May 2020

Flexible Congestion Management for Error Reduction in Wireless Sensor Networks

William Kolodzey
Clemson University, bill.kolodzey@gmail.com

Flexible Congestion Management for Error Reduction in Wireless Sensor Networks

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Engineering

by
William Kolodzey
May 2020

Abstract

The dissertation is concerned with the efficient resolution of data congestion on wireless sensor networks (WSNs). WSNs are of increasing relevance due to their applications in automation, industrial processes, natural-disaster detection, weather prediction, and climate monitoring. In large WSNs where measurements are periodically made at each node in the network and sent in a multi-hop fashion via the network tree to a single base-station node, the volume of data at a node may exceed the transmission capabilities of the node. This type of congestion can negatively impact data accuracy when packets are lost in transmission. We propose flexible congestion management for sensor networks (FCM) as a data-collection scheme to reduce network traffic and minimize the error resulting from data-volume reduction. FCM alleviates all congestion by lossy data fusion, encourages opportunistic fusion with an application-specific distortion tolerance, and balances network traffic. We consider several data-fusion methods including the k-means algorithm and two forms of adaptive summarization. Additional fusion is allowed when like data may be fused with low error up to some limit set by the user of the data-collection application on the network. Increasing the error limit tends to reduce the overall traffic on the network at the cost of data accuracy. When a node fuses more data than is required to alleviate congestion, its siblings are notified that they may increase the sizes of their transmissions accordingly. FCM is further improved to re-balance the network traffic of subtrees such that subtrees whose measurements have lower variance may decrease their output rates while subtrees whose measurements have higher variance may increase their output rates, while still addressing all congestion in the network. We verify the effectiveness of FCM with extensive simulations.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

We begin with a discussion of the importance of congestion mitigation on wireless sensor networks (WSNs) and a survey of the relevant background. We then define the model of the sensor network and the problem addressed by our data-collection scheme, flexible congestion management for sensor networks (FCM). We continue by providing details about the simulation and performance metrics used to evaluate data-collection methods. Next, we discuss work related to congestion mitigation in WSNs, including the three alternative data-collection methods we compare to our proposed method.

1.1 Study Motivation

Congestion alleviation and error mitigation are key problems in the study of data collection on WSNs, especially as the network size and the volume of data increase. Networks deployed to monitor physical conditions may help to avoid property loss and improve overall safety. Consider a network of sensor nodes deployed to monitor a physical phenomenon. E.g., a wildfire detection network. Each sensor node has the capability of local measurement of the phenomenon, data processing, and wireless communication with nearby nodes in the network. The dissertation focuses on a convergecast data-collection scenario, where measurements are taken at each node and forwarded to a unique sink node in the network (such as a cellular-network base station). It explores effective data-collection methods on networks with data bottlenecks, which are expected in inexpensive networks where the nodes have limited transmission and data-storage capabilities.

A given node in the network is able to achieve viable point-to-point communication links with only a subset of the other nodes in the network, in general, which may not include the sink node. Consequently, multi-hop routing through the network is necessary to deliver data measurements from an arbitrary sensor node to the sink. For purposes of the multi-hop convergecast, the network is organized into a spanning tree called the *routing tree*, with the sink serving as the root of the tree. Measurements travel as packets over multiple point-to-point links (hops) to reach the sink. Each node has a finite memory, and each communication link of the node has a finite achievable data rate (capacity). If the aggregate data rate into a node exceeds the capacity of either its input buffer or outgoing link, *congestion* occurs, which can be resolved in any of three ways: data summarization, a decrease in the frequency of sensor measurements, and the dropping (loss) of packets due to buffer overflow in the node. As such, data-collection methods with congestion-control mechanisms that seek to reduce or eliminate congestion need to consider both a node's input and output.

Data summarization is the representation of a data set, given as a number of packets of the same size, by a smaller number of packets of that size. A packet of data, for our purposes, is represented by the tuple $(val, count)$, where val is a measurement (or the current approximation of one or more measurements) and $count$ is the number of measurements represented by the tuple. We mostly consider one-dimensional values, but, for the most part, the fusion methods, data-collection schemes, and results mentioned in the dissertation extend to data tuples with multi-dimensional values. As an example of summarization, we can fuse two packets $(6, 1)$ and $(4, 1)$ together and represent them by a single packet $(5, 2)$, where the new *val* is the mean of the two original values, and the new *count* is the total number of measurements represented by the fused packets. The tuple can be augmented with a node ID or location, as in $(val, count, loc)$, so that some sense of the physical (or network) location of a measurement is maintained even after data fusion. If, as in a wildfire detection network, the high temperature values are more important, we could have the new *loc* set to the *loc* of the tuple with the highest *val* among the fused tuples. Alternatively, the new *loc* could be set to the average *loc* of the fused tuples or the *loc* of the node at which the fusion occurs. In this work, we do not consider the case in which location information is retained.

Wildfire detection is used as an illustrative example in the dissertation. Climate change, including global warming, increases the risk of wildfires in many regions in the world [3]. Recent wildfires in California and Australia are among the extreme events that may be related to climate change, which may exacerbate extreme weather events such as drought, high temperature, and the late onset of a rainy season. The likelihood, intensity, and at-risk regions are expected to increase, which poses greater threat to life, infrastructure, and natural resources. Early detection of wildfires or high-risk conditions could be important factors in mitigation of wildfire damage, and WSNs could help detect such conditions.

There is, in general, an error between an original measurement and its representation by a summarized value at the sink. We refer to this representation as the *estimate* of the measurement at the sink. In contrast to summarization, a decrease in the measurement frequency at each sensor node may result in less error between an original measurement and its estimate at the sink, but the timeliness of observation of the phenomenon of interest is degraded. The effect of a reduced measurement frequency may be well understood and predictable for some physical phenomena. On the other hand, packet loss due to buffer overflow has a random effect on the error at the sink for individual measurements; as a result, it has potentially the most adverse effect on the error between an original measurement and its estimate at the sink.

We consider several data-collection schemes which seek to resolve congestion by data summarization. Our proposed data-collection scheme, FCM, is compared to pure elimination (pureElim), spatio-temporal data collection (ST), and congestion-adaptive data collection (CADC). Each data-collection method is considered with data summarization that employs one of three fusion methods: the *k-means algorithm* [1] or either of two forms of *adaptive summarization* [2].

FCM is a set of management rules for data collection in a WSN, which may in principle use any fusion method to resolve congestion. It contains congestion-control variable management to balance the fusion requirements on subtrees of differing size and with different rates of variability in the data measurement values within the subtree. FCM utilizes the *fusion ratio* at each node, which is given by

$$\gamma = \frac{m_2}{m_1}, \tag{1.1}$$

where m_1 is the number of data tuples available in the input buffer and m_2 is the number of data tuples after fusion at the node. At any given time, the fusion ratio at a node lies between 0 and 1, where $\gamma = 1$ represents no summarization and $\gamma \ll 1$ represents aggressive summarization. FCM exploits the fact that a subtree with low data variability may incur low error even if its fusion ratio γ is low (that is, data is fused aggressively), and it can reduce the total error incurred by allowing parallel subtrees to use different fusion ratios. Since conditions can change on the tree, the management rules facilitate adaptation to changes in local data variability and network topology.

There are a number of network applications that call for efficient data collection, either as a primary or secondary function. Some examples of network applications with data collection as a primary function include wildfire detection, meteorology, industrial control, and sea-level monitoring. Some examples of network applications where data collection has a secondary role include autonomous-vehicle fleets and smart cities. Wireless networks are becoming increasingly pervasive with the internet of things, and many include sensor-data collection as a function.

For the purposes of simulation and performance evaluation, we adopt the example of a temperature sensor network for wildfire detection. In this example, measurements are one-dimensional, which simplifies the discussion, but most of the methods discussed, including FCM, extend to higher-dimensional data applications as well (such as wind-speed and direction measurements). We also show that FCM is scalable with respect to the size and dynamics of the network.

1.2 Model of the Sensor Network

In this section, we discuss the models used for the data-collection network in the dissertation. The network is composed of N sensor nodes that monitor some phenomenon in the environment (such as the temperature at different points in a forest) and send their periodic sensor readings to the sink via a routing tree. Our discussion includes the network topology, individual node capabilities, data flow, terminology, and assumptions used for our implementation.

1.2.1 The network tree

The network we consider is organized as a tree, where the root node alone is in layer one, its children are in layer two, their children are in layer three, and so on. A subtree rooted at node u is denoted $\mathcal{T}_u$ and the node's subtree size is $|\mathcal{T}_u|$. For the simulation in this dissertation, the number of nodes N is chosen and a pseudo-random tree is generated according to the parameters average number of children per node $\mu = 3$ and standard deviation in number of children per node $\sigma = 1.2$.

Table 1.1: Tree Notation

Symbol	Description
N	The number of nodes in the network
r	Sink node
u, v, i	Sensor nodes in the network
$\mathcal{T}_u$	Subtree of the routing tree rooted at node u
$\|\mathcal{T}_u\|$	Size of the subtree rooted at node u
μ	The mean number of a children assigned to a node
σ	The standard deviation in the number of a children assigned to a node
$\mathcal{C}_u$	The set of node u's children
$\mathcal{C}_u^-$	The set of node u's non-leaf children

1.3 Problem

In this section, the congestion-control problem is defined for convergecast data collection in a WSN. It is important to understand the impact congestion control has on data quality. Data accuracy is critical for an application running on the WSN to estimate the state of the monitored phenomenon. Congestion-control methods are used to reduce packet loss and the estimation error associated with it. However, the congestion-control methods are lossy and thus contribute to an increase in the error. Generally, an increase in error due to data summarization [2, 4] or measurement frequency reduction at the sensor nodes [5, 6, 7, 8, 9] is preferable to an increase in error due to packet loss, and the overall estimation error on the network is reduced when congestion control is implemented. The dissertation compares several congestion-control data-collection schemes which rely on data summarization for congestion control.

1.3.1 Error definitions

In each data-collection scheme, when congestion occurs at a node, the node's children are ordered to perform data summarization to reduce their transmission rates. The local measurement obtained at sensor node i in round t is denoted by $x_i(t)$, and the value of $x_i(t)$ is estimated by i's ancestor u in round t based on the received summarized data as $\hat{x}_i^u(t)$. The error is the difference between $\hat{x}_i^u(t)$ and $x_i(t)$. Table 1.2 contains terms related to the data and error in the network. The *estimation*

Table 1.2: Data and Error Notation

x_i	Measurement from node i
$\hat{x}_i^u$	Estimate of x_i at node u
e_u	Estimation error at node u
ϵ_u	Maximum tolerable error at node u
d_u	Distortion at node u
η_u	Maximum tolerable distortion at node u
c_u	Error contribution from node u
w_i	Priority coefficient of x_i generated by node i
E	Square root of average Huber error
$\mathcal{E}$	Per-experiment estimation error

error and *data distortion* are defined as follows. (We omit the parenthetical argument indicating the round in the remainder of the dissertation, except where it is needed to remove ambiguity in the discussion.)

Definition 1.3.1 (Estimation error) *The estimation error at node u in round t represents the sum of squared errors between the data values u receives from its subtree $\mathcal{T}_u$ and their actual values, in that round. I.e.,*

$$e_u(t) = \sum_{i \in \mathcal{T}_u} (\hat{x}_i^u(t) - x_i(t))^2. \tag{1.2}$$

Definition 1.3.2 (Distortion) *The data distortion at node u in round t is the sum of squared errors between the data values u receives from its subtree $\mathcal{T}_u$ and their corresponding values after fusion that are sent to its parent node v. I.e.,*

$$d_u(t) = \sum_{i \in \mathcal{T}_u} (\hat{x}_i^v(t) - \hat{x}_i^u(t))^2. \tag{1.3}$$

Definition 1.3.3 (Error contribution) *The error contribution c_u of a node u in round t measures the sum of squared errors between each measurement on u's subtree, $\mathcal{T}_u$ and the estimates of those measurements sent by u to its parent node v. I.e.,*

$$c_u = \sum_{i \in \mathcal{T}_u} (\hat{x}_i^v(t) - x_i(t))^2. \tag{1.4}$$

Numerical results in the dissertation are based on *experiments*, each of which is the operation of the network over N_R consecutive rounds. As we simulate the network, an experiment consists of many rounds of data collection given a fixed tree topology, data-collection scheme, and fusion method. We define the *per-experiment estimation error* over an experiment as follows.

Definition 1.3.4 (Per-experiment estimation error) *The per-experiment estimation error for an experiment is the square-root of the mean of the estimation error at the sink over the number of rounds in the experiment and the number of nodes in the* *network. I.e.,*

$$\mathcal{E} = \sqrt{\frac{1}{N \cdot N_R} \sum_{t=1}^{N_R} e_r(t)}. \tag{1.5}$$

In addition to the sum-squared error, we use the Huber-loss function [10] to evaluate the total error imposed on the network in our experiments. The Huber-loss function is defined for residual a as

$$L_\delta(a) = \begin{cases} \frac{a^2}{2}, & |a| \le \delta \\ \delta(|a| - \frac{\delta}{2}), & |a| > \delta. \end{cases} \tag{1.6}$$

A residual is the difference between an original measurement and the estimate of the measurement that reaches the sink. The function increases linearly with residuals greater than δ, while it increases quadratically with residuals less than δ. Figure 1.1 compares Huber loss and squared error. We see that for residual magnitudes less than the chosen value of δ, the Huber loss is half the squared error, while for residual magnitudes greater than δ, the Huber loss is linear. Here, δ is chosen to be a value

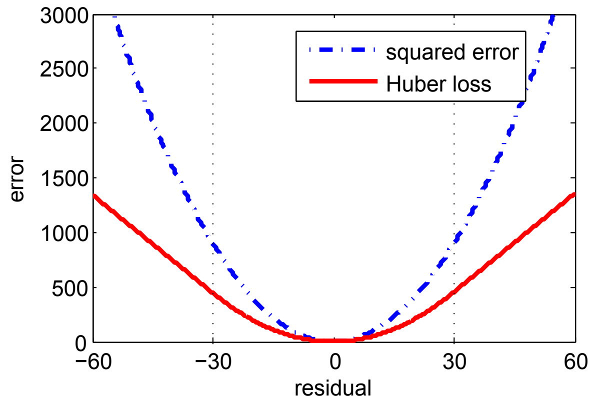


Figure 1.1: Squared error is compared to Huber loss with $\delta = 30$.

that signifies a "large" difference in measurements. For residuals smaller than δ, the Huber loss is equal to one half of the sum-squared error. The Huber-loss function is less sensitive to the impact of outliers on the total error compared with the sum-squared error. We use Huber error to determine the *experimental error* associated with an experiment, which is defined as follows.

Definition 1.3.5 (Experimental error) *The experimental error E is defined as the square-root of the average Huber loss of all residuals on the network for the duration of the experiment. I.e.,*

$$E = \sqrt{\frac{1}{N \cdot N_R} \sum_{t=1}^{N_R} \sum_{i \in \mathcal{T}_r} L_\delta(x_i(t) - \hat{x}_i^r(t))}. \tag{1.7}$$

Since δ is chosen to be large compared to the expected measurement values, the square root in the definition of experimental error acts as a proxy for the inverse of the Huber-loss function. If the value of δ were raised to be greater than all of the residuals in the experiment, the experimental error would be equivalent to $\frac{1}{\sqrt{2}}$ times the per-experiment estimation error.

Experimental error is used to measure the total error incurred in a simulated network data-collection experiment to compare the effectiveness of the various fusion methods and data-collection schemes. In comparison with the estimation error at the sink, the experimental error places less emphasis on a few large errors relative to a large number of small-to-moderate sized errors. In each of the numerical results in the dissertation, the Huber loss is considered to be $\delta = 30$.

The experimental error determined for all measurements in an experiment can be generalized to a separate value of experimental error for the measurements associated with each of several disjoint subsets of the overall measurements. This is denoted E_{bin}, where *bin* is the label for the subset of measurements in a particular priority class. To compute E_{bin} in contrast to E, instead of taking the square-root of the average Huber loss for the residuals associated with every measurement in the experiment, we use only the residuals associated with measurements in a particular priority-class *bin*. This allows us to characterize the error that results for each of multiple priority classes of data. In a portion of the dissertation, we consider a WSN that assigns a priority to each measurement based on its magnitude.

Each node estimates the priority of each data point it receives, based on its current estimated value, for the purposes of fusion. The weight a data point is given in k-means fusion is based on both the estimated priority and the number of measurements represented by a data point. The prioritized-data system used for our simulation is discussed in Section 1.4.3.

1.3.2 Objective

Simply stated, our objective is to avoid congestion while minimizing the resulting experimental error. With this objective in mind, the FCM scheme is proposed, which addresses congestion by reducing transmission rates via the use of lossy summarization while reducing network traffic according to the allowable distortion and reducing the experimental error.

