta, W. Huang, K. Sivakumar, B.-H. Park, and S. Wang, "Collective principal component analysis from distributed, heterogeneous data," in *ECML PKDD*, 2000.

[7] H. Kargupta, W. Huang, K. Sivakumar, and E. Johnson, "Distributed clustering using collective principal component analysis," *Knowledge and Information Systems*, 2001.

[8] H. Qi, T.-W. Wang, and J. D. Birdwell, "Global principal component analysis for dimensionality reduction in distributed data mining," *Statistical data mining and knowledge discovery*, 2004.

[9] S. V. Macua, P. Belanovic, and S. Zazo, "Consensus-based distributed principal component analysis in wireless sensor networks," in *SPAWC*, 2010.

[10] Y.-A. Le Borgne, S. Raybaud, and G. Bontempi, "Distributed principal component analysis for wireless sensor networks," *Sensors*, 2008.

[11] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *IPSN*, 2013.

[12] P. Appavoo, E. K. William, C. M. Choon, and M. Mohammad, "Indriya2: A heterogeneous wireless sensor network(wsn) testbed," in *TRIDENTCOM*, 2018.

[13] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden, "Distributed regression: an efficient framework for modeling sensor network data," in *IPSN*, 2004.

[14] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Communications*, 2007.

[15] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza, "Data prediction+ synchronous transmissions= ultra-low power wireless sensor networks," in *SenSys*, 2016.

[16] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *SenSys*, 2009.

[17] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *IPSN*, 2011.

[18] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *SenSys*, 2012.

[19] M. Doddavenkatappa, M. C. Chan, B. Leong *et al.*, "Splash: Fast data dissemination with constructive interference in wireless sensor networks." in *NSDI*, 2013.

[20] W. Du, J. C. Liando, H. Zhang, and M. Li, "When pipelines meet fountain: Fast data dissemination in wireless sensor networks," in *SenSys*, 2015.

[21] B. Raman, K. Chebrolu, S. Bijwe, and V. Gabale, "Pip: A connection-oriented, multi-hop, multi-channel tdma-based mac for high throughput bulk transfer," in *SenSys*, 2010.

[22] 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.

[23] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "Network-wide consensus utilizing the capture eeect in low-power wireless networks," in *SenSys*, 2017.