Energy Efficient Compressed Sensing in Wireless Sensor Networks via Random Walk

By

Robert Brian Fletcher

A Thesis
Submitted to the Faculty of
The University of Tennessee at Chattanooga
In Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

May 2011

DEDICATION

To my parents, Ken and Jacci Fletcher, without which I could not have been successful in my academic endeavors and life in general; you guys have been the best parents and I could not imagine my life without you. To Kayla Folks, for help being my backbone in those dire times and a perfect mate always.

To the Graduate Council:

I am submitting a thesis written by Robert Brian Fletcher entitled “Energy Efficient Compressed Sensing in Wireless Sensor Networks via Random Walk”. I have examined the final copy of this thesis and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a major in Computer Science.

_____

Dr. Mina Sartipi, Chairperson

We have read this thesis and recommend its acceptance:

_____

Dr. Li Yang

_____

Dr. Yu Cao

Accepted for the Graduate Council:

_____

Dean of the Graduate School

ABSTRACT

In this paper, we explore the problem of data acquisition using compressive sensing (CS) in wireless sensor networks. Unique properties of wireless sensor networks require we minimize communication cost for efficient power usage. At first, a compressive distributed sensing (CDS) algorithm is proposed but is then modified to decrease communication costs. The final algorithm presented is compressive distributed sensing with random walk CDS(RW); an algorithm that combines the data gathering and projection generation process of CDS.CDS(RW) uses rateless encoding, graph algorithms, and belief propagation decoding to improve upon the communication cost associated with CDS. In the end, we show that the communication cost of CDS(RW) versus existing CS algorithms is far superior, while still having satisfactory decoding accuracy.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Wireless sensor networks (WSNs) are being used in a broad variety of domains including scientific [1], medical [2], commercial and military applications [3]. There are a wide variety of applications for WSNs including environmental monitoring, overseeing of smart homes and offices, and managing of intelligent transportation systems. A key feature of all WSNs is that the individual wireless sensors are collecting data. This data can range from simple temperature readings in a smart home, to seismic vibration readings relating to earthquakes in an extreme outdoor environment. With the advanced state of wireless technology and computers in general, scientists are able to use WSNs to make data gathering much more robust and efficient. A typical WSN consists of *N* sensors that are constantly taking measurements of their environment or surroundings. The sensors in WSNs are generally low-processing and low-power units which means special attention must be paid to the processing and energy consumption; some common WSN platforms include National Instruments NI sensors [4] and Carnegie-Mellon's FireFly sensors [5]. WSNs have advantages over manually taking measurements for many reasons, but the top is that WSNs allow for placement of sensors in remote areas that would be otherwise difficult to reach for repeated measurements. Also, WSNs allow for continually monitoring of an area at a rate much higher than would be possible by sending a person to physically gather data from the site. For example, Harvard has a sensor network lab that is responsible for monitoring volcanic activity [6]. A visualization of Harvard's volcanic WSN can be seen in Figure 1 and aims to provide a global view of data acquisition in their WSN.

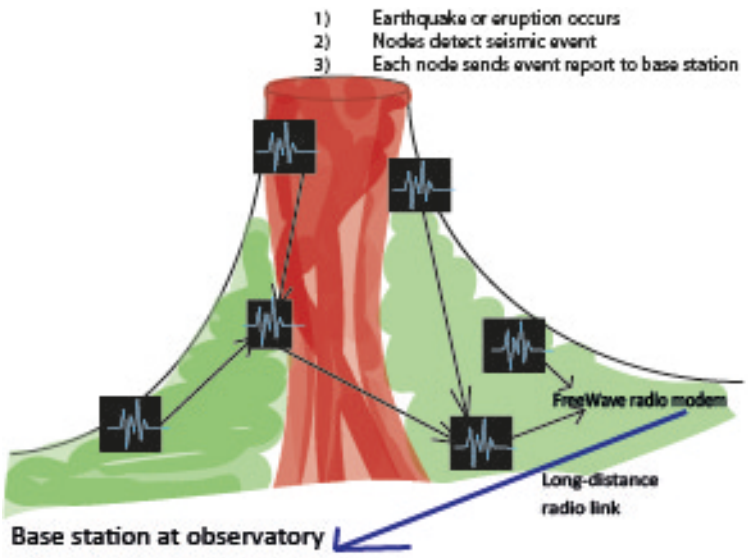

Figure 1: Harvard's Volcanic WSN [6].

In Harvard's volcanic WSN, sensors are strategically placed over the volcano. These nodes constantly take seismic measurements in an effort to be able to detect eruptions before they happen. These sensors continually send their data via neighboring sensors to, what is labeled in this image as, the FreeWave Radio Modem. The FreeWave Radio Modem in the general case is considered the sink node and is responsible for aggregating the data on the WSN so that either the data can be recovered directly from the sink node or the sink node can send data along to the base station for recovery. Constantly monitoring and transmitting data to the base station can put a burden on the sensors with limited power resources and therefore we need to minimize sensor transmissions. Another aspect of most WSNs that can be exploited is the fact that it is highly probable that sensors within a certain radius of each other will be collecting highly correlated data readings that can be exploited later on in an effort to gain information on all sensor readings without having to explicitly get data from each sensor.

Inherent limitations of WSNs mean that data acquisition algorithms must be developed in such a way that we limit processing and energy consumption. As seen in the example of Harvard's volcanic WSN, there are a large amount of intercommunications between sensors when propagating data across the network. These communications are likely the most expensive operation of a data acquisition algorithm when concerned with energy consumption and therefore need to be used intelligently. For example, if we can have each sensor send along a compressed signal that contains more than just a single data point, then we can increase the information to energy consumption ratio. However, using compressed signals comes at the cost of increased processing because of the mathematical overhead involved in the process of compressing data. In order to minimize processing and justify the use of compression the encoding and decoding complexity of the compression algorithms must shown to be low.

If it is our goal to recovery the *N* sensor readings while minimizing communication costs, we should then look to exploit the physical proximity of sensors that are taking almost identical data readings. Meaning, why should we require all N sensors to provide their data when their readings are highly correlated? In this work, we exploit the physical proximity of sensors and the highly correlated data readings of such sensors using compressive sensing (CS). The key advantage of compressive sensing is the ability to recover all N sensor readings, that are correlated, with M weighted linear combinations, also called projections, such that $M<<N$. Compressive sensing allows us to fully recover all sensor readings, within some error value, while greatly decreasing communication costs. However, current compressive sensing algorithms have the drawbacks of communication costs and encoding/decoding algorithm complexity that make their use on WSNs costly.

In this work we explore two unique algorithms that use compressive sensing combined with rateless coding for data acquisition on WSNs. Rateless coding provides a much lower complexity as compared to the coding algorithms currently used in compressive sensing algorithms; for example the joint entropy coding approach by [7]. The current algorithms that exist for data acquisition land in three main domains: conventional compression like the joint entropy coding [7] mentioned briefly above, distributed source coding like the Slepian-Wolf coding theory [8], and compressive sensing via Candès et al. [9]. The first new algorithm proposed during this work, compressed distributed sensing (CDS) involves generating *M* projections by having individual nodes send their data along to a sink node to have data linearly combined, which in turn then sends the data to the base station. CDS quickly showed that it could recover the original *N* sensor readings with high probability, but the communication costs made the applicability to WSNs not feasible. CDS was then modified to combine the projection creation and combination processes into one by using random walk on the network; the new algorithm is entitled compressed distributed sensing with random walk (CDS(RW)). By combing the two separate steps in CDS, CDS(RW) was able to greatly decrease communication costs while still recovering all *N* sensor readings with high probability. Both algorithms are independent of network topology and therefore the algorithms adapt to any unsuspected sensor failures and do not depend on a static route to the sink and/or base station.