1.4 Simulation and Performance Evaluation

This section describes the experimental design we use to evaluate the performance of each data-collection scheme in simulation. The model and experimental framework are implemented in the Matlab programming language.

1.4.1 Simulation details

An experiment begins with a fixed tree, data-collection scheme, and parameter initialization. The experiment continues for a predetermined number of data-collection rounds determined according to Section 1.4.4. The network tree is initialized beginning with the desired sink as its root. From the root, nodes are inserted up to the desired network size. Each node is assigned a random number of children from a discretized normal distribution with a mean of 3 and a standard deviation of 1.2. A sample network is visualized in Figure 1.2. The figure shows a 100-node tree with five layers. To ensure connectedness in the tree, each layer except for the last

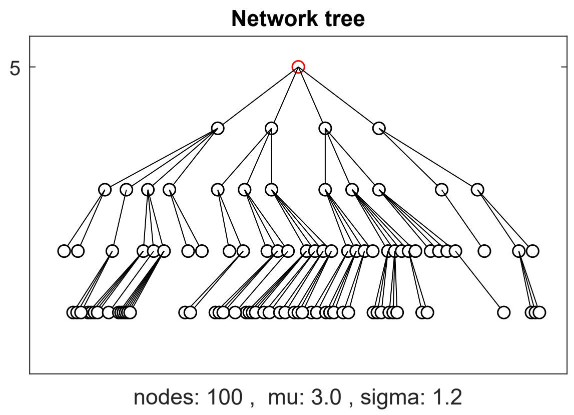


Figure 1.2: Example of a network tree generated with our method.

must contain at least one node with a child.

Measurement data at each node is one dimensional (as in temperature measurements) and is generated pseudo-randomly. Most measurements lie in the range $[0, 300]$. To mimic the time dependencies in real-world processes, the data generated at a given node varies smoothly over a sequence of data-collection rounds. With smoothly varying data measurements, we expect the variance of a set of data measurements on a given subtree in the network to vary smoothly as well. The data-collection method, FCM, takes advantage of the smoothly varying data with its slack-and-surplus mechanism described in Section 6.2.3. Details of the data-generation model are given in Appendix A.

In a round of data collection, each node receives data from its children, packets are dropped if buffer overflow results, the node summarizes the data according to its data-collection scheme and fusion method, the node prepares to send the data to its parent, and the node engages in an exchange of control packets with its children (and, in one data-collection scheme, with its parent). The node includes an input buffer and an output buffer with respective buffer capacities of mbs data tuples and mss data tuples. (We assume that the node capabilities, including mbs and mss, are the same for every node in the network.) A packet containing data that is received at a node is susceptible to loss due to buffer overflow in the node, but it is assumed that neither packets containing data nor packets containing control information are lost due to errors in the communication channel. The process for a typical node u is illustrated in Figure 1.3.

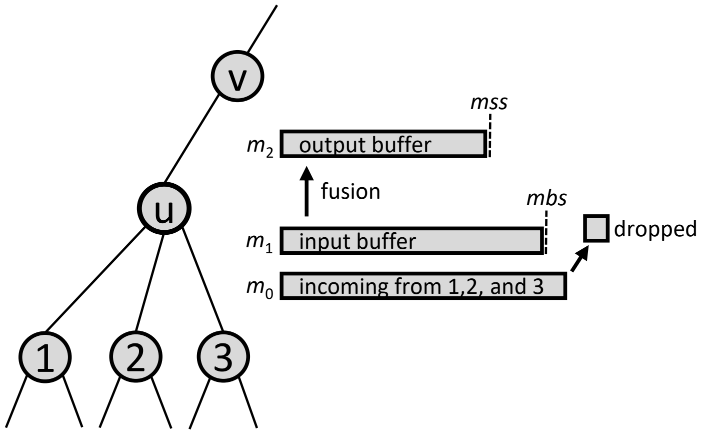


Figure 1.3: The data flow through an intermediate node in the network.

Within a round, during the phase in which data tuples are transmitted to node u from its children, a total of m_0 data tuples arrive at the node from its children and one data tuple is derived from a local measurement at node u. (The transmitted data tuples from the children are those stored in the output buffers of the children at the end of the previous phase.) The single data tuple corresponding to the local measurement is written into a memory location in the input buffer, and the data tuples received from the children are written into the remaining locations in the input buffer in the order in which they are detected at node u until all data is received or the input buffer is full. The order in which data tuples are detected is modeled as random, and all data tuples are assumed to be detected even if they are dropped due to buffer overflow.

If any additional data tuples are detected at node u after its input buffer is full, the last data tuple that was placed in the buffer is removed, and it and all subsequently detected data tuples are dropped due to buffer overflow and represented by a single proxy data tuple in that memory location in the buffer. (Consequently, either no data tuples are dropped or a minimum of two data tuples are dropped.) The resulting number m_1 of data tuples stored in the input buffer during the round is equal to at most mbs. They include the locally generated data tuple, either zero or one proxy data tuple for the received data tuples which were dropped, and either at most $mbs - 1$ received data tuples from the children (if no data tuples were dropped) or exactly $mbs - 2$ received data tuples from the children (if data tuples were dropped).

Once the data transmissions to node u from its children have been received, node u summarizes its input-buffer contents and stores the results in its output buffer. The number of data tuples stored in the output buffer and the data summarization of the input-buffer contents used to obtain them are determined by the fusion method employed, the capacities of the two buffers, and dynamic constraints placed on the node's transmission size to its parent. The constraints are determined prior to the current phase by the adaptive control of local congestion-control variables that is specific to the data-collection scheme. The resulting number m_2 of data tuples stored in the output buffer of node u during the round is equal to at most mss. The phase concludes with an exchange of control packets between node u and its children (and, in one data-collection scheme, with its parent) containing information which is used in determining the values of the congestion-control variables at the nodes in the next round.

The network includes two special cases of node data flow. A leaf node has no children, so it allocates only sufficient buffer memory for the single data tuple corresponding to its locally-generated measurement in the round. The root node has no parent, so it does not perform summarization or require a memory allocation for an output buffer.

If there were no packet loss, it would be possible to determine the per-experiment estimation error imposed on the network by having each node u report its error contribution c_u up through the network to the sink (along with the data transmissions). A proof of this fact is given in Appendix B. Each node computes its error contribution by adding its local calculation of distortion to its estimation error. The estimation error at a node is the sum of error contributions reported to it by its children. In the case of packet loss, the accuracy of the estimation error is not guaranteed.

Even in the realistic network scenario in which packet losses occur, each node may still employ an analogous mechanism to estimate the error in the local estimates of data measurements associated with its subtree. The three data-collection schemes in the dissertation besides pureElim rely on congestion-control variables initialized and maintained on each node in the network that attempt to limit the sum-squared error introduced at the nodes. The update of congestion-control variables is motivated by the fact that the total error in a node's estimates of all data measurements in the subtree rooted at the node can be approximated as the sum of errors introduced at the nodes in the subtree. If we wish to limit the maximum tolerable error for an application, we divide that value and allocate it to the nodes throughout the network, often in proportion to the node subtree size, $|\mathcal{T}_u|$. Further details of congestion-control variable initialization is detailed for the relevant data-collection schemes in Chapters 4-6.

Since there may be packet loss, a centralized view by the simulation is required to keep track of estimates and compute the experimental error. A data structure designed for this purpose is used here and described in Appendix C. It is used to track the original measurements and their estimates throughout the network during data collection. Data tuples in the simulation are augmented with a third element, *head* (that is, $(val, count, head)$), to facilitate the update of the data-tracking structure. The error on subsets of measurements can be analyzed and compared using the data-tracking structure, which is especially useful in determining the error incurred on data of a specific priority.

1.4.2 Evaluation metrics

We evaluate the various data-collection schemes by experimental error, *network overhead* (*nov*), *update overhead* (*uov*), *data-delivery ratio* (*ddr*), and performance in the priority case. Experimental error is defined in Definition 1.3.5. The network overhead is the average number of data packets sent on the network per round of data collection. The update overhead is the average number of transmissions per round sent on the network for the purposes of variable update, such as each request a node sends for one of its children to reduce its transmission rate. Typically, the *nov* is much greater than the *uov*, but if a data-collection scheme requires a large number of updates, the *uov* could degrade network performance. Generally, there is a trade-off between error and network traffic. We also consider the data-delivery ratio, as defined in equation (1.8),

$$ddr = 1 - \frac{n_{\text{drop}}}{N}, \tag{1.8}$$

where n_{drop} is the number of sensor measurements for which no corresponding data tuple is received at the sink (due to packet loss), and N is the total number of measurements on the network.

1.4.3 Data collection with prioritized data

We also consider the priority case of data collection, where some measurements are more critical to the goals of the application. For example, in the wildfire-detection network, high-temperature measurements have higher priority than low-temperature measurements, and so we prefer lower error associated with the high-temperature measurements. Different data-collection methods and fusion techniques facilitate the goals of data prioritization to different degrees.

The typical range of measurements used in the examples is 0 to 300. In the examples employing prioritization, the data is partitioned by its magnitude into ten priority classes with boundaries at integer multiples of 30 between 30 and 270. In this way, the first priority bin includes original measurements less than or equal to 30, the second bin includes original measurements greater than 30 and less than or equal to 60, and so on until the tenth priority bin, which includes original measurements greater than 270. The priority level assigned to each measurement in a priority class increases from 1 to 10 for each of the classes.

1.4.4 Experiment duration

An appropriate duration for each experiment is determined to be at least 1000 rounds of data collection. The number of rounds of data collection should be

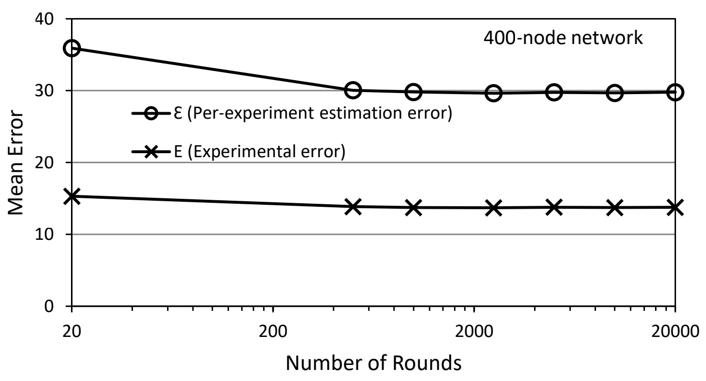


Figure 1.4: The mean error for experiments of varied duration.

sufficient to ensure result repeatability; that is, the standard deviation in a result, such as experimental error, between similar experiments should be low compared to its mean value. We consider experiments to be similar when they are performed with the same data-collection scheme, fusion method, network topology, and number of rounds. The standard deviation of the experimental error is observed to diminish up to approximately 5000 rounds of data collection for the network sizes we consider.

Figure 1.4 illustrates how the mean experimental error E and the mean per-experiment estimation error $\mathcal{E}$ vary with experiment duration. Figure 1.5 illustrates how the standard deviation between experiments decreases with the number of rounds in each experiment. In Figure 1.4 and Figure 1.5, results are shown for 20 experiments in each of which, the experimental error and the per-experiment estimation error are determined for that experiment. The mean and standard deviation of the 20 corresponding values are shown as functions of the number of rounds of data collection in Figure 1.4 and Figure 1.5, respectively. Each experiment uses the CADC scheme with k-means fusion, the same 400-node network topology, and the same experimental parameters, including experiment duration. The set of measurement values is allowed to vary between experiments. The number of rounds is plotted on a logarithmic scale to help show the decrease.

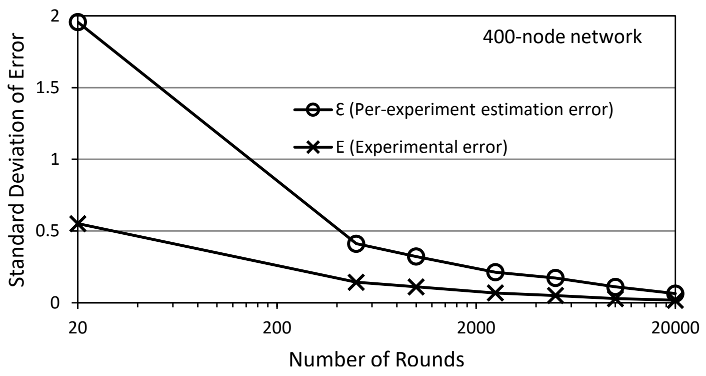


Figure 1.5: The standard deviation in error for experiments of varied duration.

1.5 Related Work

To address the problem of congestion on WSNs, many data-collection schemes have been developed. The approaches are varied, and some are better suited to a specific type of network or data-collection application. For example, centralized control may perform better for small networks and distributed control may perform better for large networks. To address bandwidth limitations, which cause congestion, most schemes control the routing of network traffic, the measurement rate at the sensor nodes, or the degree to which data is summarized at relay nodes. All three of the approaches to congestion-control are lossy, and a focus of this line of research is to minimize the estimation error associated with the physical phenomenon monitored by the sensor network.

1.5.1 Routing

Some data-collection schemes make routing, the decision of network topology and data traffic, central to the congestion-control problem. In these schemes, some routes are prioritized while others are deactivated to avoid congestion and ensure data integrity. Chang and Tassiulas [11] proposed a distributed algorithm to maximize network lifetime by solving a routing problem. Cristescu *et al.* [12] used a Slepian-Wolf coding model and a joint-entropy coding model to optimize the network topology and reduce transmission costs. Gupta *et al.* [13] attempted to minimize network-communication costs by selecting a subset of sensor nodes to represent the entire network. Han *et al.* [14] applied approximation algorithms to achieve energy-efficiency and reliability in the case of variable-power nodes in a WSN. Lee and Keshavarzian [15] proposed a multi-faceted approach including load balancing, scheduling and network-topology design to improve energy efficiency and data reliability. Park and Sahni [16] considers the routing of each message to maximize network lifetime using shortest-path computations. Wang *et al.* [17] proposed adaptive approximate data collection (ADC), which divides the network into unfixed spatially-correlated sub-clusters to reduce error and communication costs.

1.5.2 Sample-rate reduction

Several methods rely on reduction of the rate at which measurements are made at the sensor nodes. Bian *et al.* [6] proposed centralized-control protocols based on the network topology to adjust the measurement rates of the nodes in a small network. Paek and Govindan [8] proposed rate-controlled reliable transport (RCRT), a light-weight protocol, to avoid congestion on the network. Sankarasubramaniam *et al.* [5] proposed the centralized data-collection scheme, event-to-sink reliable transport (ESRT), to conserve energy expenditure on the network. Silberstein *et al.* [18] proposed the constraint-chaining (CONCH) algorithm using suppression to reduce energy costs on the network while maintaining a global view of data values. Su *et al.* [19] considered an approximate problem and its dual to find a near-optimal-performance solution to the rate-allocation problem in sensor networks. Wan *et al.*

[9] proposed congestion detection and avoidance (CODA), a data-collection scheme which seeks to mitigate congestion and reduce energy usage by congestion detection and targeted transmission-rate reduction. Zhou and Lyu [7] proposed price-oriented reliable transport protocol (PORT), which considers local and global approaches to reduce energy consumption and improve data fidelity on WSNs.

1.5.3 Data summarization

Several congestion-control approaches use data summarization to reduce the size of transmissions on the network. Ahmadi *et al.* [2] introduced the spatio-temporal data-collection scheme and considers the notion of contextual priority levels for data. Luo *et al.* [20] applied compressive-sampling theory to extend network lifetime on large sensor networks with spatially-correlated data. Wan *et al.* [21] proposed minimum-latency aggregation schedule (MLAS), which considers network topology and node-communication capabilities to reduce latency in networks subject to interference. Iri *et al.* [4] introduced the congestion-adaptive data collection (CADC) scheme, which regulates network traffic with limits on the data distortion allowed at each node.

Our work uses data summarization in a distributed data-collection scheme. We assume a pre-determined network topology and a constant sampling rate. We intend for our method to be scalable, so we use distributed congestion control, and we compare our data-collectoin scheme to schemes with similar approaches.

The rest of the dissertation is organized as follows. Chapter 2 discusses the fusion methods for data summarization, which are key components to the data collection methods. Chapters 3-6 concern the data-collection schemes. Chapter 3 is about pure elimination; Chapter 4 is about spatio-temporal data collection; Chapter 5 is about congestion-adaptive data collection; and Chapter 6 is about flexible congestion management for sensor networks. The conclusion is given in Chapter 7. The appendices include details on smooth data generation (Appendix A), a proof about error contributions summing in a network (Appendix B), a data structure to keep track of data estimates (Appendix C), and a list of abbreviations used in the dissertation (Appendix D).

