COMPRESSIVE DATA GATHERING IN WIRELESS SENSOR NETWORKS

Dariush Ebrahimi

A thesis

in

The Department

of

Computer Science & Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Doctor of Philosophy

Concordia University

Montréal, Québec, Canada

February 2016

CONCORDIA UNIVERSITY

Engineering and Computer Science

This is to certify that the thesis prepared

By: **Dariush Ebrahimi**
Entitled: **Compressive Data Gathering in Wireless Sensor Networks**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Maria Elektorowicz
_____ Chair
Dr. Pin-Han Ho
_____ External Examiner
Dr. Walaa Hamouda
_____ External to Program
Dr. Lata Narayanan
_____ Examiner
Dr. Jaroslav Opatrny
_____ Examiner
Dr. Chadi Assi
_____ Thesis Supervisor

Approved by

_____
Chair of Department or Graduate Program Director

_____ _____
Date Dean of Faculty

Abstract

Compressive Data Gathering in Wireless Sensor Networks

Dariush Ebrahimi, Ph.D.

Concordia University, 2016

The thesis focuses on collecting data from wireless sensors which are deployed randomly in a region. These sensors are widely used in applications ranging from tracking to the monitoring of environment, traffic and health among others. These energy constrained sensors, once deployed may receive little or no maintenance. Hence gathering data in the most energy efficient-manner becomes critical for the longevity of wireless sensor networks (WSNs).

Recently, Compressive data gathering (CDG) has emerged as a useful method for collecting sensory data in WSN; this technique is able to reduce global scale communication cost without introducing intensive computation, and is capable of extending the lifetime of the entire sensor network by balancing the forwarding load across the network. This is particularly true due to the benefits obtained from in-network data compression. With CDG, the central unit, instead of receiving data from all sensors in the network, it may receive very few compressed or weighted sums from sensors, and eventually recovers the original data.

To prolong the lifetime of WSN, in this thesis, we present data gathering methods based on CDG. More specifically, we propose data gathering schemes

using CDG by building up data aggregation trees from sensor nodes to a central unit (sink). Our problem aims at minimizing the number of links in the forwarding trees to minimize the number of overall transmissions. First, we mathematically formulate the problem and solve it using optimization program. Owing to its complexity, we present real-time algorithmic (centralized and decentralized) methods to efficiently solve the problem. We also explore the benefits one may obtain when jointly applying compressive data gathering with network coding in a wireless sensor network. Finally, and in the context of compressive data gathering, we study the problem of joint forwarding tree construction and scheduling under a realistic interference model, and propose some efficient distributed methods for solving it. We also present a primal dual decomposition method, using the theory of column generation, to solve this complex problem.

Acknowledgments

I would like to express my deepest gratitude to many people who supported me during my PhD and earlier studies and who helped me to complete my thesis. Their generous help made this dissertation possible.

First of all, I am deeply grateful to my advisor, Dr. Chadi Assi, for his inspiration, motivation and guidance throughout my related research. I could not have imagined having a better advisor for my Ph.D study. I would like to extend my honest appreciation to my former advisor, Dr. Maytham Safar, for encouraging me during my studies and for introducing me to the world of academic research. I am truly indebted to them for their knowledge, thoughts, and friendship.

Besides my advisors, I would like to thank my committee members, Dr. Lata Narayanan, Dr. Jaroslav Opatrny and Dr. Walaa Hamouda, for their insightful comments and encouragement and most importantly for their willingness to read through the thesis and serve on my committee board.

My sincere appreciation goes to Dr. Samir Sabbah who helped me toward this thesis in my last work related to the primal-dual decomposition method. I thank him for the insightful and enlightening discussions I had with him.

Furthermore, I am greatly thankful to all of my colleagues in the research lab at Concordia University for providing me with warm and friendly atmosphere.

Last but not least, I would like to thank my great parents, family and friends for their continues support, understanding and assistance when ever I needed them. I believe that without my parents, I could not be able to succeed throughout my life. I am always grateful to them for their encouragement and support.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

WSN ..... Wireless Sensor Network

CS ..... Compressive Sensing

CDG ..... Compressive Data Gathering

CDG/C ..... Compressive Data Gathering Centralized method

CDG/D ..... Compressive Data Gathering Decentralized method

MSTP ..... Minimum Spanning Tree Projection

eMSTP ..... extended Minimum Spanning Tree Projection

PCDG ..... Projection-Based Compressive Data Gathering

DCDG ..... Distributed Compressive Data Gathering

NC ..... Network Coding

NC-CDG ..... Network-Coding Compressive Data Gathering

NC-CDG/C ... Network-Coding Compressive Data Gathering Centralized method

NC-CDG/D ... Network-Coding Compressive Data Gathering Distributed method

OSPN ..... Optimal Selection of Projection Nodes

Opt-PCDG ... Optimal Projection-based Compressive Data Gathering

PB-CDG ..... Projection-Based Compressive Data Gathering algorithm

FTCS ..... Forwarding Tree Construction and Scheduling

D-FTCS ..... Distributed Forwarding Tree Construction and Scheduling

CG ..... Column Generation

ILP ..... Integer Linear Programming

MILP ..... Mixed Integer Linear Programming

Non-CS ..... Non Compressive Sensing

Plain-CDG .. Plain Compressive Data Gathering

Hybrid-CDG Hybrid Compressive Data Gathering

CSMA ..... Carrier Sense Multiple Access

TDMA ..... Time Division Multiple Access

MAC ..... Medium Access Control

RIP ..... Restricted Isometry Property

MST ..... Minimum Spanning Tree

SPF ..... Shortest-Path-Forest

DFT ..... Discrete Fourier Transformation

PDF ..... Probability Density Function

IN ..... Interest-Nodes

BFS ..... Breadth-First-Search

SINR ..... Signal to Interference plus Noise Ratio

RID ..... Radio Interference Detection

ND ..... Normal power Detection

HD ..... High power Detection

OTC-OLS .... Optimal Tree Construction - Optimal Link Scheduling

DTC-OLS .... Distributed Tree Construction - Optimal Link Scheduling

TCM ..... Tree Construction Model

LSM ..... Link Scheduling Model

Chapter 1

Introduction

1.1 Overview and Objectives

Wireless sensor networks (WSNs) have received significant attention due to their versatility and have been deployed widely in applications such as military surveillance, monitoring of environment, traffic and critical infrastructures, among others. Many of these applications require sensors to periodically sense and send sensory data to a remote central unit (e.g., sink) for processing, often through multi-hop paths as depicted in Figure 1.1. Once deployed, these energy-limited sensors may receive little or no maintenance; therefore energy efficient data collection protocols become of utmost importance to operate sensor networks for a long period of time.

Increasing the lifetime of a wireless sensor network depends directly on minimizing the energy consumption at sensor nodes. In a WSN, most of the power is consumed in data transmission and forwarding when compared to data sensing and computation (processing) [37]. According to [5], the energy needed to transmit a single bit is measured to be over 1000 times greater than a single 32-bit computation. Therefore, to maximize the network lifetime, one

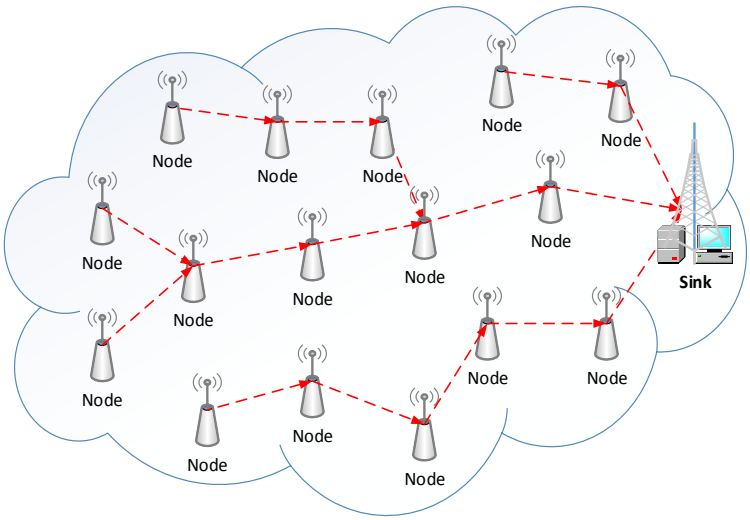


Figure 1.1: Multi-hop data transmission in wireless sensor networks.

has to address the following two challenges: 1) reducing the global network communication cost, and 2) dealing with the unbalance of energy consumption throughout the network. In a large-scale network, when individual sensors transmit their data to the sink, it is expected that a single data could be transmitted several times through multi-hop routing, which causes a large number of redundant transmissions in the network, therefore increasing the overall network communication cost. Furthermore, nodes which are closer to the sink do more forwarding tasks than other nodes. Therefore, these bottleneck nodes (i.e., neighbour nodes to the sink) consume more power and consequently run out of energy quickly, which shorten the lifetime of the network. Different methods have been proposed by researchers to maximize the lifetime of WSNs, such as, adjusting sensing ranges [10], sleep scheduling [69], clustering routing protocol [65], cross-layer network formulation [18] and data aggregation [50]. Data aggregation, unlike the other approaches, aims at reducing the amount of data to be transported, and hence significantly helps in overall energy consumption load.

Data aggregation eliminates the redundancy in transmitting data between sensor nodes and the sink and thus significantly reduces the number of transmissions in the network, yielding substantial energy savings [55]. For instance, given the spatial-temporal correlations that local sensed data may exhibit, only an aggregate and representative extract (e.g., SUM, MIN, MAX) of the measured data at various sensors may be forwarded to the sink, avoiding unnecessary transmissions in the network. Alternatively, if the sink needs to recover the set of all sensed data, more involved aggregation methods may be used, such as collaborative or non-collaborative data compression [32], where they differ in terms of practicality, complexity, and associated overhead.

Recently, compressive sensing (CS) [20] theory has emerged to provide an alternative venue for data gathering in wireless sensor networks, referred to as Compressive Data Gathering (CDG). Originally developed for signal processing [8], CS promises to efficiently recover a signal from far fewer samples than its original dimension, as long as the signal is sparse or compressible in some domain. In WSNs, CDG is one of the most efficient methods for gathering sensed data en-route to the sink [59] and has recently been receiving focal attention owing to its ability to reduce the global communication cost without incurring intensive computation or transmission overhead. With compressive data gathering, rather than receiving all readings, e.g., from n sensors, the sink will only receive few weighted (encoded) sums (e.g., m, $m \ll n$) of all the readings, from which the sink will be able to recover (decode) the original data, as long as the readings can be transformed or compressed in some sparse orthonormal transform domain [20, 59]; here, $m = O(k\log n)$ and k represents the sparsity representation of the data in the transform domain. CDG has attracted researchers' attention only recently; this technique has shown

to yield substantial energy savings, therefore extending the network lifetime, and achieve load balancing by dispersing the communication costs to all sensors along a given route [59].

1.2 Problem Statement and Motivation

In this thesis, we consider the problem of energy efficient data gathering in a network consisting of n sensors. we suppose the original sensory data is compressible in some transform domain, and it is recovered at the sink by receiving m sparse projections [73], where each projection corresponds to an aggregation of data from sensors according to the theory of compressive sensing. Here, projections are gathered by establishing forwarding trees, one tree for each projection which gather coded (compressed) data from nodes involved in the projection. Projections may be either collected by projection nodes (selected sensors), which subsequently send their collected coded measurements to the sink (e.g., through shortest paths) to recover the original data, or at the sink itself. Upon collecting all projections, the sink then attempts to recover the original data by solving a convex optimization problem [8]. Here, it should be noted that because of data aggregation in each projection, to reduce the number of transmissions along the forwarding trees, parent nodes should only transmit their measurements upon receiving measurements from their children; once downstream coded/compressed measurements are received, such measurements are combined with local measurements for uplink transmission.

Constructing efficient projections or gathering trees to collect measurements, while minimizing the cost of transmissions, is indeed a challenging problem and efficient heuristics and approximation algorithms have recently

been presented [2, 87]. Unlike previous work in the literature, the novelty of our thesis lies in utilizing independent forwarding trees, where each forwarding tree carries compressed (rather than native) data packets to the sink. These forwarding trees are constructed to ensure fewer number of transmissions as well as to evenly distribute the transmission load across the network. In the following we state different scenarios for Projection-based compressive data gathering (PCDG).

1.2.1 Distributed Compressive Data Gathering

In the centralized PCDG, initially the sink has to accomplish a topology discovery by retrieving the network wide information through deploying an all-to-all flooding (where $O(n^2)$ messages are needed), and then solves the algorithm to construct the forwarding trees required for CDG. Subsequently, for each forwarding tree, the sink sends out notification messages to all nodes in the network notifying each of its parent node and children. Clearly, the overhead associated with such centralized approach makes it difficult to implement in practice. Rather, we present a distributed approach (DCDG) for constructing the forwarding trees where each node locally decides its parent node to whom it should transmit its encoded data.

1.2.2 Network-Coding Aware Compressive Data Gathering

In PCDG, we observe that the presence of forwarding trees to collect compressed data will create opportunities for many-to-many traffic patterns in the network; such traffic patterns (which normally do not exist in wireless sensor networks) in turn create opportunities for network coding which can be

exploited by the forwarding process to carry the compressed traffic towards projection nodes. Network coding [1] has recently gained popularity for its promise to reduce the number of transmissions in wireless networks, thereby increasing their throughputs [47,67]. However, owing to the many-to-one traffic patterns typically occurring in sensor networks, network coding found only little applications in such networks; for example, to create a balance between energy efficiency and reliability of the forwarding in the presence of packet loss [48]. This thesis advocates using network coding by exploiting the overlap between forwarding trees carrying compressed traffic to their projection nodes. Here, we should note that along forwarding trees, some sensor nodes function as aggregators, combining their own sensed data with those received from downstream nodes, while other sensors act as forwarders, simply relaying the received aggregate data to parent nodes along the trees. Such nodes (forwarders) can perform network coding on the compressed traffic traversing along different forwarding trees, resulting in fewer transmissions and thus better energy efficiency for the gathering protocols. To reap the most benefits, however, forwarding or aggregation trees must be constructed to give rise to such coding opportunities in the network.

1.2.3 On the Interaction between Scheduling and Compressive Data Gathering

Once the gathering trees for PCDG are constructed, links on the constructed trees need to be scheduled for transmissions such that adjacent transmissions do not cause harmful interference on one another (thus corrupting the compressed measurements) while maintaining a maximum spatial reuse of the

wireless spectrum. Therefore, finding forwarding trees to collect measurements at the sink in the most energy efficient manner under the physical interference model becomes a complex problem of combinatorial nature. We refer to this problem as Forwarding Tree Construction and Scheduling (FTCS).

The decentralized approach of FTCS decouples the problem into two subproblems; namely, the tree construction subproblem and the link scheduling subproblem. Our decentralized tree construction is amended with refinements to help the link scheduling achieve better scheduling and thus collection latency. Our scheduling subproblem is resolved in a distributed fashion, through interference localization an coordination among links to control the level of interference.

To the best of our knowledge, our work is the first to resolve the problem of compressive data gathering under physical interference constraints in a decentralized manner without requiring to partition the unit square area into cells. Our method can be used to efficiently operate large wireless sensor networks which periodically gather sensory data in the most energy efficient manner and with gathering latency constraints.

1.2.4 A Primal Dual Decomposition Method

To solve the FTCS problem, each tree may be constructed independently and then its links are scheduled. However, when all trees are combined together, the shortest and energy efficient schedule may not be guaranteed. Further, a large number of possible forwarding trees for each projection may be considered. Both problems of enumerating forwarding trees and scheduling links for those trees are hard combinatorial problems. This is compounded by the fact that the two problems must be solved jointly, to guarantee the selection of best

forwarding trees which, when their links are scheduled, guarantee a shortest energy efficient schedule.

Solving the joint problem of Forwarding Tree construction and Scheduling (FTCS) using Mixed Integer Linear Programming (MILP) is very complex. The difficulty of FTCS problem is centered around the fact that a large number of forwarding trees may be constructed for each projection and that as many forwarding trees as the number of projections should be selected. The m forwarding trees which guarantee minimum gathering latency (under an appropriate link scheduling) should be selected. The number of such trees is exponentially large and the link scheduling itself is a known NP-hard problem [30]. Owing to the complexity and to keep track of the problem, we propose a primal-dual decomposition method using Column Generation (CG) [14]; here, the problem is divided into a Master and several Multi-Pricing sub-problems. Each sub-Pricing schedules links for one constructed tree. The Master problem checks whether the link scheduling obtained by all different sub-Pricing problems are not overlapped and their physical interference constraints are satisfied. Most importantly, the Master tries to choose a configuration from each sub-Pricing which minimizes the gathering latency. To the best of our knowledge, this problem has not been investigated in previous literature.

1.3 Thesis Contribution

The main contributions of the thesis are summarized as follows:

- We define our projection based compressive data gathering problem and describe how the trees (joint routing pathes) from nodes to the sink can be constructed to gather sensory data at the sink. First, we explain how we

divide sensors in the network into sets of interest-nodes, where each set corresponds to a projection which their data is intended to be aggregated through a forwarding tree to the sink. We present our first algorithmic method (Minimum Spanning tree projection (MSTP)) for our problem derived from current existing sparse random projections [73] which uses random projection nodes to construct the trees. Then, we modify our method and present a more efficient method (eMSTP) that improves our first method by letting the sink node to gather weighted sums directly from nodes instead of projection nodes; in this way we eliminate the traffic resulting from transmitting the corresponding projection packets from projection nodes to the sink. We present MSTP before eMSTP to show how progressively we improve the lifetime of the network. Next, we show that the efficiency of our algorithm eMSTP strictly depends on the selection of the projection nodes. Therefore, we propose an optimal selection of projection nodes algorithm (OSPN). Later, we characterize a mathematical optimization model (Opt-PCDG) for the construction of trees without using projection nodes and furthermore, we propose a heuristic algorithm (PB-CDG) which gives near-optimal solution with very fast computational time. Moreover, we analyze the time complexity of our algorithmic methods, and further, we compare our methods with compressive data gathering methods presented in the literature.

- The drawback of PCDG lies in communication and computation costs of constructing required forwarding trees for compressive data gathering. Whereas a central unit like the sink requires a complete knowledge of the network topology to construct the forwarding trees and later to notify all the nodes in the network of routing trees (for example, allowing

each node to know its parent and child). The computational cost can be defeated by using the heuristic algorithm (not the MILP model) which has much smaller complexity and runs quite fast. However, to overcome the communication drawback, we present a distributed manner (refer to as Distributed Compressive Data Gathering, DCDG) to construct the forwarding trees. Through an example, we illustrate the operation of the DCDG algorithm, then, we derive the approximation bound and analyze the message overhead of the distributed method. Furthermore, we investigate how the PCDG may reconstruct the trees in case of node(s) failure and also how DCDG may self healing reconstruct the trees. In addition, we present the Steiner-CDG method where the forwarding trees for CDG are constructed using minimum Steiner tree construction algorithm [49], and we compare the performance of all of these heuristic CDG.

- We explore the problem of network coding aware data aggregation in WSNs. We mathematically formulate the problem of optimal construction of forwarding/aggregation trees for projection based compressive data gathering in the presence of network coding. These forwarding trees are constructed to ensure fewer number of transmissions and to evenly distribute the transmission load across the network. Owing to its computational complexity, we then develop algorithmic solutions and present centralized and distributed methods for constructing forwarding trees.

- We define the problem of forwarding tree construction and link scheduling (FTCS) and we mathematically formulate the problem as a mixed integer linear program (MILP) through which we may obtain optimal solutions for small size networks. Next, we analyze the complexity of FTCS and prove its NP-hardness. To overcome the computational complexity,

we propose a distributed method that can solve for large scale networks. Later, We prove the correctness of our algorithmic method and analyze its performance. Through a large set of numerical results, we validate the efficiency and performance of our distributed FTCS method.

- After highlighting the complexity of the FTCS problem, we present a novel primal-dual decomposition method using column generation. We also highlight several challenges we faced when solving the decomposed problem and present efficient techniques for mitigating those challenges. One major advantage of our work is that it can serve as a benchmark for evaluating the performance of any low complexity method for solving the FTCS problem for larger network instances where no known exact solutions can be found.

1.4 Thesis Outline

The rest of the thesis is structured as follows. Chapter 2 presents related work, network model and background required for our investigation throughout this thesis. In Chapter 3, we present efficient algorithmic methods and mathematical formulation for our projection-based compressive data gathering. The distributed approach is given and illustrated in detail in Chapter 4. Chapter 5 investigates the joint application of compressive sensing and network coding to the problem of energy efficient data gathering in wireless sensor networks. The joint problem of compressive data gathering and scheduling under the real physical interference model is studied in Chapter 6. We highlight the complexity of the last joint problem in Chapter 7 and propose a primal-dual decomposition method using column generation. Finally, in Chapter 8 we summarize

our conclusion and provide some future directions for this research.

Chapter 2

Literature Review and Preliminaries

2.1 Related Work

Data gathering is one of the most important functions that wireless sensor networks (WSNs) are expected to perform, specially in scenarios where continuous monitoring is required. Some work is targeted at collecting the required data from individual sensor nodes [57], and others used the in-network data aggregation techniques to reduce the number of transmissions for internal nodes by aggregating all the data received from downstream nodes before forwarding them to uplink nodes. Aggregation techniques are well established methods for data gathering (e.g., [56] and [28]) and the effectiveness of data aggregation methods is strongly dependent on how the sensed data is routed to the sink (e.g., [25, 60, 70, 79, 85]). The problem of constructing aggregation trees with minimum energy cost [50] or to maximize the network lifetime has been studied in previous work [50, 79] where it was first shown that such problems are NP-complete and then approximation algorithms are presented.

Spatial correlation between sensor readings often can be exploited to perform in-network data compression to reduce the cost of communications [26]. Techniques such as entropy coding or transform coding [17, 32] are often employed for data gathering, but suffer from excessive computation and control overheads. Another category for data compression is distributed source coding which utilizes correlation at the sink [13, 16, 36]; such techniques however may not be practical due to the lack of global correlation between the sensor readings.

Recently, compressive sensing (CS) [20] has emerged as an effective approach for data gathering due to its promise to reduce the amount of traffic in the network without adapting to the data correlation structure. This technique has been receiving increased attention for its applications to data aggregation in wireless sensor networks. The CS technique first has been presented by David L. Donoho in [20] for signal processing. Bajwa et. al. in [4] introduced CS into wireless network for a single-hop star network. In [35], the authors gave a conceptual understanding of CS in a wireless sensor network. They showed that original data reading vector of sensors could be recovered at the sink with far fewer sample measurements using the same technique of CS in [20] for signals. Note that, with this technique, it appears as if the original sensors reading vector has been compressed. Hence, instead of transmitting the original data in the network, the compressed data is rather sent and thus a reduction in the traffic transmission loads in the network is expected.

The authors in [59] presented a compressive data gathering (CDG) method for larger scale wireless sensor networks; the objective is to compress sensor readings to reduce global traffic in the network and prolong the network lifetime by distributing energy consumption among sensors in the network.

The authors however decoupled the interactions between data compression and data routing. [9] analyzed theoretically the network energy consumption and showed that CDG outperforms baseline data collection through detailed analysis. Plain and Hybrid compressive sensing techniques in wireless sensor networks are studied in [61, 80] where the authors presented optimization methods for the joint problem of link scheduling and compression to minimize the network energy consumption. They improved the performance of CDG by introducing the Hybrid-CDG scheme, where it applies CS only to relay nodes that are overloaded. [80] proposed mathematical formulation and heuristic greedy algorithm for constructing routing trees for Hybrid-CDG scheme to minimize the network energy consumption. The authors in [75] presented higher level of Hybrid-CDG scheme that integrates partial nodes selection into compressive sensing by using a threshold, so as to extend the area before compressive sensing. In [7], the authors considered a scenario where a wireless sensor network exploits ZigBee protocols which guarantees energy saving. They designed a new adaptive mixed algorithm wherein each node takes a decision about which scheme to adopt among PF (Pack & Forward) and CS (Compressive Sensing) aiming at reducing the number of packets to transmit.

The authors in [73] presented a distributed algorithm based on sparse random projections that requires no global knowledge and guarantees the recovery of near optimal approximation of the original sensed data. The algorithm allows the collector to choose the number of sensors to query according to the desired approximation error; Here the sparsity of the projections greatly reduces the communication cost of pre-processing the data. However, they did not consider data gathering for each projection along the routing paths from

individual nodes to projection node. Motivated from sparse random projections, the authors of [52] presented an algorithm that computes the route for each projection greedily from each random node to the sink to minimize the communication cost. [74] used similar random routing method, but only for grid network topology. The authors in [66] presented an algorithm that, for each projection, uses random walk to collect sufficient number of sensor readings while combining them together without significantly increasing the inter-communication cost. The three random projection algorithms presented in [52], [74] and [66], use specific walk from randomly chosen source node for each projection to the sink to gather one weighted sum needed for data recovery. However, this projection walk takes long distance which results in increasing the number of transactions and thus increases the network energy cost.

The authors in [80] addressed the problem of energy efficient gathering by jointly considering routing and compressed aggregation; given the NP-complete nature of the problem, the authors presented a greedy method to achieve near optimal solutions. Computing compression trees for data gathering is studied in [54] where algorithms with provable optimality guarantees are presented in a network with broadcast communication. The authors in [62] proposed an algorithm that improves the network lifetime by dividing the sensor network into subnetworks to decrease the communication rate and to build up the data aggregation trees. The authors showed through simulations that their algorithm outperforms LEACH [37] and shortest-path routing. The authors in [89] introduced the partitioning method; where they divided the unit square area into equal cells to restrict the transmissions between adjacent cells (horizontal and vertical). In [78, 82, 83] a clustering method is used where, several

nodes are assigned for intermediate data collection at each cluster. [83] presented a hierarchical clustering architecture model, where instead of one sink node being targeted by all sensors, several nodes are assigned for intermediate data collection to gather at different hierarchical clustering levels. It is shown that the hierarchical architecture reduces the number of measurements for CS since in the proposed architecture the compressed ratio depends on the cluster size rather than the global network size. [82] proposed a clustering method that uses Hybrid-CDG for sensor networks and [78] presented an energy efficient clustering routing data gathering scheme for large-scale wireless sensor networks by obtaining the optimal number of clusters where all the cluster heads are uniformly distributed.

A number of studies in the literature considered the scheduling problem in conjunction with data gathering in wireless sensor networks. Among them, [41, 43] proposed asynchronous distributed data collection using CSMA-based MAC mechanism. The authors in [6] and [12] mathematically formulated the problem of joint long-lifetime and minimum latency data collection and aggregation scheduling respectively as a constrained optimization problem and then proposed an approximation algorithm for their problem. The trade-off between energy consumption and time latency was studied in [86]. In [68], the authors presented a distributed implementation for data collection to let each node calculate its duty-cycle locally by giving priority to sub-trees that have bigger size (they assumed the tree is given). [88] presented a novel distributed scheduling data collection algorithm, where the algorithm works periodically; in each round a TOKEN is generated by the sink which is passed in post-order to all nodes and at each round a transmission slot is assigned to nodes that have not assigned before and do not conflict with other transmissions. This

method takes lot of time to assign a time slot for all nodes in a network, since it requires so many rounds and time to pass the TOKEN.

The distributed data aggregation scheduling presented in [85] considers interference only from one-hop node. A novel cluster-based TDMA-based MAC protocol for energy-efficient data transmission has been proposed by [38]. In their protocol, for each cluster, a node with higher remaining energy level acts as a cluster head and assigns time slots to all nodes in its cell based on their needs. The authors in [34] and [44] studied the aggregation rate under interference constraint, where [34] tried to maximize the aggregated information at the sink under deadline constraint and [44] tried to minimize the sum delay of sensed data. In [89], the authors investigated the capacity and delay analysis for compressive data gathering under the protocol interference model. They used a centralized method for their data gathering by partitioning the unit square area into equal cells of a particular size under a certain probabilities.

Link scheduling under physical interference model has received increased attention due to its realistic abstraction (e.g.; [11, 40, 42, 51, 53, 84]). The problem of link scheduling in WSN under the physical interference model was proved to be NP-hard in [30]. The authors in [40] showed that the data collection rate, in addition of interference, is limited by the maximum degree of the routing tree, and proposed techniques to improve the speed of data aggregation. In all of these works [11, 40, 42, 53, 84] the network is partitioned into equal cells and the cells are assigned with colors for concurrent scheduling. In [84] and [53] the aggregation scheduling is done in levels; first aggregate data from nodes in each small area, and then further aggregate data in a larger area by collecting from those small ones. This process is repeated until the entire network as the largest area is covered. [84] constructs the tree

and then does the data scheduling in uplink manner, whereas, [53] features joint tree construction and link scheduling by assigning a nearest node to the sink as a cell head. The data collection/aggregation scheme in [11] and [42] is scheduled in two phases, where in each phase the data collection/aggregation is done in one direction (whether horizontally or vertically to next cell). The authors of [42] combined the CDG technique with pipeline technology and came up with more efficient network capacity. In [51], the authors proposed a novel technique under interference localization that allowed them to do scheduling in a decentralized manner.

2.2 Network Model

We model a wireless sensor network as a connected graph $G = (V, E)$, where V is the set of n nodes deployed randomly in a region and E is the set of links between any two sensor nodes which reside within each other's communication radius. The density of the network can be adjusted by varying the transmission power of the nodes. However, varying the transmit power yields a system model that is much harder to solve. For simplicity, we assume a fixed and uniform transmit power P for all sensor nodes and we assign the power P such that the resulting graph is connected without a single disconnected node and all transmissions within the communication range are successful. We assume each sensor at each round (period) has a data reading x_i (for example, speed, density or temperature) which it intends to send to a base station (sink) that may be located at a certain location in the network. Consequently, at each round, the sink needs to gather, in total, a data vector of size n ($X = [x_1, x_2, ..., x_n]^T$) from all the nodes in the network. Since not all the nodes may have a direct link with the sink, sensors will send their readings

over multi hop routes.

2.3 Compressive Sensing

Compressive sensing (CS) [59] promises to efficiently recover n sensors' readings at the sink with far fewer sample measurements, as long as the original readings could be transformed or compressed in some sparse orthonormal transform domain. Suppose the original data $X = [x_1, x_2, ..., x_n]^T$ has a k-sparse representation under a proper transform basis Ψ, where Ψ is a Fourier transform matrix of size $n \times n$. i.e.:

$$X = \Psi S \tag{2.1}$$

or

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \psi_{11} & \cdots & \psi_{1n} \\ \psi_{21} & \cdots & \psi_{2n} \\ \vdots & \ddots & \vdots \\ \psi_{n1} & \cdots & \psi_{nn} \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \tag{2.2}$$

where S is a k-sparse column vector representation of X, and only k coefficients of S are non-zero and $k \ll n$. According to the Restricted Isometry Property (RIP) of the CS theorem [20], the sink may receive $m = O(k \log n)$ measurements instead of n readings, where $m \ll n$; that is

$$Z = \Phi X \tag{2.3}$$

or

$$\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix} = \begin{pmatrix} \phi_{11} & \cdots & \phi_{1n} \\ \phi_{21} & \cdots & \phi_{2n} \\ \vdots & \ddots & \vdots \\ \phi_{m1} & \cdots & \phi_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \tag{2.4}$$

where Z is a column vector of sample measurement of size $m \times 1$ and Φ is a random sample basis matrix of size $m \times n$. In other words, the sink can perfectly recover the original data X by receiving $Z = [z_1, z_2, ..., z_m]^T$, where $z_t = \sum_{j=1}^{n} \phi_{tj} x_j$, $t = 1, 2, ..., m$. Each z_t represents a weighted sum of measurements from nodes in the network with non-zero coefficients in a row of the matrix Φ. Later in the thesis, we refer to these nodes as interest nodes and the data aggregated from those interest nodes as one projection. The matrix Φ has m rows, one row for each weighted sum (projection), and n columns, one column for each sensor node. Note that, the value of m is far smaller than n (i.e. $m \ll n$); as suggested by [8], $4k \geq m \geq 3k$ is sufficient.

Now, after receiving all m measurements (Z), the destination (sink), using the random sample matrix Φ and the Fourier transform matrix Ψ, recovers the sparse representation of the data $\tilde{S}$ (not the original data) by solving the convex optimization problem of (2.5).

$$\begin{aligned} \tilde{S} &= arg \min_{\tilde{S}} \|\tilde{S}\|_{l_0} \\ subject\ to \qquad Z &= \Phi X = \Phi \Psi \tilde{S} \\ \|\tilde{S}\|_{l_0} &\triangleq \|\{i : s_i \neq 0\}\| \end{aligned} \tag{2.5}$$

This $l_0 - norm$ optimization problem is NP-hard. Therefore, the approximate solution can be obtain by solving the $l_1 - norm$ convex optimization problem given in (2.6).

$$\min_{\tilde{S}} \|\tilde{S}\|_2 \qquad subject\ to \qquad Z = \Phi\Psi\tilde{S} = A\tilde{S} \tag{2.6}$$

where $\|v\|_2 = \sqrt{\sum_i |v_i|^2}$ is the $l_2 - norm$ of a vector v.

After recovering the sparse vector $\tilde{S}$, the original data (X) is obtained by letting $X = \Psi\tilde{S}$. For data recovery, the matrix $A = \Phi\Psi$ has to satisfy the RIP property. A matrix A obeys the RIP of order k if there exists a $\delta_k \in (0, 1)$ such that (2.7) holds for all k-sparse vectors S [19].

$$(1 - \delta_k)\|S\|_{l_1}^2 \leq \|\Phi S\|_{l_1}^2 \leq (1 + \delta_k)\|S\|_{l_1}^2 \tag{2.7}$$

We will loosely say that a matrix A obeys the RIP of order k if δ_k is not too close to one. More significantly, a matrix A obeys the RIP with high probability if the entries are chosen at random with i.i.d (independent and identically distributed) entries from a normal distribution with zero mean and variance $\frac{1}{m}$ or they follow a Bernoulli distribution or more generally any Gaussian distribution [20]. Note that the matrix Ψ is only required at the sink for decoding (recovering) and it is not required for encoding at the nodes. Matrix Φ is fixed and can be considered as a *priori* knowledge for the entire network [59] or, each random vector (corresponding to one sensor node) can be generated locally at each node using a predetermined seed for a pseudo random generator. Seeds may be distributed by the sink to the nodes in the networks. For more details on CS, the reader is referred to [8, 19, 20].

Finally, we note that the performance of the CS depends significantly on

the sparse representation of the data. The more sparse the data can be, the fewer sample measurements to recover the original data are needed.

2.4 Compressive Data Gathering

One of the practical applications of a WSN is to gather all sensors readings at the sink. In its simplest way, and without using compressed sensing (Non-CS), a data collection is built using tree representation as shown in Figure 2.3, where the circular nodes represent the sensor nodes with their number as sensor ID and the black square **S** represents the sink node. The routing tree can be constructed using different strategies such as shortest-path or minimum-power-greedy algorithm. After constructing the routing tree, all nodes in the network know their corresponding parent and child nodes by sending notification messages to each other. Now, in Non-CS, leaf nodes send their data readings to their parent nodes using one packet each. Subsequently, parent nodes send their readings plus the readings from their children in separate packets to their higher parents in the tree (Figure 2.1 illustrates the Non-CS for one route). Finally, n readings (packets) will be collected at the sink. Figure 2.3 illustrates an example of data collection without using compressed sensing. Here, we observe that, the nodes closer to the sink carry out many more transmissions in contrast to the leaf nodes which perform only fewer transmissions. Hence, the load in the network is greatly unbalanced.

Compressed sensing can resolve the problem of unbalanced load in the network through the so called Compressed Data Gathering (CDG) [59]. Here, the sink node receives m coded packets instead of n packets of original data from nodes and by using CS technique, the sink recovers the original n data readings. In order to do this using CDG, each node in the network multiplies its

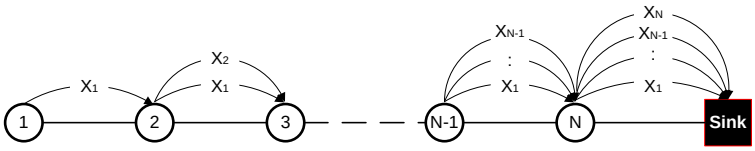


Figure 2.1: Basic Data Gathering

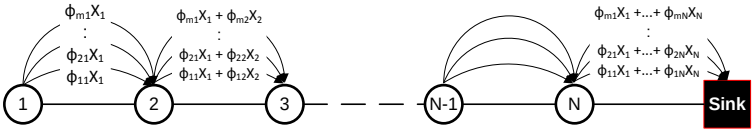


Figure 2.2: Compressed Data Gathering

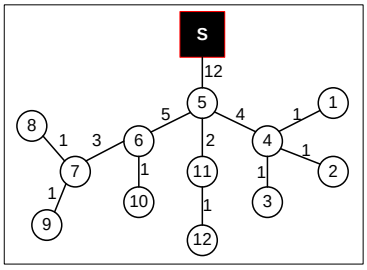


Figure 2.3: Non - Compressed Sensing

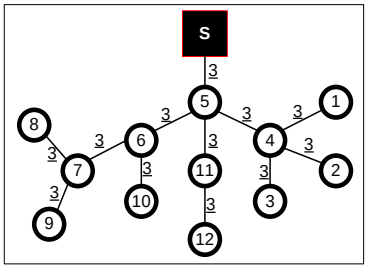


Figure 2.4: Plain - Compressive Data Gathering

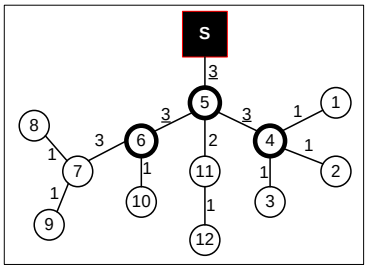


Figure 2.5: Hybrid - Compressive Data Gathering

reading (x_j) into a j column vector of basis matrix Φ (i.e., $\phi_{1j}, \phi_{2j}, ..., \phi_{mj}$) and make a vector of size m. Then, the node waits to receive all same size vectors from its child nodes and adds them to its own vector and transmits the resulting vector to its parent node using m packets. Since in matrix Φ there are n columns and m rows, each column is assigned to one node in the network and each row to one weighted sum. The idea of CDG is illustrated in Figure 2.2. All the sensor nodes transmit m weighted sums z_t , $t = 1, 2, .., m$. To transmit the t^{th} sum z_t, node 1 multiplies its reading x_1 with a random coefficient ϕ_{t1} and sends the product to node 2. Node 2 in turn, after receiving the message from node 1, multiplies its reading x_2 with the random coefficient ϕ_{t2} and adds the two products $\phi_{t1}x_1$ and $\phi_{t2}x_2$ and sends the sum $\phi_{t1}x_1$ + $\phi_{t2}x_2$ to its next node in the network. Each upstream node on the route to the sink adds its product $\phi_{tj}x_j$ to z_t. Finally, the sink node receives the t^{th} sum $z_t = \sum_{j=1}^{n} \phi_{tj}x_j$. When the sink receives all the m sums of z_t, it recovers the original data of all sensors in the network by solving the convex optimization problem as explained before. This scheme of CDG is called Plain Compressive Data Gathering (Plain-CDG). Based on Plain-CDG, all the nodes in the network transmit m packets where $m \ll n$ and all the nodes experience the same transmission load, therefore avoiding the bottleneck nodes problem. Figure 2.4 shows an example of Plain-CDG (with $n = 12$ and $m = 3$).