CHAPTER 2: LITERATURE AND ALGORITHM REVIEW

Data Acquisition Algorithms

The most basic idea behind this work is that we have a WSN that is continually taking some form of measurements and we need to collect these data measurements for analysis. We also assume that the data being collected is spatially correlated because of the physical proximity of sensors, thus the data being gathered is highly correlated. The simplest way to collect this data would be to continually save every data measurement from every sensor by having each sensor send its data along to the base station; as one can imagine this quickly becomes too expensive in regards to memory, processing, and power when applied to wireless sensors. In order to try and minimize communication costs, we analyze several different approaches for data acquisition on WSNs.

Conventional Compression

Conventional compression has several aliases including source coding, data compression, and bit-rate reduction. The conventional compression algorithm examined in [7] explores compression via joint entropy coding. As with other conventional compression algorithms, [7] requires communication between nodes and exploits correlated data in the compression process. For visualization of joint entropy encoding examine Figure 2 below.

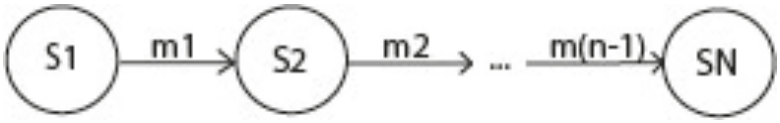

Figure 2: Joint entropy encoding visualization [10].

Joint entropy encoding begins by node s_1 encoding its data, d_1, into message m_1 using a total of $H(d_1)$ bits, where $H(d_1)$ is defined to be the entropy of the encoded data d_1

[10]. Entropy is the metric used for gauging uncertainty in a system; if entropy is high, more information is needed to fully describe that system. From there node s_2 receives message m_1 and encodes its data, d_2, with m_1 using a total of $H(d_1|d_2)$ bits, where $H(d_1|d_2)$ is the conditional entropy[27]. The joint entropy encoding continues along this path until the sink node is reached and all data as been encoded. It is shown in [10] that the key downfall with joint entropy encoding is the large number of complex computations necessary by individual sensors in the network. These complex computations increase processing and power consumption and are therefore not a good fit for WSNs.

Distributed Source Coding

Distributed source coding is based on the idea of reducing the complex computations required by the individual sensors of the network and exploiting correlated data at the sink node; two distributed source coding algorithms can be found in [11] and [12]. Both [11] and [12] are based on the Slepian-Wolf [8] coding theory that states the compression of correlated data that is encoded separately, can perform as well as data that is jointly encoded as long the data is jointly decoded [8]. These distributed source coding algorithms exploit the Slepian-Wolf theory to switch the computational complexity from the encoding of data at the sensors to the decoding of data at the base station. This switching of complexity from encoding to decoding is extremely applicable to WSNs as the individual sensors often times do not have the processing power necessary for complex coding, but the base station is usually more capable. The specific down fall of distributed source coding algorithms is that they predefine certain data to be main data and other data to be side data. This means in the case of an abnormal event, such as a sensor failure, that includes loss of main data there will be a significant negative affect on decoding accuracy. Furthermore, it is

shown in [10] that these decoding inaccuracies will also be propagated through the decoding process of other sensor readings at the base station – thus also decreasing the accuracy of other decoded data.

Compressive Sensing

In the end compressive sensing was chosen for its low complexity at the individual sensors, its ability to exploit correlated data, and the ability of compressive sensing to deal with network abnormalities elegantly. The specific algorithms in this work, CDS and CDS(RW), use compressive sensing with variations on rateless codes for encoding and belief propagation for decoding. The crux of compressive sensing, beyond the aforementioned advantages, is the ability to recover an N length signal that is sparse with M projections, such that $M << N$. Compressive sensing proves, as shown in upcoming sections, to be a good fit for WSNs regarding communication cost and the ability to exploit the correlated data in the network.

Compressive sensing is based on the observation that a sparse signal $s \in \mathbb{R}^N$ can be recovered from a small number of linear projections onto a second basis that is incoherent with the first basis [13]. In this situation, that means we can recover all N readings, with high probability, given only M projections. The mathematical definition for compressive sensing is a signal in a sparse basis induced by vectors $\{\psi_i\}_{i=1}^{N}$ or the sparsity matrix, Ψ, is represented as $s = \sum_1^n x_i \psi_i$ or $s = \Psi \mathrm{x}$ is called K-sparse if only K of the x_i are non-zero and $K << N$ [13]. The definition of compressive sensing also dictates that for a N length vector, s, with a K-sparse signal, that s can be recovered in $M = \mathrm{O}(K \log(N))$ random linear projections of *s* at the base station. These projections can be represented as $y = \Phi s = \Phi \Psi x$, where Φ is the $M \times N$ measurement matrix. As is briefly mentioned above it

is necessary to minimize incoherency between the sparsity matrix Ψ and the measurement matrix Φ in order to recover the sensor readings error free at the base station. A key feature that is different in most CS algorithms is the generation of a measurement matrix Φ for encoding. For that reason, we now identify and examine several different CS algorithms to gain insight on how they generate the measurement basis.

Independent and identical Gaussian and Bernoulli vectors provide a sufficient basis for decoding in compressive sensing [14][15]. Both the Bernoulli and Gaussian method for generating measurement matrices, Φ, are considered dense random projections (DRPs) since a majority of the elements in Φ are non-zero. However, this means for CS using DRPs that the base station must receive a significant amount of data measurements in order to recover that data because the basis is not sparse; the dense nature of the signal causes communication costs to soar.

Apparently the key downfall to generating a measurement basis using DRPs is that the inherent density of the signals means the BS station must receive a disproportionate amount of information in order for successful recover of the signal. Wang et al [16] proves that an intelligently designed sparse random projection (SRP) can reduce communication costs and perform as well, if not better, than DRP when generating a measurement basis for CS.

The final CS algorithm examined is proposed by [10] and goes about compressive sensing by first building a tree structure of the network that contains the shortest path from all nodes to the sink node. A visualization of the compressive data gathering (CDG) proposed by [10] can be seen in Figure 3 below.

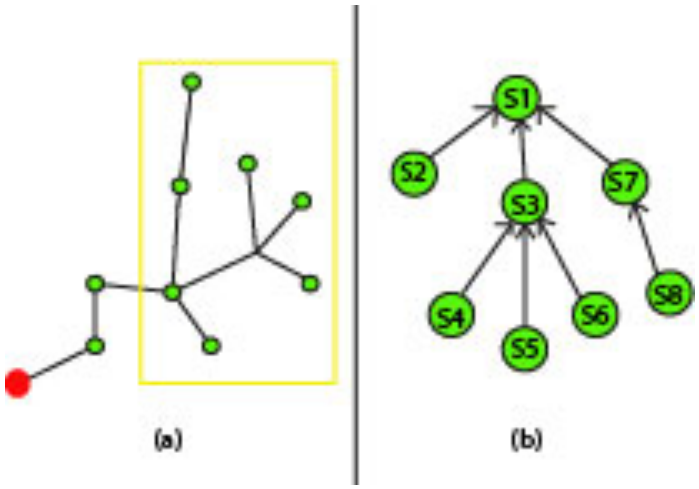

Figure 3: Visualization of CDG algorithm using a tree structure to represent shortest paths [10].