Chapter 2

Fusion Methods

Data fusion is used to summarize data and resolve congestion without packet loss. It allows a node to reduce the number of available data tuples to satisfy the output rate constraints of a node or the input rate constraints of a node's parent. Additionally, a reduction in the number of data tuples frees up bandwidth on the communication links. If there is low variability in a set of data, fusion can help reduce the network traffic with low distortion by representing the data set by a smaller number of tuples. We consider three methods of data fusion: the k-means algorithm and two versions of adaptive summarization.

2.1 The K-means Algorithm

K-means is a clustering problem where, given a set of n data points and a number k of clusters of the points, we seek the data-point cluster membership that minimizes the sum of squared distances from each point to its cluster center [1]. Each cluster center is the mean of the points belonging to the cluster. Each point lies in the cluster whose center is closest to the point. Consequently, the problem of k-means clustering is typically posed in terms of the equivalent problem of selecting k cluster centers which achieve the optimal clustering. In general, determining the optimal clustering for $1 < k < n$ is an NP-hard problem [22]. The k-means algorithm introduced in [1] is an iterative search for a locally optimal k-means solution, which converges quickly in practice. Starting with an initial guess for the location of the center of each of the k clusters, the membership of the clusters is determined by assigning each point to the cluster with the nearest center to the point. The initialization method we consider equates each of the k centers to a different point in the data set, randomly. Then, each of the n points in the data set is assigned to the cluster with the closest center, and the center of each cluster is updated to the mean of the points in the cluster. This process is repeated until the cluster memberships converge (that is, they are unaltered by an iteration of the updates). Figure 2.1 illustrates the algorithm on a 2D set of points with $n = 14$ and $k = 4$. While the algorithm is not guaranteed to converge to the optimal clustering for the given data points, it converges to a cluster membership that is locally optimal with respect to a change in the cluster membership of any single point. In this way, the cluster center represents all of the data points belonging to the given cluster. The number of iterations can be limited *a priori* to avoid the worst-case run time possible on certain input cases. Since k can be chosen between one and n inclusive, we can use the k-means algorithm to fuse data to precisely the size of data set that is needed.

There are some daunting worst-case bounds for run time discussed in [23]. Additionally, the k-means algorithm can perform much worse than optimally [24]. However, for typical data sets encountered in a data-collection scenario, the run-time and the error do not approach the worst-case values. With one-dimensional data, a k-means algorithm exists that guarantees optimal convergence with a worst-case time complexity $O(nk)$ using dynamic programming [25], where n is the number of points

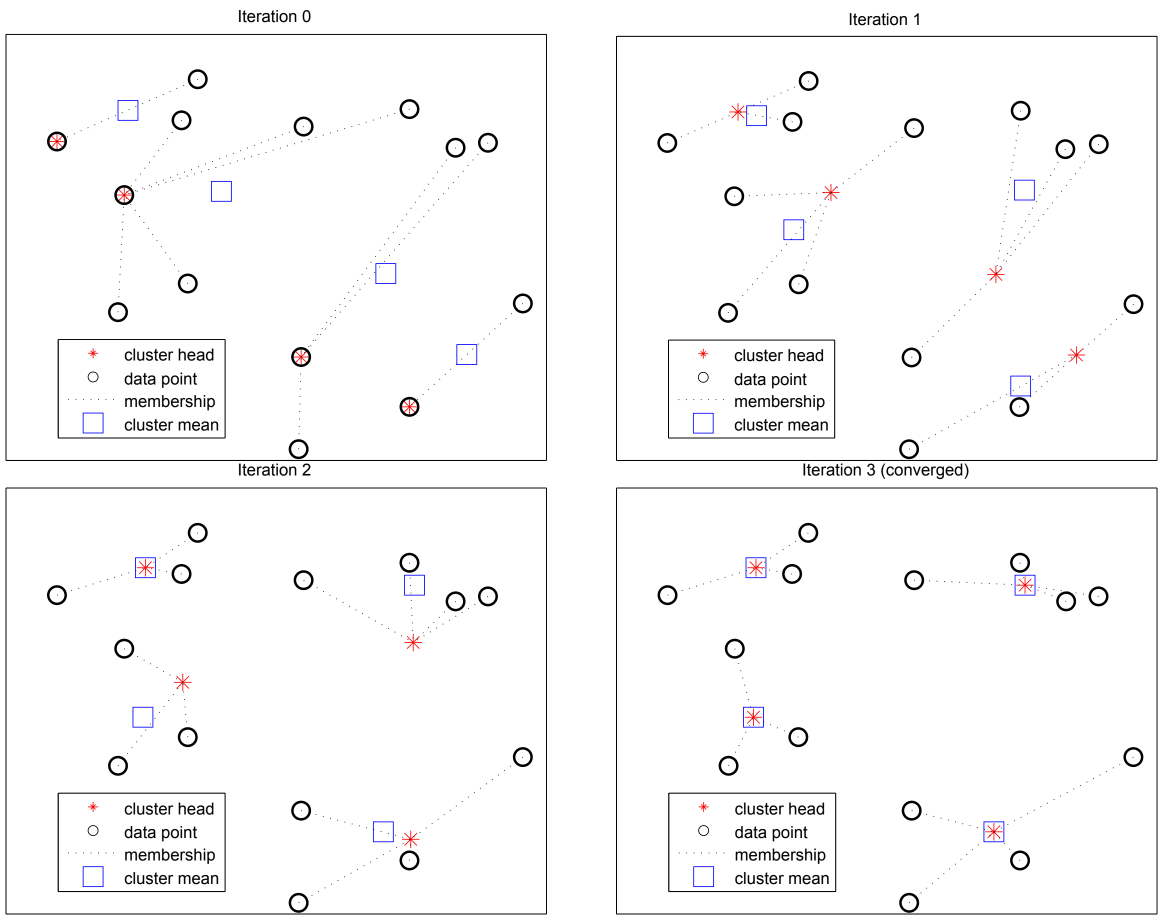


Figure 2.1: K-means demonstration.

and k is the number of cluster centers. To avoid rare cases of long convergence time with multi-variate data, a maximum number of iterations can be enforced. Even if the clusters determined by the k-means algorithm are sub-optimal, the summary is still preferable to the case of dropped packets.

In the WSN we consider, each node has a subset of the entire data set available for summarization. There are two interesting results regarding multi-stage k-means applications. If k-means fusion is performed successively as shown in Figure 2.2b, with an intermediate number of centers greater than the final number of centers, the sum of squared errors on the data after the two steps of fusion is bounded by five times the optimal sum of squared errors solution to k-means in one step [26] as shown

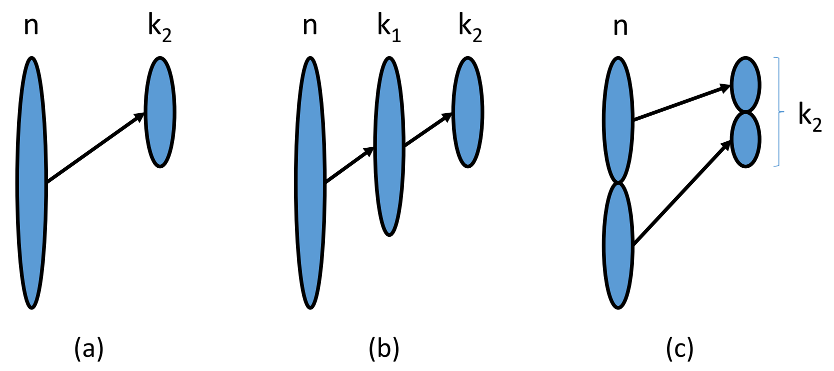


Figure 2.2: Illustration of fusion results: (a) fusion in one-step, (b) fusion in two-steps, and (c) fusion in partitions.

in equation (2.1),

$$S \leq 5OPT, \tag{2.1}$$

where S is the sum of squared errors for both optimal k-means solutions in the two-step case, and OPT is the sum squared error for the optimal solution to k-means in the one-step case (that is, the case in Figure 2.2a). The second interesting result is that when the data is split into two or more subsets, the case illustrated in Figure 2.2c, the sum of squared errors on the data in the split case may be arbitrarily worse than in the case of the whole. To illustrate this, consider the data set with eight values, $\{1, 1, 2, 2, 3, 3, 4, 4\}$, where the number of elements must be reduced by a factor of 2, we would choose $k = 4$, and the optimal center locations are $\{1, 2, 3, 4\}$, each representing two of the original elements. This summarization imposes zero error. If we instead separate the data set into the two worst-case subsets, each $\{1, 2, 3, 4\}$, we would use $k = 2$ for fusion on each set, and the final center locations become $\{1.5, 1.5, 3.5, 3.5\}$, clearly imposing some error. This is similar to the problem we face when we fuse data to resolve congestion throughout a network, and it demonstrates the possible impact of fragmented data on summarization error.

When data is prioritized (as described in Section 1.4.3), the points in k-means are given weight according to their priority, so that the cluster centers become the weighted average position of the points.

2.2 Adaptive Summarization

Adaptive summarization, outlined in [2], works by fusing each pair of adjacent data points in the buffer into a single data point. The value of the new data point is the average value of the two adjacent data points. In the priority case, the new value is the weighted average of the two adjacent data points, where the weights are the data priorities of the two points. Thus, a set of n data points stored in the buffer is reduced to $\frac{n}{2}$ data points (n even) or $\frac{n-1}{2}+1$ data points (n odd) after fusion through the sequential application of pairwise fusion of distinct pairs of points in the original set. We assume the final data point in the original set is unaltered if n is odd. A significant reduction in the error incurred with this fusion method can be achieved by first sorting the data, so this should be the first step, where possible. Adaptive summarization reduces the number of data tuples by powers of two, since it cuts the number of tuples in half (rounded up) on each iteration. If one iteration does not reduce the size of the data set sufficiently, it is reduced again, in another iteration, by half. This can be repeated as many times as required until the data set is reduced to as few as a single tuple. Adaptive summarization does not provide the degree of control of the size of the post-fusion data set that is provided by k-means, but it is faster and less computationally intensive, so it is a good option for low-energy or low-budget applications.

We consider two methods of adaptive summarization. The first method applies pairwise fusion to data tuples in the order in which they occur in the node's buffer. It is referred to as *adaptive summarization without sort* (adpSum). The second method first sorts the data tuples in order of increasing value and then applies pairwise fusion to data tuples in the sorted order. This method is referred to as *adaptive summarization with sort* (adpSum-sort).

2.3 Comparison of the Fusion Techniques

The fusion method has a significant impact on both the representation of each measurement as it flows through the network and the computational workload at each node; consequently, the different fusion methods can result in significantly different network performance. This is illustrated in Figure 2.3 for an example of one round of summarization at a single node. In the example, the received set of data values is

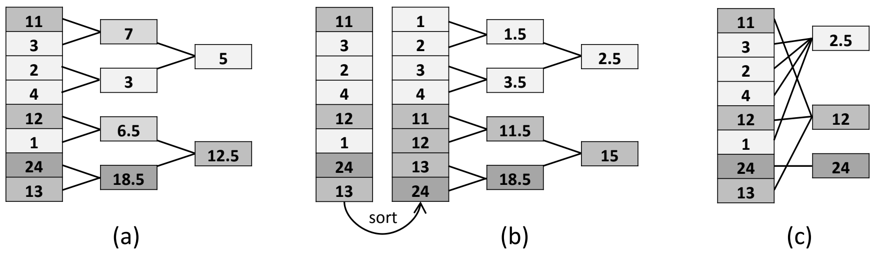


Figure 2.3: Example of data fusion using the three methods: (a) adpSum, (b) adpSum-sort, and (c) k-means.

given by $\{11, 3, 2, 4, 12, 1, 24, 13\}$. The use of adpSum, adpSum-sort, and k-means is shown in parts a, b, and c of the figure, respectively. For each case, the received data set is illustrated in the left-most column and the final summary is illustrated in the rightmost column. A shading gradient is used to highlight the difference in values.

In the scenario of Figure 2.3, it is required that the eight elements of the data set are summarized by a maximum of three elements. Adaptive summarization requires two steps of fusion, and the data must be fused down to two elements to satisfy the size requirement of the summary. K-means allows the data to be summarized by three elements. Adaptive summarization without sort incurs a sum-squared error of

315 with errors of 155 and 160 introduced in two consecutive steps. Adaptive summarization with sort incurs a sum-squared error of 115 with errors of 62 and 53 in the two steps. K-means incurs a sum-squared error of 7 in one step.

The simplest technique, adaptive summarization without sorting, has a lower computational requirement than adaptive summarization with sorting, since the latter includes the additional task of sorting. Adaptive summarization without sorting also has the benefit of only requiring two points at a time to begin summarization, so it is appropriate for applications where memory resources are limited and nodes must send data transmissions with low delay from when they receive data transmissions from their children. Both impose less computational burden on the node than does the k-means algorithm, though they offer less flexibility with regard to the size of the data set after fusion. Adaptive summarization reduces the number of tuples by powers of two by averaging adjacent tuples. As in the basic adaptive-summarization method, the number of tuples is reduced by factors of two. The most sophisticated technique, the k-means algorithm, tends to fuse data tuples close in value while not fusing outliers. Additionally, the number of data centers k can be chosen to fuse the data to any extent.

K-means has an additional advantage over adaptive summarization in the priority case, where isolated, high-priority measurements can be preserved while lower-priority data points are fused. As shown in Figure 2.3, k-means preserves the outlier value, 24, in its summary. The advantage comes at a cost, as k-means is significantly more computationally intensive than either adaptive-summarization technique, which could also contribute to delay in the data received at the sink.

The difference in outcomes in a single round at each node is reflected in differing performance at the network level depending on the fusion method. There is a trade-off between the performance, on the one hand, and the required computation and input-output delay, on the other hand, among the three fusion methods: adaptive summarization, adaptive summarization with sort, and the k-means algorithm. The performance difference is illustrated in Figure 2.4, which shows the experimental error for experiments with each of the network sizes: 200, 400, 600, 800, and 1000 nodes. The data-collection scheme, pure elimination without data-priority classes, is used for the experiments. Performance is shown for each of the three fusion methods. (The pureElim method is not designed with any inherent bias towards one of the fusion methods.)

K-means requires more processing power than adpSum and adpSum-sort, but it accommodates precise fusion ratios and almost always results in lower summarization error as seen in the example of Figure 2.4. Adaptive summarization reduces the

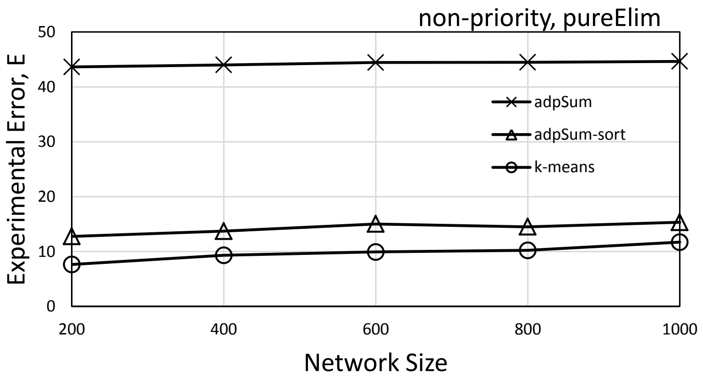


Figure 2.4: Comparison of the fusion methods under the pureElim data-collection method.

available data by powers of two, and it only considers fusing adjacent data tuples, before or after sort. A significant reduction in error in adaptive summarization is observed if the data tuples are sorted first, but k-means still provides significantly lower error. We observe that k-means is the best fusion method at reducing experimental error, followed by adpSum-sort. AdpSum incurs the highest experimental error of the fusion methods.

We repeat this analysis with the FCM method detailed in Chapter 6. The results, pictured in Figure 2.5, reinforce what is observed in the case of the pureElim method without data-priority classes. Figure 2.6 shows that the same ordering in

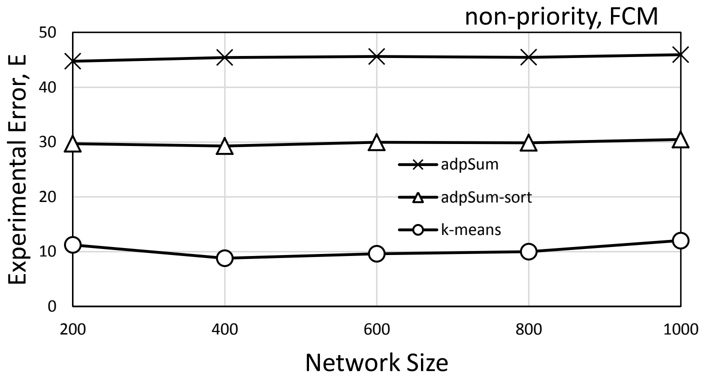


Figure 2.5: Comparison of the fusion methods under FCM in the non-priority case.

performance among networks using the three summarization methods occurs if data with multiple priority classes is used.