By inspecting the two mechanisms (Non-CS and Plain-CDG), it is clear that some nodes (especially the leaf nodes) in Non-CS transmit fewer packets than Plain-CDG. Therefore, a hybrid compressed sensing (Hybrid-CDG) is proposed in [61]. The Hybrid-CDG method uses the first method (Non-CS) for nodes that transmit equal or less than m packets and uses the second method (Plain-CDG) for nodes that transmit more than m packets. Figure 2.5 shows an example of

Algorithm 2.1 Greedy Hybrid-CDG (Taken from [81])

Require: $G(V, E), s, k$
Ensure: $Tree, A$
1: **repeat**
2: **for** all the $i \in B(A)$ **do**
3: $A_{test} = A \cup \{i\}; F_{test} = F \backslash \{i\}$
4: $\{cost_{MST}, L\} \leftarrow MST(A_{test})$
5: $\{cost_{SPF}, t\} \leftarrow SPF(F_{test}, A_{test})$
6: **if** $cost_{MST} + cost_{SPF} \leq cost$ **AND** $\min_{l \in L} t_1 \geq k - 1$ **then**
7: $cost = cost_{MST} + cost_{SPF}$
8: $A_{cand} = A_{test}; F_{test} = F_{test}$
9: **end if**
10: **end for**
11: $A = A_{cand}; F = F_{cand}$
12: **until** A unchanged;
13: $Tree = MST(A) \cup SPF(F, A)$
14: **return** $Tree$, A

this method, where the thick circles represent the aggregator nodes that use Plain-CDG and the thin circles represent the forwarder nodes that use Non-CS.

To minimize the network energy consumption through joint routing and compressed aggregation in constructing the Hybrid-CDG tree, the authors of [81] first characterized the optimal solution to the problem and then proved its NP-completeness. Later, the authors proposed a mixed-integer programming formulation to obtain the optimal solution for small scale network and a greedy heuristic that delivers near optimal solution for larger networks. In this thesis, we use their greedy heuristic algorithm to construct the Hybrid-CDG tree, which we refer to as Hybrid-CDG algorithm (Algorithm 2.1) and later we compare our new methods with this algorithm.

Note, the difficulty of Hybrid-CDG algorithm lies in partitioning the network nodes into two sets: 1) aggregator set and 2) forwarder set. The algorithm uses the Minimum-Spanning-Tree (MST) algorithm as the routing topology for

the aggregator set, and uses the shortest path from each node in forwarder set to the nearest node in aggregator set. We refer to the former set as MST and the other set as Shortest-Path-Forest (SPF).

The Hybrid-CDG algorithm is shown in Algorithm 2.1. It starts by assigning MST = S (i.e. MST contains only sink node) and SPF=V/S (i.e. SPF contains all the nodes in the network except the sink node). In each round, the algorithm moves one neighbour node from SPF to MST, if the two criteria satisfy: 1) the action leads to greatest cost reduction, and 2) the leaf nodes of MST has no less than $m - 1$ descendants. Consequently the size of the MST increases and the algorithm stops when there is no any change in MST and SPF. This algorithm is illustrated by two examples in Figures 2.6 and 2.7. In both examples, the thick circles and thick lines represent the aggregator set and MST tree respectively and the thin circles and thin lines respectively represent SPF and shortest path to MST. These joint routing and compressed aggregation shown in both figures are the minimum overall energy consumption that can be represented by Hybrid-CDG method.

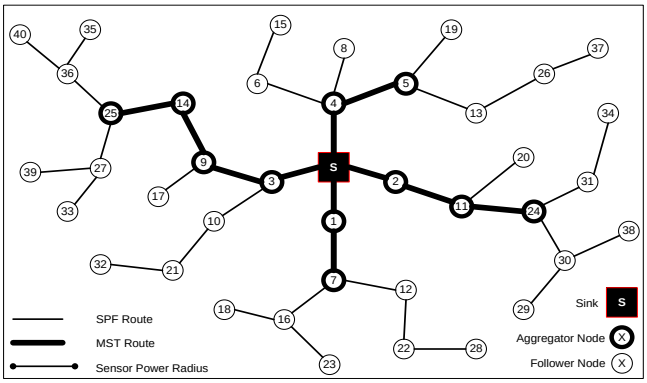


Figure 2.6: Hybrid-CDG construction using greedy algorithm. Sparse network. n=40 and m=4. Network Cost = 92 transmissions. The load for bottleneck nodes 1, 2, 3 and 4 is 4 transmissions.

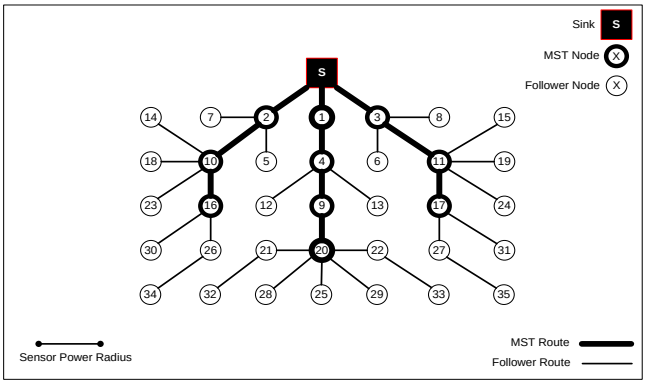


Figure 2.7: Hybrid-CDG construction using greedy algorithm. Dense (mesh) network. n=35 and m=3. Network Cost = 59 transmissions. The bottleneck loads for nodes 1,2 and 3 is 3 transmissions.

2.5 Sparse Random Projections

Data gathering with sparse random projection was first introduced in [73], where m nodes are selected at random to gather m weighted sums in the network. Each projection node gathers one weighted sum for the sink and each row of the basis matrix Φ is assigned to one projection node. First, a projection node t asks the nodes whose coefficients ϕ_{tj} are non-zero to send their data readings by one packet each through shortest path to it and after receiving all the packets, the projection node gathers all the data with its own data reading and sends the result through shortest path to the sink by a single packet. Similarly, all the other projection nodes gather and send the weighted sums to the sink. This process is illustrated for one projection node in Figure 2.8. In this figure, node 5 initializes the projection by sending requests to nodes 11, 15, and 20, where their $\phi_{tj} \neq 0$. These nodes reply to the request by sending their data readings x_j to node 5 (marked by arrows). Then, node 5 computes $\sum_{j=1}^{n} \phi_{tj} x_j$ and transmits it to the sink.

The authors in [73] claimed that there is a trade-off between the sparsity of the projections (number of nonzero coefficients in each row of the matrix

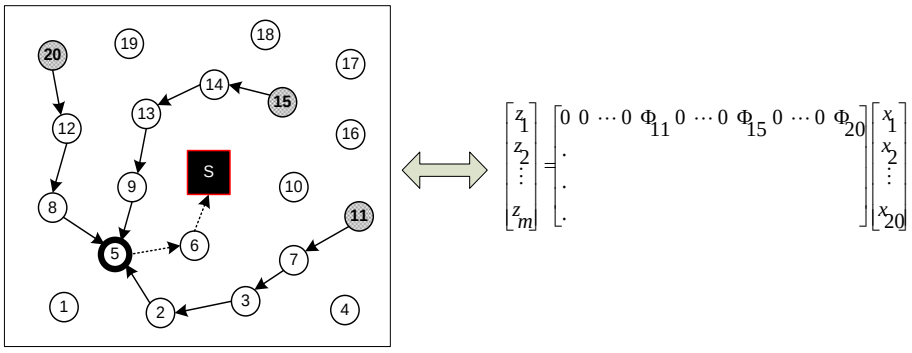


Figure 2.8: Illustration of data gathering in sparse random projection for one projection node with its dependency to matrix Φ.

Φ) and the number of projections needed (number of rows in matrix Φ). The authors have shown that the entries of the sparse projection matrix Φ can be obtained as follows:

$$\phi_{tj} = \sqrt{s} \begin{cases} +1, & \text{with prob. } \frac{1}{2s} \\ 0, & \text{with prob. } 1 - \frac{1}{s} \\ -1, & \text{with prob. } \frac{1}{2s} \end{cases} \tag{2.8}$$

That is $\phi_{tj} = \sqrt{s}$ with probability $\frac{1}{2s}$, $\phi_{tj} = 0$ with probability $1 - \frac{1}{s}$, and $\phi_{tj} = -\sqrt{s}$ with probability $\frac{1}{2s}$. They assumed the entries within each row are four-wise independent, while the entries across different rows are fully independent. This limited independence assumption allows each projection vector to be pseudo-random generated and stored in a small space. The parameter s controls the degree of sparsity of the projections. Thus, if $\frac{1}{s} = 1$, the random matrix Φ has no sparsity; and if $\frac{1}{s} = \frac{\log n}{n}$, the expected number of non-zeros in each row of the random matrix Φ is $\log n$. The sufficient number of sparse projections to recover an approximation with error comparable to the best k-sparse representation of data is $m = O(sM^2k^2 \log n)$, under the condition that the original data X satisfies a peak-to-total energy $\|X\|_\infty / \|X\|_2 \leq M$, where

$\|X\|_\infty = \max_{1\leq i\leq n}(|x_i|)$ and $\|X\|_2 = \sqrt{\sum_i |x_i|^2}$. It was shown that if $\frac{n}{s} = \log^2 n$, $sM^2 = O(1)$, if $\frac{n}{s} = \log n$, $sM^2 = O(\log n)$ and if $\frac{n}{s} = 1$, $sM^2 = O(\log^2 n)$.

Note that according to RIP [19], if the number of projections (m) or the sparsity of the projection (number of non-zeros in a row of Φ) are slightly below the minimum requirement, the sink may still reconstruct the original reading vector X with lower performance (approximation solution) if the coefficients X exhibit the power law decay [19]. The authors of [73] in their numerical results showed that when $n = 2024$, $m = 200$ and the average number of non-zeros in a row of $\Phi_{m\times n} = \frac{n}{s}$ with sparsity $\frac{1}{s} = \frac{\log(n)}{n}$, the approximation error of the data $\frac{\|X-\overline{X}\|_2^2}{\|X\|_2^2}$ is below 0.35.

In this thesis, to distribute the non-zero coefficients more evenly in the matrix Φ and make each projection as sparse as possible, the number of non-zero coefficients in each row of the matrix Φ is chosen as $\lceil \frac{n}{m} \rceil$ such that none of the columns in Φ has all-zero entries. Since the sparsity (number of non-zero coefficients) and number of projections (m) depend on the k-sparsity Fourier transform representation of sensors' readings (as we mentioned in section 2.3), the random sample matrix Φ presented here satisfies all the conditions required to fully recover the original data readings at the sink using the compressive sensing technique.

Chapter 3

Projection Based Compressive Data Gathering (PCDG)

In this chapter we present efficient algorithms to solve the problem of constructing aggregation trees for forwarding the compressed data to the sink and we formulate a mixed integer linear program (MILP) to solve the problem. We show that our algorithms have outstanding performance and order of magnitude faster than the optimal model.

3.1 Motivation

The Hybrid-CDG [61] method introduced in Chapter 2 has a major drawback; that is, the nodes near the sink which do aggregation consume more power than the leaf nodes, or far away from the sink, which do forwarding only. Therefore, the energy consumption load is still not properly balanced throughout the entire network. Furthermore, each node in Hybrid-CDG does the same job over and over at different iterations and its task as aggregator or forwarder never changes. In distributed sparse random projections algorithm [73] (refer

to Section 2.5), the authors, by choosing m projection nodes with probability $\frac{m}{n}$ at random, distributed the load more evenly throughout the network compared to Hybrid-CDG. But their algorithm cannot guarantee a minimal overall network cost, because for each random projection, the algorithm requires a large number of transmissions between the nodes to collect the data at a projection node with no en-route data gathering.

To overcome the drawbacks of the two algorithms above and improve the overall cost and load balancing in the network, we present a new data aggregation scheme which leverages the advantages of the above two methods. Our method at first uses the same strategy of the algorithm in [73] by choosing at random m nodes to do the projection in the network. Each projection node however gathers one sample measurement (weighted sum) from all the nodes in the network using compressed data gathering (CDG) and send the weighted sum in one packet through a shortest path to the sink. When the sink receives all the m weighted sums, it reconstructs the original data for all the network nodes according to compressive sensing technique.

3.2 Minimum Spanning Tree Projection (MSTP)

Selecting m projection nodes among all sensors and generating a good random basis matrix Φ for data compression are the two most critical issues that may affect the efficiency of our method. There are two different ways for selecting the m projection nodes;

- One way is to follow a decentralized approach as in [73], where among n nodes in the network, m projection nodes are selected at random with probability $\frac{m}{n}$. This method does not guarantee that exactly m projection

nodes will be selected since each node has an $\frac{m}{n}$ probability to be selected. Therefore, more or fewer nodes could be selected at random.

- Alternatively, one may fix the m nodes that do the projection in advance. Note that later, after distributing the nodes in a region, the sink node may change the projection nodes by sending a notification massage to nodes that have been re-selected as projection nodes.

In [73], the authors proposed to select at random the position of the projection nodes in a distributed manner and this indeed allowed equally each node to participate in doing the projection by periodically switching turns; this is advantageous, particularly because each projection node in their method performs more activities (i.e., the aggregation process) than the normal nodes and therefore consumes more energy; hence, by taking turns in doing projection, all nodes will participate and the consumption load will be uniformly distributed across the network. In our work, however, the m projection nodes are known and randomly selected in advance and there is no need to continuously change their roles, since in our proposed method the projection nodes need not do any extra effort which may consume additional power; they only act as the initialization point for each projection. However, it is understood that their positions in the network may affect the efficiency of the aggregation algorithm. Therefore, one may attempt to find the fixed optimal position of the projection nodes. We will generate the optimal position of projection nodes in Section 3.5.

In our MSTP method, to make each random projection as sparse as possible, we make the number of non-zero in each row of the projection matrix Φ equals to $\lceil \frac{n}{m} \rceil$, with a condition that none of the columns in Φ has full zero entries. In this way, we distribute the non-zero coefficients more even in the matrix Φ.

As an example, consider the 4×6 matrix shown in (B.5). This matrix is constructed for a network of size $n = 6$ and $m = 4$ random projections. Therefore, the number of non-zero in each row is $\lceil \frac{n}{m} \rceil = \lceil \frac{6}{4} \rceil = 2$ and none of the columns in (B.5) has all its elements zero. The non-zero entries of ϕ_{tj} are selected at random with i.i.d entries from the normal distribution with mean zero and variance $\frac{1}{m}$.

$$\Phi = \begin{pmatrix} \phi_{11} & 0 & 0 & \phi_{14} & 0 & 0 \\ 0 & \phi_{22} & \phi_{23} & 0 & 0 & 0 \\ 0 & \phi_{32} & 0 & 0 & \phi_{35} & 0 \\ 0 & 0 & 0 & \phi_{44} & 0 & \phi_{46} \end{pmatrix} \tag{3.1}$$

There are three ways, as we discussed in section 2.1.2, to construct the matrix Φ;

- First, each projection node generates the pseudo-random row sequence of matrix Φ. The authors in [73] used a similar method to generate the row vector of matrix Φ, but this method cannot guarantee the full distribution of non-zero coefficients in the entire matrix Φ, as we discussed above for the properties of our matrix Φ.

- Alternatively, the sink node can generate the matrix Φ and distribute it to all nodes in the network. However, this will increase the transmission cost for our network.

- Alternatively, a proper matrix Φ could be generated in advance and it will be stored in the memory of each node before distributing the nodes in a region, as a *pre-known* random basis Φ.

In this thesis, we use a pre-known matrix Φ as our *pre-known* projection nodes. The process of our MSTP method works as follows. Consider a WSN of size n with correlated reading values x_j, $j = 1, ..., n$. According to the theory of CS, the sink needs to receive only m weighted sums (sample measurements) to recover the readings from all the nodes (i.e., $z_t = \sum_{j=1}^{n} \phi_{tj} x_j$, $t = 1, .., m$). Since we have m projection nodes in the network, each projection node j gathers one sample measurement from all the nodes in the network. To do that, each row vector of the matrix Φ is assigned to one projection node. The size of each row vector of Φ is n (related to n nodes in the network), and the nodes whose coefficient $\phi_{tj} \neq 0$ represent the interest nodes for that projection node t. When projection node t retrieves its interest nodes from the matrix Φ, it uses the Minimum-Spanning-Tree (MST) algorithm (plus Breadth-First-Search (BFS) algorithm to find shortest paths if needed) to construct a tree t, which connects all interest nodes to the projection node. To construct the tree, projection node t first considers itself as a one-node tree. Next it expands the tree, using the MST algorithm, adding all interest nodes that can be reached directly without any multi-hop. Then, if there are more interest nodes that have not been added to the tree, using the BFS algorithm from all remaining interest nodes, the nearest one through the shortest path will be added to the tree. Next, if still more interest nodes remain, the algorithm continues using the same strategy (using MST and BFS algorithms) until it connects all the interest nodes to the tree. The projection node t represents the root for the current tree. Then, according to the routing tree, each node knows its parent and child nodes and, similar to CDG, it multiplies its reading x_j with its coefficient ϕ_{tj} and gathers its data $\phi_{tj} x_j$ with its descendants and sends the weighted sum to the parent node (as illustrated in Figure 3.1 for one projection). When the root (projection

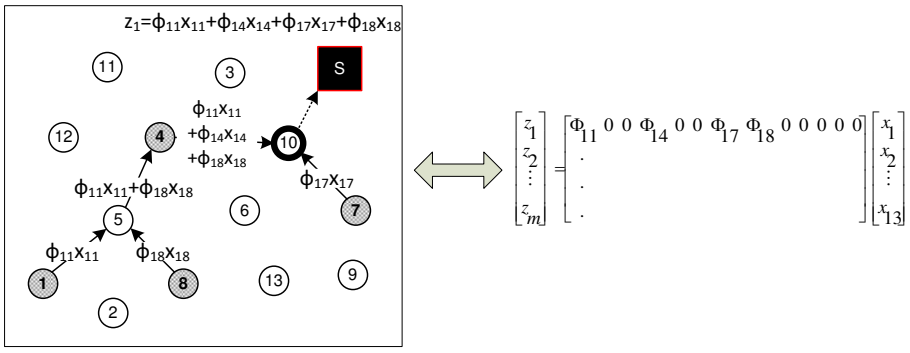


Figure 3.1: Illustration of Compressive Data Gathering.

node t) receives the weighted sum from its children, it transmits this sum $\sum_{j=1}^{n} \phi_{tj} x_j$ in one packet through the shortest path to the sink. In total, there are m such trees in the network. Each tree represents one projection (weighted sum or sample measurement z_t).

The details of MSTP method for projection nodes are given in Algorithm 3.1. The algorithm for each projection node t among the set P starts by retrieving the nodes that their coefficients in random basis matrix Φ_t are not zero and puts them in interest nodes list Int_t. Next, the algorithm puts the projection node t into MST_t list as well as into temporary queue PQ. While the PQ is not empty, the algorithm removes the top node from PQ queue and puts its neighbour nodes in MST_t and PQ and removes them from Int_t if they are in Int_t. This while loop in lines 7-14 illustrates the expansion of tree MST_t , which connects all the nodes in Int_t that can be reached directly from projection node t without multi-hop. In lines 15-28, the algorithm adds the remaining interest nodes (if available) to the MST_t. To do that, the algorithm first finds the shortest paths from all nodes in Int_t to MST_t and stores the shortest one in $Shortest_{Path}$ list. Later, the algorithm puts all the nodes from $Shortest_{Path}$ into MST_t and PQ and removes them from Int_t. Now, while the PQ is not empty, the algorithm repeats the steps in lines 8-13. This while loop in lines 15-28

Algorithm 3.1 MSTP

Require: P, Φ
Ensure: $MST_t, SP_t, t = 1, 2, ..., m$
1: **for** all $t \in P$ **do**
2: **for** all $j \in \Phi_{tj} \neq 0$ **do**
3: $Put(Int_t, j)$
4: **end for**
5: $Put(MST_t, t)$
6: $Put(PQ, t)$
7: **while** $!Empty(PQ)$ **do**
8: $CNode = Rem(PQ)$
9: **if** $Adj(CNode \in Int_t)$ **then**
10: $Put(MST_t, CNode)$
11: $Put(PQ, CNode)$
12: $Rem(Int_t, CNode)$
13: **end if**
14: **end while**
15: **while** $!Empty(Int_t)$ **do**
16: **for** all $h \in Int_t$ **do**
17: $Path(h) \leftarrow$ Find shortest path from h to MST_t using BFS algorithm
18: **if** $Shortest_{Path} > Path(h)$ **then**
19: $Shortest_{Path} = Path(h)$
20: **end if**
21: **end for**
22: $Put(MST_t, Shortest_{Path})$
23: $Put(PQ, Shortest_{Path})$
24: $Rem(Int_t, Shortest_{Path})$
25: **while** $!Empty(PQ)$ **do**
26: Execute steps 8-13
27: **end while**
28: **end while**
29: $SP_t \leftarrow$ Find shortest path from t to the sink
30: **end for**

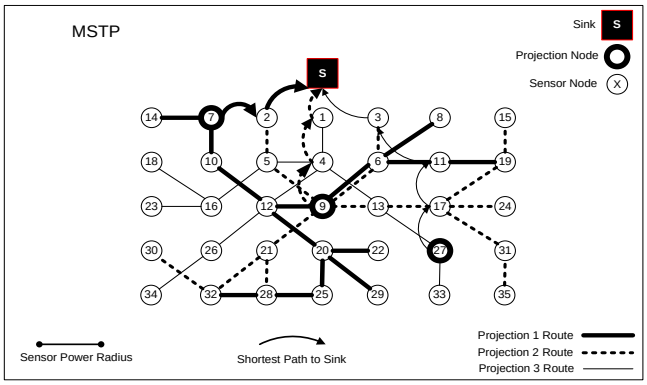


Figure 3.2: MSTP. n=35 and m=3. There are three projection nodes with three connected trees shown with different lines. Network Cost = 49 transmissions. The number of transaction for bottleneck nodes are as follows: Node 1: 3, Node2: 2, Node3: 1 packet transmissions.

will be repeated until the Int_t becomes empty. At the end, the algorithm terminates by finding the shortest path from each projection node to the sink and stores the results into SP_t. Upon termination, the algorithm returns MST_t and SP_t for all projection nodes.

We illustrate the MSTP method using the example shown in Figure 3.2. In this figure, nodes 7, 9 and 27 (shown by thick circles) are assigned to initialize the three projections. Suppose for projection one, the interest nodes (nodes with non-zero coefficients in Φ_1) are $7, 8, 9, 10, 11, 14, 19, 20, 22, 25, 29, 32$. Projection node 7 constructs a tree connecting all these nodes together. The algorithm starts by creating MST_1 that contains only node 7. Next, as explained above the algorithm expands the MST_1 using the MST algorithm, joining nodes 14 and 10 which can be reached directly without multi-hop. At this stage, the MST_1 contains three nodes 7, 10 and 14 and now, none of the neighbours of MST_1 are interest nodes. In this situation, the algorithm finds node 9 as the nearest interest node to MST_1 by exploiting BFS algorithm from all remaining interest nodes to the tree. It connects node 9 through node 12 to the tree MST_1. Now, the MST_1 tree contains 5 nodes (7, 14, 10, 12, 9). Again by

using the MST algorithm, the algorithm connects nodes 20, 22, 29 and 25 to MST_1 tree. It will expand the MST_1 until it covers all the nodes whose coefficients in ϕ_{1j} vector are non-zeros. At the end, MST_1 tree becomes as illustrated in Figure 3.2 by thick lines. Similarly MST_2 and MST_3 represent the trees for projections 2 and 3 rooted at nodes 9 and 27 respectively. MST_2 and MST_3 are marked by dashed and thin lines respectively in Figure 3.2.

After constructing the MST_t, each node knows its parent and child nodes and it will wait to collect all the data from its child nodes if it is not a leaf node. Each node in MST_t multiplies its data with its ϕ_{tj} and adds it with its child node data and transmits it to its parent node until the root node t receives the aggregate data $\sum_{j=1}^{n-1} \phi_{tj} x_j$ from all n nodes in the network. The root node (projection node) in turn adds $\sum_{j=1}^{n-1} \phi_{tj} x_j$ with its own data $\phi_{tt} x_t$. Note that, in this MST_t all nodes throughout the network use only one packet to send the aggregated data (weighted sum) to projection node t. Finally, at the last step, each projection node t transmits the weighted sum $z_t = \sum_{j=1}^{n} \phi_{tj} x_j$ in one packet to the sink using the shortest path as shown in Figure 3.2 (curved arrows). Consequently, in general there are m projection nodes and m packets will be transmitted to the sink node as in (B.6). Finally, the sink node recovers the original data as we explained in Section 2.1 using compressive sensing.

$$z_t = \sum_{j=1}^{n} \phi_{tj} x_j, \qquad t = 1, 2, ..., m \tag{3.2}$$

Time Complexity Analysis

We consider two cases to analyze the time complexity of the MSTP algorithm; In the best case, all the interest nodes could be connected directly without

multi-hop using the Minimum-Spanning-Tree (MST) algorithm. Second, in the worst case, we assume none of the interest nodes can be connected using the MST algorithm and for each interest node, the MSTP algorithm has to use the Breadth-First-Search (BFS) algorithm from remaining interest nodes which are disconnected from the tree to connect only one interest node to the tree MST_t.

Best case analysis: The algorithm at initialization checks the neighbors of a projection node and puts the ones which are interest nodes into a priority queue PQ. This runs in $O(d\rho \log \rho)$, where d is the average nodal degree of a projection node and $O(\rho)$ is needed to check whether the neighboring node is among the interest nodes and $O(\log \rho)$ to insert it into the sorted priority queue PQ, which may contain a maximum of ρ elements [15]. Next, the algorithm pops the top element of the PQ and finds its neighbors and repeats the same steps as before. This procedure is repeated until the priority queue PQ is empty, which takes $O(\rho)$ at most. Therefore, in total, the running time for m projection nodes is $O(md\rho^2 \log \rho)$.

Worst case analysis: The MSTP algorithm first finds the shortest path from each interest node to a projection node and adds the nearest one to the tree MST_t. The shortest path for one interest node takes $O(dn \log n)$. Therefore, the time complexity to find the shortest paths for ρ interest nodes to the nearest node in the tree MST_t is $O(\rho dn \log n)$. Now, MST_t contains two nodes and the rest $(\rho - 1)$ interest nodes remain to be connected to the tree. In the worst case, we assume that none of the interest nodes are connected to the tree MST_t using the MST algorithm and the MSTP algorithm consequently uses BFS algorithm among remaining interest nodes to connect one interest node to the tree MST_t. Consider

we are at stage where the tree MST_t contains $k+1$ nodes and $(\rho - k)$ interest nodes remain to be connected to the tree. At this stage, the time complexity to find the nearest $(\rho - k)$ interest nodes to tree MST_t is $O((\rho - k)dn \log n)$ as we explained above. Therefore, as we may have $(\rho - 1)$ stages, the total complexity for all stages plus the initial stage is $O(\rho dn \log n + \sum_{k=1}^{\rho-1}[(\rho - k)dn \log n]) \cong O(d\rho^2 n \log n)$ and for m projection nodes the time complexity is $O(md\rho^2 n \log n)$.

In fact, the actual time complexity of the MSTP algorithm could be much lower than the worst case, since at each step, all the remaining interest nodes may not be multi-hop away from the tree MST_t and thus every time the MSTP algorithm may not use multiple BFS algorithm but rather it uses the MST algorithm instead. On average, the MSTP algorithm connects most of the interest nodes to the tree MST_t by using the MST algorithm more than consecutive BFS algorithm, and hence the time complexity of the MSTP algorithm is closer to the best case than the worst case.

3.3 Extended Minimum Spanning Tree Projection (eMSTP)

To improve our method, we have made a slight change to the construction of MST_i. Instead of making the projection node i as a root of the MST_i, we extend the algorithm to add the sink node to MST_t and make it the root of the tree. Now the projection node t will not be the one that collects and computes the data aggregation from all the nodes in the network. Projection node t will only act as the initialization point for constructing the MST_t, and the sink node will receive the entire data aggregation directly from all the nodes. In

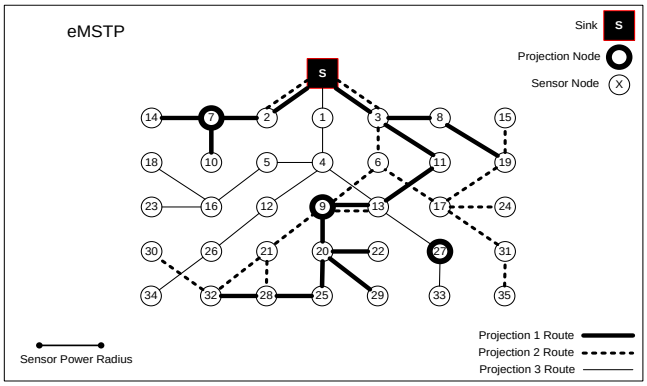


Figure 3.3: eMSTP. n=35 and m=3. There are three projection nodes with three connected trees shown with different lines. Network Cost = 43 transmissions. The number of transaction for bottleneck nodes are as follows: Node 1: 1, Node2: 2, Node3: 2 packet transmissions.

this way, we will eliminate the traffic load resulting from transmitting the corresponding projection packets from the projection nodes to the sink. The steps of eMSTP are shown in Algorithm 3.2 and its time complexity is same as the MSTP algorithm. The same example of MSTP is shown for eMSTP in Figure 3.3. In this figure, for projection 1, node 7 constructs the tree MST_1 covering all nodes in the network that their $\phi_{1j} \neq 0$, as well as the sink node. The MST_1 is shown by thin line in figure 3.3. The sink node receives directly the weighted sum 1 (sample measurement z_1) from all the nodes in MST_1, and similarly, it receives weighted sum 2 and 3 (i.e., z_2 and z_3) from MST_2 and MST_3 respectively. Consequently, the sink has all the weighted sums $z_t = \sum_{j=1}^{n} \phi_{tj} x_j$, $t = 1, 2, ..., m$, which are needed to recover the original data for all the nodes in the network.

Algorithm 3.2 eMSTP

Require: P, Φ
Ensure: $MST_t, SP_t, t = 1, 2, ..., m$
1: **for** all $t \in P$ **do**
2: **for** all $j \in \Phi_{tj} \neq 0$ **do**
3: $Put(Int_t, j)$
4: **end for**
5: $Put(Int_t, sink)$
6: $Put(MST_t, t)$
7: $Put(PQ, t)$
8: **while** $!Empty(PQ)$ **do**
9: $CNode = Rem(PQ)$
10: **if** $Adj(CNode \in Int_t)$ **then**
11: $Put(MST_t, CNode)$
12: $Put(PQ, CNode)$
13: $Rem(Int_t, CNode)$
14: **end if**
15: **end while**
16: **while** $!Empty(Int_t)$ **do**
17: **for** all $h \in Int_t$ **do**
18: $Path(h) \leftarrow$ Find shortest path from h to MST_t
19: **if** $Shortest_{Path} > Path(h)$ **then**
20: $Shortest_{Path} = Path(h)$
21: **end if**
22: **end for**
23: $Put(MST_t, Shortest_{Path})$
24: $Put(PQ, Shortest_{Path})$
25: $Rem(Int_t, Shortest_{Path})$
26: **while** $!Empty(PQ)$ **do**
27: Execute steps 9-14
28: **end while**
29: **end while**
30: **end for**

3.4 Comparison and Numerical Results of MSTP and eMSTP

In this section, first we compare our two methods MSTP and eMSTP with the three mechanisms (Non-CS, Plain-CDG and Hybrid-CDG) discussed in Section 2.2 on small network with few nodes, and then we present the numerical results.

3.4.1 Comparison

For comparison, we consider sensor nodes that measure the temperature in a field environment. We assume that the reading values have sparse representation in Discrete Fourier Transformation (DFT). We take the number of transmissions as our unit of power consumption in sensor nodes, because the other tasks of sensors are identical and their power consumption is the same. The more transmissions a sensor does, the more energy it consumes and the faster it dies out. Here, in our numerical evaluation, we are interested in two criteria, first minimizing the overall number of transmissions (transmission cost) and second, balancing the transmission load throughout the network. We consider two network topologies; 1) grid network with 35 nodes excluding the sink node at the top with $m = 3$, and 2) arbitrary network with 40 nodes plus sink node at the center with $m = 4$. For the former topology, Hybrid-CDG in Figure 2.7 costs overall 59 transactions to send all the sample data to the sink and the bottleneck nodes 1, 2 and 3 transmit three packets each. MSTP in Figure3.2, costs 49 transactions, which is less than Hybrid-CDG and the bottleneck nodes 1, 2 and 3 respectively transmit 3, 2 and 1 packets. As it is obvious, not only the overall transmission in MSTP is less than Hybrid-CDG,

but also the load on bottleneck nodes is lower. This implies that the MSTP method helps in increasing the lifetime of the network. eMSTP on the other hand (Figure 3.3) transmits in total 43 packets and the bottleneck nodes 1, 2 and 3 transmit 1, 2 and 2 packets respectively. For the other topology (arbitrary network), refer to figure 2.6 for Hybrid-CDG, Figure 3.4 for MSTP and Figure 3.5 for eMSTP. The figures in this arbitrary network topology example show a higher performance of eMSTP and MSTP over Hybrid-CDG with respect to the overall network cost and load balancing throughout the network.

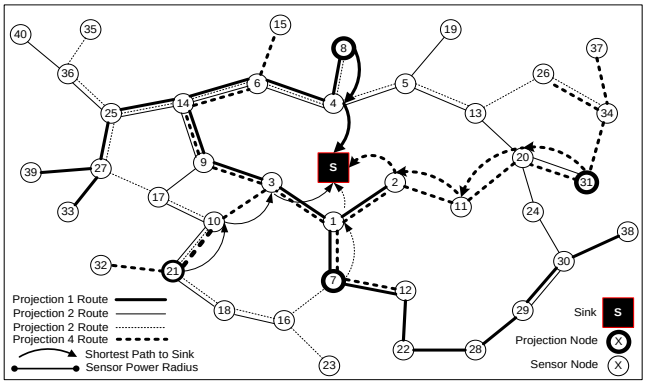


Figure 3.4: Sparse MSTP network (n=40 and m=4). Network Cost = 83 transmissions. The loads for bottleneck nodes are; Node1: 3, Node2: 3, Node3: 3 and Node4: 4 transmissions.

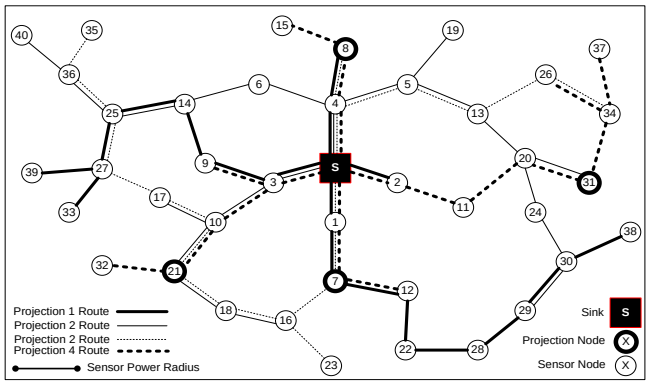


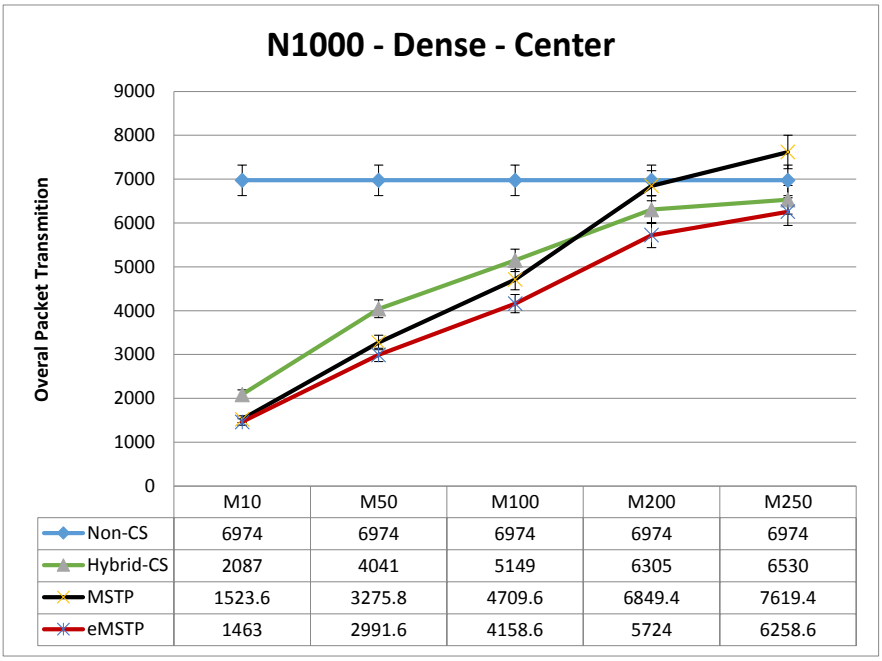
Figure 3.5: Sparse eMSTP network (n=40 and m=4). Network Cost = 75 transmissions. The loads for bottleneck nodes are; Node1: 3, Node2: 2, Node3: 4 and Node4: 4 transmissions.