In Figure 3(a) we see the topology of a WSN where the sink node is located in the center and has four neighbors; the dotted lines separating each neighbor of the sink represent the subtrees of the sink node. In Figure 3(b) we see the shortest path tree created by CDG corresponding to the boxed region in Figure 3(a); the root of the tree in Figure 3(b) corresponds to the node closest to the sink, while still within the boxed region, in Figure 3(a). Essentially, CDG starts at each of the leaf nodes and continues along the shortest path to the sink while adding its value, multiplied by a random weight, to the weighted sum it just received. In the end, the sink node contains a weight sum of the subtrees and uses those values as a measurement basis for CS. While the measurement basis, Φ, generated by CDG performs well for decoding the inherently dense nature [10] of Φ means that it suffers from the same communication cost downfalls DRPs.

Regardless which CS algorithm is used, be it SRP, DRP, or [10], general CS theory requires $M = O(K \log(N))$ projections to the base station to recover the original signal with high probability. Given the definition of M, the communication cost for SRP, DRP and [10] are given in Table 1.

Algorithm	DRP	SRP	[10]
Cost	$O((K*N)logN)$	$O((K*d)log^2N)$	$O((K*N)logN)$

Table 1: Communication cost of different CS algorithms.

In Table 1, we analyze the bit-hop communication cost associated with a network of size N with a diameter of d hops and an average distance of nodes from the sink node of $O(d)$ hops [17].

CHAPTER 3: METHODOLOGY

Compressive Distributed Sensing

The first algorithm proposed, compressed distributed sensing (CDS), looks to use compressive sensing for data acquisition on WSNs. The goal of CDS is to use a combination of rateless encoding and belief propagation decoding to accomplish several improvements:

1. Use CS to exploit spatial correlation of data.
2. Use simple encoding and decoding algorithms.
3. Lower communication cost as compared to other CS algorithms

As discussed in detail in Chapter 2, the key feature that is unique to each CS algorithm is how the measurement basis is generated for later decoding at the base station. CS theory also dictates that we need only have M projections, such that $M \ll N$, to be able to fully recover a sparse signal at the base station. In order to understand and analyze communication costs of CDS, we must first analyze the encoding and decoding algorithms used.

Compressive distributed sensing (CDS)- CS using rateless code

Rateless coding is a relatively new class of algorithms that allows for linear encoding complexity – a major enhancement over most coding algorithms. Rateless coding involves each receiver continuously reading encoded data until the receiver is able to successfully decode that data. While rateless coding is a relatively new class of codes, there already exist a few different algorithms that include LT codes [18], raptor codes [19] and Online codes [20].

The mathematics behind rateless codes define there be a degree distribution $\Omega(x) = \sum_{i=1}^{N} w_i \, x^i$ where w_i is the probability that degree i is chosen. The encoding process involves generating an independent and random packet by sampling the degree distribution $\Omega(x)$ to obtain a weight *w* between 1 and N. After packet generation, a vector $V = (v_1, \ldots, v_n)$ with a weight of *w* is chosen at random. Finally, the value of the encoded symbol is calculated as $\sum_{i=1}^{N} v_i \, m_i$. In this work we propose to use rateless encoding for generating the projections. The visualization of our proposed algorithm is shown in Figure 4. Similar to rateless encoding, node A chooses a degree i from the degree distribution $\Omega(x)$. Node A then chooses i nodes from the network completely at random. The projection generated at node A consists of the reading of Node A and the reading of the i chosen nodes.

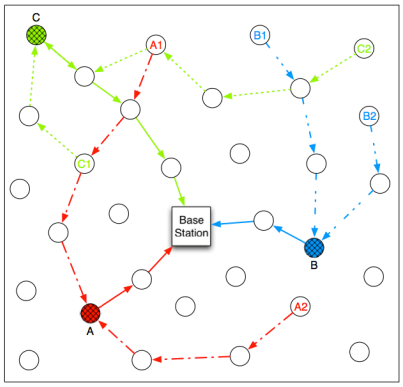

Figure 4: CS using rateless coding (CDS).

In Figure 4 we show the process of generating three projections at nodes A, B, and C. For example, node A has chosen degree 2. The nodes that contribute to node A's projection are marked as A_1 and A_2 respectively. The first step involves A requesting $A_1$and A_2 send along their data readings. From there, A_1 and A_2 send their readings along the shortest path to A - these routes are marked with dashed lines. Once all the required data readings have reached node A, A will generate the projection with the collected readings and send the projection via the shortest path to the base station.

Decoding – Belief Propagation

The decoding algorithm for CS used in this work is belief propagation decoding and is studied extensively in [21] and [22]. The belief propagation algorithm is applied to a bipartite graph with sensor nodes on one side and measurement nodes on the other. The key difference between our BP decoding algorithm and that of [22] is that:

- [22] defines BP to use a fixed degree for all measurements throughout simulations and finds the optimum value for $\Omega(\mathrm{x}) = \omega$. Our BP decoding algorithm on the other hand uses the fact that an irregular code outperforms a regular one [23], thus using a non-constant degree for $\Omega(\mathrm{x})$.

The goal of belief propagation is to approximate the marginal distribution of coefficient and state variables in a bipartite graph based on some measurement Y. The process involves iteratively passing messages between a sensor node n and one if its neighboring measurement nodes m; a message sent from node n to node m is denoted as $\mu_{n\to m}$, and a message sent from node m to node n is denoted as $\mu_{m\to n}$. The iterative process of sending messages will be repeated until the maximum iteration, maxIter, value is reached. This BP algorithm is summarized in Algorithm 1 below.

Algorithm 1 *BP Decoding Algorithm*

Require: neigh, maxIter, ϵ, and pdf-prior

$\mu_{n\to m}$ = pdf-prior

while iter < maxIter **do**

$\mu_{m\to n} = \sum_{\sim\{n\}} \left(\text{con (neigh(m))} \prod_{\omega \,\varepsilon\, \text{neigh}(m)\backslash\{n\}} \mu_{\omega\to m} \right)$

$\mu_{n\to m} = \prod_{\omega \,\varepsilon\, \text{neigh}(n)\backslash\{m\}} \mu_{\omega\to n}$

$f(n) = \prod_{\omega \,\varepsilon\, \text{neigh}(n)} \mu_{\omega\to n}$

$\hat{x} = \arg\max f(n)$

iter++

end while

Algorithm 1: Belief Propagation algorithm.

In Algorithm 1 above, $neigh(n)$ and $neigh(m)$ denote neighbors of sensor and measurements nodes on the network; $\sim\{n\}$ is the set of neighbors of m excluding n, $con(neigh(m))$ is the constraint on the set of sensor nodes $neigh(m)$. For CDS the constraint just described is defined to be $\sum_{i\in neigh(m)} x_i = y_m$ where $y_m$is the m^{th} coefficient of measurement y because of the process used to generate measurements; for example, in Figure 4 node A is connected to nodes A_1 and A_2.