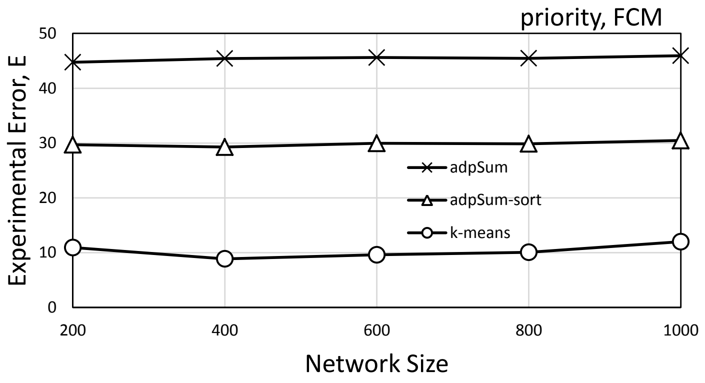


Figure 2.6: Comparison of the fusion methods under FCM in the priority case.

2.4 Fusion up to a Given Distortion

In three of the four data-collection schemes in the dissertation, we use summarization of the available data up to a given maximum distortion. To accomplish this, a node applies successive instances of the fusion method to the available data set. In each instance, the number of tuples in the summary is decreased by the minimum amount from the last instance. With k-means, we reduce the number of tuples by one in each instance. With adaptive summarization, we reduce the number of tuples by a factor of two (rounded up) in each instance. By comparing the tuples in the summary to the tuples in the available data, we determine the distortion (sum-squared error) incurred in a given instance. In general, the first instance whose distortion exceeds the given maximum distortion is rejected, and the summary of the previous instance is kept as the final summarization by fusion up to the given distortion. This summary may be used for transmission to the node's parent. However, additional fusion beyond this summary may be necessary to satisfy the output-buffer capacity. Further implementation details specific to each data-collection scheme are given in their corresponding chapters.

2.5 The Effect of Fusion on Data Accuracy

A simple example shows the effect of fusion on data accuracy. Two child nodes u_1 and u_2 transmit data packets to their shared parent node v. We assume each node is able to reduce its data output rate using adaptive summarization. Suppose that during a time slot, node u_1 has the set of data $\{1, 4\}$, and u_2 has the set $\{2, 3\}$, ready to transmit to node v, and each data item has $count = 1$. I.e., each node stores two tuples of the form $(val, 1)$. The shared parent node's input-buffer capacity can accommodate up to three tuples. To avoid congestion at node v, the child nodes need to reduce the total number of tuples they send this round, through data fusion. The adaptive-summarization fusion method computes the average of consecutive pairs of values, so there are two options to reduce the total number of tuples sent to node

v by a single tuple: one where node u_1 summarizes data tuples (1,1) and (4,1) to (2.5,2); and two where node u_2 summarizes data tuples (2,1) and (3,1) to (2.5,2). The data distortion (defined in equation (1.3)) is 4.5 in case one and 0.5 in case two. In both cases, congestion is avoided at the input to node v to the same extent (that is, the input queue length is decreased to three tuples). However, the two options incur different levels of data distortion and thus different levels of estimation error at node v.

In the above example, it is clear that we would prefer the child with greater data variability to employ less severe fusion. The mechanism used to accomplish this includes a congestion-control variable called maximum tolerable distortion and is discussed in Chapters 4-6. The idea is to decrease the fusion ratios of the children indirectly by increasing their maximum tolerable distortions. For instance, if the children u_1 and u_2 were only allowed to fuse such that the sum-squared error resulting from the fusion is less than 4, then only node u_2 would fuse its data. In this case, the congestion is resolved with the smallest possible resulting error.

Chapter 3

Data Collection with Pure Elimination

This chapter outlines the pureElim scheme [27] for data collection in wireless sensor networks (WSNs). PureElim uses simple rules to eliminate all congestion in the network by using data fusion to reduce the data rates to and from congested nodes in the network. The premise of pureElim is to fuse data as late as possible in transit through the network to the sink by fusing only as much as necessary at each node to avoid all packet loss. A node fuses incoming data to satisfy the transmission rate constraint of its output link, and a node requests that its children reduce their transmission rate until the incoming packets are manageable by the node's buffer. PureElim can be a good choice for a network with a static topology and only a single application generating data traffic in the network, but it is not appropriate for more sophisticated applications where nodes may join, exit, or move within the network, or in networks where we expect significant differences in data variance between subtrees over time. In spite of these limitations, the pureElim method provides a performance baseline for the alternative data-collection methods considered in the dissertation.

3.1 Variables and Terminology

PureElim uses the fusion-ratio limit Γ_u as the only per-node congestion-control variable. The fusion-ratio limit Γ_u at node u determines the extent of fusion at u, and thus the fusion ratio γ_u may not exceed Γ_u. However, a node is incentivized to send as much data as possible, so γ_u is as high as possible, given the fusion method and available data, without exceeding Γ_u. Each node u fuses data so that the number of outgoing tuples m_2 is the floor of Γ_u times the number of available tuples m_1 at the node. I.e., $m_2 = \lfloor \Gamma_u m_1 \rfloor$.

3.2 Initialization

The network topology is initialized according to Section 1.4, and each node u has its fusion-ratio limit Γ_u set to 1. With fusion-ratio limit $\Gamma_u = 1$, node u sends as much data as it has at each time step, up to its output-buffer capacity, mss.

3.3 Congestion-Control Mechanism

In pureElim, there are two means by which the fusion-ratio limit Γ is decreased at a congested node. First, if the number of data tuples at a node exceeds its output capacity, and its current value of Γ is too large to decrease the number of tuples in the input buffer to within the node's output capacity, the node decreases its fusion-ratio limit to reduce the number of tuples so that it satisfies the output-buffer capacity. Second, if congestion is detected at the input to the node, the node makes a request to each of its children to reduce their output rates by the factor F_R equal to the total size of the incoming data divided by the maximum input buffer size. Each child reduces its fusion-ratio limit to accommodate the parent node's input requirement.

The child node i establishes its fusion-ratio limit for the next time step, $\Gamma_i(t+1)$, based on equation (3.1),

$$\Gamma_i(t+1) = F_R(t) \cdot \Gamma_i(t), \tag{3.1}$$

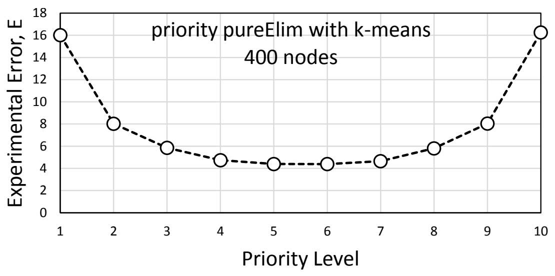
where $\Gamma_i(t)$ is its fusion-ratio limit in the current time step.

There is no relaxation mechanism for the congestion-control variables in pure elimination; that is, the fusion-ratio limit never increases at a node. On a static network using pureElim, there is no need for relaxation, but the lack of a relaxation mechanism makes the method unsuitable for a network with a dynamic topology.

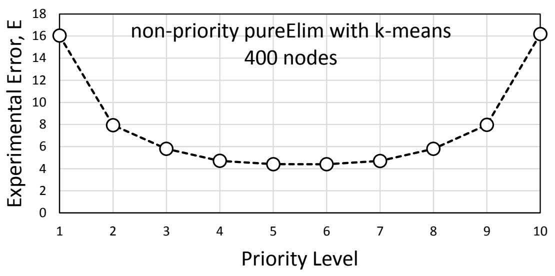
If there are significant changes to the network topology, the congestion-control variables set in pureElim may no longer be appropriate. To avoid inappropriate congestion-control variables, the network could be reinitialized periodically or when major changes to the network are detected. The reset of the fusion-ratio limit to one at each node requires some network traffic (update overhead), and we would expect to observe an increase in the number of lost packets and a corresponding decrease in data accuracy for one round of data collection after the reset. Alternative data-collection schemes, including FCM, use dynamic adjustments of per-node variables which avoid network-wide resets of the variables. They are discussed in Chapters 4-6.

3.4 Priority Case

To quantify performance in the case of prioritized-data collection in a WSN, we compare the experimental error in each of the ten priority bins in the priority case (as described in Section 1.3.1) to the experimental error in the non-priority case. To motivate the example, we consider an experiment of 5000 rounds of data collection with pureElim using the k-means fusion method on a network of 400 nodes. The experimental error for each priority bin is shown in Figure 3.1a with prioritization and in Figure 3.1b without prioritization. The difference in the experimental error ΔE



(a)



(b)

Figure 3.1: Experimental error by priority class.

between the two cases is normalized to account for differences in total experimental error between the two cases. It is given by

$$\Delta E_{bin} = \left(\frac{E^p_{bin}}{E^p} - \frac{E^n_{bin}}{E^n} \right) \left(\frac{E^p + E^n}{2} \right), \tag{3.2}$$

where the superscript n denotes the non-priority case, p denotes the priority case, and the subscript bin indicates the priority-bin data subset on which the experimental error is calculated. The first term in the product is a normalized difference in the experimental error for each priority bin between the priority case and the non-priority case. The second term in the product is the average of the two total experimental errors and it serves to re-scale the normalized difference to the units of experimental error. The result is shown in Figure 3.2.

Since pureElim does not have a mechanism to reduce error on higher priority data, the differences in experimental error observed are due to the fusion methods themselves. Figure 3.3 shows ΔE for each of the three fusion methods. The value of ΔE is small for each of the fusion methods under pureElim. We return to this method in Chapter 6 as we compare the performance with multiple priority classes

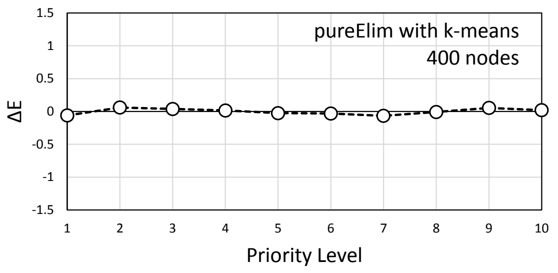


Figure 3.2: Difference in experimental error with and without prioritization.

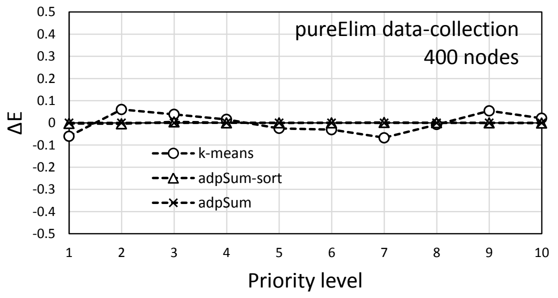


Figure 3.3: Difference in the experimental error for each of the fusion methods with pureElim data collection.

for all of the data-collection methods considered in the dissertation.

3.5 Performance Evaluation

In Chapter 6, we compare pureElim to the other data-collection schemes.

Chapter 4

Spatio-Temporal Data Collection

The spatio-temporal data-collection scheme (ST) is introduced in [2]. Each node u employs the local congestion-control variable, *maximum tolerable distortion* η_u (referred to as the "error" in [2]), which provides an upper bound on the distortion d_u at the node. If congestion is detected at a node, the maximum tolerable distortion at each node in the subtree rooted at the node is increased, which allows for more fusion at the nodes. If a node experiences no congestion, it reduces the maximum tolerable distortion at each node in the subtree.

4.1 Initialization

Each node is initialized with its maximum tolerable distortion equal to zero. With zero maximum tolerable distortion, no fusion occurs at the node, except to reduce the number of tuples to be at most the output-buffer capacity.

4.2 Congestion-Control Mechanism

As in the other data-collection methods, data is summarized using fusion to alleviate congestion. While any fusion method can be used to reduce the number of data tuples at a node with ST, its congestion-control variable update is based on adaptive summarization (adpSum and adpSum-sort). That is, the change to the maximum tolerable distortion at a node is based on power-of-two reductions in the number of data tuples.

When congestion is detected at a node u, the maximum tolerable distortion is increased to resolve the congestion based on where the congestion occurs. If congestion is detected at the input, the node sends a request to each child to increase the child's maximum tolerable distortion. If congestion is detected at the output, the node increases its own maximum tolerable distortion. In either case, the maximum tolerable distortion at node u in round $t + 1$ is given by

$$\eta_u(t+1) = \max(2\eta_u(t), \eta'_u), \tag{4.1}$$

where $\eta_u(t)$ is the maximum tolerable distortion in round t and η'_u is the distortion that would be introduced in the available data due to a single-step adaptive summarization at node u.

Data is fused at a node as much as possible by the selected fusion method (k-means, adpSum-sort, or adpSum) without exceeding the maximum tolerable distortion at the node. Generally, the fusion ratio γ decreases as the maximum tolerable distortion increases, but the fusion ratio also depends on the data variability. If there is a low variance in the data values at the node, the fusion ratio can be lower while still satisfying the maximum tolerable distortion. As the variance in the data increases, it may be necessary to increase the maximum tolerable distortion to reduce the congestion. Since we expect the variability in the data to change with time in many instances, corresponding adjustments to the maximum tolerable distortion are made, including both increases and decreases.

4.2.1 Variable relaxation

In the absence of congestion, the maximum tolerable distortion is reduced via relaxation. If the number of incoming packets to a node is less than the input-buffer capacity, then the node informs its children that they may increase their fusion ratios via maximum tolerable distortion according to

$$\eta_u(t+1) = \left(\frac{1}{\eta_u(t)+\rho}\right)^{-1}, \tag{4.2}$$

where ρ is a constant relaxation parameter set for the application. The relaxation accounts for both changes in data variability and changes to network topology. When a node's subtree size decreases, it is likely the value of η_u at the node can be decreased while still avoiding congestion. In our simulation, ρ is set to 0.001.

4.3 Priority Case

Though ST is designed to work with adaptive summarization (adpSum and adpSum-sort), it can use any fusion method, including k-means. Figure 4.1 shows the experimental error for each priority class with ST data collection and k-means fusion, in both the non-priority and priority cases. The difference, computed by formula (3.2), is shown in Figure 4.2.

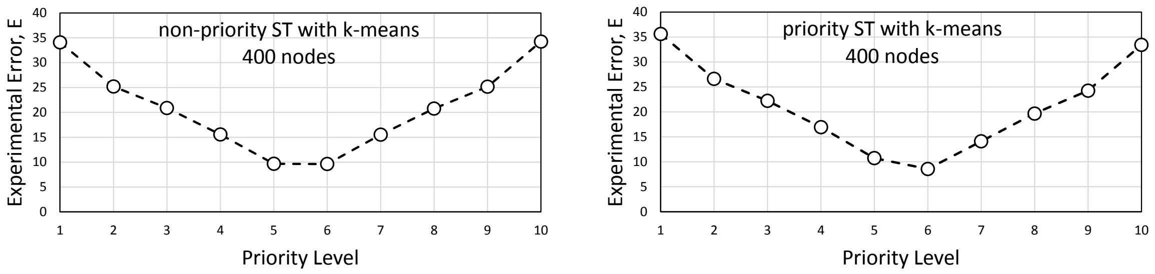


Figure 4.1: The experimental error per priority class with k-means fusion.

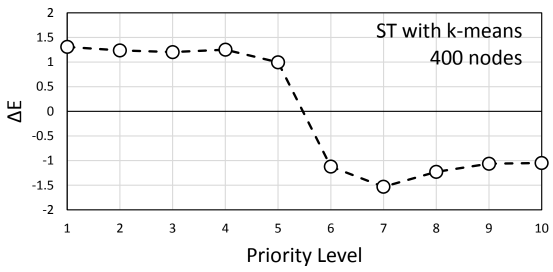


Figure 4.2: Difference in experimental error per priority class with k-means fusion.

4.4 Performance Evaluation

In Chapter 6, we compare ST to the other data-collection methods.

Chapter 5

Congestion-Adaptive Data Collection

The *congestion-adaptive data-collection* (CADC) scheme was introduced in [4]. CADC reduces the congestion on the network by data fusion which is regulated by using the maximum tolerable distortion at each node as the node's congestion-control variable. In CADC, as in the ST data-collection scheme described in Chapter 4, each node attempts to fuse data up to its maximum tolerable distortion. The *maximum tolerable error* χ at the sink is specified for the application as a static configuration parameter. The congestion-control mechanism is designed to keep the estimation error at the sink lower than the maximum tolerable error specified for the application. Each node keeps track of its contribution to the total estimation error in the network as described in Section 5.3. In cases in which the estimation error at the sink is not guaranteed to be within the maximum tolerable error specified for the application, the user of the application is notified.