3.4.2 Numerical Results

We conducted several studies to evaluate the performance of our two methods MSTP and eMSTP with Hybrid-CDG, Plain-CDG and Non-CS. We used Java developer to implement the algorithms. We considered four network topologies: 1) Dense Network with the sink node at the center, 2) Dense Network with the sink node at the top, 3) Sparse network with the sink at the center, and 4) Sparse network with the sink at the top. For each topology, we have considered four network sizes: 1) 100 nodes, 2) 200 nodes, 3) 500 nodes, and 4) 1000 nodes and for each we used different number of random samples m with compression ratio of around 20 to 4 (sizes around 5% to 25% of total nodes n). The nodes are distributed randomly in a specific fixed region and the density of the network is aligned by increasing or decreasing the communication power radios of each sensor nodes. In this sub-section, we first evaluated the overall data transmissions in the network over an average of ten runs (with 95% confidence interval) and later, we evaluated the distribution loads across the sensor nodes. We run our program on CPU with Intel Core $i7$ processor, 2.67 GHz speed, 6 GB memory ram and 64-bit windows operating system. The program for MSTP or eMSTP with a network of size 1000 nodes, in average for different topologies, run in 20 seconds time, and for Hybrid-CDG run in about 35 minutes. As we analyzed in section 5.4.3, the time complexity of the MSTP (Algorithm 3.1) or eMSTP (Algorithm 3.2) is $O(md\rho^2 \log \rho)$ in the best case and $O(md\rho^2 n \log n)$ in the worst case, where as the Hybrid-CDG (Algorithm 2.1) runs in $O((n-k)^2 n^2 + n^3)$ (as shown in [81]).

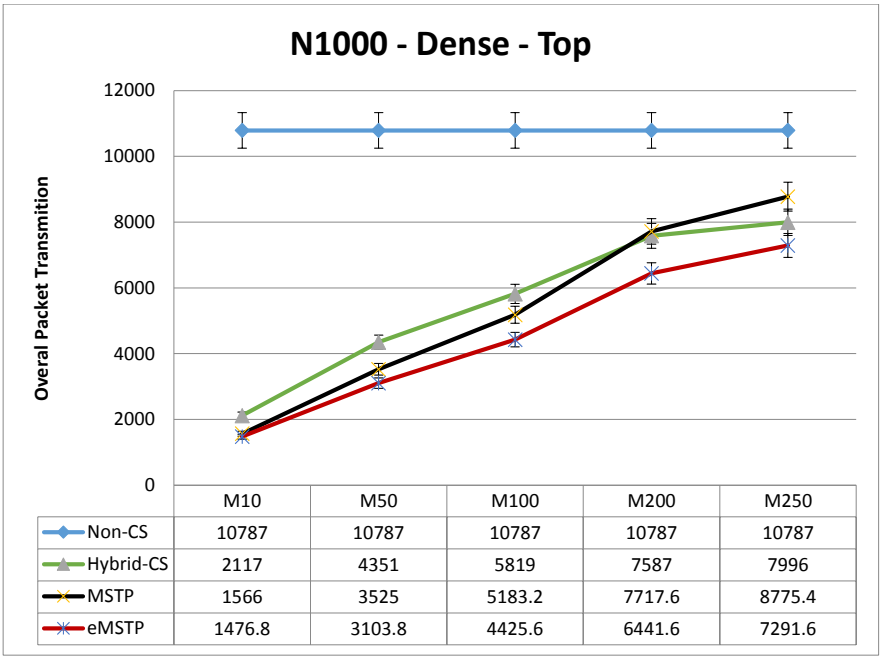
Overall Network Data Transmissions

We present the average results of five runs of MSTP and eMSTP with different random basis Φ and different projection nodes. After obtaining the results of all network topologies with different sizes ($n = [100, 200, 500, 1000]$), the results showed almost the same curves. Hence, we only present results for one network size. In Figures 3.6, 3.7, 3.8 and 3.9, for all topologies, Non-CS has almost the highest overall packet transmissions in the network and Hybrid-CDG comes at second highest transmission and then MSTP and eMSTP come at third and fourth position respectively. Furthermore MSTP falls behind the Hybrid-CDG when the compression ratio goes below 6 or 5 (size of random sample m goes above 15% or 20% of total node size n). We note that in networks where the distance from leaf nodes to the sink is too short, as in Figure 3.6 and Figure 3.8 (network with center sink), MSTP dramatically collapses and its efficiency with respect to the overall network cost drops below Non-CS when the value of m is above 20% of total nodes n. This is because, when the value of m is high, more projection nodes are needed, and for each projection node, the weighted sum is transmitted in shortest path through multi-hop nodes to the sink, therefore, the overall number of transmissions in the network increases. Our extended method eMSTP however, always has smaller overall number of transmissions and outperforms all the other methods for all tested network topologies (as shown in the figures). In case of MSTP, when the compression ratio is high, the algorithm performs better than Hybrid-CDG, Plain-CDG and Non-CS and performs worse than Hybrid-CDG in overall network transmission in case of low compression ratio. However, it outperforms Hybrid-CDG in balancing the transmission load which is the main advantage of CS as we will discuss it in the coming sub-section.



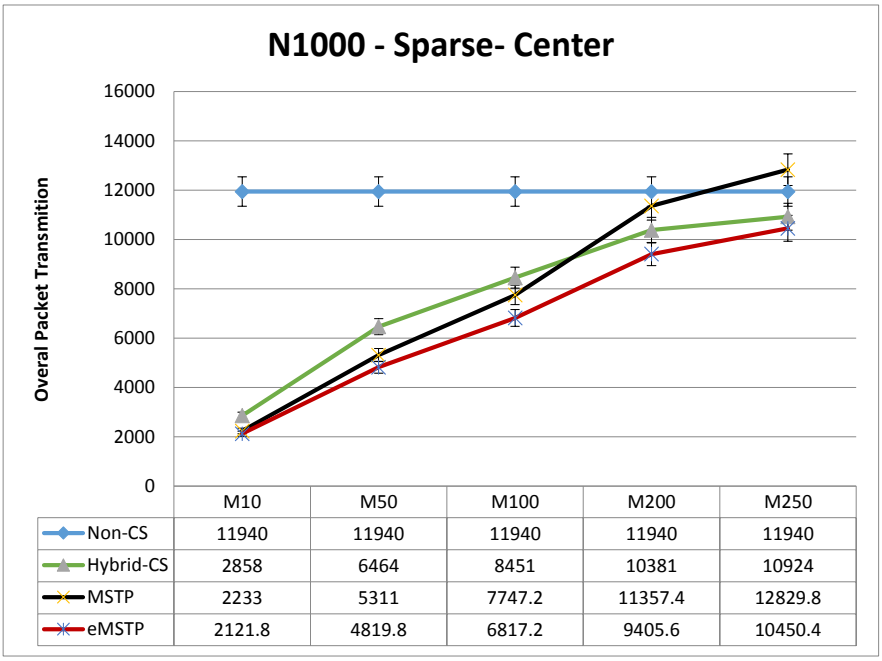
	M10	M50	M100	M200	M250
Non-CS	6974	6974	6974	6974	6974
Hybrid-CS	2087	4041	5149	6305	6530
MSTP	1523.6	3275.8	4709.6	6849.4	7619.4
eMSTP	1463	2991.6	4158.6	5724	6258.6

Figure 3.6: Overall data transmissions in a dense network with 1000 nodes and sink at the center.



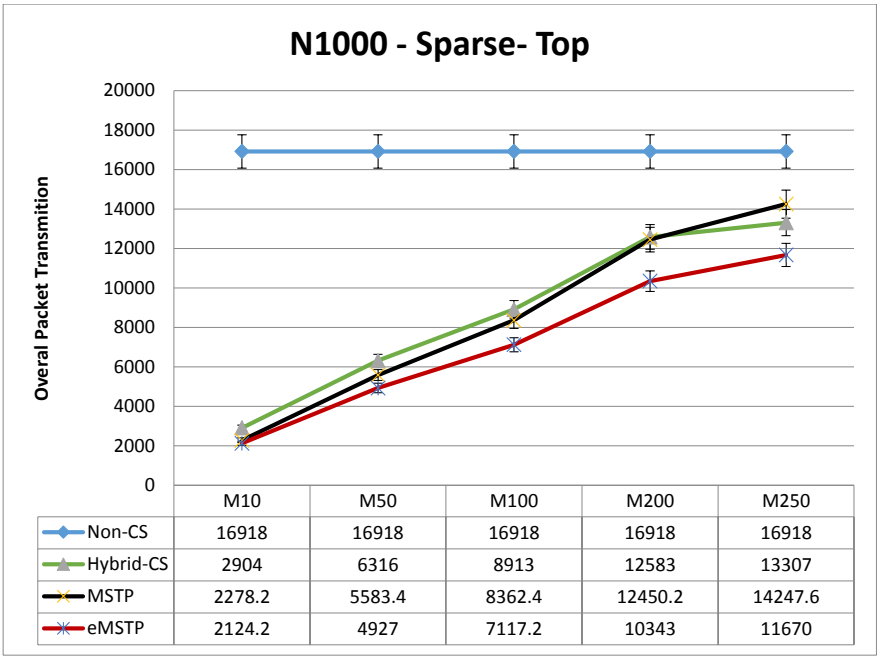
	M10	M50	M100	M200	M250
Non-CS	10787	10787	10787	10787	10787
Hybrid-CS	2117	4351	5819	7587	7996
MSTP	1566	3525	5183.2	7717.6	8775.4
eMSTP	1476.8	3103.8	4425.6	6441.6	7291.6

Figure 3.7: Overall data transmissions in a dense network with 1000 nodes and sink at top.



	M10	M50	M100	M200	M250
Non-CS	11940	11940	11940	11940	11940
Hybrid-CS	2858	6464	8451	10381	10924
MSTP	2233	5311	7747.2	11357.4	12829.8
eMSTP	2121.8	4819.8	6817.2	9405.6	10450.4

Figure 3.8: Overall data transmissions in a sparse network with 1000 nodes and sink at the center.



	M10	M50	M100	M200	M250
Non-CS	16918	16918	16918	16918	16918
Hybrid-CS	2904	6316	8913	12583	13307
MSTP	2278.2	5583.4	8362.4	12450.2	14247.6
eMSTP	2124.2	4927	7117.2	10343	11670

Figure 3.9: Overall data transmissions in a sparse network with 1000 nodes and sink at top.

Load Distribution and Load Balance

The most important advantage of CS in WSN is its ability to solve the bottleneck problem and to distribute the load more evenly throughout the network. Figures 3.6, 3.6, 3.6, 3.6, 3.10, 3.11 and 3.12 represent the un-metric Probability Density Function (PDF) of transmission per node for the entire network. From the figures, we can see that most of the nodes in MSTP and eMSTP in average transmit fewer packets in comparison to the Hybrid-CDG and Non-CS methods. This indeed shows that our two new methods MSTP and eMSTP distribute the load more evenly throughout the network and hence increases its lifetime. Also, we observe that, as the compression ratio (number of random sample measurements to total nodes) decreases, the performance of CS methods increases, since fewer transmissions are needed.

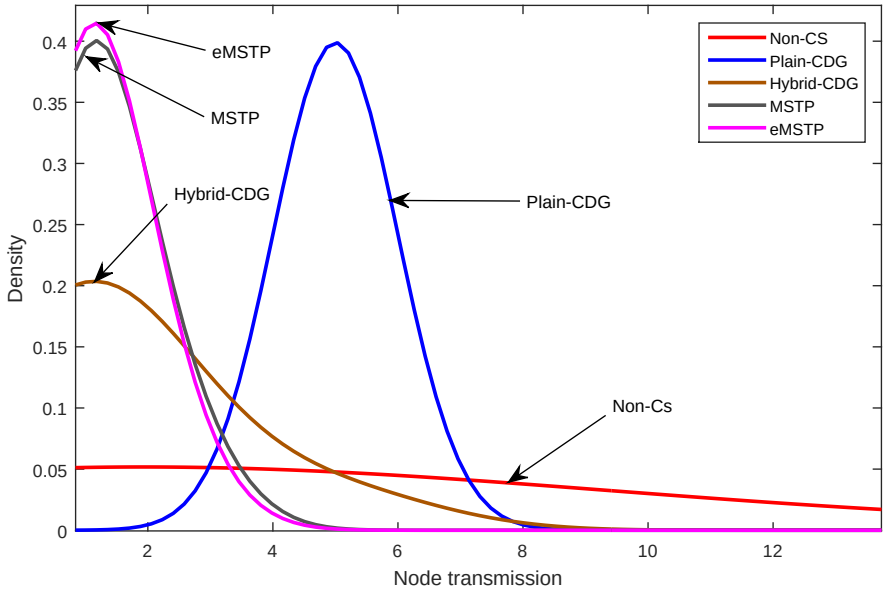


Figure 3.10: PDF for average transmission in dense network with 100 nodes, 5 random samples and sink at the center.

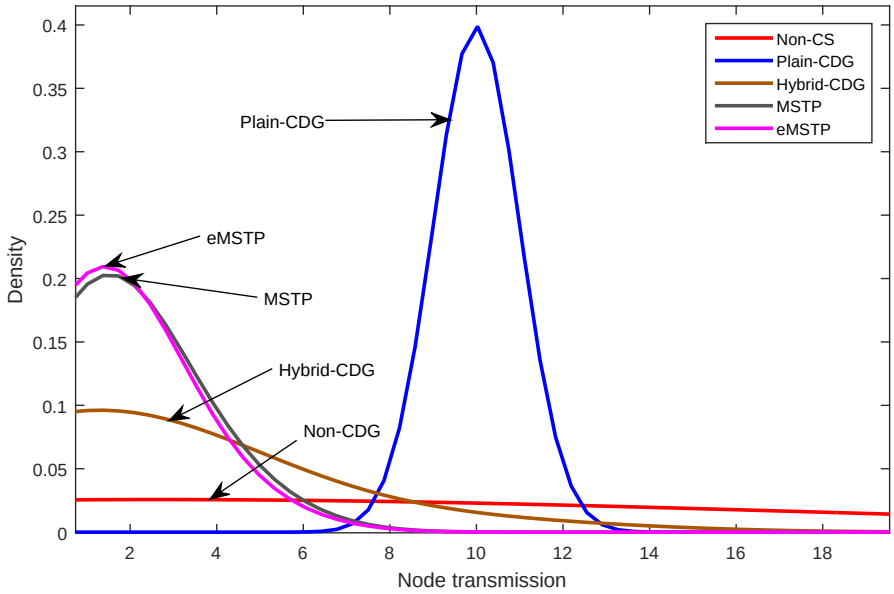


Figure 3.11: PDF for average transmission in dense network with 100 nodes, 10 random samples and sink at top.

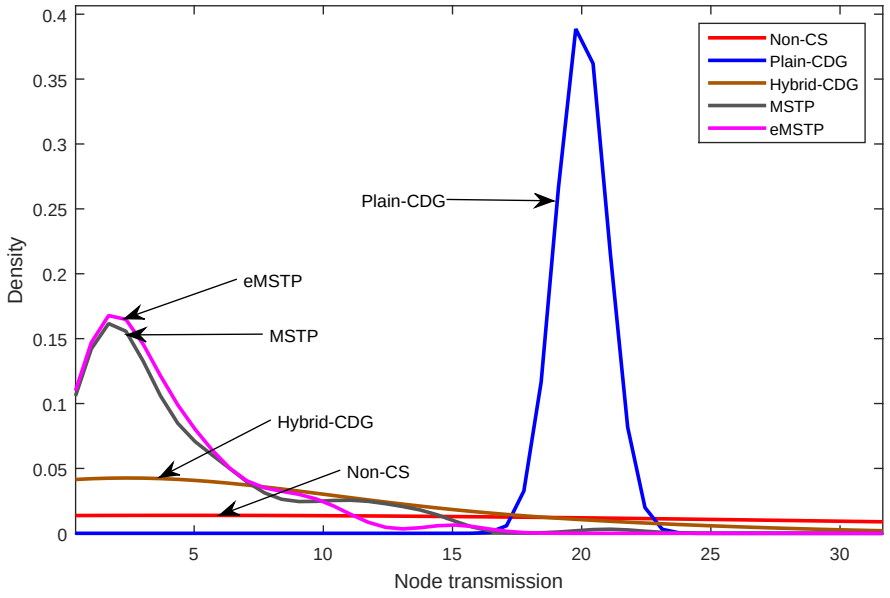


Figure 3.12: PDF for average transmission in sparse network with 100 nodes, 20 random samples and sink at top.

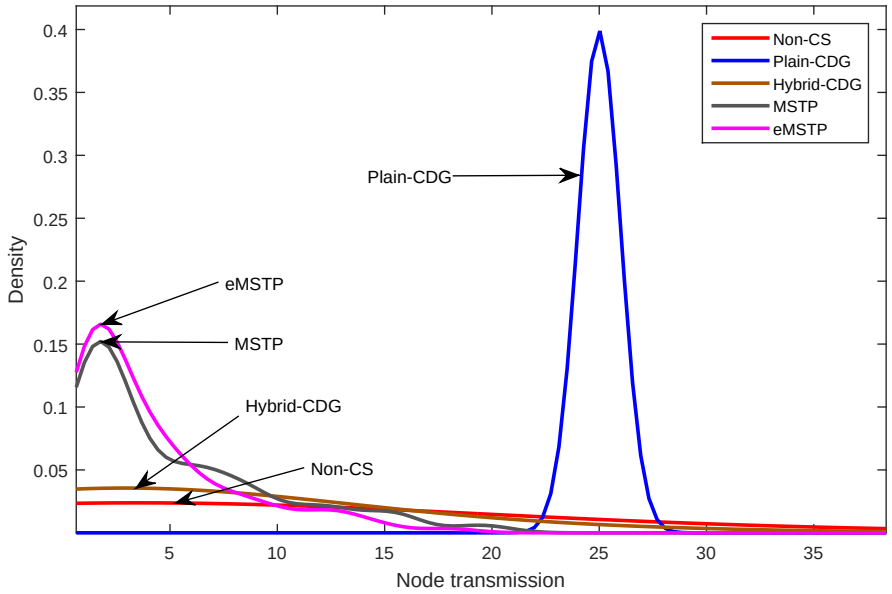


Figure 3.13: PDF for average transmission in sparse network with 100 nodes, 25 random samples and sink at the center.

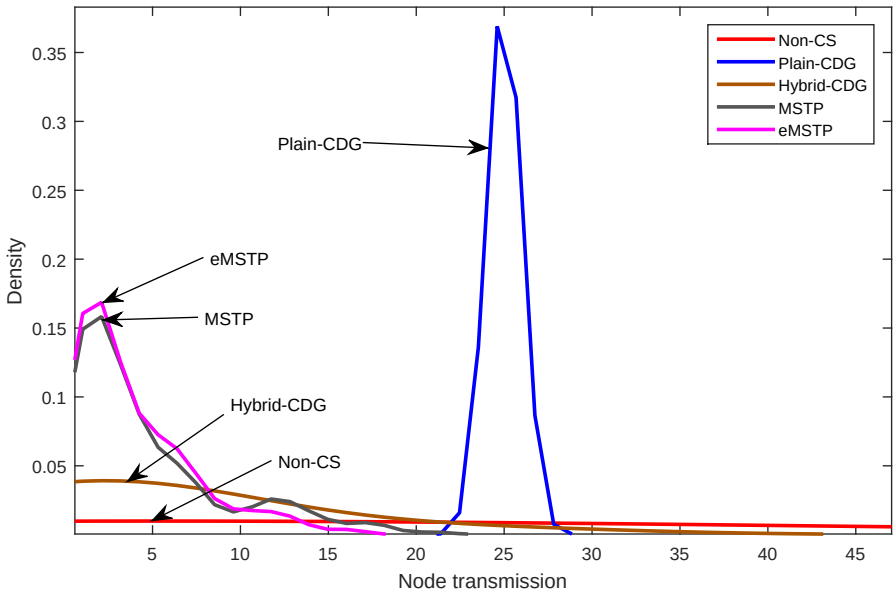


Figure 3.14: PDF for average transmission in sparse network with 200 nodes, 25 random samples and sink at top.

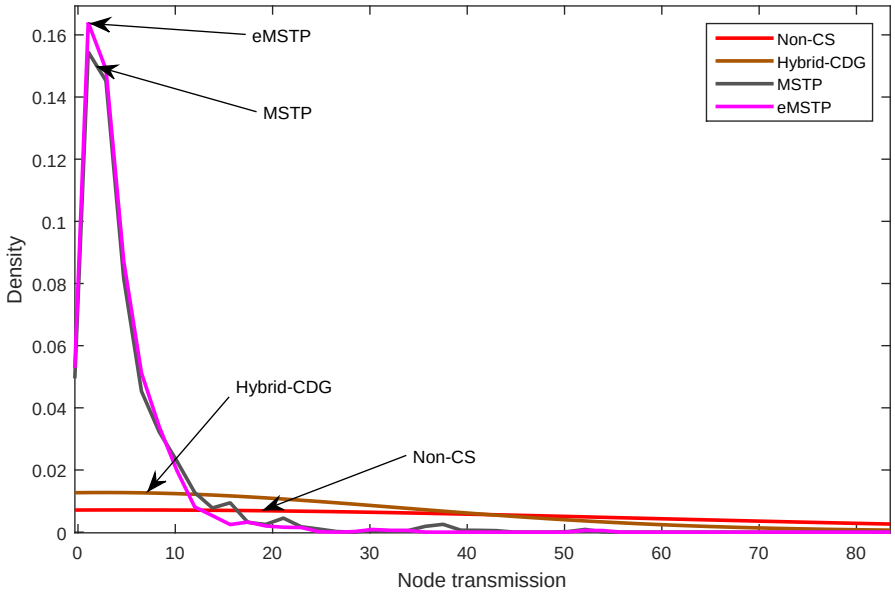


Figure 3.15: PDF for average transmission in dense network with 500 nodes, 100 random samples and sink at top.

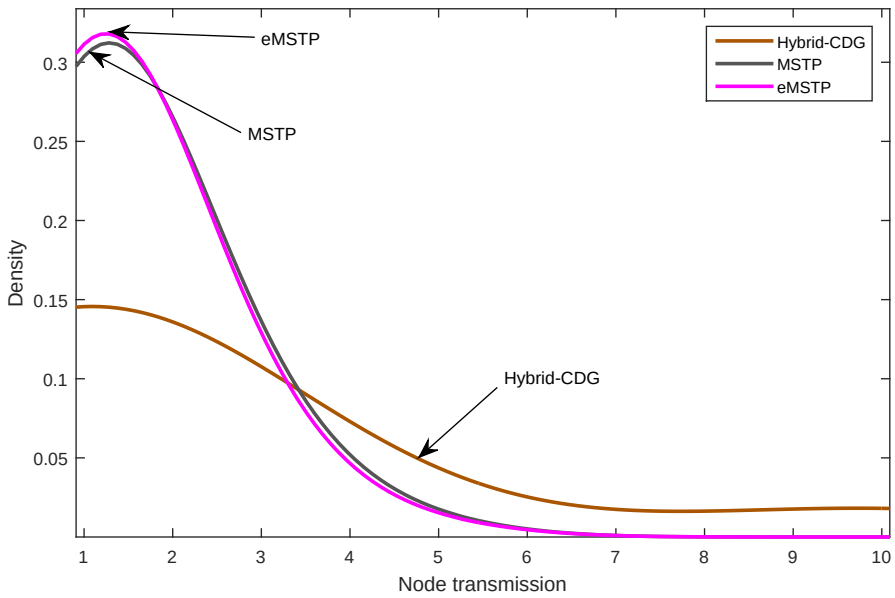


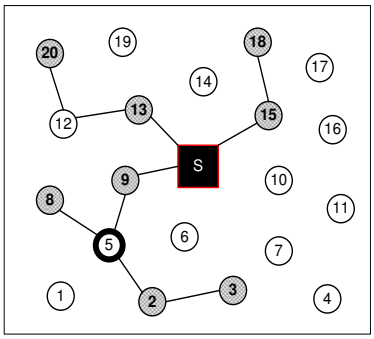
Figure 3.16: PDF for average transmission in dense network with 1000 nodes, 10 random samples and sink at the center.

3.5 Optimal Selection of Projection nodes (OSPN)

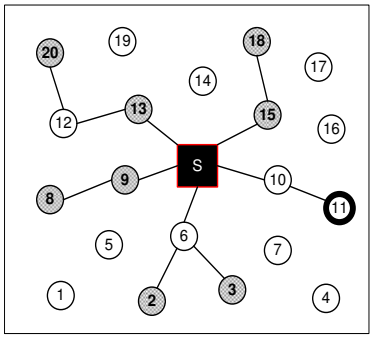
As we illustrate next, the performance of eMSTP depends on the selection of projection nodes; Fig. 3.17 shows a sample network with two different projection node selections. When node 5 is selected, the overall number of transmissions is 10 (Fig. 17(a)), and when node 11 is selected, the overall number of transmission increases to 12 (Fig. 17(b)). To determine the best set of projection nodes, one may do an exhaustive search through all the possibilities and then select the one that yields the best performance; indeed, this exhaustive search is challenged by the large number of possibilities of selecting projection nodes ($\binom{n}{m}$) making it grossly impractical. Therefore, instead of searching through this large space of possibilities, each projection tree can be constructed independent from the others without sacrificing in the performance. That is, we start by selecting the first projection node considering all nodes as candidates and the projection tree (out of n trees) that results in smallest number of transmissions from the interest nodes involved in this projection is selected. Then, the same procedure is repeated for the remaining $m-1$ projections. Therefore, instead of searching through a space of $\binom{n}{m}$ possibilities, the solution is found by exploring a total of $m \times n$ possibilities. This revised method is referred to as OSPN.

3.6 Optimal Tree Construction (Opt-PCDG)

To characterize the optimal solution, we let the sink node, instead of the projection nodes, construct the routing trees to the m sets of interest nodes and collect the m weighted sums. Let T be a set of m trees and z_{ij}^{t} be a binary variable indicating whether there is an edge between nodes i and j in tree t.



(a) 10 transmissions.



(b) 12 transmissions.

Figure 3.17: Performance of selecting different projection nodes. The thick circle is a projection node and gray ones are interest nodes.

Let x_{ij}^t be a flow of traffic between nodes i and j in tree t. Let s denote the sink node and I_t be the set of interest nodes (IN) for tree t. The objective of our problem is to minimize the total number of packet transmissions (i.e., minimize the number of edges in each of the m trees), which is formulated in the following (Opt-PCDG):

$$Minimize \qquad \sum_{t=1}^{m} \sum_{(i,j)\in E} z_{ij}^t \tag{3.3}$$

$$\sum_{j:(i,j)\in E} x_{ij}^t - \sum_{j:(j,i)\in E} x_{ji}^t = \begin{cases} -\lceil \frac{n}{m} \rceil, & i = s; \\ 1, & \forall i \in I_t; \quad t \in T \\ 0, & otherwise. \end{cases} \tag{3.4}$$

$$x_{ij}^t - z_{ij}^t \geq 0 \qquad ,\forall (i,j) \in E \quad , \quad \forall t \in T \tag{3.5}$$

$$z_{ij}^t - m^{-1} x_{ij}^t \geq 0 \qquad ,\forall (i,j) \in E \quad , \quad \forall t \in T \tag{3.6}$$

$$\sum_{j:(i,j)\in E} z_{ij}^t \leq 1 \qquad ,\forall t \in T \tag{3.7}$$

$$z_{ij}^t \in \{0,1\} \qquad ,\forall (i,j) \in E \quad , \quad \forall t \in T \tag{3.8}$$

$$x_{ij}^t \geq 0 \qquad ,\forall (i,j) \in E \quad , \quad \forall t \in T \tag{3.9}$$

Constraints (B.8) present the flow conservation constraints; they force the interest nodes that belong to one tree (projection) to have one flow each to the sink [64]. Constraints (B.9) and (B.10) make the connection between x_{ij}^t and z_{ij}^t, such that only edges carrying positive flows are indicated as tree edges; they imply that $x_{ij}^t = 0 \Leftrightarrow z_{ij}^t = 0$ and $x_{ij}^t > 0 \Leftrightarrow z_{ij}^t = 1$ [81]. Constraint (4.1) asserts that each node can have a maximum of one transmission (outgoing edge) in each tree to avoid loops. The presence of the boolean variables z_{ij}^t turns the above linear program into an NP-Hard problem. In the next subsection, we present a scalable and efficient algorithm to solve the construction of the trees for the compressive data aggregation problem.

3.7 Projection based Compressive Data Gathering Algorithm (PB-CDG)

Our algorithm uses the same steps as eMSTP with the difference that there are no projection nodes, but rather, the sink will gather the compressed data (or weighted sums). The steps are shown in Algorithm 3.3. The time complexity for the algorithm is $O(md\rho^2 \log \rho)$ in the best case and $O(md\rho^2 n \log n)$ in the worst case, same as eMSTP, where d is the average nodal degree of the nodes and ρ is the number of interest nodes.

The algorithm for each projection t at initialization retrieves the nodes whose coefficients in the basis matrix Φ_t are non-zero (i.e., I_t), adds them to the interest nodes list Int_t and then assigns the sink node s as a single node tree T_t. In the second step, the algorithm removes the nodes in Int_t which can be connected directly to tree T_t and adds them to T_t using the MST algorithm. Since interest nodes are spread throughout the network and may not

Algorithm 3.3 PB-CDG

Require: Matrix Φ.

Ensure: Set of trees $T = \{T_1, T_2, ..., T_m\}$.

1 **For** each projection t ($t = 1, 2, ..., m$) **do**:

1.1 $T_t = \{s\}$ and $Int_t = I_t = \{$ all nodes s.t. $\phi_{tj} \neq 0, j = 1, 2, ..., n\}$.

1.2 Add those nodes in Int_t into tree T_t which can be connected directly to nodes in T_t using MST algorithm and remove them from Int_t.

1.3 **While** $!Empty(Int_t)$ **do**:

1.3.1 Find the nearest node h in Int_t to T_t using BFS algorithm.

1.3.2 Add node h to T_t plus all the intermediate nodes in shortest path to T_t.

1.3.3 Remove node h from Int_t.

1.3.4 **If** Int_t is not empty, **do** step 1.2.

be reached without multi-hop, there may still be nodes in Int_t which could not be connected to T_t using MST. Therefore, as long as Int_t is not empty, the algorithm finds the nearest node in Int_t to T_t using the BFS algorithm and adds that node plus all the intermediate nodes to T_t and removes that node from the list. Again the algorithm repeats by adding more nodes from Int_t to T_t tree using MST if possible. Upon termination, the algorithm returns all the m trees T_t ($t = 1, 2, ..., m$).

3.8 Performance Evaluation

We consider an arbitrary network topology with n nodes distributed randomly in a region with a sink node at the center. We use different number of projections m and present the average results of five runs to compare between our methods with different random basis Φ and different random projection nodes

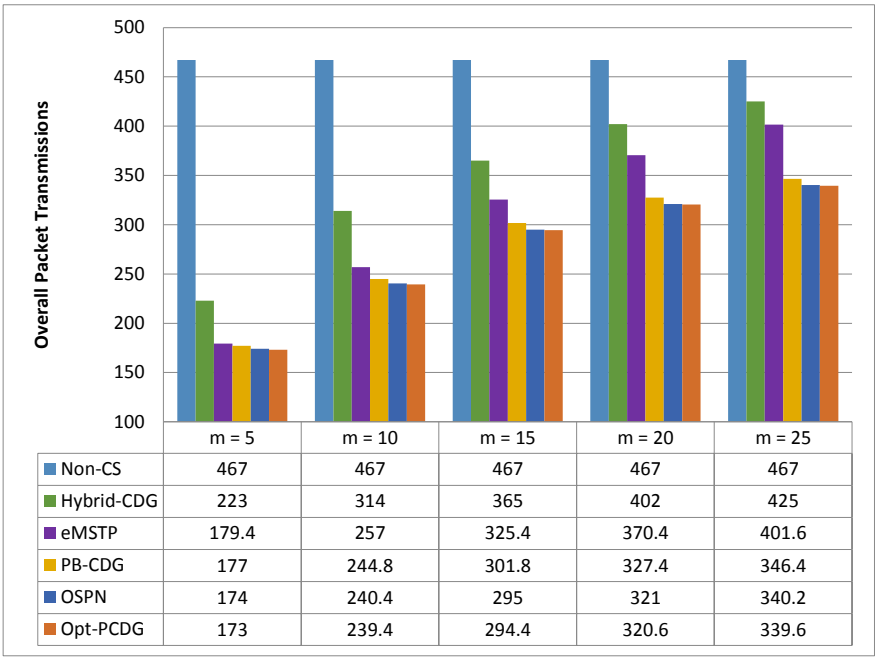
in case of eMSTP and OSPN methods.

We start by first evaluating the performance of the mathematical model (Opt-PCDG) in terms of computation complexity. For this purpose, we consider a smaller network ($n = 50$, $m = 5$) and we use CPLEX to solve the model. We use two different approaches for solving Opt-PCDG, where in the first, Opt-PCDG is solved as presented in the previous section and in the second we consider solving it sequentially, selecting one projection at a time, until all m projections/trees are determined. Both methods yield the same solution (as we explained earlier), however the difference being the running time. For the considered network instance, the first method takes 62 minutes to give back the solution whereas only 28 seconds are needed in the second approach. When $m = 10$, the running times for the first and second approaches are 4:15 hours and 48 seconds respectively. Therefore, in the rest of this proposal we follow the sequential approach for solving Opt-PCDG.

Our results in Figure 3.18 for a 100-node network illustrate the benefits of our proposed methods. Without using any gathering, the number of data transmissions in the network is excessively large; for instance, when $m = 5$, the Non-CS method performs 169% more transmissions than Opt-PCDG method. Our results show that eMSTP is far from optimal; in the worst case, eMSTP showed an 18% gap with the results obtained by the Opt-PCDG method and it outperforms the Hybrid-CDG by transmitting overall fewer packets. In addition, OSPN achieves very close results to the Opt-PCDG method with a maximum gap of 0.5%. Finally, PB-CDG showed remarkable performance, with results very close to those obtained by the optimal method (a maximum gap of 2.5% is observed between their results), but PB-CDG being much more scalable than Opt-PCDG and OSPN. For instance, when $m = 25$, PB-CDG runs in

8 milliseconds and OSPN runs in 8 seconds, whereas Opt-PCDG (depending on Φ) takes between 3 minutes and hours.

One of the advantages of using CDG is its ability to solve the bottleneck problem and to distribute the transmission load more evenly throughout the network. Figure 3.19 represents the un-metric Probability Density Function (PDF) of nodes that transmit particular number of packets in the network; that means, the fraction of nodes that send one packet (transmission), two packets, etc. By observing the peak of the curves, we can see that most of the nodes in eMSTP in average transmit fewer packets in comparison to Non-CS and Hybrid-CDG. Opt-PCDG, OSPN, and PB-CDG methods respectively distribute the load more evenly throughout the nodes with less number of transmissions and hence increase the lifetime of the network.



	m = 5	m = 10	m = 15	m = 20	m = 25
Non-CS	467	467	467	467	467
Hybrid-CDG	223	314	365	402	425
eMSTP	179.4	257	325.4	370.4	401.6
PB-CDG	177	244.8	301.8	327.4	346.4
OSPN	174	240.4	295	321	340.2
Opt-PCDG	173	239.4	294.4	320.6	339.6

Figure 3.18: Overall number of data transmission (n = 100, different m)

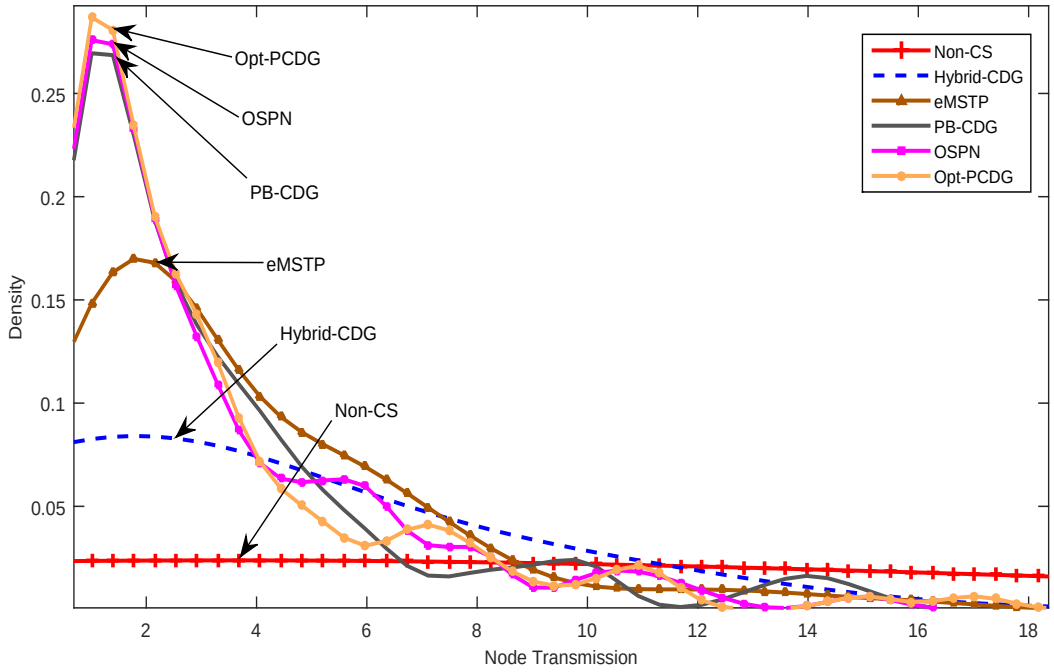


Figure 3.19: PDF for average node transmission of our different algorithms (n = 100)

3.9 Conclusion

This chapter proposed a projection-based compressive data gathering scheme by constructing routing trees in a way that minimizes the overall energy consumption and distributes the energy load more evenly throughout the network. The simulation results showed that our data gathering methods dramatically increase the lifetime of sensor networks.