In Algorithm 1, the first step is for *pdf-prior* to initialize the message sent from sensor nodes to the projection nodes. For sparse signals X, a large number of its coefficients are small valued and a small number of its coefficients are large valued. In order to accurately model this behavior, we have two probability functions - the probability mass function (*pmf*) and the probability distribution function (*pdf*).

The probability mass function (*pmf*) of state variable $Q_i = 1$ or $Q_i = 0$, represents the probability that x_i has either a large or small coefficient. In order to ensure we have a K-sparse signal, we must satisfy:

$$\Pr(Q_i = 1) = \frac{K}{N} \quad \text{and} \quad \Pr(Q_i = 0) = 1 - \frac{K}{N}$$

The probability distribution function (*pdf*) of $f(x_i|Q_i = 1)$ and $f(x_i|Q_i = 0)$ models the small and large coefficient with zero mean Gaussian distributions with high and low variances [13] must satisfy:

$$f(x_i|Q_i = 1)\sim\mathcal{N}(0, \sigma_1^2) \quad \text{and} \quad f(x_i|Q_i = 0)\sim\mathcal{N}(0, \sigma_0^2) \quad \text{where } \sigma_1 > \sigma_0.$$

This model is called a mixture Gaussian model and is widely used [22]. Using the definition from above, Algorithm 1 calculates *pdf-prior* as:

$$\textit{pdf-prior} = \frac{K}{N}\mathcal{N}(0, \sigma_0) + \left(1 - \frac{K}{N}\right)\mathcal{N}(0, \sigma_1).$$

Encoding - Rateless Code

As introduced in the rateless coding section, a key step in the encoding and projection generation process is when a node randomly selects a weight from the degree distribution $\Omega(x)$. Depending how the degree distribution $\Omega(x)$ is designed, this will result in either a dense or sparse random projection scheme [13]. While a dense projection scheme allows for more complete recovery of a sparse signal at the base station compared to a sparse projection scheme, the associated communication cost and computational complexity of dense projections is high. For this reason, it is our goal to intelligently design the degree distribution $\Omega(x)$ such that only sparse projections are generated. In order to design the degree distribution such that it generates sparse projections, CDS uses the parallel channel scheme proposed in [13] that ensures we generate a degree distribution $\Omega(x)$ for rateless coding such that the projections are sparse, thus minimizing communication costs.

The communication cost for CDS is: $O(K\log N(\mathrm{d}\omega + \mathrm{d}))$ where ω is the average row weight of the measurement matrix Φ, $\mathrm{d}\omega$ is the cost to generate each projection, and d is the cost to send the projection from the sink node to the base station. The communication cost downfall is because the gathering of data and the creation of projection are two distinctly separate processes. In order to lower communication costs we modify the way CDS creates projections to use random walk, CDS(RW).

Random Walk

Random walk on a network is the basic procedure of “walking” the network via neighboring nodes to create some path. Random walk can be used for myriad of different

applications on networks and graphs, but it is our goal to “walk” the topology of the network and collect data points as we go. The more technical definition of random walk is the sequence of selecting nodes such that the next node *j* in the path is selected from all of the previous node’s, *i*, neighbors. Random walks are Markov, since they are only dependent on the current state. The transition matrix $\boldsymbol{P} = [p_{ij}]$ is defined by the random walk next hop probability of p_{ij} and is dependent on the algorithm used. The probability that a random walk will end on node *i* is represented by the stationary distribution $\pi = [\pi_1, \dots, \pi_N]$ where π_i represents the probability that the random walk will finish at node *i*. The definition of mixing time, τ, is the number of iterations necessary for the stationary distribution to converge to a uniform stationary distribution; meaning the probability of ending on a node is the same for all nodes.

Existing Algorithms

There are several random walk algorithms available for use, a few of which include normal random walk (NRW), Maximum-Degree random walk (MDRW) and Metropolis-Hastings random walk (MHRW). Each of these algorithms offers specific advantages/disadvantages over alternates, and care must be taken to choose the algorithm that best fits our situation.

The normal random walk is the most basic algorithm that simply treats all neighbors of a node with equal probability of transition. If every node in the network/graph has the same number of neighbors, then NRW is shown to converge to the uniform stationary distribution [24], [25]. However, in our situation we do not have a-priori knowledge of the network topology and thus we are not guaranteed that every node will have the same number of neighbors.

The Maximum-Degree random walk does not treat every neighbor of a node equally. Rather, MDRW associates a maximum degree variable with each node that represent the number of neighbors that node has. Then when deciding which node is next, MDRW simply choses the neighbor with the high maximum degree variable. While this is a sophisticated algorithm that could be successfully applied to WSN and CS, there is another existing algorithm that takes MDRW a step further; that algorithm is Metropolis-Hastings random walk and is used throughout this work as the random walk algorithm.

Metropolis-Hastings

In order to disseminate data across the network, CDS(RW) uses the Metropolis-Hastings random walk algorithm and treats the network as an undirected acyclic graph. The mixing time of a graph, τ, is non-technically described as the number of hops necessary to cover the entire network with high probability. An essential step in the simulation process is to determine given a graph G of size N, what is the mixing time of that graph regardless of topology. Once we can successfully determine the mixing time of a graph, we know how far to travel in order to make sure, with high probability, that we have covered the entire network topology and gathered all pertinent data before sending data along to base station. Metropolis-Hastings defines the transition matrix $\boldsymbol{P} = [p_{ij}]$ to be:

$$p_{ij} = \begin{cases} \dfrac{1}{\max(|\mathcal{N}(i)|, |\mathcal{N}(j)|)} & if \quad j \epsilon \mathcal{N}(i) \\ 1 - \sum_{j \epsilon \mathcal{N}(i)} p_{ij}, & if \quad j = i \\ 0, & otherwise \end{cases}$$

For our situation it is necessary to randomly initialize the stationary distribution vector such that *M* of the *N* values in $\pi = [\pi_1, \dots, \pi_N]$ are set to M/N; this is because CS dictates we eventually need *M* projections to fully recover the signal.

The Metropolis-Hastings random walk algorithm is still Markov, but takes into consideration the degree of itself and the degree of its neighbors as well. Having more knowledge at its disposal, the Metropolis-Hastings random walk is better able to choose paths that result in a lowered mixing time.

A key factor later down the line is the need to know mixing time, τ, for a graph given its size *N*. This is important because we want to know how many hops, given a network of size N, is necessary to cover the entire network with high probability. To answer that question we ran the Metropolis-Hastings algorithm for different network topologies and for different network sizes; the results of the simulations can be seen in Table 2 and Table 3.

N	Grid		N	Random
50	20		50	6
100	24		100	6
200	91		200	5
300	111		300	5

Table 2: Metropolis-Hastings mixing time results for grid and random topologies.

N	Mixing Time
50	13
100	15
200	48
300	58

Table 3: Average Metropolis-Hastings mixing time results for network and random topologies.