5.1 Variables and Terminology

The two per-node congestion-control variables for CADC are the maximum tolerable error ϵ_u and the maximum tolerable distortion η_u, at node u. The maximum tolerable error ϵ_u is a limit on the total estimation error from the subtree $\mathcal{T}_u$ while the maximum tolerable distortion η_u is a limit on the distortion that can be introduced at node u. The maximum tolerable error at the sink is held constant throughout an experiment.

Auxiliary variables are stored in the memory of each node to facilitate the congestion-control mechanisms. These auxiliary variables include the variable η_u^* and the array variable $\boldsymbol{\eta}_{hist}$. Their meaning and use are discussed in Section 5.3.

CADC uses several constant parameters including the maximum tolerable error at the sink χ, the *historical window size* w, and the *relaxation constant* ρ. The parameters should be chosen appropriately for the network size, the node capabilities, and the anticipated distribution of measurements made at the nodes.

5.2 Initialization

At the start of data collection on a WSN using CADC, each node u is initialized with $\eta_u = 0$. The w entries in the auxiliary array variable $\boldsymbol{\eta}_{hist}$ at each node are set to 0.

The maximum tolerable error ϵ_u is set according to the network topology and the value of χ specified for the application. A breadth-first search is initiated from the root with $\epsilon_r = \chi$. Each parent node v sets the maximum tolerable error for each of its children. We let u denote a child of v and have

$$\epsilon_u = \alpha_u \epsilon_v, \tag{5.1}$$

where α_u is the relative weight of u's subtree size,

$$\alpha_u = \frac{|\mathcal{T}_u|}{|\mathcal{T}_v| - 1}. \tag{5.2}$$

In this way, the maximum tolerable error of a node equals the sum of the maximum tolerable errors of its children.

5.3 Congestion-Control Mechanism

Congestion control occurs at a node u in two sequential steps: the summarization step and the update step. In the summarization step, the node summarizes the available data based on its number of available data tuples m_1, the output-buffer capacity mss, and its maximum tolerable distortion η_u. In the update step, the node determines the congestion status and makes update requests to its neighbors to tune their congestion-control variables appropriately.

The enforcement of a maximum tolerable error at each node (and, ultimately, at the sink) is based on the idea of the error contribution c_u as defined in equation (1.4). As shown in [4] and repeated here, the Cauchy-Schwartz inequality is used to obtain an upper bound on the error contribution of u based on values accessible to the node,

$$\begin{aligned}
c_u &= \sum_{i\in\mathcal{T}_u}(\hat{x}_i^v - x_i)^2 \\
&= \sum_{i\in\mathcal{T}_u}((\hat{x}_i^v - \hat{x}_i^u) + (\hat{x}_i^u - x_i))^2 \\
&= \sum_{i\in\mathcal{T}_u}(\hat{x}_i^v - \hat{x}_i^u)^2 + \sum_{i\in\mathcal{T}_u}(\hat{x}_i^u - x_i)^2 \\
&\quad +2\sum_{i\in\mathcal{T}_u}(\hat{x}_i^v - \hat{x}_i^u)(\hat{x}_i^u - x_i) \\
&\leq \sum_{i\in\mathcal{T}_u}(\hat{x}_i^v - \hat{x}_i^u)^2 + \sum_{i\in\mathcal{T}_u}(\hat{x}_i^u - x_i)^2 \\
&\quad +2\sqrt{\sum_{i\in\mathcal{T}_u}(\hat{x}_i^v - \hat{x}_i^u)^2 . \sum_{i\in\mathcal{T}_u}(\hat{x}_i^u - x_i)^2} \\
&= d_u + e_u + 2\sqrt{d_u \cdot e_u} \qquad (5.3)
\end{aligned}$$

where e_u is the estimation error at node u and d_u is the data distortion due to data fusion at u. The bound is valid if there are no dropped packets in the subtree of node u or in the transmissions from node u to its parent node v. The inequality serves as a guiding approximation even if there are dropped packets (which cannot be accounted for in the distortion at the node).

5.3.1 Summarization step

In CADC, data flow at node u is regulated directly by η_u. After receiving the data tuples from its children in round t of data collection and adding its own measurement, the node applies the chosen fusion method to the available data to create a summary. As described in Section 2.4, the summary is made after successive applications of the fusion method are applied, which result in incrementally fewer tuples in the summary. The final summary is the last summary which incurs a distortion less than η_u or is comprised of no more than mss tuples, whichever is smaller.

5.3.2 Update step

In the update step of round t at node u, congestion is detected at the node if $\frac{mbs}{m_0} < 1$, where mbs is the input-buffer capacity and m_0 is the number of incoming packets (as shown in Figure 1.3). If node u is congested, each child is informed that it must increase its maximum tolerable distortion to decrease the total number of tuples incoming to node u in the next round. If node u is not congested, each child is allowed to relax its fusion requirement.

Congested case: If congestion is detected at node u, it determines the factor F required to alleviate congestion promptly at the output of its children. The factor F is based on the level of congestion at node u's input and is given by

$$F = \min\left(1, \frac{mbs}{m_0}\right). \tag{5.4}$$

Each child i of u uses its available pre-summary data from round t to determine a quantity denoted by $\eta_i^*(t+1)$, the desired maximum tolerable distortion for round $t+1$. Test-summarization fusion is applied to that data until the fusion ratio is at most $F \cdot \gamma_i(t)$, where $\gamma_i(t)$ is the fusion ratio for the summarization in round t at node i. A *test summary* is a summary made using the current set of data tuples in the node's input buffer for the purposes of congestion-control parameter update rather than for transmission to the node's parent. The resultant distortion is $\eta_i^*(t+1)$. A history of the values of η_i^* is stored in $\boldsymbol{\eta}_{hist}$ for the last w time steps. The required value of η_i^* is stored as the most recent historical value in $\boldsymbol{\eta}_{hist}$, and the oldest entry in $\boldsymbol{\eta}_{hist}$ is dropped to avoid more than w historical values as given by

$$\boldsymbol{\eta}_{hist} = \{\eta_i^*(t+1), \eta_i^*(t), ..., \eta_i^*(t-w+2)\}. \tag{5.5}$$

To avoid frequent rounds in which congestion occurs at node i, the maximum tolerable distortion for the next time step $\eta_i(t+1)$ is set to the maximum value in $\boldsymbol{\eta}_{hist}$.

An increase in the maximum tolerable distortion η_i of the child node i may cause the estimation error at its parent node u to exceed the maximum tolerable error ϵ_u, so ϵ_u is re-calculated. To alleviate all congestion, node u sets its maximum tolerable error according to equation (5.6). The value of the maximum tolerable error for the next time step, $\epsilon_u(t+1)$, is determined by node u according to

$$\epsilon_u(t+1) = \sum_{i \in \mathcal{C}_u} (\eta_i(t+1) + \epsilon_i(t+1) + 2\sqrt{\eta_i(t+1) \cdot \epsilon_i(t+1)}), \tag{5.6}$$

which is the sum of the upper bounds on the error contributions from u's children as described in equation (5.3). Each time ϵ_u is calculated at a node u, the node requests the most recent values of η_i and ϵ_i from each of its children. If the updated maximum tolerable error $\epsilon_u(t+1)$ is greater than the previous maximum tolerable error $\epsilon_u(t)$, the node makes a request to its parent v to accommodate the change.

When node u makes a request to its parent node v to accommodate a change in u's possible error contribution (due to an increase in ϵ_u or η_u), node v updates the maximum tolerable error $\epsilon_v(t+1)$ needed to satisfy the changes according to equation (5.6). Each time the new value $\epsilon_v(t+1)$ exceeds the previous value $\epsilon_v(t)$, the recursive call continues to propagate from node to parent. If a request of this type reaches the root, and the sum of the error contributions from the root's children could exceed χ (that is $\epsilon_r(t+1) > \chi$), then we say that there is a χ-*breach*, and the application is informed that the accuracy of the data may be less than the desired accuracy. Even in the case of a χ-breach, the updated values of ϵ are used for the purposes of congestion alleviation, since error due to fusion is preferable to error due to packet loss.

Uncongested case: If congestion is not detected at node u, each of node u's children is informed that it may relax its congestion control for the next time step. Specifically, the maximum tolerable distortion for the next time step $\eta_i(t+1)$ at each child i is set to the distortion resulting from the test summary made on the available input-buffer data. In the test summary, the data is fused with a fusion ratio of at most $\min(\gamma_i(t) + \rho, 1)$, where ρ is the relaxation constant. The choice of the relaxation constant ρ should be tuned for the application so that the network can adjust to changes while not quickly returning to a congested state. We find experimentally that ρ should be in the range $0.01 - 0.1$. In the numerical experiments, $\rho = w^{-1}$, where $w = 10$ is the historical window size (that is, the length of the array variable $\boldsymbol{\eta}_{hist}$). Given the relaxed value of γ_i, node i determines the appropriate level of maximum tolerable distortion η_i^* in the same way as in the case of congestion. The maximum tolerable distortion for the next round $\eta_i(t+1)$ and the most recent entry in $\boldsymbol{\eta}_{hist}$ are both set to η_i^*.

5.4 Adaptation to Changes in Network Topology

The data-collection schemes with relaxation mechanisms for their congestion-control variables, including CADC, are well suited to handle changes in the network topology. For instance, if the subtree size of a previously congested node decreases dramatically, it need not require the same fusion ratios of itself or its children to alleviate congestion. After such a change in the network, the node determines that it is not congested and allows its children to increase their fusion ratios, by means of a relaxation mechanism, until the appropriate level of congestion control is reached.

Similarly, when the subtree size of a node increases, the additional data volume increases the likelihood of congestion. The network using CADC adjusts to the congestion by decreasing the fusion ratios of nodes on the subtree according to Section 5.3.2.

5.5 Priority Case

The priority case, the case of measurement values weighted by importance, is defined in Section 1.4.3.

Figure 5.1 and Figure 5.2 show the experimental error for each priority bin with CADC data collection and k-means fusion.

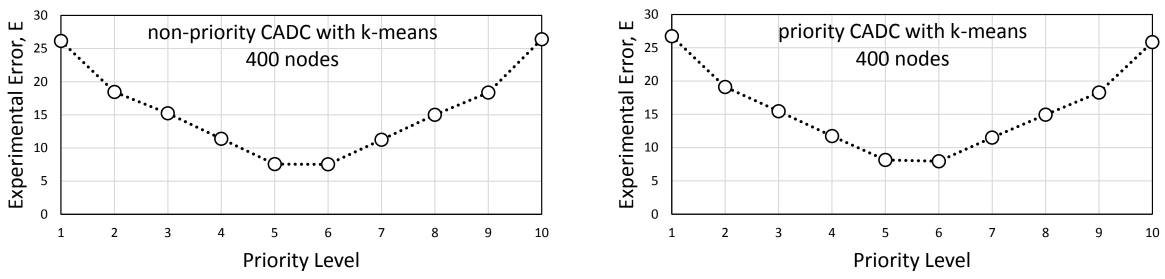


Figure 5.1: The experimental error per priority class with k-means fusion.

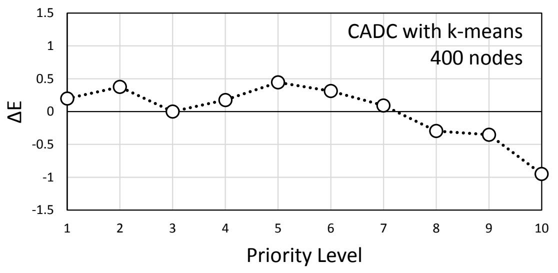


Figure 5.2: Difference in experimental error per priority class with k-means fusion.

5.6 Performance Evaluation

In Chapter 6, we compare CADC to the other data-collection methods.

Chapter 6

Flexible Congestion Management for Sensor Networks

This chapter details *flexible congestion management for sensor networks* (FCM), a data-collection scheme for error reduction on congested WSNs. FCM is a set of rules that manage the data summarization on a WSN using a fusion method such as one of those discussed in Section 1.2. FCM often outperforms the other methods considered in the dissertation by the metrics of experimental error and network overhead, or it incurs a slightly greater experimental error while requiring much less network overhead.

6.1 Variables and Terminology

The per-node congestion-control variables used in FCM are *desired output size* (dss), maximum tolerable distortion η_u, and slack σ. With careful adjustment of these variables, congestion is alleviated, opportunistic fusion is promoted, and traffic is balanced in a way to reduce overall error on the network.

6.1.1 Allowable distortion

As noted in Chapter 2, summarization incurs less error in general if it is applied to an entire data set than if it is applied separately to subsets of the data. This provides the motivation for each node to utilize its full output capacity, thereby delaying the summarization until it can be applied to a larger data set at a higher-level node in the tree, resulting in summaries that introduce less error. (With more data tuples available from the network, it is more likely that low-error fusions of two or more tuples are possible.) On the other hand, if low-error fusions are available at a node even when they are not necessitated by the node's output capacity, performing the fusion reduces the network overhead with a negligible increase in the total error.

For this purpose, an *allowable distortion* is assigned to the set of measurements obtained during a round of data collection for each application on the network. The allowable distortion is allocated across the network such that $\sum_{u \in \mathcal{T}_r} \eta_u = \phi$, where the variable η_u is the allowable incremental increase in the data distortion due to summarization at node u and ϕ is the allowable distortion for the data-collection application on the network tree rooted at node r. No summarization occurs at the root, so $\eta_r = 0$. Even when a node is not required to fuse its available data to satisfy the constraint of its output capacity or the output size requested by its parent, it still reduces the number of data tuples as much as possible with fusion as long as the resultant distortion, d_u, does not exceed η_u.

The allowable distortion is initialized throughout the network prior to the first round of data collection, starting at the nodes closest to the root, by dividing ϕ among the root's non-leaf children, where the root's non-leaf child w is assigned the temporary allowable distortion

$$\eta'_w = \frac{|\mathcal{T}_w|}{|\mathcal{T}_r| - n_F - 1}\epsilon_r, \tag{6.1}$$

where n_F is the number of leaf nodes among the root's children. Each leaf node f is assigned $\eta_f = 0$ since it takes one measurement and sends that value in each round of data collection with no fusion or distortion. Then, in a breadth-first search across the network, each non-root node v divides η'_v among itself and each non-leaf child u by

$$\begin{aligned} \eta_v &= \frac{\min(mbs, |\mathcal{T}_v|)}{D_v}\eta'_v, \\ \eta'_u &= \frac{\min(mbs, |\mathcal{T}_u|)}{D_v}\eta'_v, \end{aligned} \tag{6.2}$$

where η_v is the initialized value of allowable distortion at the node, η'_u is the temporary allowable distortion assigned to child u, and the denominator D_v is based on the subtree sizes of the node and its non-leaf children. We let $\mathcal{C}_v^-$ denote the set of v's non-leaf children and compute the denominator,

$$D_v = \min(mbs, |\mathcal{T}_v|) + \sum_{i \in \mathcal{C}_v^-} \min(mbs, |\mathcal{T}_i|). \tag{6.3}$$

The input-buffer capacity *mbs* is used in formula (6.3) to limit the weight of a subtree when the subtree-size exceeds *mbs*, which helps to distribute the allowable distortion more evenly throughout the network than if subtree size was used in each case. Accordingly, each node u without any non-leaf children has $\eta_u = \eta'_u$. In this way, we ensure that greater allowed distortion is allocated to nodes that are expected to have a greater number of available data tuples.

6.2 Congestion-Control Mechanism

The number of transmissions in the network used to update the congestion-control variables is the update overhead *uov*. In a network with a static topology, FCM is able to keep *uov* low since the congestion-control variables set after a single round of data collection alleviate all congestion. In a network with a dynamic topology, additional update messages may be needed to alleviate congestion, as we would expect when new nodes enter the network and contribute to network traffic. Further update messages may be sent on the network to re-balance the congestion-control variables on a node's children in the case of a change in the degree of data variability from the subtree rooted at each of the children. The additional update messages on the network are used to manage each node's slack, as is discussed in Section 6.2.3.

6.2.1 Requested output size

The desired output size *dss* helps regulate the output rate of each node. Figure 6.1 illustrates the use of the *dss* in the data flow at a node. Initially, each node's

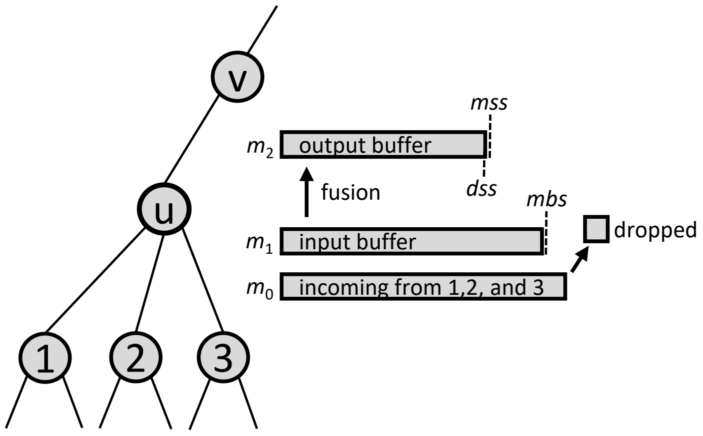


Figure 6.1: Data flow at a node labeled with sizes and capacities.

desired output size is set to the output capacity *mss*. When congestion is detected at the input to a node, the node requests that its children reduce their output sizes to satisfy the node's input-buffer capacity *mbs*. I.e., the sum of the desired output sizes of the children should not exceed the *mbs*. A larger subtree should be allowed to send proportionally more tuples to help minimize error; therefore, the *dss* allocated to each child is based on its subtree size. The allocation of the total input-buffer capacity of the congested parent across its children is managed by the algorithm described in Section 6.2.2.