Chapter 4

Distributed Compressive Data Gathering (DCDG)

This chapter presents a decentralized method for the compressive data gathering problem (DCDG). The method allows each sensor node to locally make a decision in constructing and maintaining the forwarding trees and has minimal complexity and overhead with outstanding performance.

4.1 Motivation

All methods presented in the previous chapter are centralized in nature; that is, initially the sink has to accomplish a topology discovery by retrieving the network wide information through deploying an all-to-all flooding (where $O(n^2)$ messages are needed), and then solves the algorithm to construct the forwarding trees. Subsequently, for each forwarding tree, the sink sends out notification messages to all nodes in the network informing each of its parent node and children. Clearly, the overhead associated with such centralized approach makes it difficult to implement in practice. Further, such centralized methods

do not respond well to topological changes.

With our centralized method (e.g., PB-CDG), in the case where one or more nodes leave the network, affecting one or more aggregation trees, one just needs to reconnect the tree (or trees) which contains that departed node and does not need to be concerned about the remaining trees. Furthermore, for reconstructing the affected tree, there is no need to start from its root. It is enough to join the disconnected interest nodes (caused by a departing node) to the tree by following the same steps of our algorithm. For example, as illustrated in Figure 4.1, if node 23 departs the WSN, four interest nodes (30, 32, 37and 39) will be disconnected from the tree. Hence, we only need to join these four nodes to the tree and not be concerned about other nodes which are already connected to the tree. By using our proposed algorithm (PB-CDG), node 30 will be joined to the tree. Nodes 37 and 39 are already attached to node 30 and we do not need to do anything. At last step, as illustrated in Figure 4.2, we join node 32 to the tree.

However, note that in the case of topological changes as a result of either nodes getting disconnected and/or links being removed due to poor channel conditions (fading, shadowing, etc), centralized methods have to collect new topological information and run to reconstruct the aggregation tree(s). The overhead of this is quite substantial. Alternatively, constructing the trees could be done in a distributed manner, without requiring network-wide topology information. The distributed method builds upon the deficiencies of centralized methods, where the computation is local for each node. In this chapter, we present the distributed approach (DCDG) for constructing the forwarding trees where each node locally decides its parent node to whom it should transmit its encoded data.

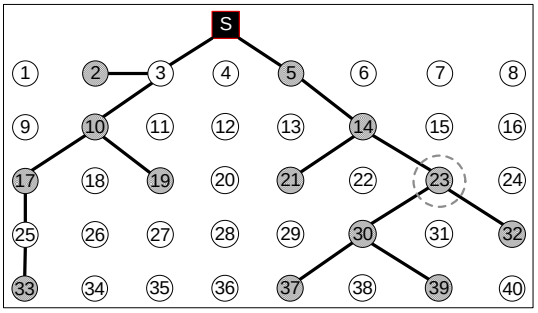


Figure 4.1: Forwarding tree example using PB-CDG

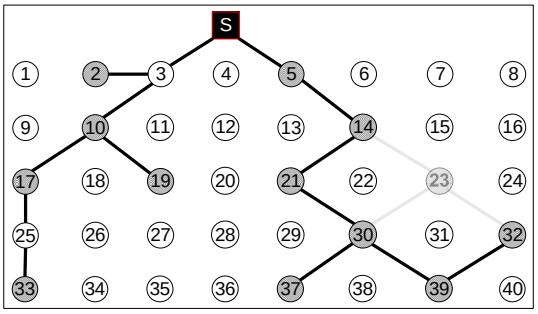


Figure 4.2: Reconstruction of PB-CDG after node failure

4.2 Overview of the Distributed Method

This chapter presents a distributed method for constructing m forwarding trees. Each forwarding tree connects the interest nodes of one particular projection to the sink and allow the sink to collect the weighted sum corresponding to this projection with minimal communication cost. Upon collecting the m encoded sums, the sink will recover/decode the original data by solving a convex problem. Below we describe the method. Initially (Phase 1), the sink starts by sending a discovery message to its neighbors. Each node, upon receiving the message, will broadcast it to allow other nodes, not close to the sink, to receive the discovery message. Hence, each node v will learn its shortest path (P_{vs}) to the sink as well as the hop count (h_v) along the path. Further, node v gets to learn its neighbors $N(v)$. Node v upon checking matrix Φ, which is stored in its

memory, determines whether node $u \in N(v)$ ($\forall u$) belongs to the set of interest nodes (I_t) of tree t or not. In Phase 2, each interest node decides its parent on the uplink path to the sink (details are provided next).

4.3 Description of the Distributed Algorithm

The steps of the method are shown in Algorithm 4.1. As explained earlier, Phase 1 consists of sending a discovery message to the network such that each node knows its shortest path and hop count to the sink. In Phase 2, tree construction starts. We explain the construction of one tree t (similar procedure is repeated for others). Once nodes in the network receive the discovery message, each node j determines whether it is an interest node or not ($j \in I_t$). For each interest node, we assign an attribute to designate its parent interest node (π_j) (note a parent interest node could be a neighbor of j or can be reached through other relay nodes) and a decision flag (F_j) to indicate whether the parent interest node of j is fixed ($F_j = 1$ indicates that the parent interest node is fixed).

Lines (7-9) show that every interest node which is a neighbor of the root selects the root as its parent node. Now, if interest node j (Lines (10-12)) is not a neighbor of the root, but has an interest node neighbor b with $F_b = 1$, then j selects b as its parent interest node and commits its decision ($F_j = 1$). In the case where none of the neighboring interest nodes (b) of j has its decision flag set (i.e., $F_b = 0$), j will select the neighboring interest node with smaller hop-count to the sink as its parent interest node (Lines 13-14). Now, when only interest node neighbors with equal hop-count to the sink as j can be found (Lines 15-16), j selects the one (b) whose successive parents reach an interest node with smaller hop-count or decision flag $F = 1$ or no parent node, and does not reach node j (to avoid loops). Note that, in this scenario when an

Algorithm 4.1 Distributed Compressive Data Gathering

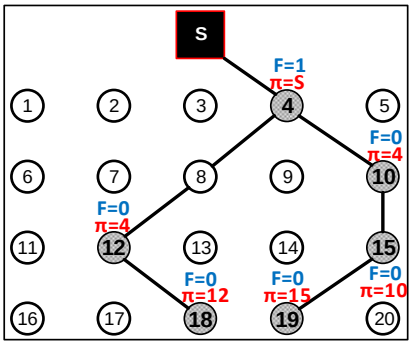
1: **Phase 1:** Start a Breath-First-Search (BFS) at the sink to disseminate the discovery message to all nodes: Each node $v \in G$ learns its shortest-path(s) (P_{vs}, $s = root$) to the root and hop-count h_v.The parameters and variables used by the algorithm are defined in Section IV.
2: **Phase 2:**
3: **for** each tree $T_t, t = 1, 2, ..., m$ **do**
4: Identify the set of interest-nodes $I_t \subseteq V$ for tree T_t.
5: Set $F_j = 0 \; \forall j \in I_t$.
6: **for** $j \in I_t (j = 1, 2, ..., \lceil \frac{n}{m} \rceil)$ **do**
7: **if** $root \in N(j)$ **then**
8: $\pi_j = s$; **(**$s = root$**)**
9: $F_j = 1$;
10: **else if** $b \in N(j)$ AND $b \in I_t$ AND $F_b = 1$ **then**
11: $\pi_j = b$;
12: $F_j = 1$;
13: **else if** $b \in N(j)$ AND $b \in I_t$ AND $h_b < h_j$ **then**
14: $\pi_j = b$;
15: **else if** $b \in N(j)$ AND $b \in I_t$ AND $h_b = h_j$ AND successive parents of b reach a parent node with smaller hop-count or $F = 1$ or non-parent and do not reach j **then**
16: $\pi_j = b$;
17: **else**
18: Run BFS from j in a radius equals to $h_j - 1$.
19: **if** we found interest-node(s) in this radius **then**
20: Connect j to nearest interest-node through shortest path.
21: **else**
22: Connect node j through its shortest path to the root.
23: **end if**
24: **end if**
25: **end for**
26: Repeat lines (6-25) until there are no changes in F_j.
27: **end for**

interest node chooses a parent with equal hop-count, it has to keep a record of its successive parents, and in case of node or link failure, it has to notify its child for any updates in route decisions. If none of the above conditions is satisfied, j runs a BFS (Breath-First-Search) to explore its neighborhood of radius $h_j - 1$ in search for an interest node b with smaller hop count to the sink otherwise, for an interest node whose decision flag $F_b = 1$ (Lines 18-20). Node j avoids selecting interest nodes b whose $\pi_b = j$ to avoid loops. Finally, if no interest node is found, j connects itself directly through a shortest path to the sink (this path is known from the discovery phase). Interest nodes with $F_j = 0$ repeat the above procedure (Lines 6-25) until no more changes in their flag attributes F_j.

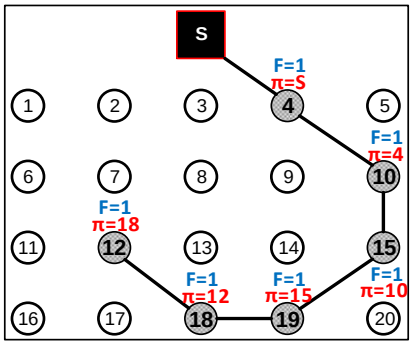
4.4 Illustrative Example

Before we illustrate, we note that upon completion of Phase 1 of the algorithm, each sensor node locally executes Lines (6-25) to cooperate in the construction of the forwarding tree. Now, we illustrate the operation of the algorithm on a sample network shown in Figure 4.3. Note that here each node is assumed to be connected to all neighboring nodes with a link of normalized distance (unitary distance) and the hop count is used to compute the path length from a source node to the destination. Gray nodes are the interest nodes which need to be connected through an efficient forwarding tree to the sink (black square S). Next, for each interest node j in the network, we show the value of π_j and F_j.

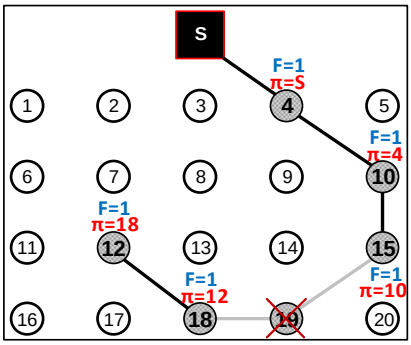
Upon completing the discovery phase, each interest node locally determines its parent node in the forwarding tree as follows (note that all interest nodes simultaneously are executing the process of Phase 2). Being a neighbor of the



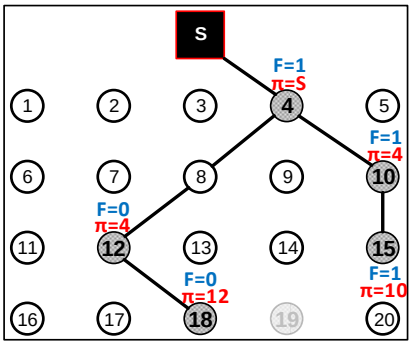
(a) After first iteration of each node. Total transmissions = 7.



(b) After second iteration of each node. Total transmissions = 6.



(c) Node 19 departs the network.



(d) upon node failure.

Figure 4.3: DCDG example

sink, node 4 sets $\pi_4 = s$ and $F_4 = 1$. Node 10, having received the discovery message(s) from the sink through node 4, will (by Lines 13-14) sets its parent node to 4 ($\pi_{10} = 4$). Node 10 then sends a notification message to its parent (node 4) to notify it of its decision and that it will transmit its data to the sink through it (here, node 4 will encode the received data with its own before forwarding to the sink). Node 4 sends back a message to its child (node 10) informing it that its decision flag $F_4 = 1$ and subsequently, node 10 sets its decision flag $F_{10} = 1$. Similar decisions are made for nodes 15 and 19. Note that once Node 10 sets its F_{10}, it will communicate with node 15 and this node sets its $F_{15} = 1$ which in turn does the same with node 19. Now, node 18 at first iteration (Figure 3(a)) receives discovery messages from nodes 12 and 19, but (Lines 13-14) selects node 12 as its parent node ($\pi_{18} = 12$) since node 12 has a

smaller hop count to the sink. Note, however, when node 19 informs node 18 (being its neighbor) that its flag is set $F_{19} = 1$, in the second iteration (Figure 3(b)), node 18 (Lines 10-12) switches its parent node to node 19 ($\pi_{18} = 19$) and fixes its flag ($F_{18} = 1$). It is to be noted that node 18 opted to use a longer route (through node 19) to the sink over the shorter path through node 12 as this achieves better aggregation gain. Finally, node 12 in the first iteration runs a BFS in its neighborhood of radius $h_{12} - 1$ to discover interest node 4 and sets it as its parent interest node ($\pi_{12} = 4$). Node 12 (being a neighbor of node 18) will receive a notification that node 18 had fixed its decision flag, therefore, in the second iteration (using Lines 10-12) it switches its path to the sink by selecting node 18 as its parent node ($\pi_{12} = 18$) and sets its decision flag ($F_{12} = 1$). As mentioned above, this decision is guided by the data gathering benefits along this route. We should note that each node runs the DCDG algorithm only once, unless it receives a notification message from its neighbors (e.g., upon a change in a flag value), or following a node (or link) failure (due to mobility or channel impairments)) occurring in the network triggering route maintenance. In both cases, the node runs the algorithm to decide a new route to the sink by selecting a new parent node.

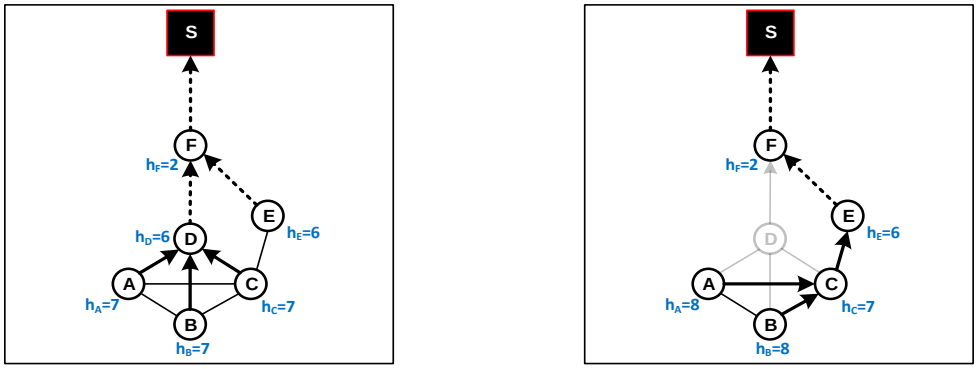
We illustrate the route maintenance in Figure 3(c). Upon the failure of node 19, both nodes 18 and 12 will be disconnected from the forwarding tree. Node 18 runs the algorithm and informs interest-node 12 that it has been selected as its parent interest node and sets $F_{18} = 0$. Node 12 after receiving a message from node 18 learns about the changes and runs the algorithm locally and selects interest node 4 as its parent interest node (Lines 18-20 of Alg. 4.1) and sets $F_{12} = 0$; after recovering from node failure, the new forwarding tree is shown in Figure 3(d).

4.5 Loop Free Tree Construction

In this section, we illustrate through examples that our distributed algorithm may not cause routing loops in case of node failure. Consider an example where nodes A, B and C are neighbors with each other, and they all use node D as their parent node. When node D fails/dies, they all need to find a new parent. It is very important that the DCDG algorithm will not allow the three nodes to choose new parents such that a routing loop occur (i.e., A connects to B, and B connects to C, and C connects to A).

In a case where nodes A, B and C (say after node failure) have different hop-distances to the sink through alternative shortest-paths (not through node D), there will not be any possibility for a loop as illustrated in Figure 4.4. Nodes A and B will choose neighbour interest-node C as their parent since node C has smaller hop-distance to the sink (i.e, $h_C < h_A$ and $h_C < h_B$), and node C will not choose any of nodes A or B because they have smaller hop-count. Note that each node at the first phase by receiving the discovery message from neighbours in different direction knows all the shortest-path(s) and hop-distances to the sink. Therefore, it does not require to receive a discovery message again in case of node failure.

Now, in case when after the failure of node D, all the neighbour nodes A, B and C have the same hop-distance to the sink, a loop only might happen however when the set of nodes have to choose neighbour parent-nodes with equal hop-distance to the sink. But, according to DCDG algorithm (Lines 15-16), a node (j) selects an interest-node parent whose successive parents reach an interest node with smaller hop-count or decision flag $F = 1$ or no parent node, and does not reach node j. This condition avoids all the loop possibilities. In example of Figure 4.5, when node D fails, node A may choose node B as its

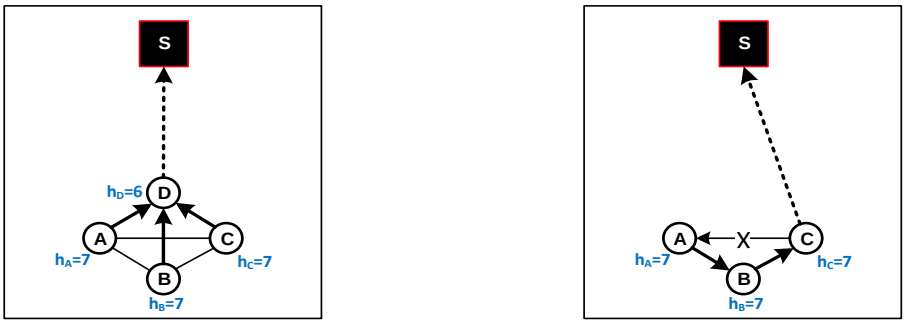


(a) Before node failure.

(b) After node failure.

Figure 4.4: There is no possibility for loop when neighbor interest nodes have different hop-distance to the sink. Line represents the link, dark and dashed flashes represent the forwarding tree's routes for link and path respectively. h represents the hop-distance to the sink for each node.

parent, and node B may choose node C, but node C can not choose node A, since node A has successive parents which reach node C. Therefore, node C runs BFS (Breadth-First-Search) in search for an interest node with decision flag $F = 1$ or smaller hop-distance to the sink.

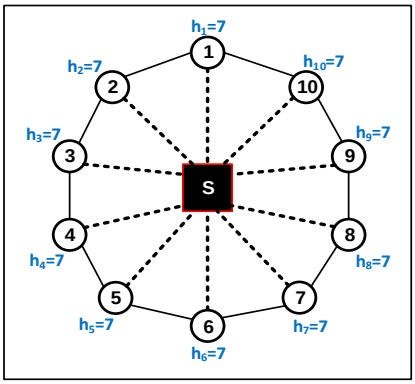


(a) Before node failure.

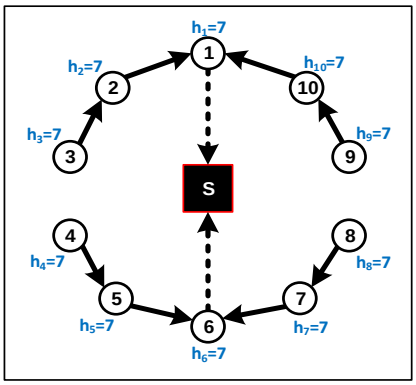
(b) After node failure.

Figure 4.5: Example of recovery after node failure.

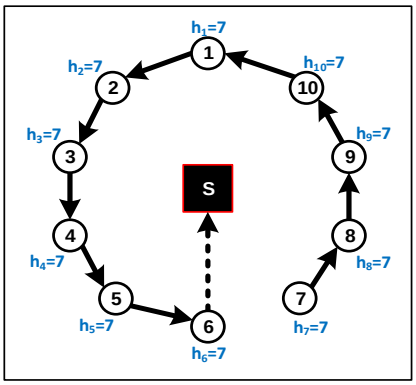
Now, let us consider a network topology where 10 interest nodes are sequentially neighbours with each other around the sink with seven hop-distance



(a) Network topology



(b) Forwarding tree example 1.



(c) Forwarding tree example 2

Figure 4.6: A sequential neighbor nodes topology example in DCDG

away as illustrated in Figure 4.6, and all the nodes do not have any other neighbour interest nodes with smaller hop-distance to the sink or decision flag $F = 1$. In this scenario, by running the distributed algorithm we might have different forwarding trees (for example Figures 6(b) and 6(c)), but clearly there are not possibilities for having loops.

4.6 Message Overhead Analysis

In the sequel we present the message overhead analysis of both centralized and distributed methods; first we clarify how the centralized and decentralized algorithms calculate the message overhead.

The following steps represent the calculation of message overhead in Decentralized Compressive Data Gathering:

1. Discovery phase: the sink initiates (broadcasts) a network discovery process, which may take up to n messages.

2. If an interest-node has neighbour interest-node(s) and assigns one of its neighbour interest-node as its parent node, then the node needs to send only one message to its parent to notify the parent of its decision.

3. In case where an interest-node i does not have a neighbour interest-node to choose for its parent interest-node, the overhead is computed as follows:

   a) Node i sends out a discovery message calling for interest-nodes for tree t in a radius equal to $h_i - 1$ (h_i is a hop distance from node i to the sink). Nodes, by receiving this discovery message, if they are in a radius $h_i - 2$, forward the message to their neighbours. In this step, in total R_i messages are required, where R_i is the number of nodes in radius $h_i - 2$.

   b) Only nodes in radius $h_i - 1$ that belong to set of interest-nodes for tree t send message (containing; node-ID, identification-flag (F), hop-distance to node i and hop-distance to the sink (h_j)) through shortest-path to node i. This step takes a number of messages equals to the

number of interest-nodes in radius h_i plus the number of hops in their way to node i.

c) Node i, after running the DCDG algorithm and choosing its parent interest-node, sends a notification message in shortest-path to the chosen interest-node, which consumes a number of messages equals to the number of hops in its shortest-path.

The following steps represent the calculation of message overhead in Centralized Compressive Data Gathering:

1. n messages are needed for the sink to broadcast the discovery message to all nodes in the network.

2. To obtain the network topology at the sink, each node in the network, based on the number of neighbours it has, sends out information messages to the sink. This step for obtaining the network topology takes a number of messages equals to the number of nodes in the network plus the number of neighbours for each node and the hop-distances to the sink.

3. After running the centralized algorithm at the sink and constructing m trees for compressive data gathering, the sink sends out a message to each node participating at each three t through shortest-path notifying of forwarding trees. This step consumes a number of messages equals to the number of trees m into the number of nodes participating in each tree t plus their hop-distances to the sink.

Now, to make the analysis more simple, we use a linear network. Note that the linear network is the worst case for the decentralized method and best case for centralized method. That is because the number of neighbours

for each node in the linear network is minimum (advantage for centralized algorithm) and the discovery radius for most of the nodes in decentralized algorithm is very big in case where they do not have a neighbour interest-nodes (disadvantage for decentralized method).

First, we calculate the number of message overhead on a linear network shown in Figure 4.7, later we show the analysis for network of size n and m number of projections. In the linear network, to get the worst case for the decentralized algorithm, we let each interest-node to be as far away as possible from others by making the distances between any two interest-nodes equal to m. In the example of Figure 4.7, the white and dark nodes represent two different sets of interest-nodes (trees). In the graph $n = 6$ and $m = 2$.

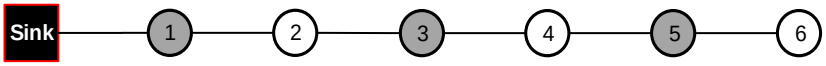


Figure 4.7: Linear Network Example

The message overhead computation and analysis of the given example for centralized and decentralized methods are given in Appendices A and B respectively.

To compare the message overhead between these two methods, we have to consider the relation between the values n and m. Based on the compressive sensing technique [20], $m = C.k.\log(n)$, where C is a constant value and k is the sparsity representation of the data. In our numerical results, the smallest value of m used is equal to 5% of n (i.e.; $m = \frac{n}{20}$). If we let $m = \frac{n}{20}$, the message overhead for both methods become as in (4.1) and (4.2), where the overhead in centralized method growths sharper than decentralized method as the number of nodes increases (refer to Figure 4.8). Based on our presented analysis and as

the Figure 4.9 shows for linear network, when $m \geq \frac{5}{100}n$, decentralized beats centralized method. Note that we emphasize here that the linear network is the worst case for the decentralized method and best case for centralized method.

$$Total\_Centralized\_Message\_Overhead(m = 10\%n) = \frac{1141n^3}{24000} + \frac{439n^2}{400} + \frac{31n}{30} - 1 \tag{4.1}$$

$$Total\_Decentralized\_Message\_Overhead(m = 10\%n) <= \frac{857n^2}{160} + \frac{73n}{8} - \frac{80}{n} + \frac{1571}{6} \tag{4.2}$$

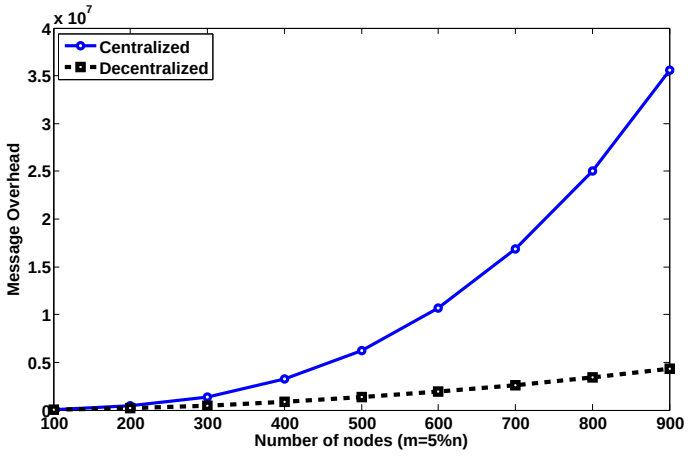


Figure 4.8: Message overhead analysis for different number of nodes

It should be noted that in a linear network the message overhead difference between the centralized and distributed method is minimum and as we change the network topology the variance increases. This is why the linear network is the best for the centralized method and worst for the decentralized method. To further clarify, consider the two uniform mesh network examples shown in Figures 4.10 and 4.11. The number of control messages required to construct the forwarding trees in Figure 4.10 for centralized method is 288 and for decentralized method is 48 messages, and in linear network the centralized and decentralized methods require 1751 and 1274 control messages respectively.

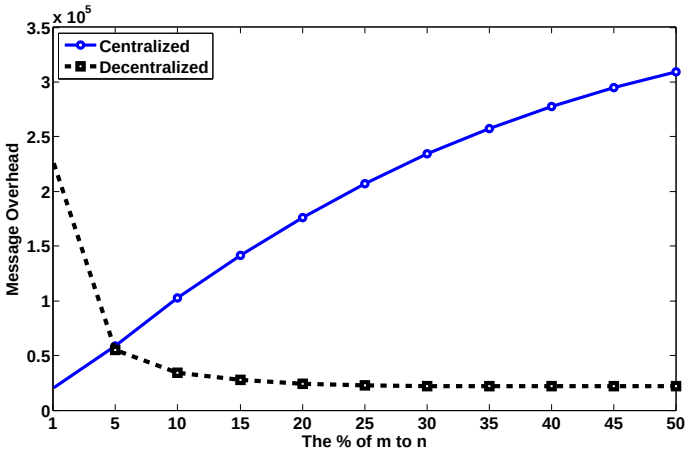
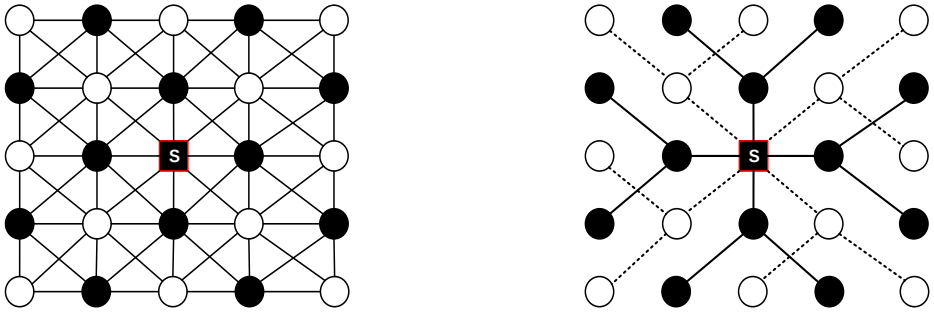


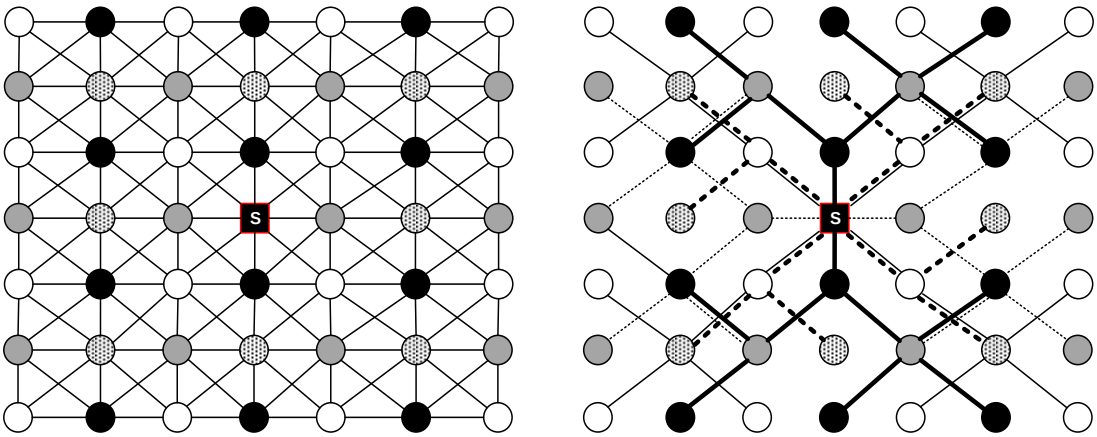
Figure 4.9: Message overhead analysis for different number of projections

The overhead differences between the centralized and decentralized methods in uniform network is $\%83.33$ and in linear network is $\%27.24$. Similarly, for Figure 4.11, the number of messages needed to construct the forwarding trees in uniform and linear networks for centralized (decentralized) method is 928 (304) and 11,191 (8,998) with $\%67.24$ and $\%19.6$ differences respectively. The reason that the linear network gives the smaller overhead gap between the centralized and decentralized method is that the number of neighbours for each node in the linear network is minimum (which is an advantage for the centralized algorithm) and the discovery radius for most of the nodes in decentralized algorithm is very big in case where they do not have a neighbour interest-nodes (a disadvantage for the decentralized method). But in a uniform or arbitrary network, each node in the decentralized method has more chances to have a neighbour interest node and the nodes which do not have a neighbour interest node, their discovery radius is not as large as most of the nodes in the linear network. Therefore, based on the current numerical results, the linear network gives the worst results for decentralized method and best for centralized method compared to other network topologies.



(a) Network topology (b) Forwarding trees

Figure 4.10: Constructing forwarding trees for mesh network with $n = 24$ and $m = 2$. Each set of interest nodes and forwarding tree have a unique color and unique line.



(a) Network topology (b) Forwarding trees

Figure 4.11: Constructing forwarding trees for mesh network with $n = 48$ and $m = 4$. Each set of interest nodes and forwarding tree have a unique color and unique line.

4.7 Numerical Results

We present simulation results to evaluate the performance of our distributed method and compare with other compressive data gathering (namely, Hybrid-CDG [81], PB-CDG and Steiner-CDG method) using metrics such as transmission cost and message overhead.

In Steiner-CDG method, forwarding trees are constructed using minimum Steiner tree algorithm [49]. The minimum Steiner tree problem is to connect a set of interest nodes $I \subset V$ such that the connected spanning tree has a minimum total distance on its edges. Algorithm 4.2 presents steps of the Steiner-CDG method. The algorithm takes $O(\rho n^2)$ to construct one tree, where ρ is the number of interest nodes. In total, the time complexity of the algorithm to construct m trees is $O(m\rho n^2)$.

Algorithm 4.2 Steiner-CDG

Require: Graph G, Matrix Φ.
Ensure: Set of trees $T = \{T_1^h, T_2^h, ..., T_m^h\}$.

1 **For** each projection t ($t = 1, 2, ..., m$) **do**:

1.1 Construct the undirected distance graph $G_i^1 =< S_i, E_i, d >$ that contains only interest nodes S_i, where E_i is the set of links between two interest nodes with d hops away from each other.

1.2 Find the MST t_i^1 from G_i^1.

1.3 Construct the graph G_i^2 from G by replacing each edges in t_i^1 by its corresponding shortest path in G.

1.4 Find the MST t_i^2 from G_i^2.

1.5 Construct tree t_i^h from t_i^2 by removing leaf nodes which are not interest nodes with their edges in a way that all leaves in t_i^h are interest nodes.

We consider arbitrary connected networks where nodes are generated and distributed randomly in a 700×700 unit field using the uniform distribution

and we assume all the nodes have unique communication range. In our simulation, we change the node density and we average the results over ten runs with different random sample matrices Φ (our results are shown with 95% confidence interval). Figures 4.12 and 4.13 show the overall number of transmissions (cost) vs number of projections (m) and node density respectively. Clearly Non-CS incurs the highest cost followed by Hybrid-CDG since this method did not exploit projection based gathering, rather it only relied on constructing one forwarding tree to collect all weighted sums. PB-CDG achieves minimal transmission cost with Steiner-CDG being close to it (difference of 2.1% in Figure 4.12 and 2.5% in Figure 4.13). DCDG differs from the previous methods in that it is completely distributed but yet achieves very close performance to PB-CDG and Steiner-CDG with maximum gap of 3.8% and 7.5% in Figure 4.12 and Figure 4.13 respectively. Observe that a lower compression ratio (i.e., higher m) in Figure 4.12 results in a higher number of transmissions (cost) than when the compression ratio is high (i.e., lower m) and this is due to the fact that a higher m means more projections and therefore more transmissions in the network to collect the weighted sums. Finally, we compare the overhead incurred by DCDG and PB-CDG for constructing the forwarding trees and the results are depicted in Figure 4.14 for different network sizes. The figure clearly shows that DCDG has a lower communication overhead than PB-CDG and both methods have an overhead that grows polynomially with the size of the network (the analysis is beyond the scope of the letter). DCDG enjoys an overhead that is 53% to 65% less than PB-CDG.

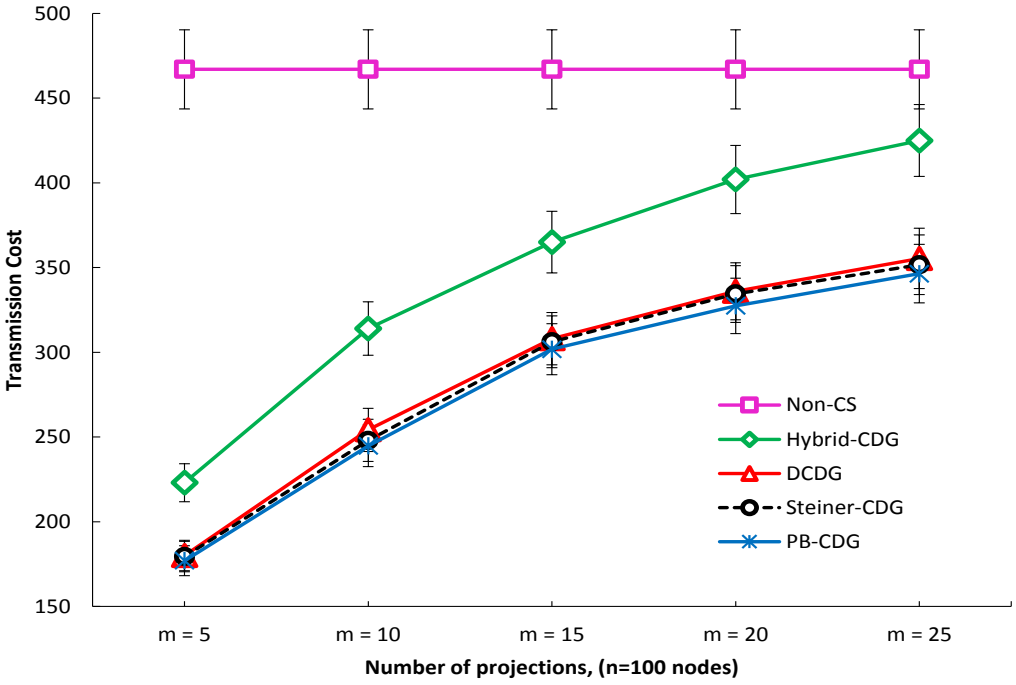


Figure 4.12: Different number of projections Vs. transmission cost (DCDG)

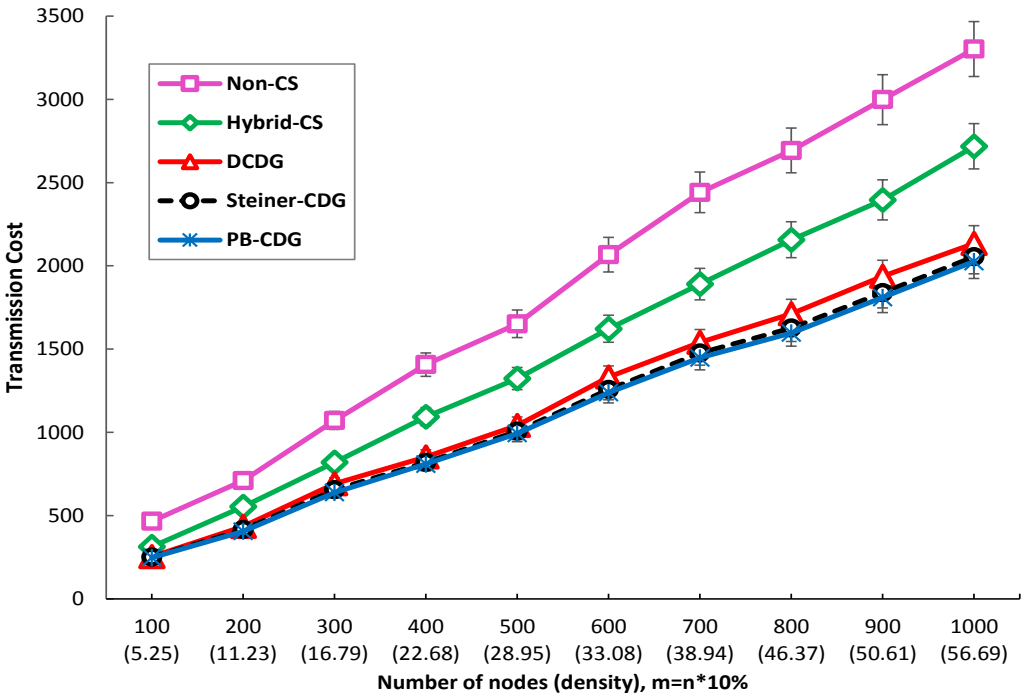


Figure 4.13: Different network Density Vs. transmission cost (DCDG)

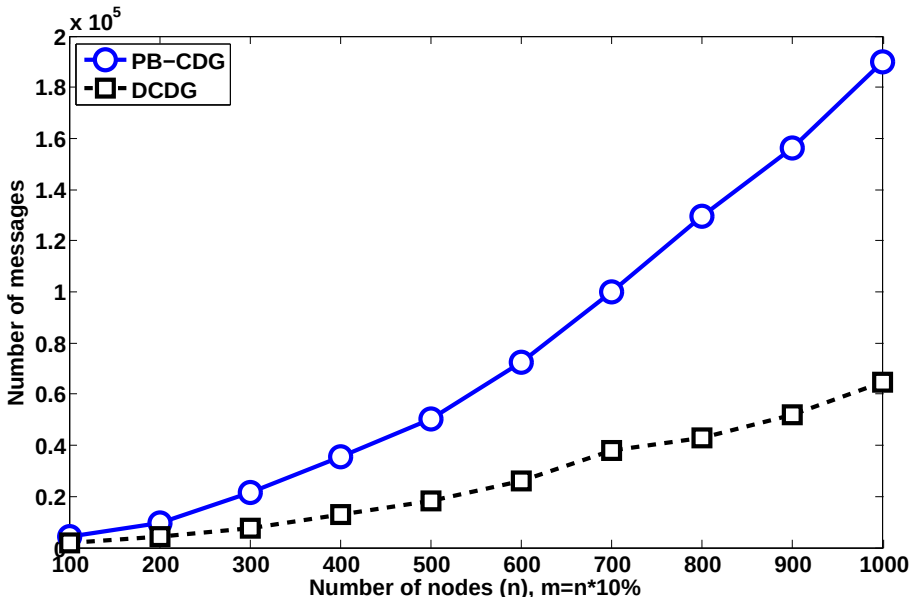


Figure 4.14: Message Overhead Vs. number of nodes (DCDG)

4.8 Conclusion

We proposed an efficient distributed CDG scheme where each sensor node independently finds its parent node and constructs part of the routing tree without requiring a central unit to construct all the forwarding trees. Through simulations we showed that DCDG performs (in terms of transmission cost) very close to the best centralized methods but outperforms them in terms of communication overhead.