Compressive Distributed Sensing with Random Walk

Compressed distributed sensing with random walk CDS(RW) combines the projection generation and data collection process, which was shown to be the downfall in

the original CDS algorithm. The key difference between CDS and CDS(RW) is that instead of having data points independently send their data to a collection node, we simply walk around the graph, collecting data as we go. However, it is not simple enough to walk the minimum distance required to gather the necessary each data points, say q hops. If we simply walk *q* hops and collect that data, it is fairly obvious we will not have enough information to recover the global data of the graph since we will be restricted to a *q*-hop radius from our first node. The mixing time of a graph, τ, is the number of hops necessary to cover the entire topology of the graph with high probability. In order to ensure that we cover a satisfactory amount of the network, we must travel a minimum τ hops and collect *q* data points. If $\tau > q$, then we will skip $ceil(\tau/q)$ nodes between each data point collected. Once we have arrived at the *qth* data point to be collected, CDS(RW) sends along all the collected information to the base station for decoding. It is important to note that rateless coding, all be it a modified rateless coding algorithm, and belief propagation were also used for CDS(RW) just as in CDS; details in the CDS section above. A visualization of CDS(RW) can be seen in Figure 6 below.

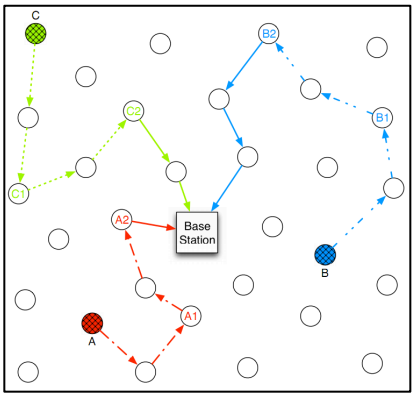

Figure 5: Visualization of CDS using random walk (CDS(RW)).

In Figure 6, we can see nodes A, B, C initialize projections to the base stations. Examining the projection generated by node A, we see the contributing data points

correspond to data readings at nodes A_1 and A_2. CDS(RW) starts by randomly walking the network at node A. A key feature to note is the skipping of nodes data measurements; this corresponds to the mixing time being longer than the number of required measurements for projection generation. Therefore, nodes must be skipped in between each data measurement so that we ensure coverage of the entire network with high probability. Once the CDS(RW) associated with node A has reached the final node contributing a data measurement, considered the sink node, the projection is generated and sent along the shortest path to the base station. As long as we travel at least the length of the mixing time for a graph, CDS(RW) consistently outperforms CDS from a communication standpoint and is satisfactory in error recovery; analysis is performed in the next section.

Generating Networks for Simulation

A clear step in the simulation process is the need to generate maps of networks to perform simulations on. In this paper we considered both grid networks and random networks. The grid network generation, as seen in Figure 7 was a simple static algorithm that was comprised of a while loop and a few if statements. However, the random network generation proved to be much more complicated.

For the simulation process, the networks were represented as undirected, un-weighted graphs via adjacency lists. For a given graph $G = (V, E)$ is made up of set of *V* vertices and a set *E* of edges that connect these verticies. Adjacency lists represent graphs as an individual node with an associated list of neighbors.

This contrasts representing graphs as adjacency matrices, where a given graph *G* of size *N* is represented as an *NxN* Boolean valued matrix $M = (m_{ij})$ in which entry m_{ij} is TRUE if there is an edge connecting vertices *i* and *j* , and FALSE otherwise. While an

adjacency matrix representation of networks is certainly more intuitive and straightforward to program, scalability problems relating to memory allocation quickly became apparent. Representing the network as an adjacency list results in much smaller memory requirements but also complicates operations like matrix multiplication and matrix algebra.

Grid Topology

In order to ensure the accuracy of our simulations, regular networks and random networks needed to be generated and run through all the same simulations; this made sure the results were consistent regardless of network topology. A regular network is traditionally defined as any network/graph such that all nodes have the same number of neighbors and a minimum spanning tree exists. Regular networks help the distribution of probabilities across the network when performing simulations and guarantee good connectivity. While not completely regular, for our purposes we decided to use a grid graph. In a grid graph all internal nodes have same connectedness and it is only the outer border nodes that cause the graph to not be completely regular. An example of the grid network structure generated for our simulations can be found in Figure 7 below.

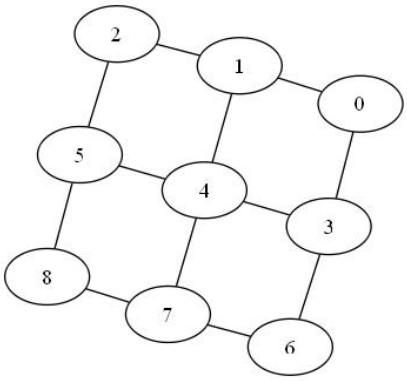

Figure 6: Example of grid network generated

Random Topology

Generating the random graphs was a little more complicated as it is necessary to generate graphs that have high connectivity even though they are not regular. The method to generate the maps was to choose a radius *r* that ensured connectivity and then generate *N* random *(x, y)* points that represent the traditional Euclidean tuple. Each point then calculates the Euclidean distance to all other points, and any point with a distance that is less than or equal to the radius *r* is determined to be a neighbor. It was important to minimize the number of orphan sections on the graph, so it was necessary to vary the radius until we reached the desired connectivity described above.

The radius used in generating random networks was a key issue that majorly affects the connectivity of the graphs. The connectivity of the graphs for CDS and CDS(RW) becomes important and can greatly affect performance in the end. In order to minimize the affect connectivity had on simulation results, we simply aimed to generate random graphs with approximately the same connectivity as a grid graph of similar size. Connectivity is formally defined as the average number of neighbors per node in a network. An example of a random graph generated can be seen in Figure 8:

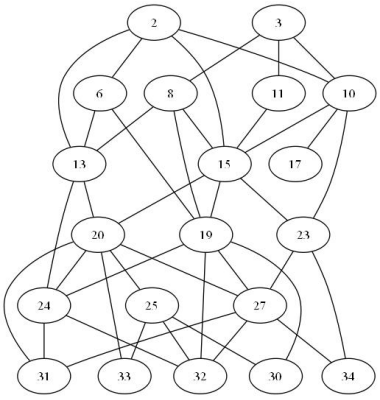

Figure 7: Example of random network generated

CHAPTER 4: ANALYSIS OF RESULTS

In the section on CS, in Chapter 2, we compared the cost of CDS with that of existing algorithms for CS – specifically, DRP, SRP, and [10]. It was shown that CDS has the lowest cost when all algorithms require $M = O(K \log(N))$ projections to recover the original signal with high probability. In this section we focus on comparing the costs between CDS and CDS(RW) as we have already shown that CDS is more efficient compared to existing CS algorithms.

In order to analyze CDS and CDS(RW) we must not only examine the projected communication costs but must also compare the error percentage recovered in decoding so we can ensure that data is recovered with high probability. In compressive sensing the main metric for analysis is the ability to recovery a sparse signal within some error percentage. For that reason we ensure that the projections created generate an error percentage that is acceptable to existing algorithms. As covered in depth up to this point, one of our main concerns with data acquisition in WSNs is the communication cost, so we also carefully analyze the communication costs of CDS and CDS(RW) as well.