6.2.2 Allocation algorithm

A parent node congested at its input updates the requested output size *dss* of each of its children using an algorithm that allocates its input-buffer capacity *mbs* in units of data tuples. We reserve at least one data tuple for each child, which is possible as long as *mbs* is greater than the number of children. (We assume a node's input-buffer capacity is much greater than the number of its children.) The tuples are distributed to the children according to a greedy algorithm in the order of their subtree sizes.

Algorithm inputs: The inputs to the algorithm include the number of tuples to be allocated N, and three $M \times 1$ arrays regarding the M children. The three arrays are the subtree size of each child $\mathbf{c}_{\text{size}}$, the minimum number of tuples to be assigned to each child $\mathbf{c}_{\text{min}}$, and the maximum number of tuples that can be assigned to each child $\mathbf{c}_{\text{max}}$. We assume that the sum of the children's maximum number of assigned tuples is greater than N.

From the inputs, we initialize the number of tuples that remain to be allocated *pot* and the number of tuples reserved for allocation to the remaining children *res*. We normalize the array $\mathbf{c}_{\text{size}}$, so that its elements sum to one. The number of tuples allocated to each child $\mathbf{c}_{\text{alloc}}$ is initialized to all zeros. The set of formulas (6.4) define the initialization of these variables.

$$\begin{aligned} pot &:= \min\left(N, \sum_{i=1}^{M} \mathbf{c}_{\max}(i)\right) \\ res &:= \sum_{i=1}^{M} \mathbf{c}_{\min}(i) \\ \mathbf{c}_{\text{size}} &:= \frac{\mathbf{c}_{\text{size}}}{\sum_{i=1}^{M} \mathbf{c}_{\text{size}}(i)} \\ \mathbf{c}_{\text{alloc}} &:= \mathbf{0} \end{aligned} \tag{6.4}$$

First pass: For each child i in descending order of subtree size, we adjust res, determine the number of tuples to allocate the child, add that number to the total number of tuples allocated to the child, and update the number tuples remaining to be allocated. The updates are shown in the list of formulas (6.5).

$$\begin{aligned} res &:= res - \mathbf{c}_{\min}(i) \\ alloc &:= \min\left(pot - res, \mathbf{c}_{\max}(i), \max(\text{nint}(N\mathbf{c}_{\text{size}}(i)), \mathbf{c}_{\min}(i), 1)\right) \\ \mathbf{c}_{\text{alloc}}(i) &:= \mathbf{c}_{\text{alloc}}(i) + alloc \\ pot &:= pot - alloc \end{aligned} \tag{6.5}$$

The nearest-integer operation, $\text{nint}(\cdot)$, rounds its argument to the nearest integer.

Subsequent passes: If any tuples remain to be allocated (that is, if $pot > 0$) after the first pass, one-or-more subsequent passes of allocation are used to distribute the remaining tuples. In a subsequent pass of the algorithm, we set the number of unallocated tuples $N_s = pot$. The remaining tuples are then assigned to the child nodes in nearly the same process as in the first pass. In each subsequent pass, there are no tuples reserved for allocation (that is, we have $res = 0$) and the maximum number of tuples that can be assigned to child i now takes into account how many tuples have already been assigned, $\mathbf{c}_{max}(i) - \mathbf{c}_{alloc}(i)$. The nodes are again visited in order of descending subtree size, and the updates are shown in the list of formulas (6.6).

$$\begin{aligned} alloc &\coloneqq \min\left(pot, \mathbf{c}_{\max}(i) - \mathbf{c}_{\text{alloc}}(i), \max(\text{nint}(N_s \mathbf{c}_{\text{size}}(i)), 1)\right) \\ \mathbf{c}_{\text{alloc}}(i) &\coloneqq \mathbf{c}_{\text{alloc}}(i) + alloc \\ pot &\coloneqq pot - alloc \end{aligned} \tag{6.6}$$

This process continues until there are no unallocated tuples in *pot*.

When we use this algorithm to set the *dss* of each child node, we ensure that the total number of incoming tuples in the next round of data collection does not exceed the input-buffer capacity *mbs*. The discrete nature of the allocation algorithm allows for tighter control and higher average output rates for the children than the multiplicative rate reductions used in the other data-collection methods. The allocation algorithm is particularly advantageous in the case where a small number of data tuples are transmitted, and therefore, each data tuple has greater significance.

6.2.3 Surplus and slack

Another per-node congestion-control variable, *slack* σ, is introduced for use in conjunction with *dss*. When the number of data tuples at a node after summarization is smaller than its value of *dss*, the node has a transmission rate surplus that can be distributed as slack to its siblings. This allows one or more of the siblings to transmit more tuples than the *dss* without reintroducing congestion at the parent. When slack is assigned to a node, that node may transmit a greater number of tuples than the value of dss by the amount of slack, up to the output capacity of the node.

If there is a surplus, a notification is sent to the parent, which distributes the slack to any children without surplus according to the same algorithm used to distribute dss. We allow the slack to be negative, where negative slack corresponds to a surplus, and we have that the sum of the slack associated with all of a node's children is never more (but sometimes less) than zero. With slack σ, we have the output of a node limited by $m_2 \leq \max(dss, dss + \sigma) \leq mss$. As network conditions change, a node's surplus may decrease, in which case the corresponding slack on its siblings is removed with another call to the slack balancing algorithm.

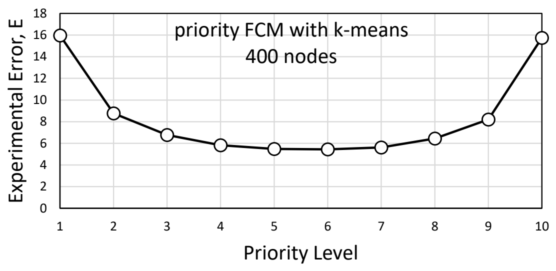
6.3 Adaptation to Changes in Network Topology

An important feature of a versatile data-collection scheme is adaptability to changes in network topology, such as nodes entering or leaving the network and connections changing. These changes in network topology are expected in a wide range of circumstances such as robot networks, hazardous conditions, network expansion, and aging infrastructure. The data-collection scheme should be able to support uninterrupted service in the case of node failure, and there should not be a lengthy reinitialization process or a drop in performance after network reorganization, therefore, the scalability is also an important feature of the data-collection scheme.

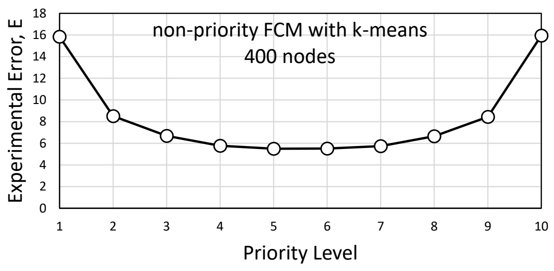
FCM supports changes in the network topology, since it scales well and has lightweight, distributed variable maintenance. After a change in the topology, each affected node reassesses its subtree size. The congestion-control variables dss and σ re-balance naturally as rounds of data collection continue on the network.

6.4 Performance in the Priority Case

The experimental error for each priority bin is shown in Figure 6.2a with prioritization and in Figure 6.2b without prioritization. The measurements are generated



(a)



(b)

Figure 6.2: Experimental error by priority class.

pseudo-randomly in the approximate range 0 to 300 according to the method described in Appendix A. The priority bin divisions range from 30 to 270 in increments of 30. In the non-priority case, the curve is symmetric about the midpoint, 150. In the priority case, the error for the higher-priority bins decreases relative to the non-priority case, as shown in Figure 6.3. The difference between experimental errors ΔE

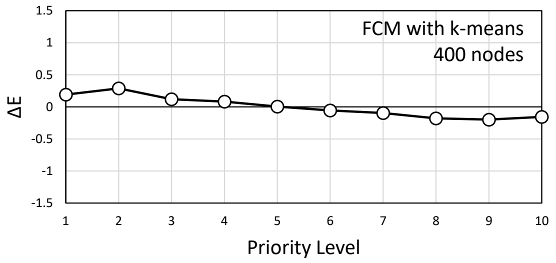


Figure 6.3: Difference in experimental error with and without prioritization.

is determined according to formula (3.2).

The high average error observed on the outer-priority levels 1 and 10 is explained by the average distance to other data points from points in these priority groups. Therefore, when data points from priority levels 1 or 10 are fused, they tend to be fused with other points further away than when points from inner-priority levels are fused. The average measurement value in our simulation is 150, so values further from 150 tend to incur more error when fused, that is, it is more likely that the value 150 can be fused with a value near to it (say, a value within a radius of 5) than it is for the value 300. Data tuples tend to fuse with values close to one another, when available, except in the case of adaptive summarization without sort.

We compare the priority case performance for the four data-collection methods in Figure 6.4.

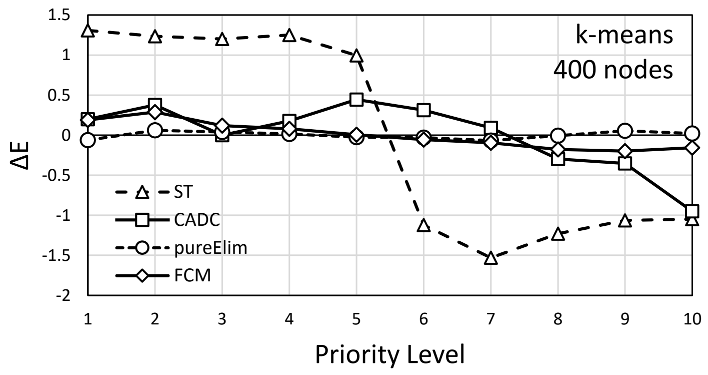


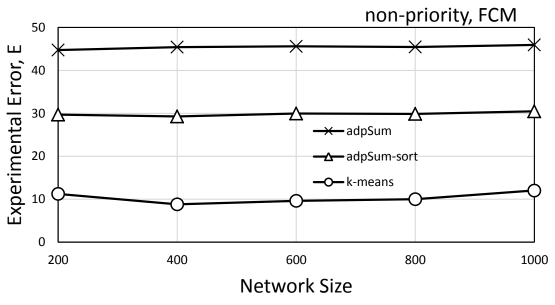
Figure 6.4: Difference in experimental error for each data-collection method using k-means fusion.

6.5 Results and Analysis

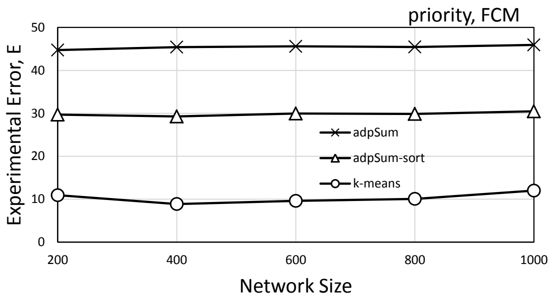
This section presents, with analysis, the key findings of the dissertation.

6.5.1 Experimental error

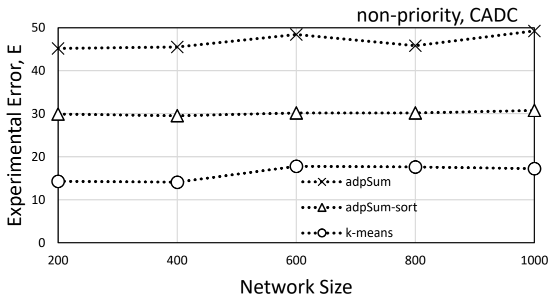
In Figure 6.5, we show how the experimental error E is affected by the fusion method for each data-collection scheme in both cases of data without prioritization and with prioritization. We note that for each data-collection scheme, the order of lowest to highest experimental error between the fusion methods is k-means, adpSum-sort, and adpSum. While every data-collection scheme performs better with k-means, pureElim has the least increase in experimental error when it uses adpSum-sort.



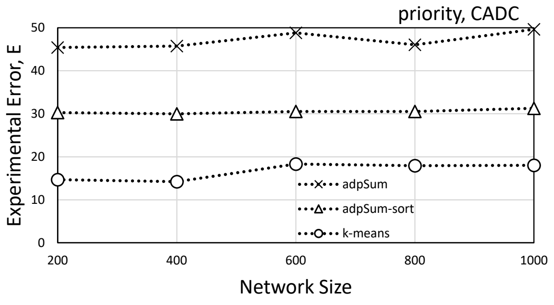
(a)



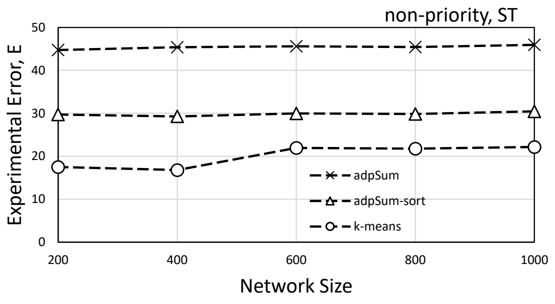
(b)



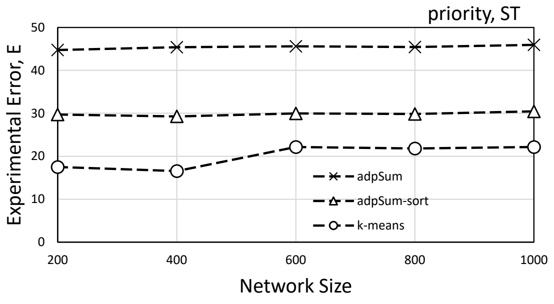
(c)



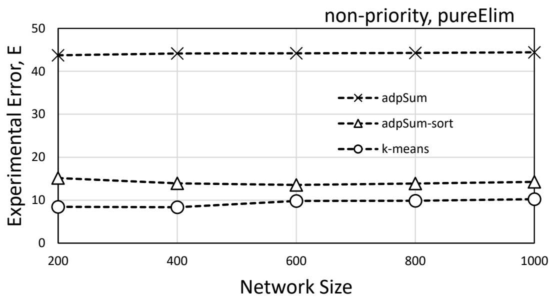
(d)



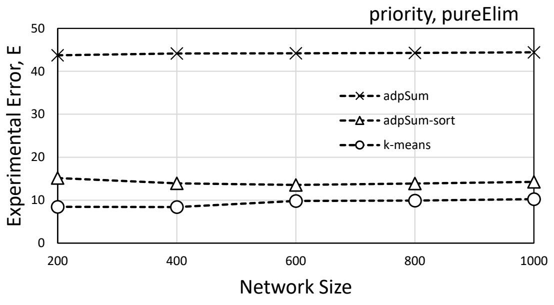
(e)



(f)



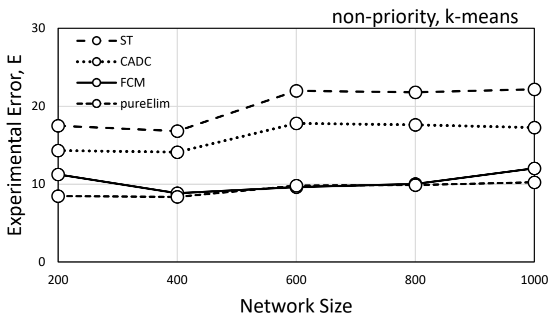
(g)



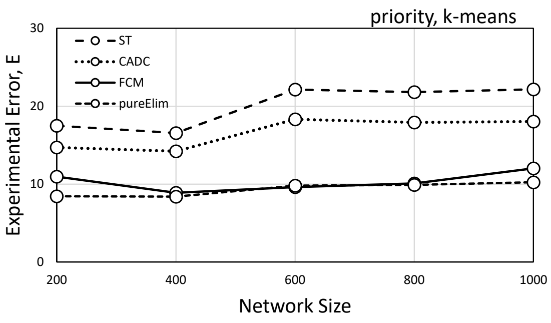
(h)

Figure 6.5: Experimental error versus network size with each fusion method.