Chapter 5

Network-Coding Aware Compressive Data Gathering (NC-CDG)

In this chapter we investigate the joint application of compressive sensing (CS) and network coding (NC) to the problem of energy efficient data gathering in wireless sensor networks. We consider the problem of optimally constructing forwarding trees to carry compressed data to projection nodes; each compressed data refers to a weighted aggregation (or sum) of sensed measurements from network sensors collected at one projection node. Projection nodes then forward their received compressed data to the sink, which subsequently recovers the original measurements. This aggregation technique based on compressive sensing is shown to reduce significantly the number of transmissions in the network. We observe that the presence of multiple forwarding trees gives rise to many-to-many communication patterns in sensor networks which in turn can be exploited to perform network coding on the compressed data being forwarded on these trees. Such technique will further

reduce the number of transmissions required to gather the measurements, and consequently result in a better network-wide energy efficiency. The chapter addresses the problem of network coding aware construction of forwarding/aggregation trees. We present a mathematical model to optimally construct such forwarding trees which encourage network coding operations on the compressed data. Owing to its complexity, we further develop algorithmic methods (both centralized and distributed) for solving the problem and analyze their complexities. We show that our algorithmic methods are scalable and accurate, with worst case optimality gap not exceeding 3.96% in the studied scenarios. We also show that when both network coding and compressive data gathering are considered jointly, performance gains (reduction in number of transmission) of up to 30% may be attained. Finally, we show that the proposed methods distribute the work load of data gathering throughout the network nodes uniformly, resulting in extended network life times.

5.1 Network Coding Model

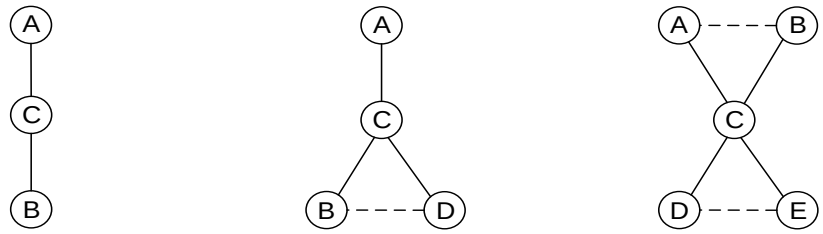
Network coding, originally developed by [1], has shown to yield substantial increase in the throughput of both wired and wireless networks both for multicast and unicast sessions [71, 72]. The basic idea of network coding is that a relay node combines several packets, which are intended for various receivers, into one packet and broadcasts it. Provided that each recipient has a priori knowledge of other packets (through overhearing), it can decode the desired packet from the aggregate packet. Therefore, the relay node is capable of forwarding more data within one transmission which eventually improves the overall throughput and reduce the cost of communications. In our work, we consider a simple network coding mechanism [47] where packets are linearly

coded through a simple operation (e.g., modulo-2 or XOR addition). We further assume that each coded packet is decodable at the next hop of a broadcast transmission. Another type of network coding, which is worth mentioning, is analog NC, which is a physical-layer technique and was introduced and discussed in [45]. Although analog NC seems simple to implement, this technique has many disadvantages (e.g., noise amplification, as the need to deal with symbol, phase, and frequency synchronization [46, 58].

Several coding topologies/structures can be constructed in the network for relaying the traffic (see Figure 5.1). In each coding structure, an *edge node* is a transmitter and/or receiver of different packets. A recipient edge node must know any uncoded packet except its desired one through either: 1) overhearing the link through which the packet was transmitted, or 2) being its previous transmitter. Coding structures constructed based on the former scheme of obtaining knowledge, are referred to as *network coding with opportunistic listening*, while the latter one is known as network coding without opportunistic listening in the literature [47]. In any particular coding structure, the node responsible for combining native received packets from other nodes in the same coding structure (edge nodes), is referred to as the *relay node.* We denote that in each coding structure, only packets from different flows can be encoded together. Figure 5.1 illustrates various coding structure, constructed based on the above rules, which we explain as follows.

Chain structure

Here, two packets from two flows traversing in reverse directions are coded without opportunistic listening (Figure 1(a)). For instance, relay node C upon



(a) Chain (b) Bell (c) X

Figure 5.1: Different coding components. Solid lines show intended transmission links and dashed lines show overhearing links. Note: Links are not necessarily symmetric.

receiving packets from both A and B (e.g., in two consecutive time slots) performs XOR operation and broadcasts the coded packet for both nodes A and B (in the third time slot). These two edge nodes subsequently can decode the coded packet by XORing it with their own native one to extract their desired packet. This coding structure reduces the required number of transmissions from 4 to 3, which is a 25% improvement [47].

X structure

A maximum of two packets which are sent in two consecutive time slots and intersecting at the relay node, are encoded (Figure 1(b)). The destination of each packet obtains the other unintended native packet by listening to its transmission (opportunistic listening). Later, the overheard packet is used to decode the intended packet. The performance of X component is similar to the Chain and provides 25% improvement by reducing the required number of transmissions in the network from 4 to 3.

Bell structure

A maximum of two packets are encoded where only one of the destinations obtains its unintended packet through opportunistic listening (Figure 1(c)).

As with the previous coding structures, this structure reduces the number of transmissions by 25%.

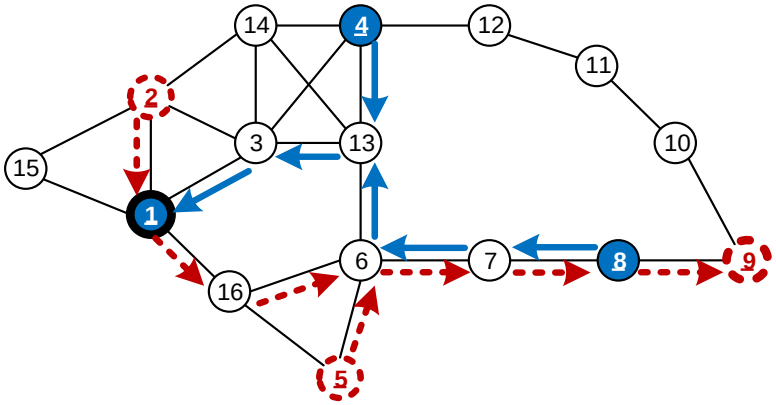
5.2 Problem Description and Motivation

Given a connected graph G of n nodes (sensors), a sink, and a sparse matrix Φ, our problem consists of finding m forwarding trees to collect measurements from n sensors in the most energy efficient manner; each tree T_t corresponds to one projection which gathers one weighted sum z_t from interest nodes (nodes with non-zero coefficients in a corresponding row of matrix Φ) at a random projection node P_t as explained in Section 3.2 for MSTP method. Here, our objective is to construct those trees such that the total number of transmissions in the network is minimized, by exploiting both projection based compressive data gathering and network coding techniques. Note that, similar to [73] the projection nodes are selected randomly (for example, can be considered as a priori knowledge for the entire network). However, selecting the appropriate projection nodes may have different impact on the number of data transmissions, but require more time, energy and efforts to find the most appropriate ones. We have studied the problem of finding the best projection nodes without considering the NC in Section 3.5, where, with the presence of NC, this problem is beyond the scope of this thesis.

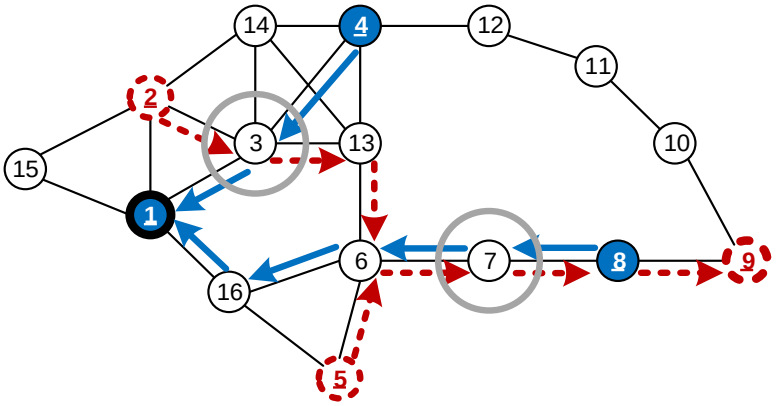
Clearly, the existence of forwarding trees creates opportunities for many-to-many traffic patterns which can be exploited to perform network coding on the (compressed) traffic belonging to different projections. Here, we distinguish between forwarder nodes and interest nodes. Interest nodes are those involved in data aggregation; that is, those nodes whose random coefficients are not zero and thus prior to forwarding the received aggregate/compressed

data received from downstream nodes, they combine their own measurement readings (as explained before) and then forward the aggregate data to their parents. Forwarder nodes, on the other hand, are those nodes whose random coefficients are zeros (for a particular projection), and thus they simply forward the received downstream data without doing any aggregation functions. Such forwarder nodes if they appear on more than one forwarding tree (or if forwarding trees are constructed in a way to include such forwarder nodes), they can perform network coding on the traffic flowing through the corresponding trees. We illustrate in the following example.

Consider the scenario shown in Figure 5.2, where projection node 9 needs to gather data from nodes 2 and 5, and similarly projection node 1 requires to gather data from nodes 4 and 8. The data for both projections may be gathered as shown in Figure 2(a), where two trees are constructed (optimal construction of trees is used [21]), tree 1 (dashed arrows) contains 7 links and thus requires 7 transmissions and tree 2 (dark arrows) requires 6 transmissions. In total, $7+6 = 13$ transmissions are required to gather the data for the two projections. Figure 2(b) shows a different construction of the trees which benefits from the existence of two coding components, namely a chain and a X structure at nodes 7 and 3 respectively. Traffic forwarded on both trees is network-coded (e.g., using XOR operations, as discussed earlier) at these forwarder nodes; for instance, it can be easily observed that the chain (6-7-8) reduces the number of transmissions by 1. In total, 11 transmissions are needed to gather the data from the interest nodes for the two projections, a total gain of 15.38% over the previous solution where network coding is not considered. The gain achieved using this simple method motivated us to study the optimal construction of network coding-aware aggregation trees.



(a) Total number of transmissions: 13



(b) Total number of transmissions:11

Figure 5.2: Data transmission scenario with and without network coding.

It should be noted here that the m projection nodes may be selected at random with a probability $\frac{m}{n}$ [73]; alternatively, the position of the projection nodes may be part of the design problem but their optimal selection is beyond the scope of this thesis. We assume in our work such positions are predetermined. In this work, we do not make any particular assumption about the link scheduling method; once forwarding trees are constructed, we assume any scheduling approach (e.g., TDMA-like or random-like) may be used to activate the wireless links connecting the adjacent sensors.

5.3 Optimal Tree Construction

In this section, we present a mathematical formulation for the problem of optimal construction of forwarding trees as a mixed integer linear program (MILP). Each tree connects a set of interest nodes to one pre-selected projection node; the projection node collects a weighted sum of measurements from the interest nodes through compressive data gathering. The trees are constructed to exploit network coding on weighted sums from different sets of interest nodes. We refer to this model as coding aware compressive data gathering. Projection nodes transmit the received weighted sums on shortest paths to the sink, which upon receiving all weighted sums, will decode to obtain the original data measurements. Our objective is to perform data collection in the most energy efficient manner, or alternatively, using the minimum number of transmissions.

The notations used throughout this section are listed in Table 6.1. Let T be a set of m trees and z_{ij}^t be a binary variable indicating whether there is an edge between nodes i and j in tree t. Let x_{ij}^t be a variable which represents the flow of traffic between nodes i and j in tree t. Let P_t denote a projection node (i.e., the root of tree t) and I_t denote the set of interest nodes for tree t. That is, $I_t = \{\text{nodes with non-zero coefficients } \phi_{ti}, i = 1, 2, ..., n\}$, $t = 1, 2, ..., m$. Let C_i denote the amount of network coding at node i; for example, C_i represents the number of times node i performs network coding on the traffic/packets traversing through it. Then, our objective is to construct trees that minimize the number of transmissions, which can be achieved by simultaneously minimizing the total number of edges in each of the m trees and maximizing the

Table 5.1: Notation Used in the Optimization Model of NC-CDG

V	Set of nodes in the network.
E	Set of edges in the network.
n	Total number of nodes.
m	Total number of projections.
l_{ij}	The directed link connecting node i to j.
Γ_{jik}	The directed segment connecting node j to k through node i.
x_{ij}^t	The amount of flow on link l_{ij} and tree t.
z_{ij}^t	Indicating whether link l_{ij} in tree t is active.
P_t	The projection node (root) for tree t.
I_t	The set of interest nodes for projection or tree t.
T	The set of m forwarding trees.
b_j^t	Indicating whether node j in tree t has more than one child.
f_{ij}^t	Indicating whether link l_{ij} in tree t is a forwarder link (i.e., node j in tree t has only one child i).
F_j	The number of forwarder links to node j .
w_{jik}^t	Indicating whether segment Γ_{jik} in tree t is a forwarder-segment
W_{jik}	Total amount of forwarder-segments traversing the segment Γ_{jik}
$N(i,j)$	Parameter indicating whether nodes i and j can hear each other
$p(i)$	Set of segments that traverse intermediate node i
$c_{jij'}^{kik'}$	The number of times node i has been intersected by two directed forwarder segments $\Gamma_{jij'}$ and $\Gamma_{kik'}$.
C_i	The total number of network coding instances at node i.

number of possible network coding at each (forwarder) node:

$$Minimize \qquad \sum_{t=1}^{m} \sum_{(i,j)\in E} z_{ij}^t - \sum_{i=1}^{n} C_i \tag{5.1}$$

Flow conservation constraints:

The flow conservation stats that at each node the total incoming flow plus the flow originating at the current node equals the total outgoing flow. The following constraints force all interest nodes (in I_t) belonging to one tree (projection)

to have each one flow to a projection node P_t:

$$\sum_{j:(i,j)\in E} x_{ij}^t - \sum_{j:(j,i)\in E} x_{ji}^t = \begin{cases} -I_t, & i = P_t; \\ 1, & \forall i \in I_t; \\ 0, & otherwise. \end{cases} \qquad \forall t \in T \tag{5.2}$$

Connectivity constraints:

These constraints create the relation between flow variables x_{ij}^t and link variables z_{ij}^t. The edges which have positive flows are indicated as tree links. This implies that $x_{ij}^t = 0 \Leftrightarrow z_{ij}^t = 0$ and $x_{ij}^t > 0 \Leftrightarrow z_{ij}^t = 1$. Therefore, we have:

$$\begin{cases} x_{ij}^t - z_{ij}^t \geq 0 \\ z_{ij}^t - m^{-1} x_{ij}^t \geq 0 \end{cases} , \forall (i,j) \in E \quad , \quad \forall t \in T \tag{5.3}$$

Transmission constraint:

This constraint asserts that each node should have a maximum of one transmission (i.e., outgoing edge) in each tree to avoid loops.

$$\sum_{j:(i,j)\in E} z_{ij}^t \leq 1 \qquad , \forall t \in T \tag{5.4}$$

Directional constraint:

Links in each tree must have only one direction. The following constraint forces each edge to have a maximum of one direction in each tree.

$$z_{ij}^t + z_{ji}^t \leq 1 \qquad \forall (i,j) \in E \quad , \quad \forall t \in T \tag{5.5}$$

Next we present the constraints necessary to find the amount of traffic to be coded at each forwarder node. Such forwarder nodes are referred as relay nodes in the rest of the chapter.

Constraints to find forwarder links to relay nodes:

We start by identifying all the links (transmissions) to relay nodes which are not going to be used for compressive coding/gathering but rather such transmissions are intended to be forwarded by relay nodes to next (parent) nodes. Such links are referred to as forwarder-links.

Let the binary variable f_{ij}^t denote whether the link between two nodes i and j on tree t is a forwarder-link or not. A link l_{ij} in tree t is a forwarder-link for node j if node i is the only child for j. In other words, when link l_{ij} is the only incoming edge to node j in tree t, this link is a forwarder-link since there are no other transmissions incoming to node j which should be combined (gathered using compressive sensing method) with the transmission or packet arriving from node i. Let b_j^t be a binary variable which indicates whether the number of links of tree t incoming at node j is greater than one or not. b_j^t is defined as follows:

$$b_j^t = \begin{cases} 1, & if \quad \sum_{i \in V} z_{ij}^t > 1 \\ 0, & otherwise. \end{cases} \tag{5.6}$$

Using Linear Programming (LP) notations, the constraints for finding b_j^t are as follows:

$$\begin{cases} b_j^t \leq \sum_{i \in V} z_{ij}^t \\ b_j^t \leq |\sum_{i \in V} z_{ij}^t - \frac{1}{m}| \qquad ,\forall j \in V \quad , \quad \forall t \in T \\ b_j^t \geq \frac{\sum_{i \in V} z_{ij}^t - 1}{m} \end{cases} \tag{5.7}$$

Now, an edge l_{ij} in tree t is a forwarder link (i.e., $f_{ij}^t = 1$) if the number of incoming links to node j in tree t is not more than one (i.e., $b_j = 0$) and edge l_{ij} is a selected link for tree t (i.e., $z_{ij}^t = 1$) and node j is not the projection node P_t (i.e, the root of t) or any of the interest nodes in I_t. The variable f_{ij}^t indicating whether l_{ij} is a forwarder-link is defined as follows:

$$f_{ij}^t = \begin{cases} 1, & if \quad b_j^t = 0, z_{ij}^t = 1, j \notin \{I_t \& P_t\} \\ 0, & otherwise. \end{cases} \tag{5.8}$$

The corresponding LP constraints are:

$$\begin{cases} f_{ij}^t \leq z_{ij}^t \\ f_{ij}^t \leq 1 - b_j^t \quad , \forall (i,j) \in E \quad , j \notin \{I_t, P_t\} \quad , \forall t \in T \\ f_{ij}^t \geq z_{ij}^t - b_j^t \end{cases} \tag{5.9}$$

Therefore, the total number of forwarder-links for node j can be obtained by summing all the forwarder-links coming to node j on all forwarding trees:

$$F_j = \sum_{t \in T} \sum_{i \in V} f_{ij}^t \qquad , \forall j \in V \tag{5.10}$$

Forwarder-segment constraints:

The term forwarder-segment is defined as a two-hop path or segment of a path on a tree that a packet is expected to traverse. This segment contains a relay node that will simply forward the incoming traffic without any aggregation function; such node may be exploited to perform network coding on the transiting traffic/transmissions. Let w_{jik}^t be a binary variable which indicates whether the segment Γ_{jik}^t in tree t is a forwarder-segment or not (j and k are

child and parent nodes respectively of i and i is a relay node). Then, $w^t_{jik} = 1$ for a forwarder-segment and 0 otherwise. Γ^t_{jik} is a forwarder-segment if both f^t_{ji} and z^t_{ik} are simultaneously 1. Here, $f^t_{ji} = 1$ implies we have an incoming forwarder-link to node i, and $z^t_{ik} = 1$ indicates an outgoing link from node i to node k. The mathematical constraints for a forwarder-segment are:

$$\begin{cases} w^t_{jik} \leq f^t_{ji} \\ w^t_{jik} \leq z^t_{ik} \\ w^t_{jik} \geq f^t_{ji} + z^t_{ik} - 1 \end{cases} , \forall \begin{matrix} (i,j) \in E \\ \\ (i,k) \in E \end{matrix} , \forall i \in V \quad , \forall t \in T \tag{5.11}$$

Summing over all forwarding trees, we obtain the number of forwarder segments traversing the structure Γ_{jik} in the network. Let W_{jik} be such number:

$$W_{jik} = \sum_{t \in T} w^t_{jik} \qquad \forall (j,i) \in E \quad , \quad \forall (i,k) \in E \tag{5.12}$$

Network Coding participation constraints:

Let $p(i)$ denote the set of segments which traverse intermediate node i (for example, Γ_{AiB} or Γ_{BiA} in Figure 5.3). Let $N(i,j)$ (a binary parameter) denote whether two nodes i and j can listen to each other's transmissions. That is:

$$N(i,j) = \begin{cases} 1, & \text{if } i = j \text{ or } i \text{ and } j \text{ are neighbours} \\ 0, & \text{otherwise.} \end{cases} \tag{5.13}$$

Both parameters $p(i)$ and $N(i,j)$ are computed offline for a given connected graph .

Let $c^{kik'}_{jij'}$ denote the number of times a node i has been intersected/crossed by two directed forwarder-segments $\Gamma_{jij'}$ and $\Gamma_{kik'}$ (i.e., there is a likelihood of

network coding at i). The following give the coding participation constraints for any two segments intersecting at i:

$$\begin{cases} c_{jij'}^{kik'} \leq W_{jij'}.N(j,k') \\ c_{jij'}^{kik'} \leq W_{kik'}.N(k,j') \end{cases} \quad \forall \begin{array}{l} (j,i,j') \in p(i) \\ (k,i,k') \in p(i) \end{array} \quad ,\forall i \in V \tag{5.14}$$

Constraints (5.14) assert that when two forwarder-segments $\Gamma_{jij'}$ and $\Gamma_{kik'}$ intersect at intermediate node i, then if two nodes j and k', as well as k and j', can listen to each other's transmission, node i may perform network coding on the traversing traffic.

Amount of network coding at each node:

Figure 5.4 and Figure 5.3 show two different network coding opportunities with and without opportunistic listening respectively. The tables next to the figures show the different values that variable c can obtain. Clearly, for each coding structure, there is a duplicate value for the variable c. For example, in Figure 5.3 c_{BiA}^{AiB} is the duplicate of c_{AiB}^{BiA}. Therefore, the total amount of network coding a node can perform on the traversing traffic is computed as:

$$C_i = \frac{\sum_{(j,i,j') \in p(i)} \sum_{(k,i,k') \in p(i)} c_{jij'}^{kik'}}{2} \quad ,\forall i \in V \tag{5.15}$$

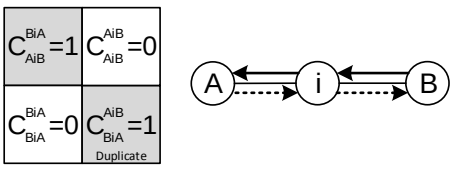


Figure 5.3: Network coding without opportunistic listening

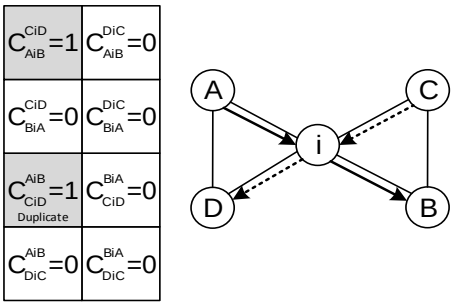


Figure 5.4: Network coding with opportunistic listening

Maximum network coding bound:

The total amount of network coding at each node should not exceed half of the total number of forwarder links to a node. An example is illustrated in Figure 5.5 where node i may do coding for traffic traversing segments Γ_{AiB} and Γ_{FiE}, or Γ_{AiB} and Γ_{DiC}, or Γ_{FiE} and Γ_{DiC}. The above presented constraints will calculate three different coding amounts for node i in this example. However, in reality, node i should only choose one coding structure (to avoid one transmission being network coded in more than one coding component). Therefore, the following constraint will limit the upper bound on the number of coding at each node:

$$C_i \leq \frac{F_i}{2} \qquad ,\forall i \in V \tag{5.16}$$

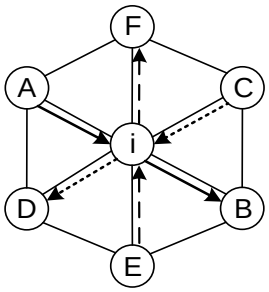


Figure 5.5: Maximum network coding bound

The value ranges of decision variables:

$$x_{ij}^{t} \geq 0 \qquad ,\forall (i,j) \in E \quad AND \quad \forall t \in T \tag{5.17}$$

$$z_{ij}^{t} \in \{0,1\} \qquad ,\forall (i,j) \in E \quad AND \quad \forall t \in T \tag{5.18}$$

$$b_{j}^{t} \in \{0,1\} \qquad ,\forall j \in V \quad AND \quad \forall t \in T \tag{5.19}$$

$$f_{ij}^{t} \in \{0,1\} \qquad ,\forall (i,j) \in E \quad AND \quad \forall t \in T \tag{5.20}$$

$$F_{j} \geq 0 \qquad ,\forall j \in V \tag{5.21}$$

$$w_{jik}^{t} \in \{0,1\} \qquad ,\forall (j,i) \in E \ ,\forall (i,k) \in E \ ,\forall t \in T \tag{5.22}$$

$$W_{jik} \geq 0 \qquad ,\forall (j,i) \in E \quad AND \quad \forall (i,k) \in E \tag{5.23}$$

$$c_{jij'}^{kik'} \geq 0 \qquad ,\forall (j,i,j') \in p(i), \ \forall (k,i,k') \in p(i) \tag{5.24}$$

$$C_{i} \geq 0 \qquad ,\forall i \in V \tag{5.25}$$

5.4 Algorithmic Solutions

In this section, we present our algorithmic approach for solving the coding-aware forwarding tree construction for compressive data gathering (NC-CDG). We present both centralized and distributed methods for solving the NC-CDG problem. The NC-CDG calls for constructing m forwarding trees to gather data from the sensors in the most energy efficient manner. As in the previous section, the objective is to minimize the total number of transmissions for collecting the sensed data. Before presenting our methods, we note the similarity between our problem and the Steiner tree problem;the Steiner tree problem finds a tree in a graph $G(V, E)$ that spans $S \subseteq V$ with minimum total distance on its edges. For each projection, if we let S be the set of interest nodes plus the projection node, the Steiner tree problem will be similar to our problem of

constructing one forwarding tree.

5.4.1 Centralized Method:

The centralized algorithm has three phases. Initially, phase 1 constructs the m forwarding trees. Phase 2, based on the constructed trees, calculates and obtains the total amount of network coding at each node separately, and phase 3 updates the routes of the trees to further reduce the overall number of transmissions. The details of each phase are provided next.

The steps of phase 1 are shown in Algorithm 5.1. At initialization, the algorithm for each projection t retrieves the nodes whose coefficients in the basis matrix Φ_{ti}, $i = 1, 2, ..., n$ are non-zero, adds them to the interest nodes list (Int_t) and then assigns the projection node P_t as a single node tree T_t (P_t is selected at random). In the second step, the algorithm removes the nodes in Int_t which can be connected directly to tree T_t and adds them to T_t using the Minimum-Spanning-Tree (MST) algorithm. Since interest nodes are spread throughout the network and may not be reached directly without multi-hop, there may still be nodes in Int_t which could not be connected to T_t using MST. Therefore, as long as Int_t is not empty, the algorithm finds the nearest node in Int_t to T_t using the Breadth-First-Search (BFS) algorithm and adds that node plus all the intermediate nodes to T_t and removes that node from the list. Again, the algorithm repeats by adding more nodes from Int_t to T_t tree using MST if possible. Upon termination, the algorithm returns all the m trees T_t ($t = 1, 2, ..., m$). The time complexity is $O(md\rho^2 \log \rho)$ in the best case (when all nodes can be connected to the projection nodes using MST) and $O(md\rho^2 n \log n)$ in the worst case (when non of the interest nodes can be connected to the trees using MST), where d is the average nodal degree of the nodes and ρ is the

Algorithm 5.1 NC-CDG: Constructing m forwarding trees (Phase 1)

Require: Graph $G(V, E)$, Matrix Φ, Set of projection nodes P_t ($t = 1, 2, ..., m$), and the sink.

Ensure: Set of trees T_t.

1: **for** each projection t ($t = 1, 2, ..., m$) **do**
2: Let tree $T_t = \{P_t\}$, and
3: set of interest nodes $Int_t = I_t = \{$ all nodes s.t. $\phi_{ti} \neq 0, i = 1, 2, ..., n\}$.
4: Add those nodes in Int_t into tree T_t which can be connected directly to nodes in T_t using MST and remove them from Int_t.
5: **while** $!Empty(Int_t)$ **do**
6: Find nearest node h in Int_t to T_t using BFS from nodes in Int_t to T_t.
7: Add node h to T_t plus all the intermediate nodes in shortest path to T_t
8: Remove node h from Int_t.
9: **if** Int_t is not empty **then**
10: execute step 4.
11: **end if**
12: **end while**Connect the projection node P_t to the sink through shortest-path.
13: **end for**

number of interest nodes. For time complexity analysis refer to Section 3.2.

Algorithm 5.2 shows the steps of phase 2. At each node, the algorithm determines the maximum amount of traffic (transmissions) which can be coded and which is obtained from the m forwarding trees. We explain the steps at a node i (a similar procedure is repeated for others). Each node maintains two lists: the network coding list and the segment list, both are initially empty (i.e., $NCL_i = \varnothing$ and $Candidate_i = \varnothing$); initially, the total number of codings (transmissions which can be coded) at node i is set to zero (i.e.; $C_i = 0$). Then, for each tree t the algorithm checks whether the total number of incoming tree links to node i is greater than one or not (b_i^t indicates that). In other words, if $\sum_{j=1}^{n} z_{ji}^t > 1$, $b_i^t = 1$ and zero otherwise (lines 4-8). Next (lines 9-14), for each incoming link to node i (i.e.; $z_{ji}^t = 1$), if $b_i^t = 0$, a forwarding-link variable f_{ji}^t is set to one (i.e.; $f_{ji}^t = 1$). Now, for each forwarder-link variable where $f_{ji}^t = 1$, if node i is not an interest or projection node for tree t (i.e., node i

has an outgoing link in tree t and will not aggregate or compress any packets), node i might use the forwarder-segment Γ^t_{jik} (obtained by joining incoming tree link l_{ji} with outgoing tree link l_{ik}, where k is the parent of node i in tree t) for network coding with forwarding-segment of other trees. Therefore, node i puts the forwarder-segment Γ^t_{jik} whose $w^t_{jik} = 1$ in candidate list $Candidate_i$ for possibility of network coding with other forwarder-segments (lines 16-19).

Algorithm 5.2 NC-CDG: Calculating network coding for each node (Phase 2)

Require: Set of trees T_t.
Ensure: The total and set of network coding at each node.

1: **for** each node $i \in V$ **do**
2: Set network coding list $NCL_i = \varnothing$, $Candidate_i = \varnothing$, $C_i = 0$.
3: **for** each tree $t \in T$ **do**
4: **if** $\sum_{j=1}^{n} z^t_{ji} > 1$ **then**
5: $b^t_i = 1$.
6: **else**
7: $b^t_i = 0$.
8: **end if**
9: **for** each $z^t_{ji} = 1$ **do**
10: **if** $b^t_i = 0$ **then**
11: $f^t_{ji} = 1$.
12: **else**
13: $f^t_{ji} = 0$.
14: **end if**
15: **end for**
16: **if** $i \notin \{P_t \& I_t\}$ AND $f^t_{ji} = 1$ **then**
17: Let k be the parent for node i.
18: Add forwarder-segment Γ^t_{jik} to candidate list $Candidate_i$.
19: **end if**
20: **end for**
21: **for** each two different forwarder-segments $\Gamma^t_{jij'}$ and $\Gamma^{t'}_{kik'}$ in $Candidate_i$ list, where $j \neq k$ and $j' \neq k'$ **do**
22: **if** $N(j, k')$ AND $N(k, j')$ **then**
23: Add $c^{kik'}_{jij'}$ to NCL_i list, and remove $\Gamma^t_{jij'}$ and $\Gamma^{t'}_{kik'}$ from $Candidate_i$ list
24: Increment the total number of network coding $C_i = C_i + 1$.
25: **end if**
26: **end for**
27: **end for**
28: **return** NCL_i and C_i.

From the $Candidate_i$ list, the algorithm chooses two different segments $\Gamma^t_{jij'}$ and $\Gamma^{t'}_{kik'}$, where $j \neq k$ and $j' \neq k'$ (line 21). If nodes j and k can listen to the transmissions of k' and j' respectively (line 22), i may combine and transmit the two packets coming from j and k into one coded transmission. Consequently, the algorithm adds $c^{kik'}_{jij'}$ into the NCL_i list, increments the total number of network coding C_i by one, and removes the two segments from $Candidate_i$ list (lines 23-24). The last step (lines 21-26) will be repeated until there are no two forwarder-segments in the $Candidate_i$ list which can be coded together. At termination, the algorithm returns the total number of network coding C_i and the list NCL_i for each node $i = 1, 2, ..., n$. At each node, the algorithm takes $O(md^2)$ and hence the overall time complexity is $O(nmd^2)$.

In phase 3 (steps shown in Algorithm 5.3), each node i attempts to discover a new route (if that exists) to the projection node of each tree in a way to improve the overall transmission cost. Algorithm 5.3 starts by checking if node i in tree t is one of the interest nodes in set I_t and its parent (π^t_i) does not belong to any interest node set I_t (for example, node 5 in Figure 5.6; in this figure, dark nodes are interest nodes and arrows represent the forwarding tree). If these conditions are satisfied for node i, then the algorithm (in line 4) removes all the successive tree links from node i to a node that is either an interest node or has more than one child (an example is illustrated in Figure 5.6; for node 5, the path shown by arrows with 'x' from node 5 to node 2 is removed; note that node 2 has two children). Let b represent the node that has more than one child. Let R be the total number of removed links. The total amount of network coding at those nodes which are on the removed path is calculated and stored in a variable RC. In this step, interest node i and its descendants are disconnected from the main tree t (e.g., node 5 in Figure 5.6 which has been

disconnected from the tree). Next, to discover an alternative path to connect node i to the main tree t, node i (lines 8-9) searches in a radius equals to R, using Breadth-First-Search (BFS), for a node(s) in tree t (if any) such that the cost of transmission (e.g., number of transmissions) is improved (e.g., in Figure 5.6, node 5, which has been disconnected, can connect to the tree through different paths; for example, connecting to node 4 through the intermediate node 6, as it is shown in the figure with dashed arrows, or to node 2 or 7. Here, since all the paths have the same hop distance, the algorithm will choose a path which has better coding capabilities). To find a better path, for each node g found on tree t in a radius R, the algorithm calculates the hop distance (h_g) and the total number of possible network coding (O_g) from node i to g. Then, the algorithm will choose a path whose overall gain from network coding is better than other paths (lines 10-17). The time complexity for the algorithm is $O(\rho m \delta)$, where δ is the average number of nodes for different values of R. This is because, each interest node (among ρ) for each tree (among m), using BFS (Breath First Search Algorithm), searches for nodes in radius R to find a better route. BFS algorithm has time complexity of O(number of nodes in the radius search). Therefore, the total time complexity will be $O(\rho m \delta)$.