Error Recovery

While the decoding process for CDS and CDS(RW) is very similar, the encoding differences are what eventually prove to be the mitigating factor when it comes to overall communication cost. In this section we carefully analyze the error recover capability of both CDS and CDS(RW). For the error recovery comparison and performance evaluation of the rateless coding process we used minimum-mean squared error (MMSE) where $\|X\|_2^2$ is the norm 2 function:

$$\widehat{x}_{\text{MMSE}} = \underset{x'}{\operatorname{argmin}}\, E\|X - x'\|_2^2 \quad \text{s.t.} y = \Phi x'$$

Where the error is:

$$E\|X - \widehat{x}_{\mathrm{MMSE}}\|_2^2$$

In Figures 9-14 we have six different network configurations and the graphs that result from the comparison of the original signal to the recovered signal for each situation. In the graphs there is a baseline that has data point “spikes” rising from the baseline that represent the original signal. Scattered along the baseline and near the peaks of the “spikes” are small circles that represent the recovered signal. In the case of near perfect recovery, the graphs would consist of a baseline with “spikes” and circles at centered around the peak of each spike and all along the baseline – indicating that the recover signal matched exactly with the original signal.

In Figures 9-14, the three essential variables that need to be considered when analyzing the graphs are the size of the graph N, the sparsity of the graph s, and the number of projections provided for decoding M. The sparsity value s is defined to be $s = \frac{K}{N}$ where K is the K-sparse value associated with CS and the input signal. The most dynamic value, usually based on empirically observations of error recovery, is the number of projections, M. Compressive sensing dictates the number of projections must be at least $M = \mathrm{O}(K \log(N))$ in order to recover the signal with high probability. In the end, M was empirically chosen to balance the communication cost versus the error recovery capability.

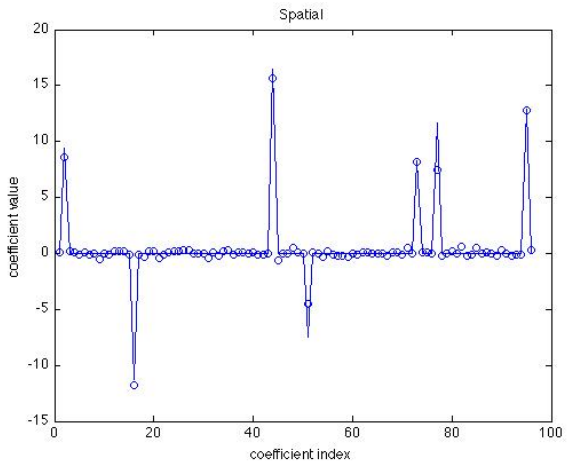

Figure 8: CDS Error Recovery for N=96, s=0.0729, M must be at least $\boldsymbol{M = O((s * N)\log(N))}$ =13.872727, M=40 was chosen to maximize error recovery while still staying within communication bounds.

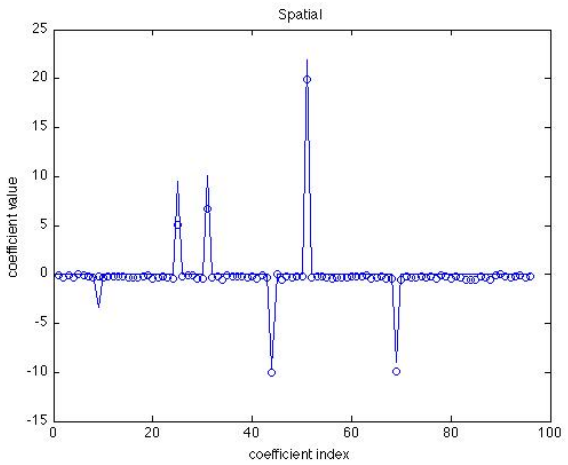

Figure 9: CDS(RW) Error Recovery for N=96, s=0.0729, M must be at least $\boldsymbol{M = O((s * N)\log(N))}$=13.872727, M=40 was chosen to maximize error recovery while still staying within communication bounds.

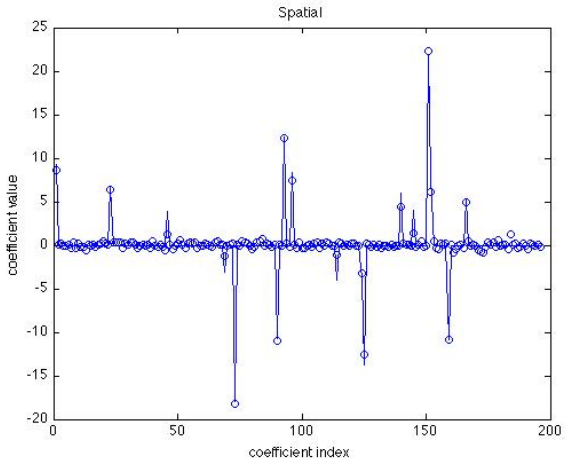

Figure 10: CDS error recovery for N=196, K=0.0714, M must be at least $\boldsymbol{M = O((s * N)\log(N))}$=32.0787, M=80 was chosen to maximize error recovery while still staying within communication bounds.

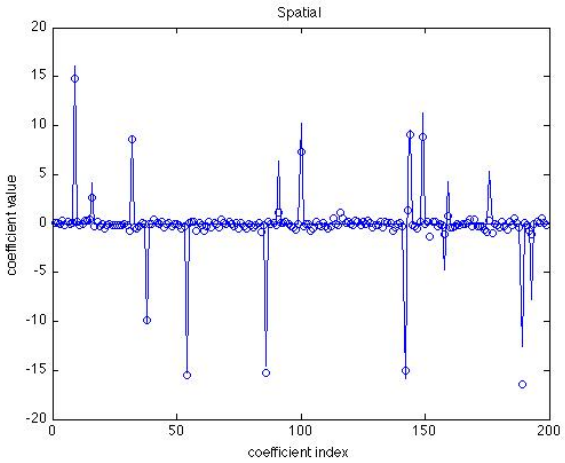

Figure 11: CDS(RW) recovery for N=200,K=0.0854, M must be at least $\boldsymbol{M = O((s * N)\log(N))}$=39.302, M=80 was chosen to maximize error recovery while still staying within communication bounds.

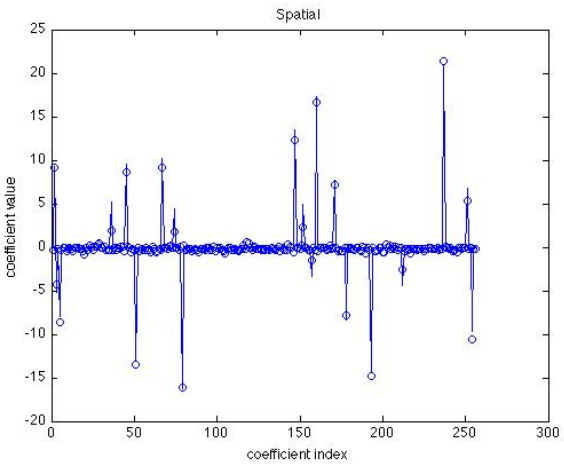

Figure 12: CDS error recovery for N=256, K=0.0742, M must be at least $\boldsymbol{M = O((s * N) \log(N))}$=45.7445, M=120 was chosen to maximize error recovery while still staying within communication bounds.