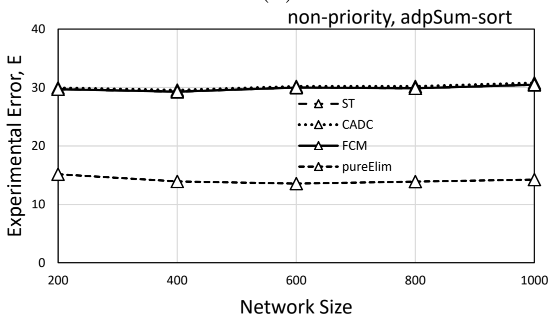
The same experimental-error results are given in Figure 6.6, where in each sub-figure, the fusion method and the choice between unprioritized and prioritized data are fixed.



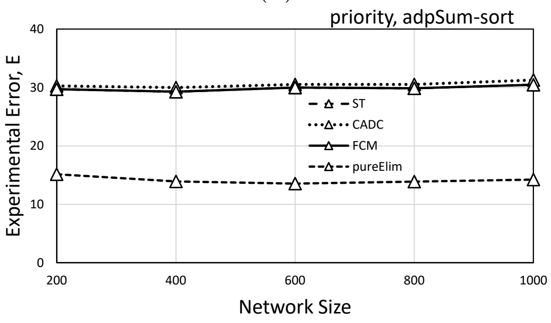
(a)



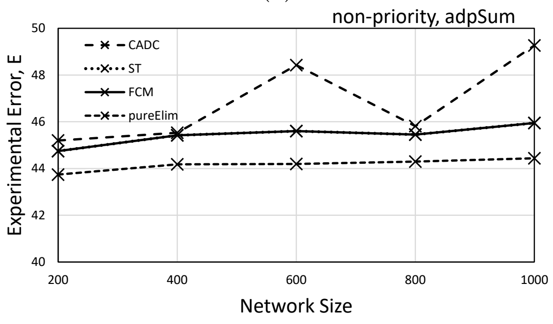
(b)



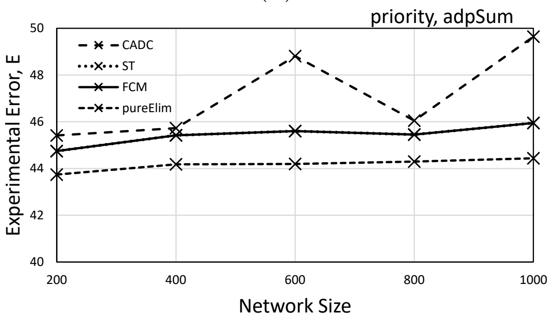
(c)



(d)



(e)



(f)

Figure 6.6: Experimental error versus network size with each data-collection scheme.

6.5.2 Data-delivery ratio

Figure 6.7 shows the data-delivery ratio *ddr* versus network size for several data-collection schemes and fusion methods. A low *ddr* is indicative of more dropped packets in the network and is a source of experimental error. While there is some

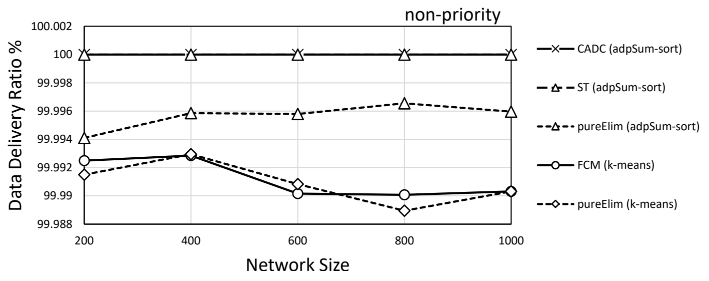


Figure 6.7: Data-delivery ratio comparison.

variation between the methods shown, *ddr* is very close to one in each case. The *ddr* is significantly lower, around 91%, for CADC paired with k-means fusion. In the CADC scheme, a node's output is regulated solely by the maximum tolerable distortion. As a consequence, when the variability in the data available to the node decreases, it may send a greater number of data tuples and cause buffer overflow at its parent's input.

6.5.3 Comparison of select methods

The pureElim and FCM methods have the lowest observed error incurred at the sink. CADC outperforms ST. The experimental error also depends on the fusion method paired with the data-collection scheme, so we compare the experimental error across several combinations of data-collection scheme and fusion methods. The results are visualized in the non-priority case in Figure 6.8 and in the priority case in Figure 6.9.

6.5.4 Sensitivity to allowable application error

The data-collection schemes CADC and FCM use constant parameters for data collection on a network to regulate the extent of allowable distortion on each node.

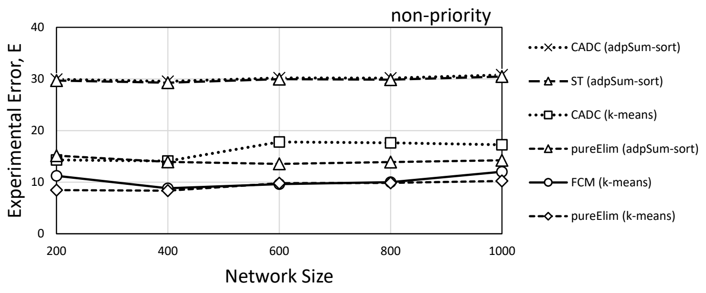


Figure 6.8: Performance comparison in the non-priority case.

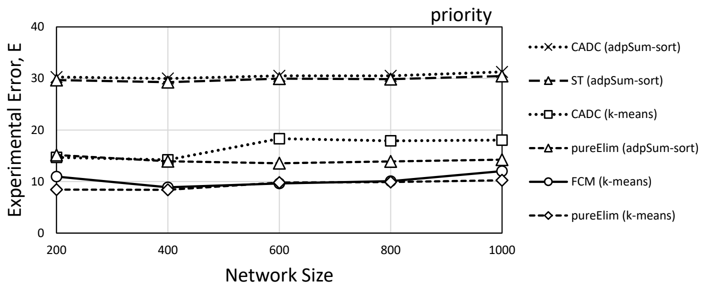


Figure 6.9: Performance comparison in the priority case.

The parameters are maximum tolerable error χ for CADC and allowable distortion ϕ for FCM. While their usage is slightly different, both parameters are comparable, in terms of scale, to the estimation error at the sink. These parameters encourage fusion on the network when the resulting distortion is within some tolerable bound given by η_u for each node u. In FCM, the allowable distortion at a node, η_u, is initialized based on ϕ according to the protocols of FCM, and it is constant for the duration of an experiment. In CADC, η_u is the congestion-control variable whose value is adjusted based on the network conditions and the value of maximum tolerable error ϵ_u and the protocols described in Chapter 5. Figure 6.10 shows the experimental error versus ϕ for FCM and χ for CADC using k-means fusion given a 600-node network with 2000 time-step experiments, which illustrates the sensitivity of the error incurred to the allowable application error (that is, χ or ϕ), and Figure 6.11 shows how the network overhead *nov* varies with the allowable application error for the same experiment.

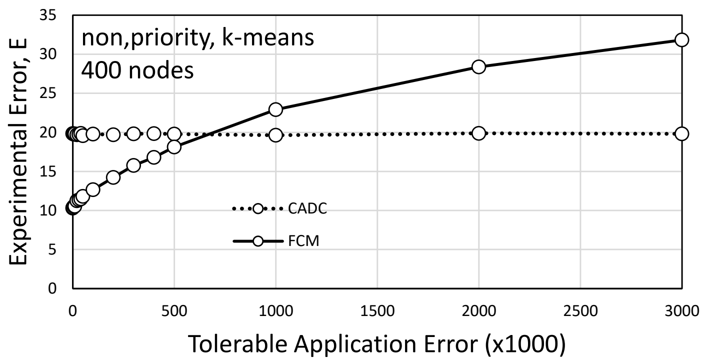


Figure 6.10: Experimental error sensitivity to allowable application error.

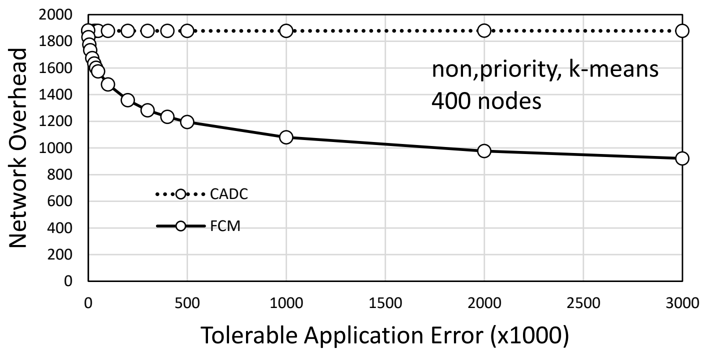


Figure 6.11: Network overhead sensitivity to allowable application error.

As expected, the network overhead decreases significantly with increasing ϵ_r in the FCM case. The experimental error is lower in the network using FCM even as the network overhead is significantly lower than in the network using CADC.

Chapter 7

Conclusion

In cyber-physical systems, estimation accuracy is critical to the monitoring of environmental phenomena, whether in wildfire detection or industrial control applications. Wireless sensor networks have many important cyber-physical system applications. We highlight wildfire detection since the threat from wildfire disasters is of increasing relevance with rising global average temperatures [3]. Early detection and accurate condition monitoring may help mitigate the loss of life and property caused by wildfires and other natural disasters. Data collection is a key feature of WSNs, but the quality of data may be compromised by poor network conditions, including congestion. Our investigation considers several data-collection schemes that use congestion control including our proposed flexible congestion management for sensor networks (FCM). We conclude the dissertation with some key insights and discussions from the development of FCM and from the other data-collection schemes.

FCM implemented on a congested WSN has advantages over the other three data-collection schemes considered in the dissertation: pureElim, ST, and CADC. Its advantages include scalability, low experimental error, low network overhead, and robustness to changes in network conditions such as network size, topology, and data variability.

We see that k-means fusion has advantages over the alternative fusion methods, adpSum-sort and adpSum. The advantages include low experimental error, better control of transmission sizes, and the ability to introduce less error for higher-priority data points. The k-means fusion method is more computationally intensive, but it is still viable in a wide range of network applications.

Each data-collection scheme discussed in the dissertation provides some valuable insight into congestion mitigation. PureElim (Chapter 3), due to its simplicity, provides a baseline performance especially suitable for static-topology networks. PureElim excels at minimizing error by fusing as little as possible at each node in the network so that better summaries can be made closer to the root. ST (Chapter 4) introduces distortion-based fusion, which allows similar data values to be fused in early phases of a round of data collection, which reduces overall network traffic. CADC (Chapter 5) provides insight into the estimation error at the sink by tracking estimated upper bounds for estimation error at each node over the course of data collection. (The bounds on estimation error are estimates because the values of dropped packets are not accounted for.)

FCM (Chapter 6) balances the advantages of delayed fusion (as in pureElim) and fusion up to a tolerable distortion (as in ST and CADC) to reduce both error and traffic on the network. The extent of early fusion can be controlled for the FCM application through the allowable-distortion parameter ϕ. In a single network with multiple concurrent data-collection applications, the relative traffic can be controlled with adjustment to the allowable distortion for each application. A decrease in the allowable distortion tends to reduce early fusion and experimental error at the cost of additional network overhead. FCM also introduces a discrete output management system (similar to a "cap and trade" system) for the packets sent from a node's children. This system, described in Section 6.2, is more appropriate for small numbers of packets than the ratio-based output limits in pureElim and the other data-collection schemes. The system is especially appropriate when the variance in the data is different from one subtree to another, since one feature of the system is to allow a child of high data variability to send more than it's initial requested limit when one or more of its siblings are able to send less than their requested limit as a result of their opportunistic low-distortion fusions.

7.1 Advantages of FCM

FCM has several features that make it a better choice than alternative data-collection schemes. The goals of our data-collection scheme are for it to eliminate congestion and minimize the experimental error incurred by data summarization. FCM accomplishes these goals while also reducing network traffic where possible to save energy or accommodate multiple applications vying for bandwidth on the network. The congestion-control mechanisms in FCM account for changes in data variability, network topology, and data priority.

The results in Section 6.5 show that FCM consistently outperforms ST and CADC in terms of experimental error. CADC's performance may suffer as network size increases since some updates require multi-hop communication sequences that reach the sink from arbitrary nodes in the network giving a worst-case $O(\log N)$ time complexity to any given update step, as compared to worst-case constant time updates in the other data-collection schemes. The relatively simple data-collection scheme PureElim results in low experimental error on static-topology networks. FCM often boasts similarly low experimental error, but, unlike pureElim, FCM is designed to account for local changes to data variability and to adjust to changes in the network topology, allowing FCM to outperform pureElim in a range of network conditions, both in reducing experimental error and in reducing network traffic.

In the case of multiple applications running on the same network, it may be appropriate to adjust the tolerable errors for the individual applications so that the less important application uses less bandwidth. An an increase in the tolerable application error tends to reduce the network overhead of the data collection for that application.

7.2 Fusion Method Choice with FCM

K-means fusion is most suitable for use with the FCM data-collection scheme. The mechanics of FCM are designed to take advantage of the precise fusion ratios allowed by k-means, as opposed to the power-of-two reductions inherent to adpSum and adpSum-sort. We observe that pureElim outperforms FCM with lower experimental error and less update overhead when the fusion method used is adpSum or adpSum-sort. Since k-means has greater requirements on node memory and processing, FCM may also be most suitable as the data-collection scheme where the nodes have enough resources to support k-means fusion.

7.3 Variations on Adaptive Summarization

In applications where k-means is unfeasible, such as when the memory or computational power of the individual nodes is limited, adaptive summarization without sort and adaptive summarization with sort are good alternative fusion-method options. These methods can be extended or modified in a few simple ways to accommodate the requirements of the application.

One way to modify adaptive summarization is to average every three points instead of every two points. This results in power-of-three reductions in the number of tuples instead of power-of-two reductions. It is possible to achieve other fixed factors of reduction with the adaptive summarization method, even fractional factors. If the reduction factor were variable, it would accommodate the precise fusion requirement of each node. Adaptive summarization with a variable reduction factor would help reduce error on the network while keeping the nodes' demands on computational resources low compared to k-means fusion.

While both adaptive-summarization methods simply average adjacent pairs of data tuple values, before or after sort, a weighted average could be taken, instead. The average could be weighted by one or both of the number of measurements represented and priority level. Weighting the average in this way helps to further reduce the estimation error resulting from the fusion.

7.4 Future Work

Future work for FCM could include the consideration of field testing, simultaneous applications, increased data dimensionality, greater network size, dynamic network topology, mobile agents, multiple sinks in the network, and unequal node capabilities. Simultaneous applications further limits the bandwidth available for transmissions, but FCM can help reduce the traffic on the less important application to free up bandwidth for the more important data, while mitigating the error significantly in both cases. Topology changes could include node failure, network expansion, and network reorganization. In our work, all nodes have the same capabilities, however, nodes closer to the sink would benefit more from increased data-storage and transmission capabilities. In mobile-node networks, the nodes could reorganize themselves to facilitate better network performance, including lower total error between measurements and their final estimates.

Another problem of interest is modeling the physical phenomenon based on the estimates that reach the sink. Each estimated value can be associated with a level of confidence. Modeling techniques such as Kalman filter [28] estimation could provide better estimates of the original measurements than the reported estimates alone.

The method of η initialization, that is, the precise allocation of ϕ throughout the network, was chosen in a way that made sense to the author, but may not be optimal. A more appropriate distribution of ϕ throughout the network may result in better performance. One alternate initialization is given here,

$$\eta_u = \phi \frac{|\mathcal{T}_u|}{S}, \tag{7.1}$$

where S is the sum of subtree sizes on the network discounting the root and leaf nodes,

$$S = \sum_{i \in \mathcal{T}_r \setminus \{r, \mathcal{L}\}} |\mathcal{T}_i|, \tag{7.2}$$

and where $\mathcal{L}$ is the set of all leaf nodes in the network.

The choice of data-collection scheme and fusion method are highly application-dependent; however, the dissertation suggests that FCM paired with k-means fusion is an excellent choice for congestion control in a wide variety of WSN applications.

Appendices

Appendix A

Random Motion

This appendix presents a method for the generation of random 1D motion using state-transition equations with Gaussian noise.