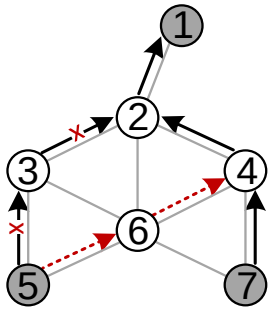


Figure 5.6: Updating route example in NC-CDG

Algorithm 5.3 NC-CDG: Updating the routes of the trees (Phase 3)

Require: Set of trees T_t ($t = 1, 2, ..., m$), Set of coding list NCL_i ($i = 1, 2, .., n$).
Ensure: Updated trees T_t.

for each node $i \in V$ **do**
  for each tree $t \in T$ **do**
    if $i \in I_t$ AND $\pi_i \notin I_t$ **then**
      Remove all the successive tree links from node i to first interest node b or has more than one child in tree t.
      $R \leftarrow$ Total number of removed links.
      $RC \leftarrow$ Total number of network coding on nodes which have been removed from tree t.
      $B_i = RC - R$.
      Run Breath-First-Search (BFS) algorithm from i in a radius equals to R.
      if node(s) other than b on disconnected main tree t is found in this radius **then**
        for each found node g **do**
          $h_g \leftarrow$ Hop distance from i to g.
          $O_g \leftarrow$ Total number of network coding on nodes from i to g.
          if $(O_g - h_g) > B_i$ **then**
            $B_i = O_g - d_g$.
            $b \leftarrow g$.
          end if
        end for
      end if
      Connect node i to b in shortest-path.
      Update the list and number of network coding on nodes based on the path changes.
    end if
  end for
end for

5.4.2 Distributed Method:

A drawback of the centralized approach is that a central unit performs a topology discovery by retrieving network wide information through an all-to-all flooding, and then solves the algorithm to construct the forwarding trees and subsequently notifies each node in the network with necessary information to execute the route process. Clearly, the overhead associated with such centralized approach makes it costly for large networks and does not respond well to topological changes (e.g., in the presence of node or link outages). Therefore, it is more desirable to distribute the computation on individual nodes. In this subsection, we present a distributed algorithm for constructing forwarding trees, where each node locally makes a decision in the routing process and for each tree (projection), the node decides to whom it should transmit its data packet.

The distributed method consists of four phases. The first two phases are related to constructing forwarding trees and the steps are given in Algorithm 5.4. The last two phases for calculating the network coding and updating the routing trees are the same as Algorithms 5.2 and 5.3, since both algorithms could be executed locally at each node. For both algorithms, nodes may retrieve information needed for their computation (e.g., network coding list NCL) from nearby nodes by sending request messages. Below we describe the tree construction for the distributed method.

Initially (Phase 1), each projection node (P_t) starts by sending a discovery message to its neighbors. Each node, upon receiving the message, will broadcast it to allow other nodes, not close to the projection node, to receive the discovery message. Hence, each node i will learn its shortest path(s) ($Spath^t_{iP_t}$) to the projection node P_t as well as the hop count (h^t_i) along the path. Further,

Algorithm 5.4 NC-CDG: Distributed Forwarding Tree Construction

1: **Phase 1:**
2: **for** each projection-node P_t ($t = 1, 2, ..., m$) **do**
3: Disseminate the discovery message to all nodes by running BFS algorithm. Each node $i \in V$ learns its shortest-path(s) ($Spath_i^t$) and hop-count h_i^t to the root (P_t).
4: **end for**
5: **Phase 2:**
6: **for** each node $i \in V$ **do**
7: **for** each tree $t \in T$ **do**
8: **if** $i \in I_t$ **then**
9: **if** $root \in N(i)$ **then**
10: $\pi_i^t = P_t$;
11: Set and broadcast $flag_i^t = 1$;
12: **else if** $b \in N(i)$ AND $b \in I_t$ AND $flag_b^t = 1$ **then**
13: $\pi_i^t = b$;
14: Set and broadcast $flag_i^t = 1$;
15: **else if** $b \in N(i)$ AND $b \in I_t$ AND $h_b^t < h_i^t$ **then**
16: $\pi_i^t = b$;
17: Set $flag_i^t = 0$;
18: **else if** $b \in N(i)$ AND $b \in I_t$ AND $h_b^t = h_i^t$ AND successive parents of b reach a node with smaller hop-count or $flag = 1$ or non-parent and do not reach i **then**
19: $\pi_i^t = b$;
20: Set $flag_i^t = 0$;
21: **else**
22: Run BFS from i in a radius equals to $h_i^t - 1$.
23: **if** interest-node(s) in this radius found **then**
24: Connect i to nearest interest-node through shortest path.
25: Set $flag_i^t = 0$;
26: **else**
27: Connect node i through its shortest path to the root.
28: **end if**
29: **end if**
30: **if** node i receives a notification message of any changes **then**
31: Repeat lines (8-33).
32: **end if**
33: **end if**
34: **end for**
35: **end for**

node i discovers its neighbor set $N(i)$. Node i, upon checking matrix Φ, which is stored in its memory, determines whether node $u \in N(i)$ ($\forall u$) belongs to the set of interest nodes (I_t) of tree t or not. In Phase 2, each node i for each tree t, if it is an interest node, decides its parent on the uplink path to the projection node P_t. For each interest node, we assign an attribute to designate its parent interest node (π_i^t) (note a parent interest node could be a neighbor of i or can be reached through other relay nodes) and a decision flag ($flag_i^t$) to indicate whether the parent interest node of i is fixed. Lines (9-11) show that every interest node which is a neighbor of the root selects the root as its parent node and sets and distributes its decision flag $flag_i^t = 1$. Now, if interest node i (Lines (12-14)) is not a neighbor of the root, but has an interest node neighbor b with $flag_b^t = 1$, then i selects b as its parent interest node and commits its decision ($flag_i^t = 1$). In the case where none of the neighboring interest nodes (b) of i has its decision flag set (i.e., $flag_b^t = 0$), i will select the neighboring interest node with the smaller hop-count to the sink as its parent interest node (Lines 15-17). Now, when only interest node neighbors with equal hop-count to the sink as i can be found (Lines 18-20), i selects the one (b) whose successive parents reach an interest node with smaller hop-count or decision flag $flag = 1$ or no parent node, and does not reach node i (to avoid loops). If none of the above conditions is satisfied, i runs a BFS to explore its neighborhood of radius $h_i^t - 1$ in search for an interest node b with smaller hop count to the sink; otherwise, for an interest node whose decision flag $flag_b^t = 1$ (Lines 22-25). Node i avoids selecting interest nodes b whose $\pi_b^t = i$ to avoid loops. Finally, if no interest node is found, i connects itself directly through a shortest path to the sink (this path is known from the discovery phase). Node i will repeat the route discovery in lines (8-33) if it receives a notification message from its neighbors

indicating that there is a change in the network, e.g., change in a decision flag, or following a node or link failure due to mobility or channel impairments occurring in the network triggering route maintenance. The time complexity for distributed algorithm in phase 1 for each projection node is $O(n)$, and $O(mn)$ for all projection nodes. In phase 2, the time complexity for each interest node i in tree t is $O(1)$ in the best case and $O(\gamma)$ in the worst case, where γ is the number of nodes around node i and within a radius $h_i^t - 1$. Note that nodes in the distributed approach simultaneously execute the algorithm.

5.4.3 Performance Analysis:

In this section, we attempt to derive theoretical performance bounds on the algorithmic solution we presented above. We start by noting that our problem for constructing each aggregation tree is similar to the Steiner tree problem in that we connect all interest-nodes $I \subseteq V$ and the sink together such that the constructed spanning tree has a minimum total distance on its edges. The difference between our tree construction with the minimum Steiner tree is that our tree is rooted at the sink which makes a difference in selecting the appropriate minimum spanning tree when there are several minimum spanning trees possible for a given graph or network. In [49], the authors proved that the edges on the minimum steiner tree in the worst case have a total distance no more than $2(1 - \frac{1}{l})$ times that of the optimal tree, where l is the number of leaves in the optimal tree. In our problem, the worst case occurs when the overlaps of the trees do not make opportunities for NC, and hence we have zero NC in the network. Therefore, the worst case performance of our NC-CDG algorithmic method will not be worse than $2(1 - \frac{1}{l})$. Recall that the total number of leaves in a tree is equal to or less than the number of interest-nodes, and

there are $\lceil \frac{n}{m} \rceil$ interest-nodes. Thus, our NC-CDG method in the worst case performs no worse than $2(1 - \frac{1}{\lceil \frac{n}{m} \rceil})$. i.e.;

$$\frac{NC_{CDG-alg}}{NC_{CDG-opt}} \leq \frac{Steiner_{alg}}{Steiner_{opt}} \leq 2(1 - \frac{1}{\lceil \frac{n}{m} \rceil}) \tag{5.26}$$

For example, if $n = 100$ and $m = 20$, the upper bound performance of our algorithmic method is $2(1 - \frac{1}{5})$, which is $\frac{8}{5}$-approximation. We should note however that the upper bound given above is not the tightest possible bound.

5.5 Performance Evaluation

This section presents numerical and simulation results obtained by solving the various methods presented earlier; namely, we numerically study the performance of Network Coding aware tree construction for Compressive Data Gathering (NC-CDG) and compare it with a method that does not exploit network coding for tree construction (CDG) [21]. We also compare centralized and distributed algorithmic implementations of both NC-CDG and CDG. We consider networks of different sizes; each network is randomly generated and nodes are uniformly distributed over a region such that the resulting graph is connected. We assume all nodes use the same transmit power. The metrics of comparisons are 1) gain achieved from network coding 2) the total number of transmissions 3) transmission load distribution across the sensors.

NC-CDG Vs. CDG:

We first start by comparing the optimal forwarding tree construction using the 20-node network topology shown in Figures 5.7 and 5.8. In this example, the number of projections (trees) is $m = 4$ and for each tree, a projection

node (with dashed border line) is required to gather a weighted sum from four interest nodes (with same color). Figure 5.7 illustrates the optimal forwarding trees without considering network coding (CDG) [21] where it is easy to verify that in total 30 transmissions are required to fulfill the gathering at projection nodes. Note that, a total of 16 transmissions are needed to forward the weighted sums from projection nodes to the sink through shortest paths. Therefore, Figure 5.7 overall requires 46 transmissions. In Figure 5.8, the forwarding trees are constructed using NC-CDG; as one can observe, the optimal routing trees use different paths to allow four nodes (1, 4, 7 and 8) to perform XOR-coding. This optimal tree construction gathers all weighted sums at their projection nodes with $30 - 4 = 26$ transmissions (and at sink with 42 transmissions); hence, this method outperforms the former one and yields a gain of 8.7% in transmission cost reduction.

Now, we evaluate the performance of NC-CDG for larger networks ($20 \leq n \leq 50$) with different number of projections ($m = 5$, $m = 6$) and present the average results of five runs varying different matrix Φ and projection nodes for comparison. Table 5.2 depicts the overall number of data transmissions required to gather all the sensed data at the sink; the table shows a base model where compressive sensing is not used for data gathering (Non-CDG) and simple shortest-paths are used for collecting the data. The results indicate that both NC-CDG and CDG outperform the base model; for instance, the gains of NC-CDG over non-CDG range between 11% to 47.88% whereas the gains of NC-CDG over CDG vary between 3.08% and 12.11%. It should be noted that these gains strongly depend on the size of the network, the projection nodes and matrix Φ (i.e., position of interest nodes and projection nodes). Indeed, the larger the network is, and the more forwarding trees (projections) there are,

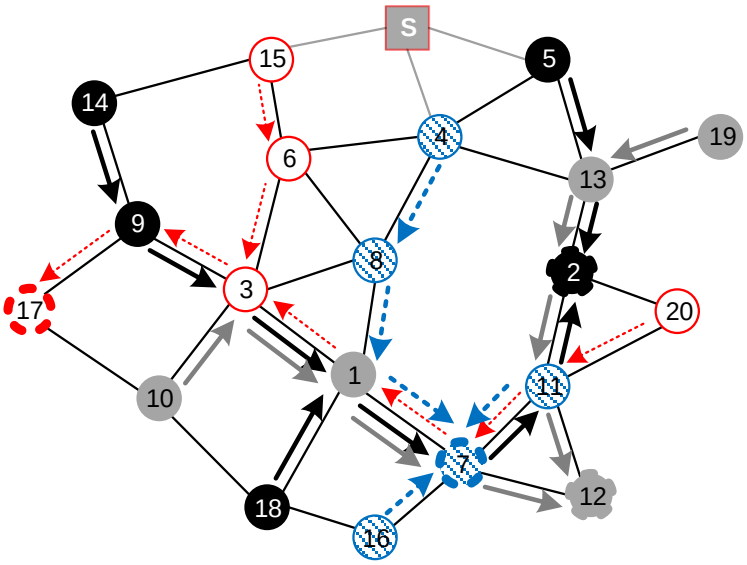


Figure 5.7: Optimal tree construction without Network Coding

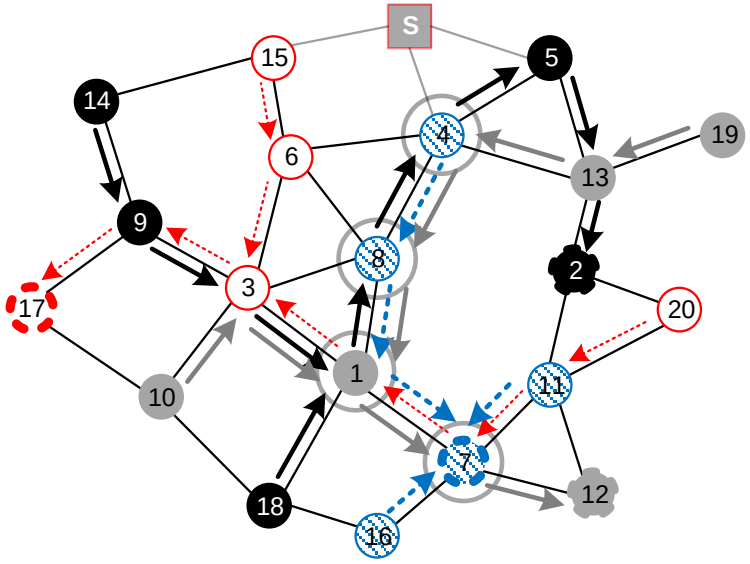


Figure 5.8: Optimal tree construction with Network Coding

the more coding opportunities there could be between the forwarding trees and thus the higher the gains are. Given the complexity of the models, we were not able to run them for larger network instances.

Table 5.2: Overall number of data transmissions (NC-CDG vs CDG)

# Nodes	# Projections	NC-CDG	CDG	Non-CDG
$n = 20$	$m = 5$	44.6	47.2	60
	$m = 6$	53.4	56.2	60
$n = 30$	$m = 5$	70.6	74	114
	$m = 6$	79	85	114
$n = 40$	$m = 5$	100.6	103.8	193
	$m = 6$	110	115.8	193
$n = 50$	$m = 5$	114.6	127.6	206
	$m = 6$	135	153.6	206

Performance of the algorithms:

This section will evaluate the performance of the algorithmic methods we presented earlier. We compare NC-CDG with the centralized method (NC-CDG/C) and the distributed method (NC-CDG/D). The objective of this comparison is to showcase the effectiveness of both methods in reaching solutions which are close to those obtained by the NC-CDG optimal method. We also compare CDG/C [22] and CDG/D [23] with the CDG [21] method. The results (overall number of transmissions for delivering the sensed data) of these comparisons are presented in Tables 5.3 and 5.4; clearly, the results indicate that both centralized and distributed methods (both with and without network coding) achieve very close performance to those obtained in the models with a worst case gap of 5.78% (for CDG/D to CDG) and 3.29% (for NC-CDG/D to NC-CDG) in the studied scenarios. We build on such results to study the performance of the coding-aware CDG on larger networks using the algorithmic methods.

Note that the optimal solution of NC-CDG using Cplex solver takes time in average between one and half minutes for 20-node network size to two hours and twelve minutes and sometimes over a day for a 50-node network. Whereas, the algorithmic (heuristic) method takes between one, two to four seconds to solve for 20-node, 50-node to 100-node network size respectively. However, NC-CDG optimal was incapable to solve for 100-node network. We run our program on CPU with Intel Core i7 processor, 2.67 GHz speed, 6 GB memory ram and 64-bit windows operating system.

Table 5.3: Overall number of data transmissions (NC-CDG vs Algorithms)

# Nodes	# Projections	NC-CDG	NC-CDG/C	NC-CDG/D
$n = 20$	$m = 5$	44.6	45.6	45.2
	$m = 6$	53.4	54.4	54
$n = 30$	$m = 5$	70.6	71.2	71.6
	$m = 6$	79	81.2	81.6
$n = 40$	$m = 5$	100.6	102.4	102.6
	$m = 6$	110	112.2	113.4
$n = 50$	$m = 5$	114.6	117.6	116.4
	$m = 6$	135	137.2	136.4

Table 5.4: Overall number of data transmissions (CDG vs Algorithms)

# Nodes	# Projections	CDG	CDG/C	CDG/D
$n = 20$	$m = 5$	47.2	48	48
	$m = 6$	56.2	57	57.2
$n = 30$	$m = 5$	74	74.8	76.2
	$m = 6$	85	85.4	87.4
$n = 40$	$m = 5$	103.8	105.6	109.8
	$m = 6$	115.8	117.4	120.6
$n = 50$	$m = 5$	127.6	129.4	131.2
	$m = 6$	153.6	156.6	159.4

Performance on larger networks:

This section will evaluate the performance of network coding aware tree construction for compressive data gathering on larger networks using the algorithmic methods, both centralized and distributed. For comparison purposes, we also use a tree construction method for CDG which relies on using the Steiner method (Steiner-CDG). Figure 5.9 shows the overall number of transmissions for different networks (n range from 100 to 500 nodes) with total projection nodes of $m = n \times 10\%$. We observe that our centralized (NC-CDG/C) and distributed (NC-CDG/D) algorithms almost equally outperform the Steiner-CDG method, and as the number of nodes in the network increases, they start to gradually outperform Steiner-CDG; the figure shows a minimum gain (for smaller network sizes) of 2.43% and a maximum gain of 13.26% over Steiner-CDG. It should be noted here that the Steiner-CDG method is NP-complete since the Steiner problem is itself NP-complete. On the other hand, NC-CDG/C and NC-CDG/D exhibit substantial performance gains over CDG/C and CDG/D respectively with gains ranging from 11.88% to 22.89% for centralized methods and from 16.21% to 27.39% for the distributed methods. The reason being that as the size of the network increases, more forwarding trees are constructed ($m = n \times 10\%$) and therefore more chances for constructing such trees to exploit the network coding opportunities.

Next, we vary the number of projections (m) and study its impact on the performance gains. We consider a network of 400 nodes and the results are depicted in Figure 5.10. The algorithmic methods are compared against each other and against the Steiner-CDG heuristic. Intuitively, the larger the value of m, the more forwarding/aggregation trees are needed to gather the sensed data, and hence the higher is the likelihood to construct such trees to promote

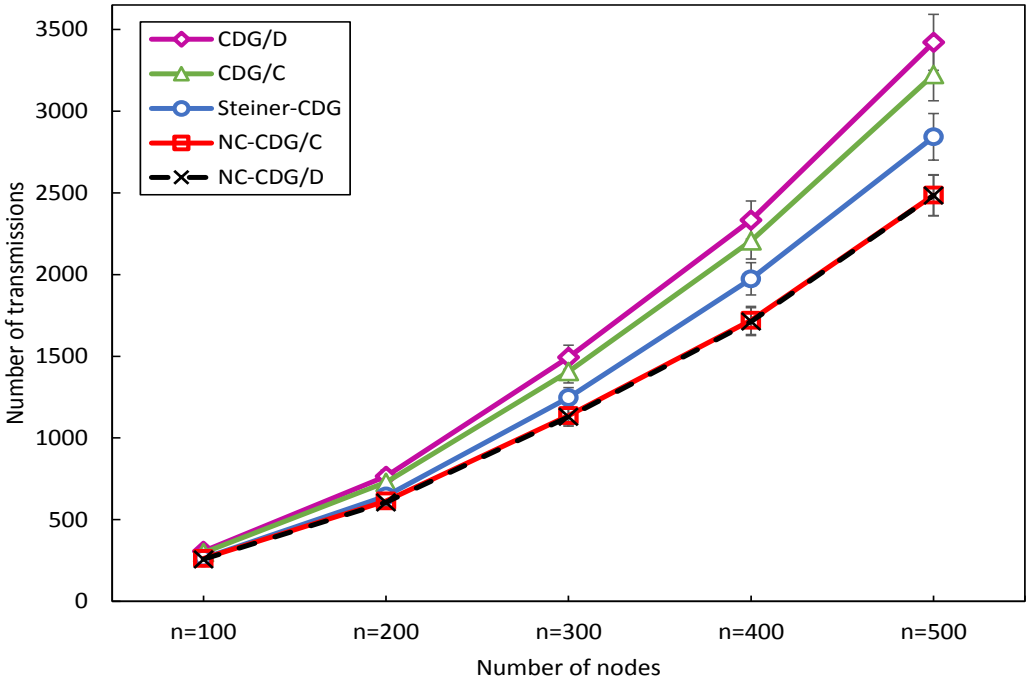


Figure 5.9: NC-CDG: Cost of transmissions Vs. number of nodes ($m = 10\%n$)

or exploit coding opportunities. First, as the number of projections increases (from $m = 20$ to $m = 100$), both NC-CDG/D and NC-CDG/C performs similarly and they both significantly outperform the CDG methods, with NC-CDG/D (NC-CDG/C) showing gains (reduction in the total number of transmissions required) ranging from 23.26% to 28.82% (resp. 16.71% to 24.8%) over CDG/D (resp. CDG/C). Both NC-CDG methods outperform the Steiner-CDG method. Here, it should be noted that the Steiner-CDG method does not exploit the network coding opportunities, but rather construct trees in a more optimal manner (i.e., minimize the number of transmissions per each aggregation tree). Not surprisingly, the NC-CDG methods exhibit a maximum gain of 13.7% and a minimum gain of 8.79% over Steiner-CDG.

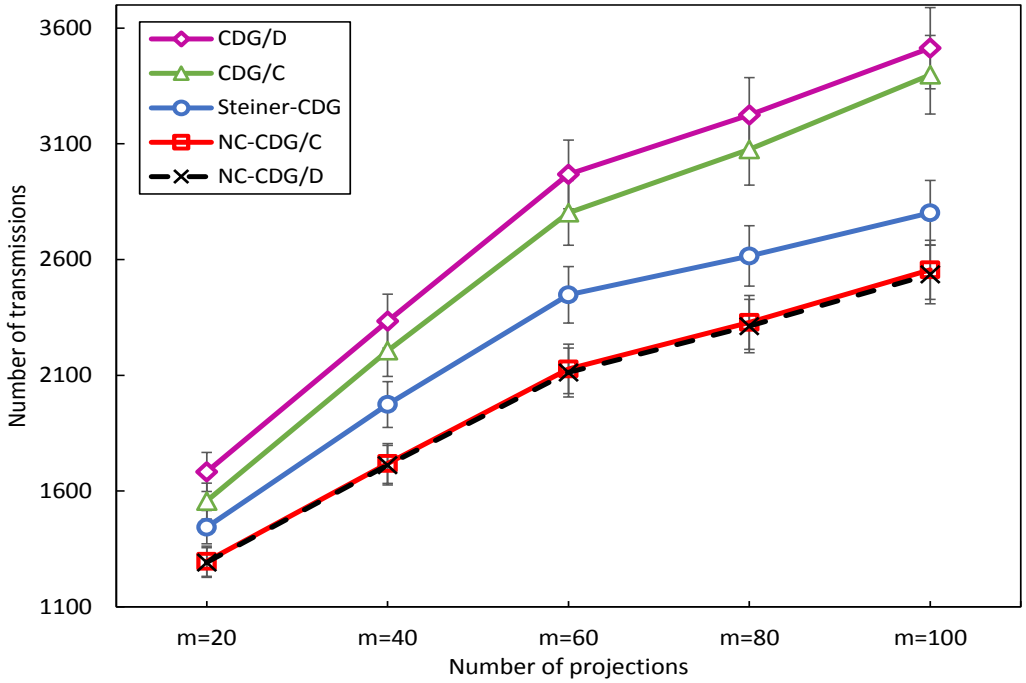


Figure 5.10: NC-CDG: Cost of transmissions Vs. number of projections for n=400 nodes

Transmission load balancing:

In addition to reducing the cost of transmissions, CDG methods attempt to distribute the load of aggregation and forwarding among all sensors in the network. The advantage of this is that all sensors more or less equally consume similar amount of energy which result in extending the lifetime of the network and avoid having nodes deplete their batteries earlier than others. To study this effect, we look at the distribution of number of transmissions (PDF) at all nodes in the network. This distribution is depicted in Figure 5.11 for a network of 300 nodes ($m = n \times 10\%$) and using NC-CDG/D, CDG/D, Steiner CDG and the Non-CDG method. The three methods show much better energy consumption distribution than the Non-CDG method. It is clear that with the NC-CDG method, the average transmission load per node is smaller than other

aggregation methods, followed by Steiner-CDG, CDG and Non-CDG. With the Non-CDG method, the variance of transmission load is very large implying that some nodes may deplete their energy much earlier than others, resulting in shorter network lifetimes. Conversely, the NC-CDG method yields the most balanced transmission load distribution, owing to the capabilities of the method to distribute the load cross all sensors.

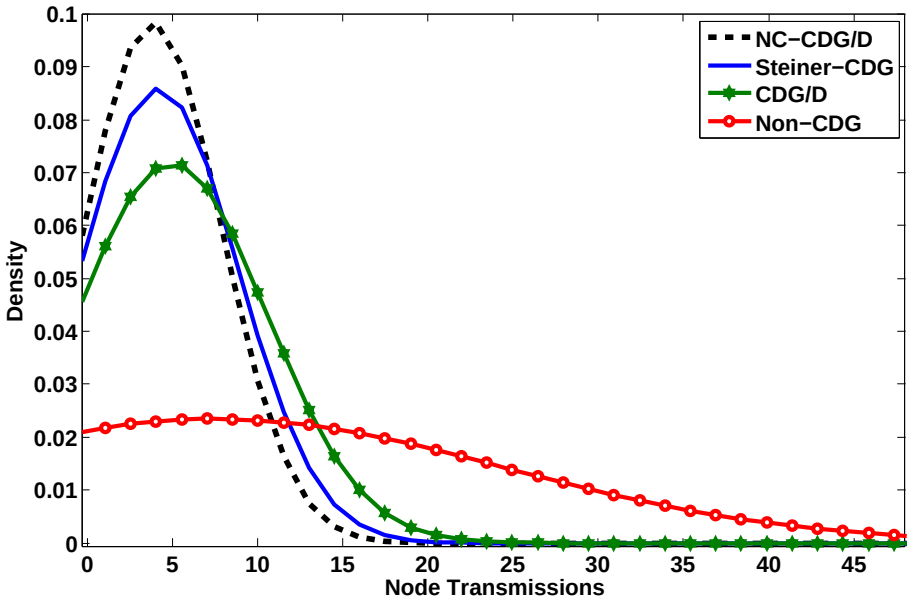


Figure 5.11: Probability Density Function (n=300)

5.6 Conclusion

We studied the problem of compressive data gathering (CDG) in wireless sensor networks. Gathering measurements from the network relies on constructing forwarding/projection trees, each tree corresponds to a projection for collecting a weighted/compressed data. The existence of such trees creates opportunities for many-to-many communication patterns, which in turn gives rise to network coding operations; such operations if exploited will further reduce

the number of transmissions needed to collect the sensory data. We developed a mathematical model for constructing trees which maximally exploits the coding opportunities on compressed data being routed on these forwarding/projection trees. We have shown that when network coding and compressive data gathering techniques are jointly considered (NC-CDG), gains (reduction in number of transmission) of up to 47.67% (resp. 10.71%) can be observed over networks (up to 50 nodes) which do not implement compressive sensing for data gathering (resp. with only CDG). Owing to its complexity and to evaluate NC-CDG over larger networks, we developed both centralized and distributed algorithmic methods for solving the NC-CDG problem. We showed that our algorithmic methods are scalable and accurate, with worst case optimality gap not exceeding 3.96% in the studied scenarios. We also showed that NC-CDG yields performance gains of up to 30% may be attained.

Chapter 6

Forwarding Tree Construction and Scheduling (FTCS)

In this chapter we study the problem of constructing forwarding trees for collecting and aggregating sensed data in the network under the realistic physical interference model. More specifically, we jointly address the problem of tree construction and link scheduling for our problem PCDG proposed in Chapter 3. With PCDG, multiple forwarding trees are constructed, each for aggregating a coded or compressed measurement, and these measurements are collected at the sink for recovering the uncoded transmissions from the sensors.

The problem of gathering tree construction and link scheduling is addressed jointly, through a mathematical formulation, and its complexity is underlined. Our objective is to collect data at the sink with both minimal latency and fewer transmissions. We show the joint problem is NP-hard and owing to its complexity, we present a decentralized method for solving the tree construction and the link scheduling sub-problems. Our link scheduling sub-problem relies on defining an interference neighbourhood for each link and coordinating transmissions among network links to control the interference. We prove the

correctness of our algorithmic method and analyze its performance. Numerical results are presented to compare the performance of the decentralized solution with the joint model as well as prior work from the literature.

6.1 Link Scheduling in Physical Interference Model

We consider a Time Division Multiple Access (TDMA) based MAC access where time is divided into slots of equal length; we define the set of links which can be active concurrently in the same time slot as a ***configuration***. Here, a configuration consists of links/transmissions from multiple forwarding trees which may be active simultaneously, such that no one parent (in one tree) is scheduled for transmission before it receives transmissions from its children. Let d_{ij} be the Euclidean distance between two nodes i and j and let G_{ij} be the channel gain from a transmitter node i to a receiver node j, (e.g., $G_{ij} = d_{ij}^{-\alpha}$, α is the path lost exponent). Now, under the physical interference model [33], in the presence of concurrent transmissions, a receiver j can successfully receive the transmission from node i if the signal to interference plus noise ratio (SINR) at j is above a certain threshold β, which is formulated as:

$$SINR_{(i,j)} = \frac{P\,G_{ij}}{\eta + \sum_{\forall(h,k)\in E: h\neq i} P\,G_{hj}} \geq \beta \qquad \forall (i,j) \in E \tag{6.1}$$

where η is the background noise. In general, we refer to the number of time slots needed to schedule the links in all forwarding trees (to collect all compressed measurements) as a round. The size of a round determines the latency for collecting the measurements. We further assume all packets (each carrying a compressed measurement) are of equal size.

6.2 Problem Description

We are interested in gathering, in each round, measurements at the sink from all the sensors. We assume sensors have finite battery lifetime. We also assume transmissions in the network can interfere with one another and therefore an access scheme should be in place to coordinate the transmissions.

Problem Definition 1 (Forwarding tree construction in PCDG): *Given a connected graph G of n sensor nodes, a sink, and a sparse matrix Φ, the problem of finding tree construction in projection based compressive data gathering (PCDG) consists of finding m forwarding trees, each tree to collect coded measurements from a subset of nodes (nodes with non-zero coefficients in a corresponding row of matrix Φ, where such nodes are referred to as interest nodes) en-route to the sink in the most energy efficient manner.*

Here, each tree t ($1 \leq t \leq m$) corresponds to one projection which gathers one weighted sum z_t from a set of interest nodes at the sink. Our objective is to construct these trees such that the total number of transmissions in the network is minimized. The gathering on each routing tree is performed based on the Compressive Sensing technique.

Problem Definition 2 (Scheduling): *Given a set of forwarding trees, the scheduling problem consists of finding maximal size sets (where a set is a configuration[1] of active links in one time slot) and allocating time slots for them such that the resulting schedule length is minimized. Such problem guarantees the delivery of compressed measurements to the sink with minimal latency.*

Problem Definition 3 (FTCS): *The joint problem of forwarding tree construction and scheduling (FTCS) is the combination of problems 1 and 2.*

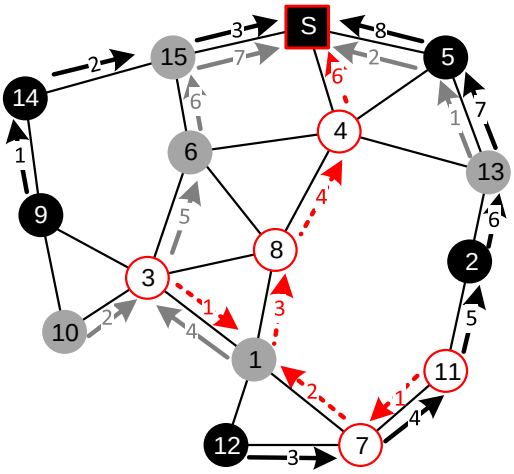
[1] A configuration is formally defined in Section 6.1.

We illustrate the operation of FTCS on the sample network shown in Figure 6.1; namely, we illustrate the interaction between the tree construction and link scheduling and highlight the impact on the data gathering latency (or the schedule length). We compare a joint FTCS method with one that constructs trees and schedule them separately. The results are depicted in Figures 1(a)-1(b). The example shows how to gather data at the sink from all sensors using three projections. As the figures show, both methods require the same number of transmissions (links) to gather the data, however, Figure 1(a) shows that the trees in the joint FTCS can be scheduled in only 8 time slots, whereas, the disjoint method, as Figure 1(b) shows, requires 9 time slots to collect the measurements. This is due to the fact that trees are constructed without considering the requirements for achieving shorter schedule length. Such insights will be exploited as we develop our decentralized method in subsequent sections. Figure 1(c) shows a tree construction using a distributed (algorithmic) method, where the scheduling length of this method depends on the radius of the interference neighbourhood of each link. Our distributed method as well as the interference neighbourhood will be properly introduced and explained in Section 7.3.

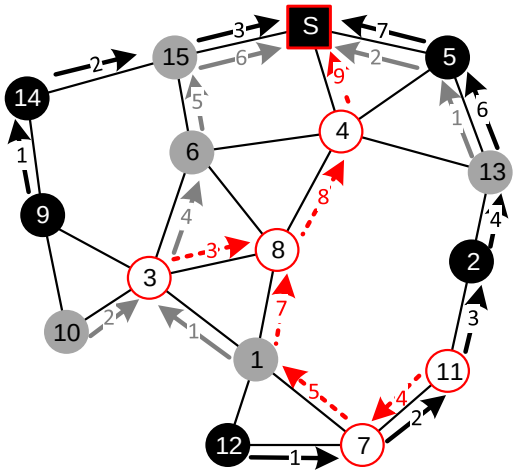
6.3 Problem Formulation

In this section, we formulate FTCS as an optimization problem whose objective is to obtain a set of forwarding trees which can be scheduled to deliver measurements to the sink in the shortest schedule period to achieve a balance between lower latency delivery and energy efficient gathering. We mathematically formulate the problem as a mixed integer linear program (MILP).

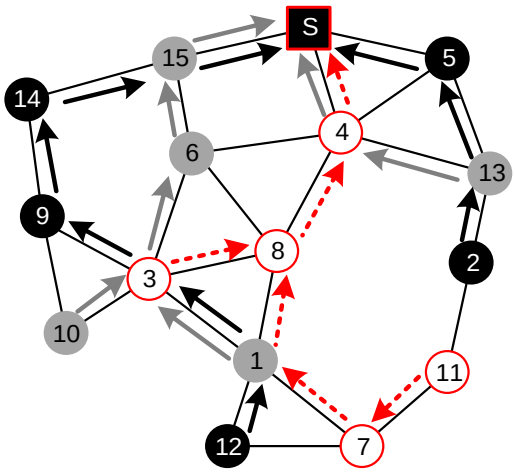
x_{ij}^{t} is a binary variable which indicates whether there is an edge between



(a) Joint tree construction and scheduling



(b) Disjoint tree construction & scheduling



(c) Algorithmic tree construction

Figure 6.1: FTCS, $m = 3$ ($m = 20\%n$). In the network, the black square S is the sink which intends to gather data from all nodes. Same colour arcs represents aggregation tree for one projection. Each set of interest nodes is illustrated with same colour. The numbers on the arcs represent the time slot when the tree links are active.