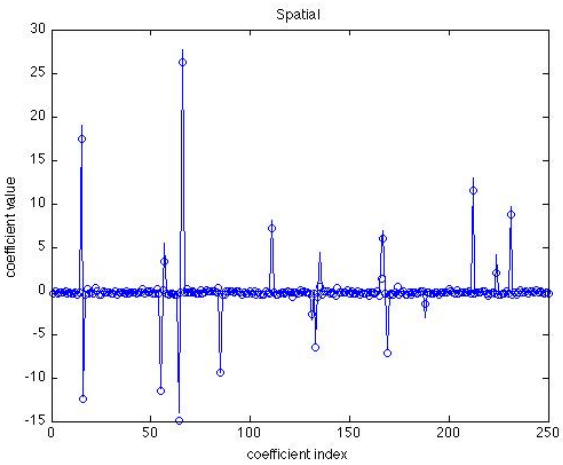

Figure 13: CDS(RW) recovery for N=250, K=0. 0720, M must be at least $\boldsymbol{M = O((s * N) \log(N))}$=43.1629, M=120 was chosen to maximize error recovery while staying withing communication bounds.

While one can extrapolate information from the graphs in Figures 9-14, we cannot rely solely on visualizations without having also analyzing concrete numbers – for that reason we carefully detail one set of the graphs in Table 4. In Table 4 we have a series of ten different simulations of CDS and CDS(RW) where each time $M = 40$ projections were generated for a size of $N = 96$ network with a sparsity value of $s = 0.0729$. The "begin" value is the norm-2 of the signal that essentially refers to the error recovery if we randomly guessed the values in the signal. The "end" value represents the error recovery after using the CDS or CDS(RW) projections in the decoding process. The summary of these runs can be found in middle section of Table 4, where we list all the averages and also include a percentage of difference from the beginning error percentages versus the ending error percentages.

CDS(RW) Begin	*CDS(RW) End*	*CDS Begin*	*CDS End*
29.5073	8.2696	29.4894	7.2263
29.0035	9.9183	30.2244	5.6476
29.524	10.3277	29.1587	6.449
29.1776	9.7763	30.3121	7.1271
29.573	9.4178	29.0892	6.9328
30.3327	10.7532	29.0567	5.2237
29.6605	6.4261	29.19	10.0886
29.4118	7.393	29.6201	7.4674
29.8098	9.4969	29.2678	5.2679
28.892	10.7901	29.5642	6.2484
Averages of data above			
CDS(RW) Begin	*CDS(RW) End*	*CDS Begin*	*CDS End*
29.48922	8.26507	29.49726	6.76788
Percentage Difference in Averages			
CDS(RW) Percentage Difference		*CDS Percentage Difference*	
0.719725717		0.770559028	

Table 4: Recovery error statistics for CDS and CDS(RW)

The most telling data in Table 4 is the percentage difference from beginning to end for CDS, about 77%, and CDS(RW) of about 72%. CDS and CDS(RW) are very comparable but CDS has the edge when it comes to its error recovery from the projections provided. In summary, the error recovery of CDS(RW) is comparable to that of CDS. As will be shown in the next section, the communication cost of CDS(RW) if far less than that of CDS therefore mitigating the advantage that CDS has in error recovery.

Communication Cost

To analyze the communication cost of CDS versus CDS(RW) we consider how many “hops” or sensor communications are required to propagate the necessary data across the network. In order to do successfully analyze communication cost a key step involved finding the shortest paths on dynamic networks between nodes. Since our

networks were represented as undirected graphs with no edge costs, we simply used a breadth first search until the destination node was reached or no more nodes could be visited, in which case the node was not found. The breadth first search algorithm starts at a source node and expands its frontier to each unvisited neighboring node and is unaffected by changes in the network topology as it is Markov. The algorithm returns an integer that represents the shortest number of hops necessary to reach destination node from source node regardless of the specific path taken.

The shortest path algorithm is used in both CDS and CDS(RW) in several different ways. In CDS, the shortest path algorithm was used very heavily as it became necessary for the all requested data from nodes had to first be sent to a intermediary node via the shortest path, then linearly combined and sent along to the base station via the shortest path. For CDS(RW) the shortest path algorithm was only used to calculate costs between the last node in our random walk to the base station. Tables 5, 6, and 7 summarize average communication costs for CDS and CDS(RW) on grid and random network topologies based several different network sizes *N*.

Communication Costs	CDS	CDS(RW)
Grid Network Topology	4,320	1,160
Random Network Topology	2,172	968

Table 5: Communication cost for CDS and CDS(RW) with N=96, M=40 projections

Communication Costs	CDS	CDS(RW)
Grid Network Topology	89,520	5,120
Random Network Topology	12,640	4,960

Table 6: Communication cost for CDS and CDS(RW) for N=200, M=80 projections

Communication Costs	CDS	CDS(RW)
Grid Network Topology	252,000	8,640
Random Network Topology	30,360	7,320

Table 7: Communication cost for CDS and CDS(RW) for N=300, M=120 projections

For clarity, we detail a sample calculation for a network size of $N=300$ on a grid network. As seen in the Chapter 3, the communication cost for CDS is:

$$O\left(K\log N(d\omega + d)\right) = O\left(M * (d\omega + d)\right)$$

Where $K\log(N) = M$ projections need for recovery at BS according to CS, $d\omega$ is the cost associated with generate a projection, and d is the cost from the sink node to the base station.

For the random topology situation in Table 7 using CDS we see $O\left(K\log N(d\omega + d)\right) = O\left(M * (d\omega + d)\right) = O(120 * (250 + 3) = 30{,}360$ hops to disseminate data for compressive sensing. CDS(RW) communication cost is calculated differently and uses the average mixing time, τ, described in the Methodology section earlier. Since the data collection and projection generation process are combined we need simply consider the mixing time used, given that $\tau < q$ – the number of individual data points required for each, and then the cost from the final node in the random walk to the base station. Let τ be the mixing time described in Methodology chapter, let d be the cost from the last node in the random walk path to the base station (the last node in the random walk path acts as the sink node), and let $K\log(N) = M$ be the number of projections needed for CS; then for CDS(RW) we have a communication cost of:

$$O\left(K\log N(\tau + d)\right) = O\left(M * (\tau + d)\right)$$

For the random topology situation in Table 7 using CDS(RW) we see $O\left(K\log N(\tau + d)\right) = M * (\tau + d) = 120 * (58 + 3) = 7320$ hops to disseminate data for compressive sensing. The results for CDS and CDS(RW) on a random network follow the same pattern as on grid networks except the disparity between CDS and CDS(RW) is not as drastic because of the higher connectivity of random networks; making finding shortest paths to a destination node much more efficient because of the increase in neighbors and thus overall connectivity. Another key feature of CDS(RW) over CDS is the ability for CDS(RW) to adapt

to network topology. This is best illustrated in the similar communication costs for grid and random topologies for each of the three situations provided in Tables 5,6, and y. To summarize the details in Tables 5,6 and 7 we provide Table 8 which lists averages for CDS and CDS(RW) on grid and random network topologies. We can see that CDS(RW) performs, on average, significantly better than CDS. As briefly mentioned above, we can see that CDS(RW) also performs better comparatively to CDS when changing from grid to random topology; this is an indicator that CDS(RW) will handle dynamic topologies and errors in the WSN gracefully.