The model used to generate the random 1D motion has two parts. The first part is the *state*, defined by

$$\mathbf{x} = \begin{bmatrix} r \\ v \\ a \\ m \\ k \end{bmatrix} = \begin{bmatrix} position \\ velocity \\ acceleration \\ mass \\ time\ step \end{bmatrix}. \tag{A.1}$$

The second part is the set of *state-transition equations* with Gaussian noises added. The state-transition equations are defined in equations (A.2)-(A.6).

$$r_{t+1} = r_t + v_t \tag{A.2}$$

$$v_{t+1} = v_t + a_t \tag{A.3}$$

$$a_{t+1} = \frac{1}{m}\left(2 - \frac{2}{1+e^{-r}} - \frac{2}{1+e^{-v}} + \mathcal{N}(0,1)\right) \tag{A.4}$$

$$m_{t+1} = 5 + 3\sin(0.1k) \tag{A.5}$$

$$k_{t+1} = k_t + 1 + 0.1\mathcal{N}(0,1) \tag{A.6}$$

The position is updated according to the last known velocity. The velocity is updated according to the last known acceleration. The mass oscillates as a sinusoid between 2 and 8 with a frequency dependent on the time-step variable, k. Gaussian noise is introduced to the acceleration and to the time step. The Gaussian noise in equation (A.4) is the only source of acceleration when both position and velocity are zero. The sigmoid terms in the acceleration update equation (A.4) tend to force the object's position back to zero and to slow the object down. Without the sigmoid terms, the object's path behavior would vary wildly with time and would tend not to stay near its original position. The noise in the time step corresponds to changes in the frequency with which the mass oscillates.

Figure A.1.1 shows sample output from the implementation of the random-1D-motion generator.

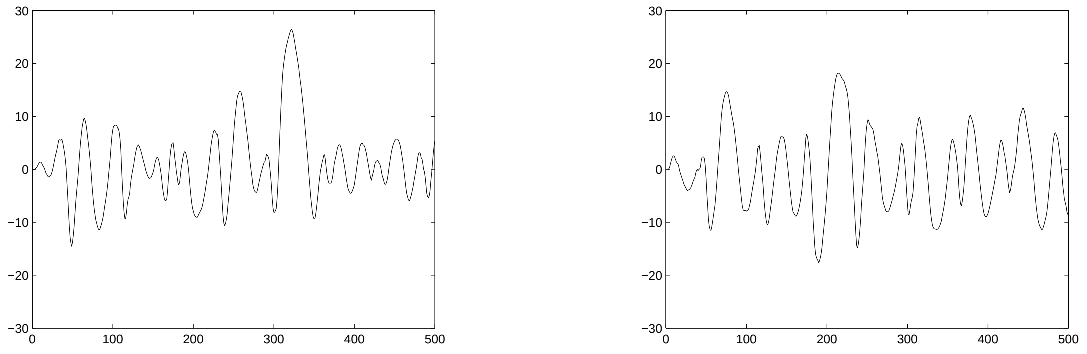


Figure A.1: Two random-motion sequences with $t = 500$ steps.

Appendix B

Proof that Sequential Distortions Sum to Total SSE

We show that the sum of successive distortions (sum of squared errors imposed by fusing points to their means) equals the total error (sum of squared errors between initial and final data). Consider an initial set of N points (in one or several dimensions) denoted by x_i for each $i \in (1, N)$. Let us assume that there are M intermediate means and $\bar{x}'_i$ is the intermediate mean that represents x_i, and $\bar{x}_i$ is the final mean that represents x_i. For simplicity, and without loss of generality, we assume that all points are represented by the same final mean. I.e., $\bar{x}_i = \bar{x}$ is a constant.

We wish to show:

$$\sum_{i=1}^{N}(x_i - \bar{x}'_i)^2 + \sum_{i=1}^{N}(\bar{x}'_i - \bar{x}_i)^2 = \sum_{i=1}^{N}(x_i - \bar{x}_i)^2$$

so, we begin by expanding terms, and we continue by canceling terms from each side.

$$\sum_{i=1}^{N}(x_i^2 - 2x_i\bar{x}_i' + \bar{x}_i'^2 + \bar{x}_i'^2 - 2\bar{x}_i'\bar{x}_i + \bar{x}_i^2) = \sum_{i=1}^{N}(x_i^2 - 2x_i\bar{x}_i + \bar{x}_i^2)$$

$$\sum_{i=1}^{N}(-2x_i\bar{x}_i' + 2\bar{x}_i'^2 - 2\bar{x}_i'\bar{x}_i) = \sum_{i=1}^{N}(-2x_i\bar{x}_i)$$

$$\sum_{i=1}^{N}(-2x_i\bar{x}_i' + 2\bar{x}_i'^2 - 2\bar{x}_i'\bar{x}_i) = -2N\bar{x}\sum_{i=1}^{N}x_i$$

$$\sum_{i=1}^{N}(x_i\bar{x}_i' - \bar{x}_i'^2 + \bar{x}_i'\bar{x}_i) = N\bar{x}\sum_{i=1}^{N}x_i$$

$$\sum_{i=1}^{N}(x_i\bar{x}_i' - \bar{x}_i'^2) + N\bar{x}\sum_{i=1}^{N}(\bar{x}_i') = N^2\bar{x}^2$$

We use the fact that the weighted average of intermediate means equals the mean, $\sum_{i=1}^{N}(\bar{x}_i') = \bar{x}$, to evaluate the second summation term, which cancels with the right-hand side.

$$\sum_{i=1}^{N}(x_i\bar{x}_i' - \bar{x}_i'^2) + N^2\bar{x}^2 = N^2\bar{x}^2$$

$$\sum_{i=1}^{N}(x_i\bar{x}_i') = \sum_{i=1}^{N}(\bar{x}_i'^2)$$

We expand the sums on each side to include one intermediate mean at a time. We let $\bar{x}_k'$ represent the kth intermediate mean with $k\epsilon\{a, b, ..., M\}$. (Note that this is a change in the meaning of the intermediate-mean subscript for the remaining lines of the proof.) We define the set S_k as the indices of the points that belong to the kth intermediate mean and let n_k be the cardinality of the set S_k.

$$n_a\bar{x}'_a \sum_{i\epsilon S_a} x_i + n_b\bar{x}'_b \sum_{i\epsilon S_b} x_i + ... + n_M\bar{x}'_M \sum_{i\epsilon S_M} x_i = n_a^2\bar{x}'^2_a + n_b^2\bar{x}'^2_b + ... + n_M^2\bar{x}'^2_M$$

We use the fact that $\sum_{i\epsilon S_k}(x_i) = n_k\bar{x}'_k$ to simplify the left-hand side.

$$n_a^2\bar{x}'^2_a + n_b^2\bar{x}'^2_b + ... + n_M^2\bar{x}'^2_M = n_a^2\bar{x}'^2_a + n_b^2\bar{x}'^2_b + ... + n_M^2\bar{x}'^2_M$$

The two sides are equal. Q.E.D.

Note that there are simpler proofs of this fact. One simplification that does not diminish the generality of the result is to allow the overall mean $\bar{x}$ to equal zero.

Appendix C

Data-Tracking Structure for Sensor-Network Simulation

This appendix details the data structure developed for WSN simulation to track the estimates of each measurement value generated in the network. Whenever there is data summarization or packet loss in the network, the data estimates change, and the structure is updated. Given the original measurement values and their estimates, it is possible to calculate error on the data set or on subsets of that data set defined by properties of interest such as initial value, origin, and data priority. The data-tracking structure is an important and powerful tool that helps us to compare and evaluate data-collection scheme performance in WSN simulation and to help make important decisions prior to network deployment, such as the hardware and software requirements of the nodes.

In simulation, we have access to a centralized view of a WSN that would be unavailable to us in the real-world deployment of the network, where we rely on the information transmission between adjacent nodes. With the communication limitation of a real-world application, it is possible to calculate estimation error by summing the errors introduced at each node, a result demonstrated in Appendix B, but this result only holds if, firstly, there is no packet loss, and secondly, the summarization of multiple data points is by their mean value. Even in the case where this result holds, it is difficult to track the error on subsets of the data (such as data sharing a particular origin or initial value). The data-tracking structure addresses these limitations during a network simulation by keeping a centralized account of all data across the network including current value and origin.

The structure contains each original measurement value and its current estimate in the network. As data points are fused, and multiple original measurements are estimated by a single value, the structure keeps track of which measurements belong to which *summary group*. Essentially, the data-tracking structure acts a disjoint-set data structure, where each row begins in a separate subset, and the subsets are joined as summary groups merge. Data fusion performed at a node prompts summary groups to merge. The data-tracking structure is designed for quick updates any time there is a change to the data estimates and summary-group membership.

The data-tracking structure is organized in a 2D or 3D array with three fundamental columns of information: the original data, the current estimates of the data, and the pointers that form cycles of grouped data. Additional columns may be added to keep track of other features of the data, such as whether a packet containing the data was dropped (in which case, the current estimate is independent of its original measurement value). The third temporal dimension to the array may be added when it is convenient to distinguish between the data-tracking which occurs in different time-steps of the simulation. The general structure is shown in Figure C.1.1.

Data is generated over time, so it is often convenient to consider the third dimension to represent the time step at which a certain batch of data is generated and collected. This gives an intuitive indexing option to distinguish data temporally. Each

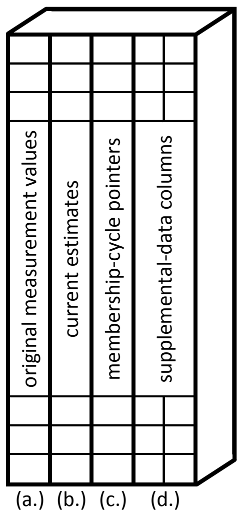

original measurement values
current estimates
membership-cycle pointers
supplemental-data columns
(a.)
(b.)
(c.)
(d.)

Figure C.1: The general form of the data-tracking structure.

row in a given time step represents a measurement and may be referenced uniquely by its index (row number) and time step. Error incurred by fusion is calculated by comparing the original data to the current estimates, and this can be done for any subset of data.

During the simulation, we maintain the structure after each change to the data. Upon initialization, the original data and current estimates match, and each point is its own predecessor (that is, its pointer goes to itself). In a simple round of summarization where only two points are fused, their current estimates are updated, and the pointer of the lower-index element is set to the higher-index element and the pointer of the higher-index element is set to the lower-index element to form a cycle. When two or more summary groups are merged, the pointers of the resulting summary group form a cycle. The pointer of the element with the smallest index points to the element of the next-smallest index and so on until the point of the element with the largest index points to the element with the smallest index. Merging the cycles in this way keeps elements sorted by index within their summary group and takes $O(n)$ time where n is the number of elements in the merging cycles. While it is possible to maintain cycles with unsorted indices, sorted indices facilitate the merger of two or more groups at once and can be helpful for error detection.

When two or more summary groups merge, the current estimate for each element in those summary groups is set to the new summary value and the pointers in the third column of the data-tracking structure are updated to form a single cycle. The cycle merger begins with a temporary pointer at the head of each cycle. The new cycle head is the element of lowest index among those pointed to by the temporary pointers. The temporary pointer for this element is incremented to the next in its original summary group. The *next element* in the new cycle is pointed to by the most recently added element to the new cycle. In each step of the merger, the next element is the lowest index among those pointed to by the temporary pointers, and that temporary pointer moves to that element's successor in the pre-merged summary group, until the cycle is exhausted (that is, the pointer would return to the first element). Once all indices in the merged group have been visited, the final pointer is set to the summary-group head to complete the merged cycle.

The data-tracking structure is useful in the simulation of WSNs to make error calculations on data subsets. Error calculations are an important metric for the evaluation of data-collection schemes, such as FCM. The data-tracking structure proposed here is a light-weight and useful tool for performance evaluation, taking advantage of the global information available in simulation. Once deployed, it is cumbersome and expensive to evaluate network data-collection schemes in the same way, so the results in simulation are important to understand the real-world network operation.

Appendix D

Abbreviations

AdpSum	adaptive summarization
AdpSum-sort	adaptive summarization with sort
CADC	congestion-adaptive data collection
ddr	data-delivery ratio
dss	desired output size
FCM	flexible congestion management for sensor networks
K-means	k-means algorithm
mbs	input-buffer capacity
mss	output-buffer capacity
nov	network overhead
PureElim	pure elimination
ST	spatio-temporal data collection
uov	update overhead
WSN	wireless sensor network

Bibliography

[1] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, March 1982.

[2] H. Ahmadi, T. F. Abdelzaher, and I. Gupta, "Congestion control for spatio-temporal data in cyber-physical systems." in *Proc. of ICCPS*, 2010.

[3] J. T. Abatzoglou and A. P. Williams, "Impact of anthropogenic climate change on wildfire across western us forests," *Proceedings of the National Academy of Sciences*, vol. 113, no. 42, pp. 11 770–11 775, 2016. [Online]. Available: https://www.pnas.org/content/113/42/11770

[4] N. Iri, L. Yu, H. Shen, and G. Caulfield, "Congestion-adaptive data collection with accuracy guarantee in cyber-physical systems," in *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, June 2015, pp. 82–90.

[5] Y. Sankarasubramaniam, O. B. Akan, and I. F. Akyildiz, "Esrt: event-to-sink reliable transport in wireless sensor networks." in *Proc. of MobiHoc*, 2003.

[6] F. Bian, S. Rangwala, and R. Govindan, "Quasi-static centralized rate allocation for sensor networks," in *Proc. of SECON*, 2007, pp. 361–370.

[7] Y. Zhou, M. R. Lyu, J. Liu, and H. Wang, "Port: A price-oriented reliable transport protocol for wireless sensor networks." in *Proc. of ISSRE*, 2005.

[8] J. Paek and R. Govindan, "Rcrt: Rate-controlled reliable transport protocol for wireless sensor networks." *TOSN*, vol. 7, no. 3, 2010.

[9] C. Y. Wan, S. B. Eisenman, and A. T. Campbell, "Coda: congestion detection and avoidance in sensor networks." in *Proc. of SenSys*, 2003.

[10] P. J. Huber, "Robust estimation of a location parameter," *Ann. Math. Statist.*, vol. 35, no. 1, pp. 73–101, 03 1964. [Online]. Available: https://doi.org/10.1214/aoms/1177703732

[11] J. Chang and L. Tassiulas, "Maximum lifetime routing in wireless sensor networks." *TON*, vol. 12, no. 4, pp. 609–619, 2004.

[12] R. Cristescu, B. Beferull-Lozano, and M. Vetterli, “On network correlated data gathering.” in *Proc. of Infocom*, 2004.

[13] H. Gupta, V. Navda, S. R. Das, and V. Chowdhary, “Efficient gathering of correlated data in sensor networks.” *TOSN*, vol. 4, no. 1, 2008.

[14] K. Han, L. Xiang, J.Luo, M. Xiao, and L. Huang, “Energy-Efficient Reliable Data Dissemination in Duty-Cycled Wireless Sensor Networks,” in *Proc. of MobiHoc*, 2013.

[15] H. Lee and A. Keshavarzian, “Towards energy-optimal and reliable data collection via collision-free scheduling in wireless sensor networks,” in *IEEE INFOCOM*, 2008.

[16] J. Park and S. Sahni, “An online heuristic for maximum lifetime routing in wireless sensor networks,” *IEEE TOC*, vol. 55, no. 8, pp. 1048–1056, 2006.

[17] C. Wang, H. Ma, Y. He, and S. Xiong, “Adaptive approximate data collection for wireless sensor networks.” *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 6, pp. 1004–1016, 2012.

[18] A. Silberstein, R. Braynard, and J. Yang, “Constraint chaining: on energy-efficient continuous monitoring in sensor networks.” in *Proc. of SIGMOD*, 2006.

[19] L. Su, Y. Gao, Y. Yang, and G. Cao, “Towards Optimal Rate Allocation for Data Aggregation in Wireless Sensor Networks,” in *Proc. of MobiHoc*, 2011.

[20] C. Luo, F. Wu, J. Sun, and C. W. Chen, “Compressive data gathering for large-scale wireless sensor networks.” in *Proc. of MobiCom*, 2009.

[21] P. J. Wan, S. C. H. Huang, L. Wang, Z. Wan, and X. Jia, “Minimum-Latency Aggregation Scheduling in Multihop Wireless Networks,” in *Proc. of MobiHoc*, 2009.

[22] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, “Clustering large graphs via the singular value decomposition,” *Machine Learning*, vol. 56, no. 1, pp. 9–33, Jul 2004.

[23] S. Har-Peled and B. Sadri, “How fast is the k-means method?” *Algorithmica*, vol. 41, no. 3, pp. 185–202, Mar 2005. [Online]. Available: https://doi.org/10.1007/s00453-004-1127-9

[24] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, “A local search approximation algorithm for k-means clustering,” *Computational Geometry*, vol. 28, no. 2, pp. 89 – 112, 2004, special Issue on the 18th Annual Symposium on Computational Geometry -

SoCG2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925772104000215

[25] H. Wang and M. Song, "Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming," *The R journal*, vol. 3, no. 2, pp. 29–33, December 2011.

[26] B. C. Dean, "Successive k-means," Jun 2018, unpublished.

[27] Y. Zhuang, L. Yu, H. Shen, W. Kolodzey, N. Iri, G. Caulfield, and S. He, "Data collection with accuracy-aware congestion control in sensor networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 5, pp. 1068–1082, May 2019.

[28] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.