Table 6.1: Notations Used in problem formulation for FTCS

Parameters		
V	The set of nodes in the network.	
E	The set of edges in the network.	
n	Total number of nodes.	
m	Total number of projections (trees).	
$\|I_t\|$	Total number of interest nodes in set I_t.	
P	Node power transmission.	
G_{ij}	Channel gain from transmitter i to receiver j.	
β	SINR threshold.	
η	Background noise.	
S	The set of a large number of time slots sufficient for one round of data gathering (for all transmissions).	
T	The set of m trees required for compressive data gathering.	
ω	Weight of each term in the objective function. ($0 \leq \omega < 1$)	
Variables		
f_{ij}^t	$\in \mathbb{N}$	The amount of traffic flow (data traffic load) on link (i,j) in tree t.
x_{ij}^t	$\in \{0,1\}$	Indicating whether link (i,j) is in tree t.
$a_{ij}^{t,s}$	$\in \{0,1\}$	Indicating whether link (i,j) in tree t is active in time slot s.
λ_s	$\in \{0,1\}$	Indicating if at least one link is active at time slot s.

nodes i and j in tree t, and let $a_{ij}^{t,s}$ indicates whether link (i,j) in tree t is active (scheduled) during time slot s or not. In addition, we define S to be a large number of time slots which is sufficient to gather data for all trees. We use a binary variable λ_s to assert if a time slot s has at least one active link. That is:

$$\lambda_s = \begin{cases} 1, & \text{if at least one link active at time slot s;} \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

The notations used throughout this section are listed in Table 6.1. The objective of our design is to construct trees which achieve a balance between the number of links needed to gather the measurements (and thus energy expended for data gathering) and the required number of time slots needed to

schedule the constructed trees (i.e., gathering latency):

$$Minimize \qquad \omega \sum_{t \in T} \sum_{(i,j) \in E} x_{ij}^{t} + (1 - \omega) \sum_{s \in S} \lambda_s, \tag{6.3}$$

The first sum corresponds to the total number of links in the constructed trees and the second one depicts the scheduling length. The parameter ω ($0 \leq \omega < 1$) indicates the weight of each term in the objective. Depending on the task, if one of the terms (whether energy efficiency or time efficiency) is more important than the other, we give more weight for that particular term. Otherwise, we assign equal weight to both terms ($\omega = 0.5$). The following are the constraints for our problem:

Traffic Flow conservation constraints:

These constraints assert that the total incoming traffic flow (data traffic load) plus the traffic flow originating at a particular node is equal to the total outgoing traffic flow. Let $f_{ij}^{t} \in \mathbb{N}$ being the data traffic load (number of packets) imposed by certain routing on edge (i, j) or between nodes i and j in tree t. The following constraints, for each tree t, force the set of interest nodes (vector set I_t) which belong to one tree (projection) to have one data flow from each interest node to the sink:

$$\sum_{j:(i,j) \in E} f_{ij}^{t} - \sum_{j:(j,i) \in E} f_{ji}^{t} = \begin{cases} -|I_t|, & i = sink; \\ 1, & \forall i \in I_t; \qquad \forall t \in T \\ 0, & otherwise. \end{cases} \tag{6.4}$$

Tree link creation constraints:

These constraints create forwarding links for a tree. Let $x_{ij}^t \in \{0,1\}$ indicate whether there is a link between nodes i and j in tree t. $x_{ij}^t = 1$, if there is a positive traffic flow from i to j, and zero otherwise. This implies that $f_{ij}^t = 0 \Leftrightarrow x_{ij}^t = 0$ and $f_{ij}^t > 0 \Leftrightarrow x_{ij}^t = 1$ which is achieved by the following inequalities: (note that n (number of nodes) is always greater than any f_{ij}^t)

$$\begin{cases} f_{ij}^t - x_{ij}^t \geq 0 \\ x_{ij}^t - \frac{f_{ij}^t}{n} \geq 0 \end{cases} \qquad \forall (i,j) \in E, \quad t \in T. \tag{6.5}$$

Outgoing link constraints:

These constraints assert that each node can have a maximum of one outgoing transmission (link) in each tree to avoid loops. Otherwise, data is not aggregated to a root (sink).

$$\sum_{j:(i,j)\in E} x_{ij}^t \leq 1 \qquad \forall i \in V, \quad t \in T. \tag{6.6}$$

Half duplex constraints:

The half duplex constraints ensure that a node may not transmit and receive in the same time slot.

$$\sum_{t\in T} a_{ij}^{t,s} + \sum_{t\in T} a_{jk}^{t,s} \leq 1 \qquad \forall (i,j) \in E,\ (j,k) \in E,\ s \in S. \tag{6.7}$$

Transmitting constraints:

These constraints ensure that a transmitter cannot simultaneously transmit to multiple receivers in the same time slot.

$$\sum_{t \in T} \sum_{j:(i,j) \in E} a_{ij}^{t,s} \leq 1 \qquad \forall i \in V, \quad s \in S. \tag{6.8}$$

Receiving constraints:

These constraints ensure that a receiver cannot simultaneously receive from multiple senders in the same time slot.

$$\sum_{t \in T} \sum_{i:(i,j) \in E} a_{ij}^{t,s} \leq 1 \qquad \forall j \in V, \quad s \in S. \tag{6.9}$$

Link scheduling constraints:

These constraints are required to force a link in a tree to be scheduled only once in a time slot.

$$\sum_{s \in S} a_{ij}^{t,s} = x_{ij}^{t} \qquad \forall (i,j) \in E, \quad t \in T. \tag{6.10}$$

Transmission order constraints:

These constraints are required to ensure that a node in a tree cannot transmit unless it receives all packets from its children. That is, a link (i,k) in a tree t at time slot s can be scheduled, if all links to its children have been scheduled prior to time slot s (i.e., in time slots between 1 and $s-1$). In other words, $a_{ik}^{t,s} = 1$, if $\sum_{\bar{s}=1}^{s-1} \sum_{j:(j,i) \in E} a_{ji}^{t,\bar{s}} \geq \sum_{j:(j,i) \in E} x_{ji}^{t}$.

In LP format, the above condition is written as follows:

$$\sum_{\bar{s}=1}^{s-1} \sum_{j:(j,i)\in E} a_{ji}^{t,\bar{s}} + B(1 - a_{ik}^{t,s}) \geq \sum_{j:(j,i)\in E} x_{ji}^{t} \qquad \forall (i,k) \in E, \quad s \in S, \quad t \in T. \tag{6.11}$$

B is a big constant, which is bigger than the total number of links in any combination of m trees. When $a_{ik}^{t,s} = 0$, inequality (6.11) is always satisfied. But, when $a_{ik}^{t,s} = 1$, (6.11) reduces to $\sum_{\bar{s}=1}^{s-1} \sum_{j:(j,i)\in E} a_{ji}^{t,\bar{s}} \geq \sum_{j:(j,i)\in E} x_{ji}^{t}$ which implies that the summation of all links coming to node i had to be activated at time slots between 1 and $s-1$, otherwise, node i can not transmit (or, link (i,k) can not be active, i.e., $a_{ik}^{t,s} \neq 1$) at the current time slot s.

SINR constraints:

The following constraints make sure that the SINR for each active link is above the thresholdβ.

$$P\,G_{ij} + B_{ij}^{t,s}(1 - a_{ij}^{t,s}) \geq \beta(\eta + \sum_{\bar{t}\in T} \sum_{(k,h)\in E; k\neq i} P\,G_{kj}\, a_{kh}^{\bar{t},s}) \qquad \forall (i,j) \in E, s \in S, t \in T. \tag{6.12}$$

where $B_{ij}^{t,s}$ is a constant and satisfies the following:
$B_{ij}^{t,s} \geq \eta + \sum_{t\in T} \sum_{(k,h)\in E; k\neq i} P\,G_{kj}\, a_{kh}^{t,s}$

In (6.12), if link (i,j) in tree t is active in time slot s (i.e., $a_{ij}^{t,s} = 1$), then (6.12) reduces to expression (6.1).

Finding occupied time slots in a schedule:

The following constraints check whether a time slot s has at least one active link or not.

$$\lambda_s \geq a_{ij}^{t,s} \qquad \forall s \in S\ , \quad t \in T\ , \quad (i,j) \in E. \tag{6.13}$$

Note, after solving the above problem, the time slots which have no active links are removed from the schedule and the remaining time slots form the corresponding scheduling solution.

6.3.1 NP-hardness

The authors of [29] have shown that the data gathering in WSN under SINR is NP-hard through a reduction from the *max-connections* problem [3]. The max connection problem is to select a maximal set or configuration size under the physical interference model. Our problem however is different from [29] in that we construct multiple forwarding trees (rather than only one) to collect the coded measurements; we also differ in that a node waits for its children's measurements to compress them (with its own) into one packet for upward transmission. This makes the scheduling problem more difficult. Here, we try to show that the problem of forwarding tree construction and scheduling (FTCS) is very difficult to solve. Below is our informal methodology for highlighting this difficulty.

The FTCS problem has two combined objective terms (constructing m aggregation trees with minimum links, and scheduling these links based on SINR constraint in a shortest time length). Now, according to the weight given

to each term, the problem gives different results. Without loss of generality, let us first assume the trees are given. We may show that the minimum link scheduling problem is NP-hard by reducing from the One-Shot Scheduling problem which has been shown to be NP-hard in [30]. The One-Shot Scheduling problem is to pick a subset of weighted links such that the total weight is maximized and the SINR at the receiver of each link is above the threshold β. In other words, attempting to use one slot to its full capacity. It should be noted that in our problem links have equal weights. Therefore, we give a weight of one to each link and the problem of one-shot scheduling becomes of picking a maximum number of links in one slot that satisfies the SINR constraint.

The problem of finding the minimum scheduling length among all m data aggregation trees can be decomposed into a series of one-shot scheduling subproblems. In each one-shot scheduling subproblem, an auxiliary graph is constructed (in polynomial time) from a set of links in m aggregation trees that do not have child links for data aggregation. In other words, an edge is added to the auxiliary graph if the corresponding link on any aggregation tree is connected to a leaf node. After resolving the one-shot scheduling subproblem on the auxiliary graph, the scheduled links are removed from the aggregation trees. This step is repeated until no links remain in any tree. Then, the number of iterations is the total number of time slots required for trees scheduling. Therefore, scheduling the problem of finding the minimum scheduling length is NP-hard.

On the other hand, if we give the highest weight to minimizing the total links in constructing the m aggregation trees, we may show the problem of constructing each aggregation tree is NP-hard by reducing from the minimum

Steiner tree problem which is known to be NP-hard problem [39]. The minimum Steiner tree problem is to connect a set of interest nodes $I \subseteq V$ such that the connected spanning tree has a minimum total distance on its edges. Now, from the minimum Steiner tree problem, if we let one of the nodes in the tree act as a root, the minimum Steiner tree is converted to one tree construction of our problem. Selecting a root (which is the sink node) can clearly be done in polynomial time. We require m such trees for our compressive data gathering. Therefore, the tree construction is also NP-hard.

6.4 Algorithmic solution

To overcome the computational complexity of the FTCS problem, we decompose it into two subproblems, namely the forwarding tree construction and the link scheduling subproblems and present decentralized methods for solving them.

6.4.1 Distributed Tree Construction

Our objective is to construct forwarding trees in a decentralized manner. Each forwarding tree will carry a compressed measurement from the network to the sink; our objective is to obtain energy efficient trees which deliver data to the sink with minimal latency.

The compressive data gathering tree construction consists of three phases: 1) disseminating discovery messages; 2) route discovery; 3) search for more efficient routes, to leverage them in the scheduling subproblem. Initially (**Phase 1**), the sink starts by sending a discovery message to its neighbours. Each node, upon receiving the message, will broadcast it to allow other nodes, not

close to the sink, to receive the discovery message. This procedure is similar to traversing the network using a breadth-first search (BFS) algorithm [15]. Hence, each node v will learn its shortest path (P_{vs}) to the sink as well as the hop-count along the path. Further, node v discovers its neighbour set $N(v)$. Node v, upon checking matrix Φ, which is stored in its memory, determines whether node $u \in N(v)$ ($\forall u$) belongs to the set of interest nodes (I_t) of tree t or not. The time complexity for **Phase 1** (similar to BFS) is $O(n)$.

In **Phase 2**, each node v for each tree t (if it is an interest node), after running Algorithm 6.1, decides its parent on the uplink path to the sink. For each interest node, we assign an attribute to designate its parent interest node (π_v^t) (note, a parent interest node could be a neighbour of v or can be reached through other relay nodes) and a decision flag ($flag_v^t$) to indicate whether the parent interest node of v is fixed. Lines (1-3) show that every interest node which is a neighbour of the root selects the root as its parent node and sets and distributes its decision flag $flag_v^t = 1$. Now, if interest node v (Lines (4-6)) is not a neighbour of the root, but has an interest node neighbour b with $flag_b^t = 1$, then v selects b as its parent interest node and commits its decision ($flag_v^t = 1$). In the case where none of the neighbouring interest nodes (b) of v has its decision flag set (i.e., $flag_b^t = 0$), v will select the neighbouring interest node with the smaller hop-count to the sink as its parent interest node (Lines 7-9). Now, when only interest node neighbours with equal hop-count to the sink as v can be found (Lines 10-12), v selects the one (b) whose successive parents reach an interest node with smaller hop-count or decision flag $flag = 1$ or no parent node, and does not reach node v (to avoid loops). If none of the above conditions is satisfied, v runs a BFS to explore its neighbourhood of radius $h_v^t - 1$ in search for an interest node b with smaller hop-count to the sink; otherwise,

it searches for an interest node whose decision flag $flag_b^t = 1$ (Lines 14-17). Node v avoids selecting interest nodes b whose $\pi_b^t = v$ to avoid loops. Finally, if no interest node is found, v connects itself directly through a shortest path to the sink (this path is known from the discovery phase). Node v will repeat the route discovery (Algorithm 6.1) if it receives a notification message from its neighbours indicating that there is a change in the network, e.g., change in a decision flag, or following a node or link failure due to mobility or channel impairments occurring in the network triggering route maintenance. The time complexity for **Phase 2** is $O(1)$ in the best case (when a node chooses a neighbour node) and $O(\gamma)$ in the worst case (when node does not have a neighbour interest node), where γ is the number of nodes around node v and within a radius $h_v^t - 1$. Note that nodes in the distributed approach simultaneously execute the algorithm.

Tree Construction Refinement:

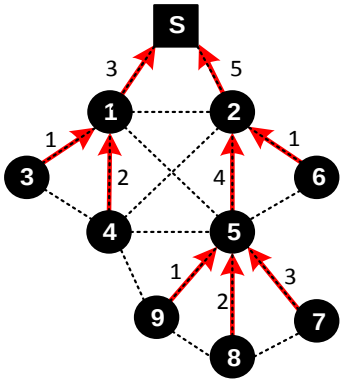
We motivate our refinement phase through an illustrative example shown in Figures 2(a)-2(b). The intuition for refining the tree selection is that the forwarding trees should have fewer links for energy efficiency and should be scheduled in a shorter time period for latency efficiency. For instance, node 9 may select either node 4 or node 5 as its parent node. Either selection will result in a forwarding tree with same number of links, however, the trees corresponding to the two selections will differ in their data collection latency, obtained from the scheduling subproblem (going through node 5 requires a total of 5 times slots, and through node 4 only 4 time slots). Clearly, if a parent in a tree has a higher node degree, with multiple incoming transmissions, then those transmissions will be scheduled sequentially, and therefore this should be avoided. Clearly, this suggests a thinner but a larger tree height. The

Algorithm 6.1 Route discovery at node v (**Phase 2**) for FTCS

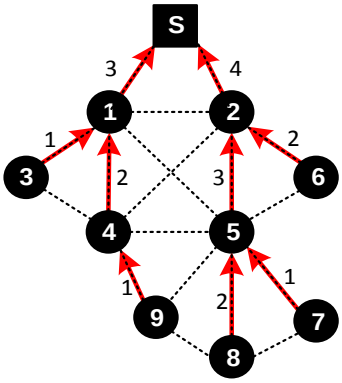
1: **if** $root \in N(v)$ **then**
2: $\pi_v^t = s$;
3: Set and broadcast $flag_v^t = 1$;
4: **else if** $b \in N(v)$ AND $b \in I_t$ AND $flag_b^t = 1$ **then**
5: $\pi_v^t = b$;
6: Set and broadcast $flag_v^t = 1$;
7: **else if** $b \in N(v)$ AND $b \in I_t$ AND $h_b^t < h_v^t$ **then**
8: $\pi_v^t = b$;
9: Set $flag_v^t = 0$;
10: **else if** $b \in N(v)$ AND $b \in I_t$ AND $h_b^t = h_v^t$ AND successive parents of b reach a node with smaller hop-count or $flag = 1$ or non-parent and do not reach v **then**
11: $\pi_v^t = b$;
12: Set $flag_v^t = 0$;
13: **else**
14: Run BFS from v in a radius equals to $h_v^t - 1$.
15: **if** interest node(s) in this radius found **then**
16: Connect v to nearest interest node through shortest path.
17: Set $flag_v^t = 0$;
18: **else**
19: Connect node v through shortest path to the root.
20: **end if**
21: **end if**

larger tree height however may in turn suggest longer schedule period; this is because a parent node along a path towards the sink will have to wait until all downstream measurements are collected before it forwards its own measurement. Recall, measurements have to be compressed, to reduce the number of transmissions in the network. This is depicted in Figures 3(a)-3(b), where selecting a subtree with larger height increases the scheduling period, and thus collection latency. Motivated by these observations, our tree construction should be refined to yield more efficient forwarding trees, and this is elaborated in **Phase 3**.

In **Phase 3**, each node v checks whether it is among the interest nodes in set I_t. If yes, node v runs Algorithm 6.2 searching for a more efficient route

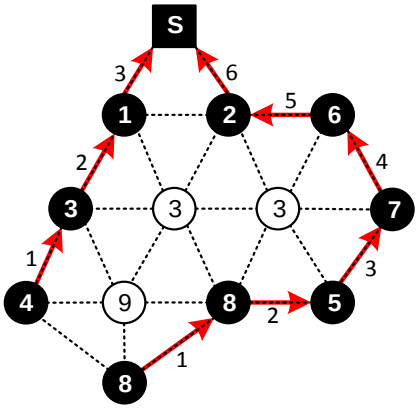


(a) Number of time slots=5

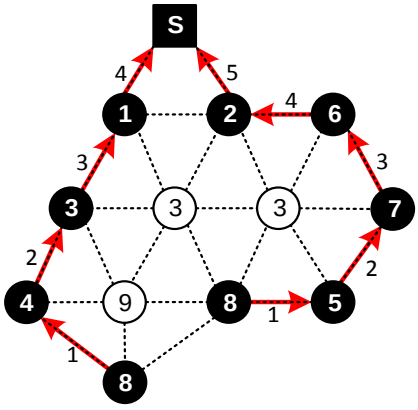


(b) Number of time slots=4

Figure 6.2: An example of balancing the node degree in a tree. In the network, black nodes are interest nodes. The directed arcs denote the links on the data aggregation tree. The active time slot for each arc is shown next to it.



(a) Number of time slots=6

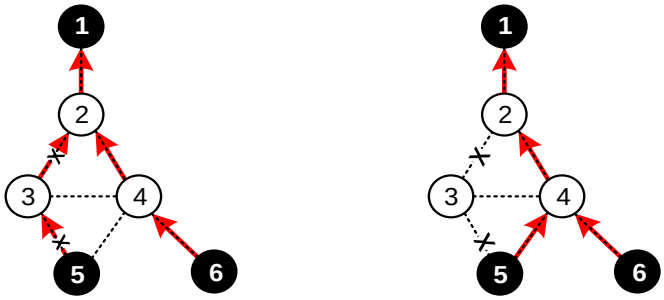


(b) Number of time slots=5

Figure 6.3: An example of minimizing the height of a subtree. Black nodes are interest nodes, while white nodes are relay nodes. The directed arcs denote the links on the tree. The active time slot for each arc is shown next to it.

or a parent that potentially can reduce the scheduling length as discussed above. Algorithm 6.2 removes all the successive tree links from node v to a node that is either an interest node or has more than one child (an example is illustrated in Figure 4(a); for node 5, the path shown by arrows with 'x' from node 5 to node 2 is removed (note that node 2 has two children)). Let b represent the node that has more than one child. Let R be the total number of removed links. In this step, an interest node v and its descendants are disconnected from the main tree t (e.g., node 5 in Figure 4(a) which has been disconnected from the tree). Next, to discover an alternative path to connect v to the main tree t, v searches in a radius equals to R, using Breath-First-Search (BFS), for a node(s) in tree t (if any) that can improve the schedule length and reduce the number of transmissions (e.g., in Figure 4(b), node 5, which has been disconnected, can connect to the tree through node 4; hence, the overall number of transmissions decreases from 5 to 4). To find a better path, the algorithm adds the nearest candidate nodes found on tree t in a radius R into a $Candidates$ list. Furthermore, for each node g in $Candidates$ list, it retrieves the nodal degree D_g and its hop-distance to the sink H_g. This information can be obtained from each node where they have been obtained from **Phase 2**. As discussed earlier, the candidate that minimizes the nodal degree and the height of the subtree will be selected as the new parent (refer to lines 8-13 in Algorithm 6.2).

Let δ to be the number of nodes within a radius R, it takes $O(\delta)$ to traverse all nodes in radius R using BFS algorithm. Finding best candidate among nodes in the $Candidates$ list takes $O(\rho)$, where ρ is the size of the $Candidates$ list. Therefore, Algorithm 6.2 takes $O(\delta + \rho)$, where δ is bigger than ρ, since ρ is a subset of δ. Thus, the time complexity for the algorithm is $O(\delta)$.



(a) Before **Phase3** (b) After **Phase3**

Figure 6.4: An example of removing successive links in a tree. The arcs with X sign denote the removed tree links.

Algorithm 6.2 Tree refinement at node v (**Phase 3**) for FTCS

1: Remove all the successive tree links from node v to first interest node or node that has more than one child in tree t. Let b be the found node.
2: $R \leftarrow$ Total number of removed links.
3: $Best_{candidate} \leftarrow b$.
4: Run BFS algorithm from v in a radius equals to R.
5: **if** node(s) other than b on disconnected main tree t is found in this radius **then**
6: $Candidates \leftarrow$ put the nearest candidate nodes into the list.
7: $Best_{weight} =$ Infinity.
8: **for** each node g in the $Candidates$ list **do**
9: $H_g \leftarrow$ Hop-count from g to the sink.
10: $D_g \leftarrow$ Degree of node g.
11: **if** $(D_g + H_g) < Best_{weight}$ **then**
12: $Best_{weight} = D_g + H_g$.
13: $Best_{candidate} \leftarrow g$.
14: **end if**
15: **end for**
16: **end if**
17: Connect node v to $Best_{candidate}$ in shortest-path.

6.4.2 Distributed Link Scheduling Algorithm

We consider a Time Division Multiple Access (TDMA)-based access method, and assume time is divided into slots of equal length; we assume each time slot is divided into a scheduling period and a transmission period. A schedule is constructed during the scheduling period where a configuration of links (a configuration is defined earlier) which may be scheduled concurrently is determined. During the transmission period, links in the selected configuration transmit their packets, one packet each, containing their compressed measurements. In this section, we present our decentralized scheduling algorithm, where the objective is for each link to locally schedule its transmission while not violating 1) the order of transmissions and 2) the interference constraints for transmissions to be successful. To achieve this objective, we define for each link an interference neighbourhood, which is centered around the receiver of the link. We shall determine (and control) the cumulative interference caused by active sensors falling in the interference neighbourhood of a link. Further, all links whose transmitters are inside the interference neighbourhood of a link l will be able to exchange information (therefore coordinate) with the transmitter of l for scheduling purposes.

For each link l of length d_l (e.g., a transmission between a transmitter\child i and a receiver\parent j), an interference neighbourhood with a radius $K_l \times d_l$ around the receiver of link l, and using the interference localization method presented in [51], is constructed. The neighbourhood for each link is constructed such that interference beyond this neighbourhood only has negligible impacts on its received signal [51]. For a transmission to be successful on a link l, the maximum interference that can be tolerated at the receiver of link l

is:

$$I_l^{max} \triangleq \frac{P d_l^{-\alpha}}{\beta} \tag{6.14}$$

where P is the transmit power, α is a power loss exponent and β is a predetermined SINR threshold required for an acceptable bit error rate. The authors of [51] showed that given a constant ϵ, where $0 < \epsilon < 1$, for a link l to be feasible, the upper bound on the interference coming from the transmitters of active links located outside the interference neighbourhood of link l should not exceed ϵI_l^{max} and the total interference coming from transmissions inside the interference neighbourhood cannot exceed $(1-\epsilon)I_l^{max}$. The radius of the interference neighbourhood ($K_l \times d_l$) certainly depends on the value of ϵ. The smaller the value of ϵ, the larger the interference neighbourhood, and thus the higher the scheduling overhead. The value of ϵ can be used to control the scheduling overhead. In addition, the receiver of each link can estimate the interference power created by the transmitter of each link in the interference neighbourhood using the Radio Interference Detection (RID[2].) [90]. For more details about the interference localization and RID methods, we refer the reader to [51] and [90] respectively. It should be noted that other approaches (e.g., FlashLinQ [77] and ITLinQ [63]) have been shown to perform very well in terms of interference management and can be used for our link scheduling subproblem.

We now propose our distributed scheduling algorithm. Let Δ_l be the set of

[2]The RID protocol is only used to let the receiver estimates the interference caused by any transmitter. The basic idea of RID is that a transmitter broadcasts a High Power Detection (HD) packet, and immediately follows it with a Normal Power Detection (ND) packet. The HD packet contains the transmitters ID, from which the receiver knows from which transmitter the following ND packet comes. The receiver estimates possible interference caused by the transmitter by sensing the power level of the transmitters ND packet. For more details we refer the reader to [90]

Algorithm 6.3 Distributed Scheduling Algorithm at link l for FTCS

1: Transmitter of link l broadcasts $SchReq$ to all links in Δ_l.
2: Receiver of Links $k \in L \bigcap \Delta_l$ calculate the interference I_k^{tem} after adding link l temporary to L.
3: **if** any receiver of link k has $I_k^{tem} > (1-\epsilon)I_k^{max}$ **then**
4: Link k sends an $NotAcc$ message to link l.
5: **end if**
6: **if** link l receives at least one $NotAcc$ message **then**
7: Link l does not add itself to schedule L.
8: Link l broadcasts $RemSch$ message.
9: All links k upon receiving $RemSch$ message remove link l from current schedule L.
10: **else if** Link l receives no $NotAcc$ messages **then**
11: Link l is added to the current schedule L.
12: Link l broadcasts $AccSch$ message.
13: All links k upon receiving $AccSch$ message update their schedule L by adding link l to L.
14: **end if**

all links k such that the transmitter of link l is in their interference neighbourhood. Let L be the set of links for the current schedule; at the beginning of each time slot, L is empty. At a high level, links to leaf nodes or links whose children do not have data to transmit will go into a ready state (since they do not have to wait for any downstream data); transmitters of such links broadcast their priority information to all nodes in their interference neighbourhood Δ_l. The priority of each node is estimated based on two criteria: (1) its parent nodal degree and (2) its hop-count to the root (this information is obtained from the tree construction phase). For instance, the priority of a node can be quantified by combining (1) and (2). A node with bigger parent nodal degree and larger hop-count to the sink assigns itself a higher priority. The tie can be broken by the transmitter's node ID (node with bigger ID has higher priority). The priority information of a link l is disseminated to all links (transmitters) within the interference neighbourhood of l. Now, each link l in ready state which has

the highest priority among all links (in ready state) in its interference neighbourhood, if its cumulative interference I_l is not exceeding $(1-\epsilon)I_l^{max}$ and its receiver has not already been scheduled for any other link, can simultaneously run Algorithm 6.3 to add itself to the current schedule L. This process continues until no more ready state links can be added to the current schedule L. For the next time slot, new links will be added to the ready state if their predecessor links have been scheduled in the previous time slots. Accordingly, the above procedure will be repeated until no more links are left unscheduled.

6.5 Performance Analysis

In this section, we prove the correctness of our algorithmic method and analyze its efficiency by giving the approximation ratio of the algorithmic tree construction to the optimal one and analyze the performance bounds of the link scheduling algorithm with respect to the aggregation latency.

6.5.1 Correctness

Our distributed method, as discussed above, consists of two (tree construction and link scheduling) parts, where the former part has three phases. We prove the correctness of each part or/and phase using the following theorems.

Theorem 6.5.1. *(Correctness of phase 1). The sink disseminates the discovery message, and all the nodes in the network receive it and hence update their information.*

Proof. Sensor nodes in the network are connected and thus there is at least one path from each node to the sink. If nodes upon receiving the discovery message, broadcast it, this guarantees that all the nodes will receive this discovery

message, and by updating the hop-counter, nodes learn their distances to the sink, as well as the number of neighbors, since they receive one message from each neighbor. It should be noted that nodes do not re-broadcast the discovery message with equal or higher hop-count, and this proves the termination of the discovery message phase. □

Theorem 6.5.2. *(Correctness of Phase 2). In Algorithm 6.1, each interest node v that belongs to tree t finds its route to nearest interest node parent.*

Proof. In Algorithm 6.1, a node has to choose an interest node parent with smaller hop-count (nearer to the sink) or a parent with a decision flag equals one (i.e.; $flag^t_{parent} = 1$). When the node chooses a parent with $flag^t_{parent} = 1$, this guarantees that the route will reach the sink (a node can set its decision flag equal one if its ancestors reach the sink); otherwise, the node will select the parent with smaller hop-count to the sink. The selected parent will repeat the same procedure until the route to the sink is discovered. □

Theorem 6.5.3. *(Correctness of Phase 3). A node in Algorithm 6.2 can enhance the forwarding tree by finding a more efficient route, if any.*

Proof. After removing the successive tree links from node v and disconnecting it from the forwarding tree t, Algorithm 6.2 examines all the paths to nearest node(s) in tree t and finally chooses the best one and connects node v to the tree. Hence, reconnecting the disconnected node to the tree ensures the termination of the algorithm. □

Theorem 6.5.4. *(Correctness of Link scheduling). The distributed link scheduling algorithm in Section 6.4.2 can correctly schedule the links in all the m trees under the physical interference model.*

Proof. Algorithm 6.3 guarantees that each link in the ready state which has the highest priority among others and its cumulative interference does not exceed the maximum interference which can be tolerated, will be added to current scheduling list and removed from the ready state, and hence will be scheduled once. At the end of each round, the ready state will be updated and links that have not been assigned a time slot remain for future rounds. Finally, all the links will be added to the schedule list and the algorithm terminates. □

6.5.2 Performance bounds of the link scheduling algorithm

In this section, we discuss the theoretical lower and upper bounds on the latency for data aggregation on the constructed m forwarding trees.

Theorem 6.5.5. *Given a set of m trees T for compressive data gathering, the lower bound on the required time slots to schedule all the links in T is*

$$max(m, D_i^t + H_i^t)(\forall i \in V, t \in T) \tag{6.15}$$

D_i^t and H_i^t are respectively the nodal degree and hop-count to the sink for node i and tree t.

Proof. In any tree, a parent node (doing data aggregation) has to wait until it receives data from all of its children and then forward the aggregated data to its upper node (if it is not a sink node). Therefore, the minimum number of time slots required for a node i in tree t is D_i^t (i.e., the number of neighbors of node i in tree t). This D_i^t includes the time slot to transmit data from node i to its parent, since its parent has been counted as one of its neighbors in D_i^t.

Now, the minimum time required to send data from a node to the sink is

equal to the hop-distance of a node to the sink, which is represented by H_i^t. Therefore, in total a minimum of $D_i^t + H_i^t$ time slots is needed for node i in tree t to send its aggregated data to the sink. If all the transmissions occur in a way that the SINR constraint is satisfied at each receiver, then the possible lower bound on the required number of time slots is $max(D_i^t + H_i^t)(\forall i \in V, t \in T)$. We should note here that the sink can receive only one transmission in each time slot, hence for m trees a minimum of m time slots is required for the sink to receive all the aggregated data from m trees. Therefore, the final lower bound on time slot for CDG is (6.15). □

It should be noted that at each time slot, the maximum number of links from m trees which are at the ready state and their SINR is below the threshold β, are going to be scheduled and removed from the trees to let the remaining links to be scheduled in the next rounds. It is possible that in the worst case (because of lack of fulfilling the SINR constraint, common node transmission or receiver among ready state links), no more than one link could be scheduled at each time slot. Intuitively, at least one link can be scheduled at each time slot and thus, the worst case performance of the link scheduling algorithm for m trees under the physical interference model is bounded by the total number of links in all m trees.

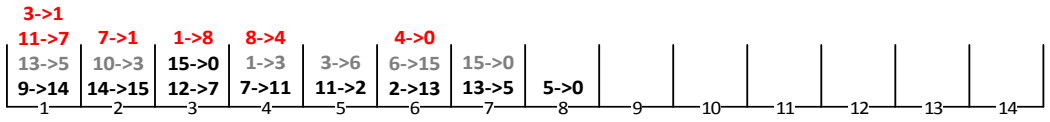
6.6 Performance Evaluation

We study the performance of the joint design method under optimal formulation and compare it with the decentralized solution we proposed. We also study the performance of FTCS under optimal tree construction and optimal scheduling, separately. Finally, we compare the performance of our FTCS with

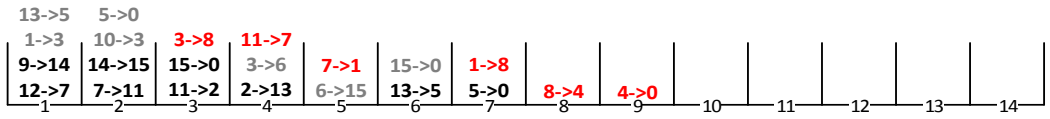
LLHC-MWF [29], which does data gathering but not compressive data gathering. Our metrics for comparisons are the number of transmissions and schedule length required to gather data under various network sizes, topologies, and number of projections (for compressive data gathering). For numerical results, we generate arbitrary networks with n nodes where nodes are randomly distributed over a region of 700×700 unit distance, such that the resulting graph is connected. The density (average nodal degree) of the network is tuned by increasing or decreasing the communication range of a node. Further, we randomly assign each node in the network to m sets of interest-nodes where each set contains $\lceil \frac{n}{m} \rceil$ nodes. Note that based on the number of n nodes and m sets, a node might be included in more than one set. We assume all nodes use the same normalized transmit power $P = 1$. Moreover, we assume a path loss exponent $\alpha = 3$ and the SINR threshold for successful transmission $\beta = 2$; we assume the background noise is negligible; we also assume a single transmit rate and hence only one threshold β. We further assume $\omega = 0.5$, giving equal weights to both terms in the objective (i.e., energy efficiency and time efficiency are equally important). We use CPLEX to solve our optimization model and JAVA to simulate the operation of our distributed algorithms. We run our program on CPU with Intel Core i7 processor, 3.6 GHz speed, 8 GB memory ram and 64-bit windows operating system.

Evaluation on a small network

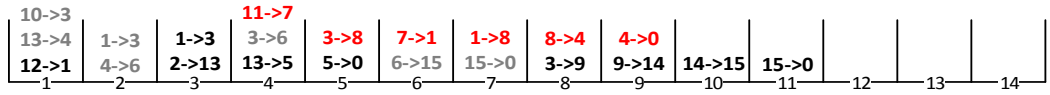
We start by examining the results obtained by solving the FTCS jointly using the MILP model and compare it with our decentralized solution, using the 15-node network shown in Figure 1(a). Clearly, both methods construct forwarding trees with same number of links (and thus same number of transmissions



(a) FTCS joint

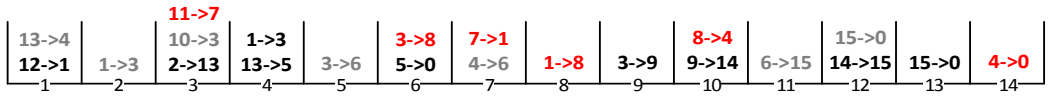


(b) FTCS disjoint



(c) FTCS Decentralized, $\epsilon = 0.3$

(d) FTCS Decentralized, $\epsilon = 0.6$



(e) FTCS Decentralized, $\epsilon = 0.9$

Figure 6.5: Link scheduling solution

to gather the sensory data), however both methods differ in their link scheduling performance as depicted in Figure 6.5. First, the MILP solution yields an optimal link scheduling (schedule length = 8 time slots), however its solution is centralized. The decentralized solution (see Figure 1(c)) varies according to the value of ϵ; a smaller ϵ indicates a larger interference neighbourhood and thus larger area to coordinate transmissions and as a result obtain better solutions than larger values of ϵ. However, this better performance comes at the expense of larger scheduling overhead [51]. When $\epsilon = 0.3$, a schedule length of 11 time slots is obtained which is around 27% far from the optimal solution. The MILP however, being a centralized method, exhibits a much higher computational complexity.

Centralized Vs. Distributed

Now, we compare the performance of the decentralized solution of FTCS (D-FTCS) with two other disjoint methods, namely, methods that solve the problems of tree construction and scheduling separately using either the centralized optimal model or distributed algorithmic method. The first one is centralized and solves the two subproblems optimally (OTC-OLS) and the second one solves only the (centralized) scheduling subproblem optimally (DTC-OLS). It should be noted that the link scheduling under interference model is an NP-hard problem, as shown before. The results, averaged over five runs, are shown in Table 6.2. We should recall that the schedule length highlights the gathering latency in the network. It is observed that OTC-OLS and DTC-OLS, being able to provide optimal solutions to the link scheduling subproblem, resulted in shortest schedule length and thus faster collection time for the measurements. It is interesting to note that DTC-OLS for larger network instances resulted in slightly shorter schedules and this is due to the tree construction refinement phase of the distributed algorithm. In OTC-OLS, however, trees are constructed to contain minimum number of edges without any refinement. D-FTCS on the other hand achieved notably a good performance with worst case gap to other solutions not exceeding 27%. In terms of computation complexity, our decentralized algorithm obtained solutions in less than 2 seconds (for a 40 nodes network) whereas OTC-OLS obtained a solution for a 40 nodes network after a day; namely, the tree construction took only few seconds and the link scheduling took 1.5 days. For smaller networks (e.g., 20 nodes), the decentralized method returned the solution in less than one second and OTC-OLS took in the order of minutes. These results confirm that the link scheduling under interference constraints is indeed very complex to solve in a centralized

setting. Finally, we should note that all three methods constructed trees with same number of edges, consuming the same number of transmissions. We will further examine and compare the performance of D-FTCS in terms of schedule length and number of transmissions with other method in the literature at the end of this section.

Table 6.2: FTCS performance (number of time slots, $m = 20\%n$)

#Nodes	Avg. nod. deg.	#Trans.	D-FTCS	DTC-OLS	OTC-OLS
n=10	2.9	10.8	**6**	**5.6**	**5.6**
n=15	3.12	20.6	**9.6**	**9.6**	**9.4**
n=20	3.14	31.4	**14.6**	**12**	**11.4**
n=25	3.46	44.8	**19**	**15.4**	**15.4**
n=30	3.55	63.4	**22.4**	**17.8**	**18.4**
n=35	3.33	82.2	**30.6**	**25.2**	**26**
n=40	5.22	76.6	**29.4**	**21.4**	**22.4**

Exploring more forwarding trees

At this stage, it should be clear that the scheduling performance depends entirely on the structure of the forwarding trees. In a general graph, more than one forwarding tree with minimum edges may be constructed to gather data from a set of interest nodes to the sink. Therefore, to obtain a more efficient (shorter) schedule, one may first construct all the possible minimum forwarding trees and then solve the scheduling subproblem for all tree combinations, each for a multi-set of interest nodes (or projection), and then choose a combination that gives the best schedule length among others. Let $I_1, I_2, ..., I_m$ represent interest nodes sets for projections $1, 2, ..., m$ respectively. For each set (I_t), we may have different minimum trees (i.e,; $\tau_t = \{t^1, t^2, ...\}$). To find the best scheduling, we have to solve for $\tau_1 \times \tau_2 \times ... \times \tau_m$ combination of trees. Algorithm 6.4 shows the steps to find all the minimum forwarding trees.