Average Communication Costs	CDS	CDS(RW)
Grid Network Topology	115,280	4,973.33
Random Network Topology	15,057.33	4,416

Table 8: Single-hop average communication cost for CDS and CDS(RW) on grid and random network topologies.

In the end while CDS has the advantage in error recovery, CDS(RW) has a dominating advantage in regards to communication costs. While performance is still important, our main concern is to minimize communication costs, and thus CDS(RW) is the apparent winner over CDS. This balance between communication costs and error recovery can be affected in a number of ways, and it is possible to decrease the error recovery of CDS(RW) by increasing the number of projections provided to the base station. By CDS(RW) increasing the number of projections to the base station in an effort to decrease error recovery, there is an obvious increase in communication cost associated with the collection and generation of the projection. Another way to decrease error recovery of CDS(RW) compared to CDS is by increasing the mixing time and skip-hop value in the random walk so that the sample data points used in the generation of projections are more likely to be less correlated due to increased physical proximity.

CHAPTER 5: CONCLUSION

This work takes an in depth exploration into data acquisition algorithms for WSNs and ends at an innovative solution that combines compressive sensing and random walk algorithms in CDS(RW). The first algorithm proposed, CDS, is essentially compressive sensing with rateless codes and was shown to already have a lower communication cost than currently existing CS algorithms. While the cost was low, we were eventually able to do better by incorporating random walk with CS. In the end, CDS(RW) showed to be dominant in communication costs as compared to existing CS algorithms while still being satisfactory in decoding accuracy.

Future work includes rigorous simulations on different network topologies to gain further insight on performance gains found in this work. Also, increasing the number of measurements provided to the base station for CDS(RW) is an option since the communication cost is so low. This means that CDS(RW) has the capability to possibly perform better in terms of recovery error as compared to CDS while still be significantly less in communication cost. Finally, this work can be extended to an actual WSN and then analyzed for performance in a real-world situation.

REFERENCES

[1] Kandris, D.; Tsioumas, P.; Tzes, A.; Pantazis, N.; Vergados, D.D.; , "Hierarchical energy efficient routing in Wireless Sensor Networks," *Control and Automation, 2008 16th Mediterranean Conference on* , vol., no., pp.1856-1861, 25-27 June 2008.

[2] Dayu He; , "The ZigBee Wireless Sensor Network in medical care applications," *Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010 International Conference on* , vol.1, no., pp.497-500, 24-26 Aug. 2010.

[3] Diamond, S.M.; Ceruti, M.G.; , "Application of Wireless Sensor Network to Military Information Integration," *Industrial Informatics, 2007 5th IEEE International Conference on* , vol.1, no., pp.317-322, 23-27 June 2007.

[4] "NI Wireless Sensor Networks", http://www.ni.com/wsn/, April, 2011.

[5] "Real-Time Wireless Sensor Network Platform", http://www.ece.cmu.edu/firefly/, April 2011.

[6] "Harvard Sensor Networks Lab", http://fiji.eecs.harvard.edu/Volcano, Aprol 2011.

[7] R. Cristescu, B. Beferull-Lozano, M. Vetterli. "On Network Correlated Data Gathering with explicit communication: Np-completeness and algorithms" IEEE/ACM Trans. On Networking, 14(1):41-54, Feb. 2006.

[8] D. Slepian, J.K. Wolf. "Noiseless Encoding of Correlated Information Sources" 19:471-480, Jul. 1973.

[9] E. J. Candès, M.B. Wakin. "An Introduction to Compressive Sampling." IEEE Signal Processing Magazine, 25(2):21-30, Mar. 2008.

[10] C. Luo, F. Wu, J. Sun, and C. W. Chen, "Compressive data gathering for large-scale wireless sensor networks," *InProc. of MobiCom*, pp. 145–156, 2009.

[11] J. Chou, D. Petrovic, and K. Ramchandran. A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks. In Proc. of IEEE Infocom, pages 1054–1062, Mar. 2003.

[12] G. Hua and C. W. Chen. Correlated data gathering in wireless sensor networks based on distributed source coding. Intl. Journal of Sensor Networks, 4(1/2):13–22, 2008.

[13] M. Sartipi and R. Fletcher, "Energy-Efficient Data Acquisition in Wireless Sensor Networks Using Compressed Sensing," *Data Compression Conference (DCC), 2011* , vol., no., pp.223-232, 29-31 March 2011

[14] E. Candes, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," IEEE Transactions on Information Theory, vol. 52, no. 2, pp. 489–509, 2006.

[15] S. Chen, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," SIAM Journal on Scientific Computing,, pp. 33–61, 1998.

[16] W. Wang, M. Garofalakis, and K. Ramachandran, "Distributed sparse random projections for refinable approximation," *In Proc. of the ACM/IEEE International Symposium on Information Processing in Sensor Networks*, pp. 331–339, 2007.

[17] S. Lee, S. Pattem, M. Sathiamoorthy, B. Krishnamachari, and A. Ortega, "Compressed sensing and routing in multi-hop networks," USC Technical Report, 2009.

[18] M. G. Luby, "LT codes," *In Proc. of the* 43rd *IEEE Symposium on the Foundations of Computer Science (STOC)*,pp. 271–280, 2002.

[19] A. Shokrollahi, "Raptor codes," *In Proc. of IEEE International Symposium on Information Theory*, p. 36, 2004.

[20] P. Maymounkov, "Online codes," *NYU Technical Report TR2003-883*, 2002.

[21] W. Xu and B. Hassibi, "Further results on performance analysis for compressive sensing using expander graphs," In Proc. of the Forty-First Asilomar Conference on Signals, Systems and Computers, pp. 621–625, 2007.

[22] D. Baron, S. Sarvotham, and R. Baraniuk, "Bayesian compressive sensing via belief propagation," IEEE Transactions on Signal Processing, vol. 58, no. 1, pp. 269–280, 2010.

[23] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 569 – 584, February 2001.

[24] Y. Lin, B. Liang, and B. Li, "Data persistence in large-scale sensor networks with decentralized fountain codes," In Proc. of Infocom, 2007.

[25] D. Vukobratovic, C. Stefanovic, V. Crnojevic, F. Chiti, and R. Fantacci, "A packet-centric approach to distributed rateless coding in wireless sensor networks," In Proc. of Conference on Sensor, Mesh, and Ad-Hoc Networks and Communications, 2009.

[26] F. Zhang and H. D. Pfister, "Compressed sensing and linear codes over real numbers," *In Proc. of Information Theory and Applications*, Feb. 2008.

[27] Shannon, Claude E.: Prediction and entropy of printed English, The Bell System Technical Journal, 30:50-64, January 1951.

VITA

Robert Brian Fletcher was born in Santa Cruz California and lived in Arcadia, California with his parents Ken and Jacci Fletcher. He moved to Knoxville, Tennessee in 2003 and graduated from Karns High School in Knoxville in 2004. After high school, Robert attended University of Tennessee – Knoxville and received his Bachelor of Science in Computer Science with a minor in mathematics from University of Tennessee Knoxville. Robert then received his Master of Science in Computer Science from University of Tennessee at Chattanooga. Robert recently, February 2011, was hired at Cisco Systems, Inc. as a Software Engineer II. Robert is also employed as a graduate research assistant at University of Tennessee at Chattanooga. Robert enjoys working out and has been a fantastic relationship with his girlfriend, Kayla Folks, for over seven years.