Algorithm 6.4 Steps to get all minimum forwarding trees in FTCS

1 Construct the optimal forwarding tree. (e.g., (6.3)-(6.6) without the second sumation of (6.3).
- **1.1** Add the tree into $MinimumTreesSet$.
- **1.2** Let $NLinks$ = number of links in the tree.

2 Remove links one by one from the optimal tree.
- **2.1** Construct tree without that removed link.
- **2.2** If the number of links in the obtained tree is equal to $NLink$, and the obtained tree is not in $MinimumTreesSet$:
  - **2.2.1** Put the obtained tree into $MinimumTreesSet$.
  - **2.2.2** Put the obtained tree into $CheckTreesSet$.

3 While $CheckTreesSet$ is not empty;
- **3.1** Remove one tree from the $CheckTreesSet$.
- **3.1** Repeat step 2 for this tree.

Recall that a primary objective in a WSN is to minimize the total number of transmissions (links in the forwarding trees) for energy efficiency, and later schedule those trees to obtain a shortest schedule for efficient data gathering latency. If the primary objective is latency, then more trees may be enumerated (step 2.2 in Algorithm 6.4 can be updated to accept trees with larger size (we add μ to $NLinks$, where, μ indicates the number of edges that is acceptable if the obtained tree has links more than optimal tree)). It might be possible that trees with larger size yield a better schedule length. If we let the value of μ to be large enough (i.e. $\mu \geq$ number of edges in the network), the algorithm will find all possible forwarding trees without a cycle. Let $\overline{\tau_i} = \{\overline{t^1}, \overline{t^2}, ...\}$ be the set of all forwarding trees for each set I_t, the scheduling length is obtained by solving the scheduling subproblem for all $\overline{\tau_1} \times \overline{\tau_2} \times ... \times \overline{\tau_m}$ combinations. Let $S^* = \{\tau_1^*, \tau_2^*, ..., \tau_m^*\}$ indicate the optimal tree combinations yielding optimal schedule, τ_t^* for interest nodes set t (obtained through MILP or exhaustive), and let S^τ and $S^{\overline{\tau}}$ be the best scheduling found for τ and $\overline{\tau}$ respectively. Then, $S^* \leq S^{\overline{\tau}} \leq S^\tau$.

Table 6.3 shows the results (number of time slots, time complexity and

number of constructed forwarding trees combination) for the same instances used in Table 6.2. In this table, the scheduling problem for each trees-combination is solved using the two methods (optimal model and distributed algorithm). The results show that constructing multiple trees for each projection improves the performance of the disjoint methods. However, it significantly increases by computational complexity. For instance, the multiple trees construction with optimal scheduling method (for 20-node network) performed 12% better than OTC-OLS, but obtained the solution after days (res. near an hour) when taking all forwarding tree combinations (res. minimum tree combinations). On the other hand, when solving the scheduling subproblem with distributed algorithm, the multiple trees construction method solved the 20-node network in minutes (much faster than when solving the scheduling subproblem optimally), whereas D-FTCS took less than a second with a worst gap scheduling performance not exceeding 12% (but, equal number of transmissions in case of Min-Trees). It should be noted that the computational complexity of all the multiple trees construction methods grow exponentially with the size of the network, whereas D-FTCS is scalable for very large networks due to the fact that each node in a network can do the tree construction and scheduling locally.

Table 6.3: FTCS performance using combinations of multiple forwarding trees ($m = 20\%n$, time is shown by h:m:s)

#Nodes	**Optimal (MILP)**		**Optimal Scheduling Subproblem**				**Algorithmic Sheduling Subproblem**					
			All-Trees		**Min-Trees**		**All-Trees**			**Min-Trees**		
	#Slots	**Time**	**#Slots**	**Time**	**#Slots**	**Time**	**#Tran.**	**#Slots**	**Time**	**#Tran.**	**# Slots**	**Time**
n=10	**5.6**	0:00:23	**5.6**	0:00:59	**5.6**	0:00:24	10.8	**5.6**	0:00:27	10.8	**5.6**	0:00:14
			44.80 tree comb.		16.00 tree comb.		44.80 tree comb.			16.00 tree comb.		
n=15	**7.8**	13:50:37	**7.8**	0:29:35	**7.8**	0:03:16	21.2	**9**	0:01:13	20.6	**9.4**	0:00:38
			427.40 tree comb.		74.2 tree comb.		427.40 tree comb.			74.20 tree comb.		
n=20	Exp. 10	Out of Memory	**10**	61:40:26	**10**	0:40:01	31.4	**12.6**	0:03:07	31	**13**	0:00:51
			3468.40 tree comb.		53.00 tree comb.		3,468.40 tree comb.			53.00 tree comb.		
n=25	-		-		-		44.8	**16**	0:15:08	44.4	**16.4**	0:02:21
							29,425.60 tree comb.			2,034.00 tree comb.		
n=30	-		-		-		-			62.8	**20.8**	0:05:14
							239,831.20 tree comb.			5,301.00 tree comb.		

Our distributed method Vs. [29]

Next, we examine and compare the performance of our distributed solution for FTCS (D-FTCS) with the centralized data gathering algorithm (LLHC-MWF) presented in [29] in terms of schedule length and number of transmissions required to complete one round of data gathering. It should be noted however that LLHC-MWF algorithm does not use compressive data gathering; it constructs only one tree for data gathering, where each node in the network chooses a parent node that minimizes the maximum subtree size and introduces a new link that is compatible with most links in the constructed tree.

Figures 6.6, 6.7, 6.8 and 6.9 depict the results of comparison between D-FTCS (with ϵ=0.0, ϵ=0.25, ϵ=0.5) and LLHC-MWF under different network topologies (sparse and dense) and varying number of sensor nodes (100 to 500) with communication radius ranges from 45 to 100 units for sparse and 75 to 150 units for dense networks, and different number of projections (m=$10\%n$ and m=$20\%n$) with an average of ten runs. As shown in the figures, our D-FTCS (with any value of ϵ) outperforms LLHC-MWF and achieves much shorter schedule lengths (thus lower collection latency). For instance, when $n = 500$ and $\epsilon = 0.5$, D-FTCS in the sparse network performs 25% and in the dense network performs 21% better than LLHC-MWF. It should be noted here that such gains are attributed to compressive data gathering, a feature lacking in the LLHC-MWF method. LLHC-MWF on the other hand constructs only one tree and is oblivious to the order of transmissions when performing link scheduling; in other words, in LLHC-MWF, a parent node does not need to wait for its children's measurements since it is not performing any compression, thus its scheduling is more flexible. Nonetheless, our D-FTCS outperformed LLHC-MWF. It is also notable that D-FTCS performs much better than LLHC-MWF

when the number of projections is smaller (around 15% to 25% when $m = 20\%n$, and 41% to 52% when $m = 10\%n$ for different network sizes). With fewer projections, fewer forwarding trees are constructed, and when constructed efficiently, they result in much shorter schedule. Moreover, the curves in the figures show that the performance of D-FTCS over LLHC-MWF in sparse networks increases steeper than dense networks, specially in large networks. The reason goes for the advantage of compressive data gathering, where in the sparse networks, because of deficiency of interference, more links can be scheduled in fewer time slots. In addition, the figures confirm that the results of D-FTCS vary according to the value of ϵ as explained earlier, where a smaller ϵ achieves shorter schedule length.

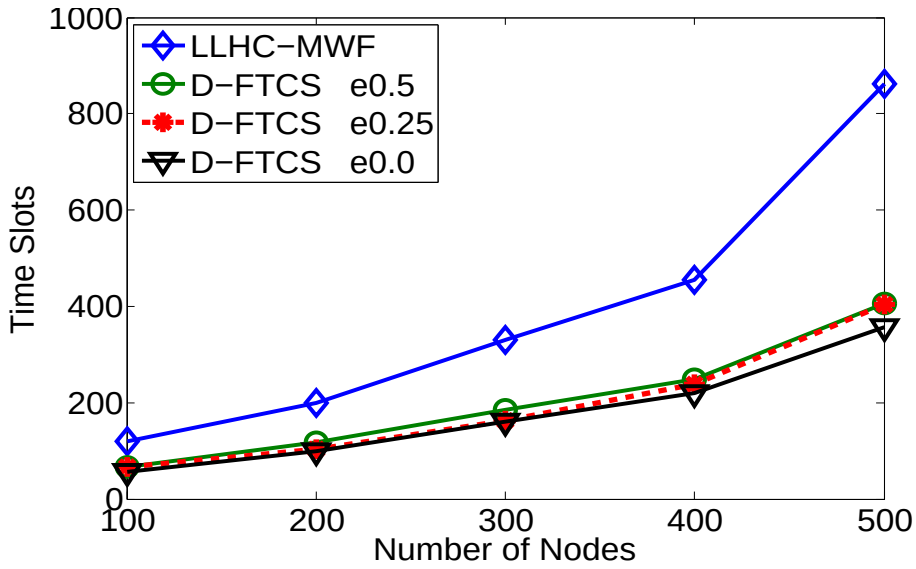


Figure 6.6: FTCS Vs. LLHC-MWF: # slots in sparse network, m=10%n

Finally, we consider a network of 200 nodes and we compare the performance of D-FTCS with LLHC-MWF [29] as we vary the number of projections (m) used for FTCS. The results (schedule length and number of transmissions) are shown in Figure 6.10. The number of transmissions and time slots for different number of projections (m) in LLHC-MWF are both uniform, since

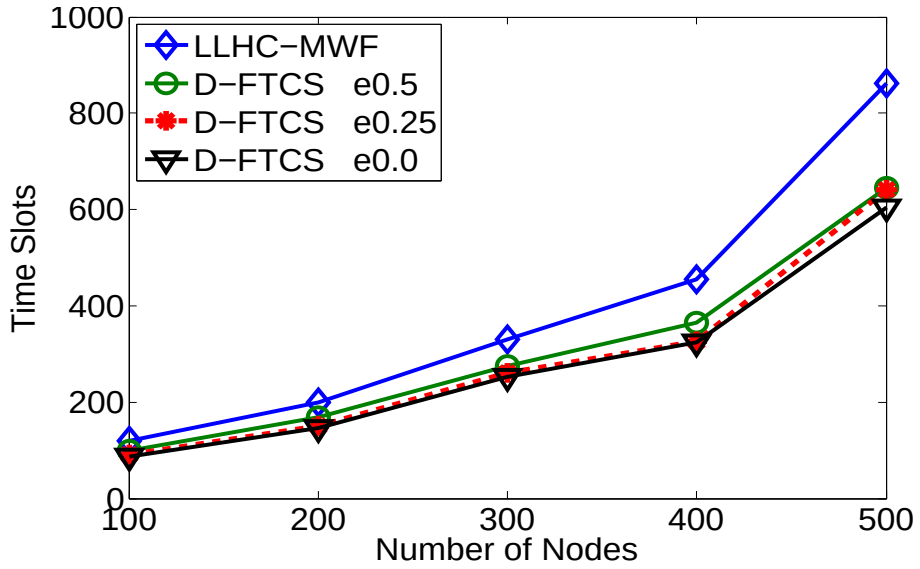


Figure 6.7: FTCS Vs. LLHC-MWF: # slots in sparse network, m=20%n

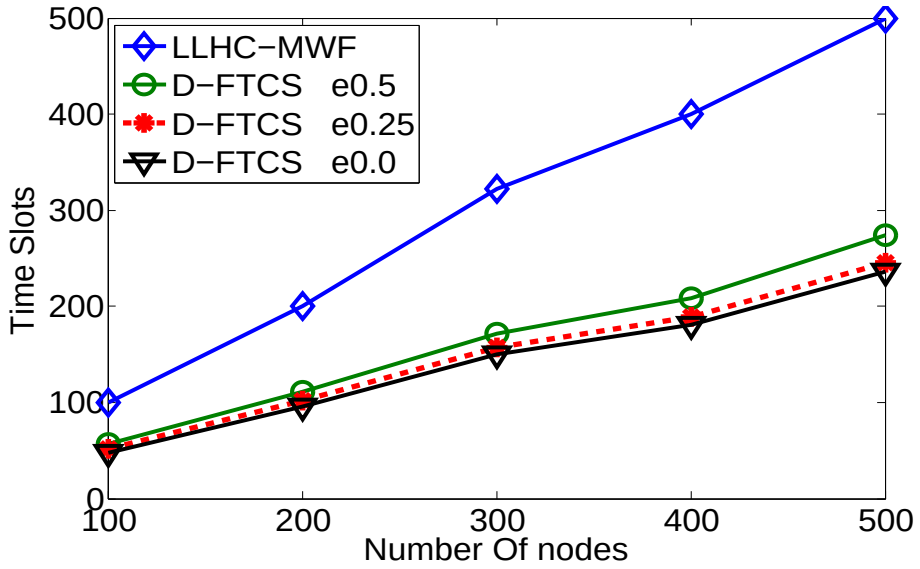


Figure 6.8: FTCS Vs. LLHC-MWF: # slots in dense network, $m = 10\%n$

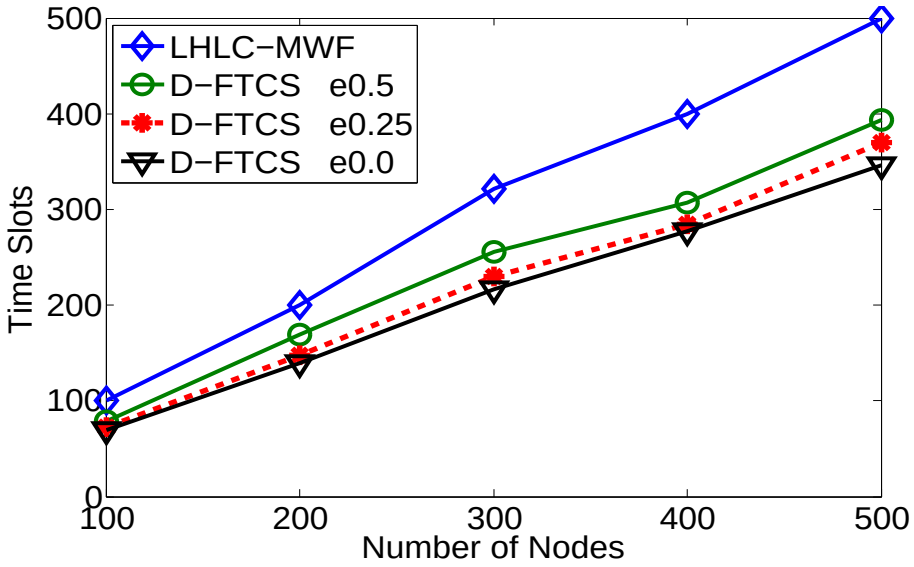


Figure 6.9: FTCS Vs. LLHC-MWF: # slots in dense network, $m = 20\%n$

LLHC-MWF method does not rely on compressive data gathering technique and thus is not affected by different number of projection. The figure shows that when the number of projections is small, D-FTCS substantially outperforms LLHC-MWF both in terms of number of transmissions and schedule length. For instance, when m=5$\%n$ and $10\%n$, with D-FTCS, few trees are constructed to collect the data from the network (respectively 10 and 20 trees), and owing to compressive data gathering, much fewer transmissions are needed to collect the data (resp. 58% and 45% less transmissions), where such transmissions can be scheduled effectively in a very short period of time (resp. performs 67% and 50% better). The schedule length is either smaller than half or close to half that of LLHC-MWF. However, as the number of projections increases, then more forwarding trees are constructed and hence more transmissions will be needed. Accordingly, the length of schedule as well as number of transmissions start to increase. As Figure 6.10 shows, when $m = 40\%n$ or bigger, our algorithm performs slightly worse than LLHC-MWF. Alternatively, if the number of projections is kept smaller, then D-FTCS outperforms substantially the

performance of LLHC-MWF, as depicted in Figure 6.11, for varying network sizes. For example, when m=20%n, FTCS achieves gains that vary between 29% and 44% over LLHC-MWF.

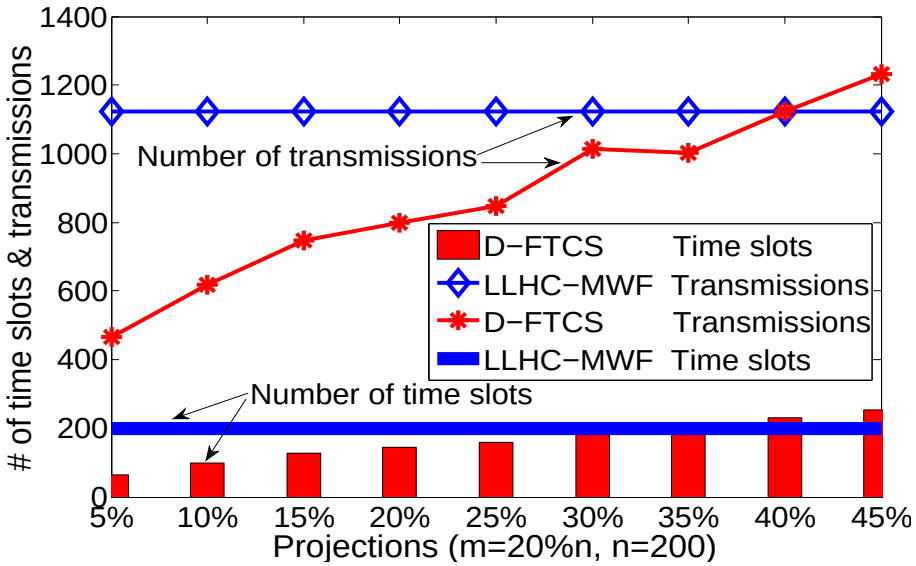


Figure 6.10: FTCS vs. LLHC-MWF: schedule length Vs. # transmissions

6.7 Conclusion

In this chapter, we considered the problem of projection-based compressive data gathering and scheduling in wireless sensor networks under the physical interference model. We formulated the problem of joint forwarding tree construction and link scheduling mathematically with the objective of achieving energy efficient data gathering with minimal collection latency. We highlighted the complexity of the problem, and then we presented our decentralized algorithm for solving it. Our decentralized approach decouples the problem into two subproblems; namely, the tree construction subproblem and the link scheduling subproblem. Our decentralized tree construction is amended with

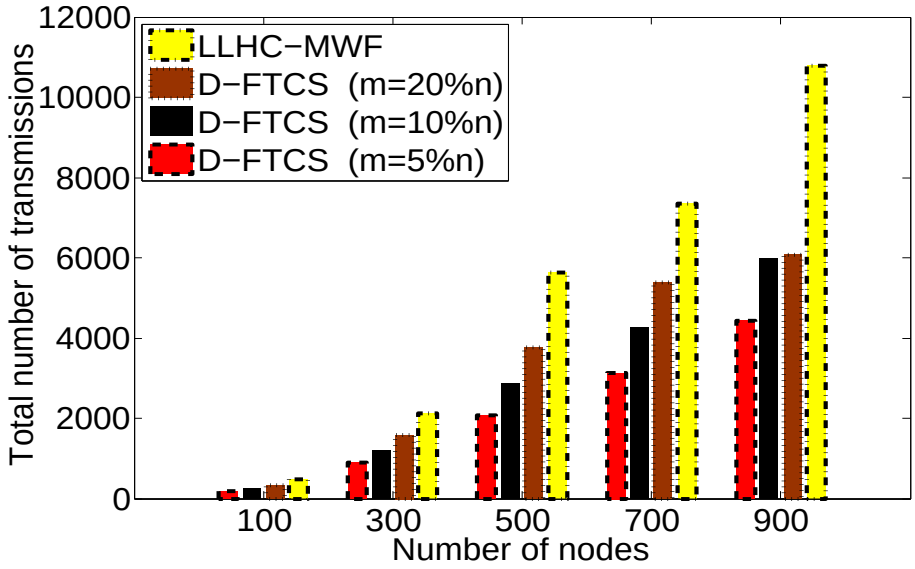


Figure 6.11: FTCS vs. LLHC-MWF: # nodes Vs. # transmissions

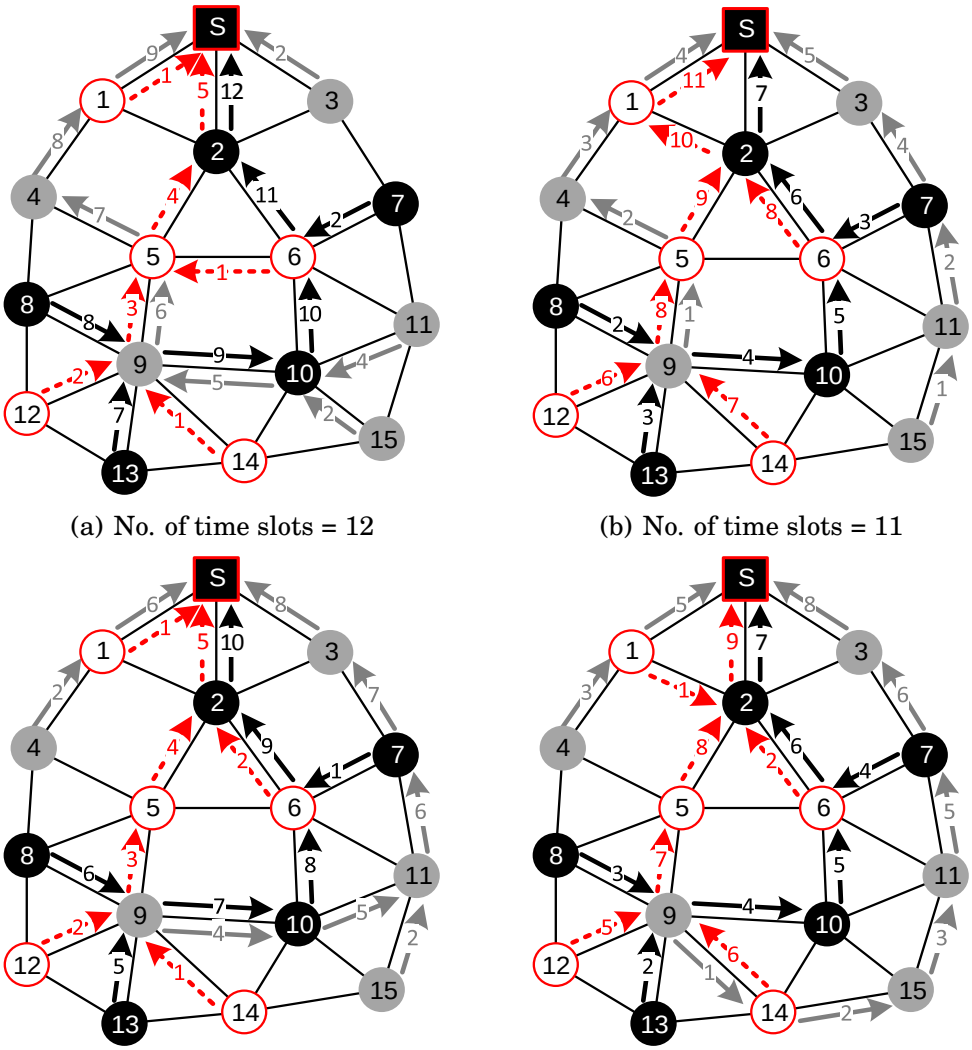
refinements to help the link scheduling achieve better scheduling and thus collection latency. Our scheduling subproblem is resolved in a distributed fashion, through interference localization and coordination among links to control the level of interference. Our distributed method showcased the benefits of compressive data gathering in collecting measurements and has been shown to be scalable with outstanding performance in terms of energy efficiency (number of transmissions) and gathering latency (time to gather data from sensors).

Chapter 7

A Column Generation (CG) Approach for FTCS

In the problem of constructing multiple forwarding trees and scheduling (FTCS), presented in Chapter 6, each tree may be constructed independently and then its links are scheduled. However, when all trees are combined together, the shortest and energy efficient schedule may not be guaranteed. Further, a large number of possible forwarding trees for each group of sensors may be considered. Both problems of enumerating forwarding trees and scheduling links for those trees are hard combinatorial problems [24]. This is compounded by the fact that the two problems must be solved jointly, to guarantee the selection of best forwarding trees which, when their links are scheduled, guarantee a shortest energy efficient schedule.

Figure 7.1 illustrates an example on the interaction between the tree construction and link scheduling and highlights the impact on the data gathering latency (or the schedule length). The example shows how to gather data at the sink from all sensors using three groups of interest-nodes. As the figures



(a) No. of time slots = 12

(b) No. of time slots = 11

(c) No. of time slots = 10

(d) No. of time slots = 9

Figure 7.1: An example of tree construction and link scheduling; (d) joint and (a)-(c) disjoint with different tree constructions. Here, $m = 3$ ($m = 20\%n$). In the network, S is the sink which intends to gather data from all nodes. Same color arcs represents aggregation tree for one set. Each set of interest nodes is illustrated with same color. The numbers on the arcs represent the time slot when the tree links are active.

show, all four tree constructing methods require the same number of transmissions (links) to gather the sensory data; however, Figure 1(d) shows that the trees in the joint FTCS can be scheduled in only 9 time slots, whereas, other figures (Fig. 1(a) - Fig. 1(c)) require more time slots to collect the measurements. This is due to the fact that trees are constructed without considering the requirements for achieving a shorter schedule length.

In this chapter, after highlighting the complexity of the FTCS problem, we present a novel primal-dual decomposition method using column generation. We also highlight several challenges we faced when solving the decomposed problem and present efficient techniques for mitigating those challenges. One major advantage of our work is that it can serve as a benchmark for evaluating the performance of any low complexity method for solving the FTCS problem for larger network instances where no known exact solutions can be found.

7.1 Problem Formulation and Complexity

The FTCS problem, as we mentioned earlier, has two sub-problems: (1) Finding m forwarding trees, each connects one set of interest-nodes to a sink node, and (2) Scheduling the links on these forwarding trees. The joint FTCS problem can be modeled as follows:

Objective:

$$\min_{\overrightarrow{\mathcal{T}} \in \overrightarrow{T}} f(\overrightarrow{\mathcal{T}}) \tag{7.1}$$

Subject to:

1. Routing (tree construction) constraints.
2. Link scheduling constraints.
3. SINR constraint.

In (7.1), we assume $f(.)$ is a cost function incorporating both energy consumption and gathering latency. $\overrightarrow{\tau} = (\tau_1, \tau_2, ..., \tau_m)$ is set of forwarding trees, each for a set of interest-nodes (I_t), and $\overline{T} = \bigcup_{t=1}^{m} \overline{T}_t$, where $\overline{T}_t$ is the set of all possible forwarding trees for interest-nodes set I_t. The above FTCS model has been mathematically formulated in chapter 6 (and in our previous work [24]) as an ILP model. Our objective function aims at minimizing the number of links in the constructed trees to reduce the number of transmissions for data gathering (thus, conserving energy) and the required number of time slots needed to schedule the constructed trees (i.e, gathering latency). Clearly, this ILP model is complex and hard to scale for networks of reasonable sizes. The complexity of the problem can be categorized as follows: 1) Complexity of constructing and enumerating forwarding trees; this problem, similar to the Steiner tree problem, is NP-complete and its NP-hardness has been shown in [39]. 2) Complexity of link scheduling under the physical interference model; the problem is shown to be NP-complete, e.g., in [30]. 3) Complexity of finding multiple trees which, collectively, guarantee a minimum schedule length; it has been shown in [29] that this problem is NP-hard as well. Note that, in addition to the listed complexities, the above model has to jointly construct and schedule links for multiple trees. Table 7.1 illustrates the ILP solutions for three network sizes. As the table shows, the ILP model failed to find solutions for networks with 20 nodes or larger due to the large number of mapping possibilities that the model had to search through to find the optimal solution. Therefore, to address the scalability issue, in the following section we introduce a primal-dual decomposition approach using Column Generation (CG) [14].

Table 7.1: Performance of ILP model ($m = 20\%n$)

# Nodes	# Trans.	# Slots	CPU Time
n=10	11	6	12 seconds
n=15	20	8	10 hours
n=20	Out of memory after passing 8 hours		

7.2 Decomposition method

To design a more efficient method, we decompose our problem into sub-problems, using the technique of CG [14]. CG is an efficient method for solving large scale LP problems by decomposing the original problem into two sub-problems, a Master and a Pricing. The two sub-problems are solved iteratively until an optimal criteria is met. The Master (LP) is initialized with a subset of configurations (columns) that satisfy all the constraints in the Master model (a feasible solution is obtained). The Pricing (ILP), which is a separation model for the dual LP of the Master, iteratively generates and adds columns that improve the solution of the Master problem. A very few number of these columns is usually sufficient for the Master sub-problem to obtain the optimal LP solution. In some cases, in order to obtain an $epsilon$ optimal solution for the ILP model, it is enough to solve the Master problem using the columns associated with the optimal LP solution.

Given the nature of our FTCS problem, we decompose it into a Master and a set of Pricing sub-problems, where each Pricing constructs (for each group of interest nodes) a tree and schedules its links. Pricing sub-problems, at each iteration, generate configurations for the Master, and the Master problem chooses the best combination of configurations among all the feasible ones gathered by the Pricing problem. Figure 7.2 illustrates an example of how Master columns (configurations) look like in our problem. As the figure shows, for each interest-nodes set (I_t), each configuration is generated by solving a

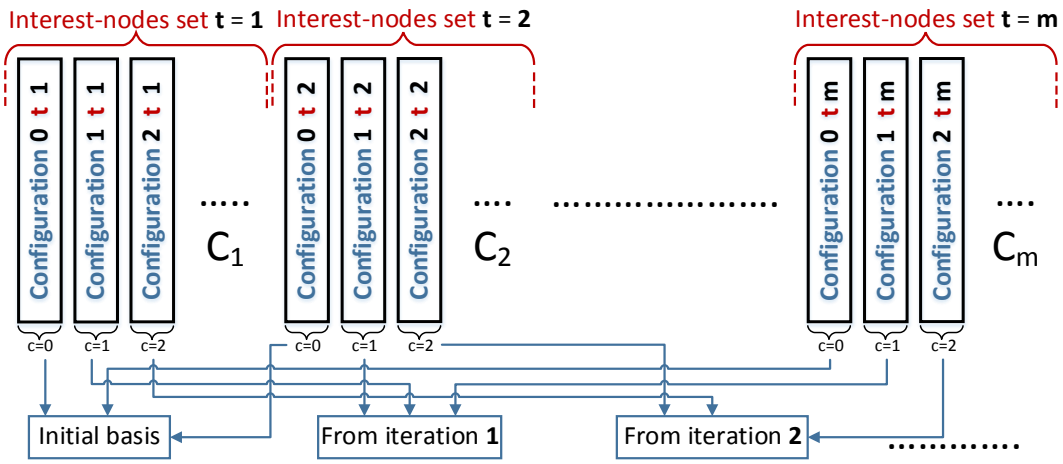


Figure 7.2: Example of master columns/configurations.

Pricing sub-problem at each iteration. Let C_t represent the set of configurations for set of interest-nodes I_t (recall, each configuration is a tree and a schedule for I_t). The set of all possible configurations is depicted by $C = \bigcup_{t=1}^{m} C_t$. Among these configurations, the Master sub-problem should select one and only one configuration for each I_t. Furthermore, the Master should forbid the selection of 1) configurations which contain the same link that is scheduled at the same time slot along the scheduling horizon; 2) configurations in which the same node receives from more than one child (each child in different configuration) in the same time slot; 3) configurations in which a node is receiving and transmitting at the same time slot; 4) configurations which violate the SINR constraints.

At the beginning of our decomposition method, to get an initial basis (configurations) for solving the Master problem, we start by independently constructing m minimum spanning trees (MSTs), one tree for all interest-nodes in each set (I_t, $t = 1..m$), to a sink node using an ILP model (Tree Construction Model (TCM), see Appendix C). Then, for each tree, using a separate ILP

model (Link Scheduling Model (LSM), refer to Appendix D), we find the minimum number of time slots required to schedule the links. The initial configurations of the Master are created by concatenating schedules of all trees to avoid scheduling conflicts among links or transmissions of different trees (inter-tree interference). Now, as we explained earlier for CG, the Master and sub-Pricing problems iteratively alternate until a stoping criteria is met [14]. The Master problem can be formulated as follows:

7.2.1 The Master Problem

• **Parameters:**

Table 7.2: Common parameters used throughout the chapter

V:	Set of vertices in the graph.
E:	Set of edges in the graph.
$\mid I_t \mid$:	Total number of interest-nodes in set I_t.
S:	A large number of time slots for one round of data gathering.
P:	Power transmission for each node.
G_{ij}:	Channel gain from transmitter i to receiver j.
β:	Minimum SINR threshold.
η:	Background noise.
B:	Big number.
ε:	$0 < \varepsilon < 1$.

λ_c^t : Number of consecutive time slots required to schedule configuration c for interest-nodes set t.

$d_{ij,c}^{s,t}$: Indicate whether link (i, j) at time slot s for configuration c of interest-nodes set t is active or scheduled.

$b_{i,c}^{s,t}$: Indicate whether node i at time slot s for configuration c of interest-nodes set t is busy (transmitting or receiving).

$F_j^{s,t}$: Interference caused by other scheduled links on node j at time slot s on tree selected by interest-nodes set t.

$$a_{ij,c}^{s,t} = \begin{cases} PG_{ij} - \beta F_j^{s,t}, & \text{if link } (i,j) \text{ is active at time slot } s \text{ for configuration } c \\ & \text{of interest-nodes set } t; \\ -\beta F_j^s, & \text{otherwise.} \end{cases}$$

- **Decision Variables:**

L :Total number of required slots.

$$Z_c^t = \begin{cases} 1, & \text{if configuration } c \text{ for interest-nodes set } t \text{ is selected;} \\ 0, & \text{otherwise.} \end{cases}$$

H_{ij}^s : Indicate whether any configuration is using link (i,j) at time slot s.

- **Mathematical Model:**

$$Minimize \qquad L \tag{7.2}$$

Subject to:

$$\gamma : \quad L \geq \lambda_c^t \ Z_c^t \qquad \forall c \in C, t = 1..m. \tag{7.3}$$

$$\delta : \quad \sum_c Z_c^t = 1 \qquad t = 1..m. \tag{7.4}$$

$$\theta : \quad \sum_{t,c} b_{i,c}^{s,t} \ Z_c^t \leq 1 \qquad \forall i \in V, s \in S. \tag{7.5}$$

$$\omega : \quad \sum_{t,c} a_{ij,c}^{s,t} \ Z_c^t \geq \beta \ \eta + (H_{ij}^s - 1)B \qquad \forall (i,j) \in E, s \in S. \tag{7.6}$$

$$\mu : \quad H_{ij}^s \geq \varepsilon \sum_{t,c} d_{ij,c}^{s,t} \ Z_c^t \qquad \forall (i,j) \in E, s \in S. \tag{7.7}$$

$$\nu : \quad H_{ij}^s \leq \sum_{t,c} d_{ij,c}^{s,t} \ Z_c^t \qquad \forall (i,j) \in E, s \in S. \tag{7.8}$$

Note that λ_c^t, $b_{i,c}^{s,t}$, $d_{ij,c}^{s,t}$ and $a_{ij,c}^{s,t}$ are parameters in the Master problem which are obtained after solving the LSM or/and the multi-Pricing sub-problems. Each Pricing solves its decision variables λ, b_i^s, d_{ij}^s and a_{ij}^s, which correspond respectively to parameters in the Master problem. At each iteration, columns (a column for each interest-nodes set) of these parameters are added to the basis of the Master problem. Therefore, the number of configurations ($C = \bigcup_{t=1}^{m} C_t$) in the Master problem is increased by one for each interest-nodes set t (I_t) at each iteration. $a_{ij,c}^{s,t}$ is a parameter which the Master needs to calculate the interference on link (i,j) (belonging to configuration c for interest-nodes set t) scheduled at time slot s. When link (i,j) is active, $a_{ij,c}^{s,t}$ is the received power at node j minus interference caused by other links active in the same time slot of the same configuration. When link (i,j) is not active, $a_{ij,c}^{s,t}$ depicts only interference coming from other active links at time slot s (i.e., $a_{ij,c}^{s,t} = -\beta F_j^{s,t}$).

The objective of the Master problem is to choose best configurations obtained from the multi-Pricing sub-problems which minimize the total number of time slots required for scheduling collectively forwarding trees for all interest-nodes sets. Constraint (7.3) finds the number of consecutive time slots required for scheduling each configuration of a group of interest-nodes. Constraint (7.4) asserts that for each group of interest-nodes only one configuration is selected. Constraint (7.5) makes sure that nodes are not active (scheduled) for more than one activity (transmit/receive) at a time slot. Constraints (7.6), (7.7) and (7.8) enforce that the SINR constraint for each link (i,j) is satisfied when all m forwarding trees for all sets of interest-nodes are selected. As we explained earlier in section 6.1, the SINR constraint (6.1) for

link (i,j) is satisfied if

$$\frac{PG_{ij}}{\eta + F_j^s} \geq \beta \tag{7.9}$$

where F_j^s is the aggregate interference caused by other concurrent active links at receiver node j in time slot s. Hereafter, we explain how constraints (7.6), (7.7) and (7.8) lead to SINR constraint (7.9). First, for each link, we check whether it is active in any configuration in a time slot; this can be represented by H_{ij}^s and obtained through (7.7) and (7.8). Then, if the link is not active (i.e., $H_{ij}^s = 0$), constraint (7.6) is always satisfied (i.e., no need to check the SINR constraint). Else, if it is active (i.e., $H_{ij}^s = 1$), constraint (7.6) reduces to

$$\sum_{t,c} a_{ij,c}^{s,t} \;\; Z_c^t \;\geq\; \beta \;\; \eta \tag{7.10}$$

Now, when link (i,j) is active in a forwarding tree of interest-nodes set t, $a_{ij,c}^{s,t} = PG_{ij} - \beta F_j^{s,t}$, otherwise, $a_{ij,c}^{s,t} = -\beta F_j^{s,t}$. We know that link ($i$,$j$) can be active in only one tree of the interest-nodes set (refer to constraint (7.5)), and only one configuration (c) is going to be chosen for each interest-nodes set (forced by constraint (7.4)). Therefore, only one term in the left hand side of inequality (7.10) (i.e., $\sum_{t,c} a_{ij,c}^{s,t} \;\; Z_c^t$) should be positive (this corresponds to a tree of interest-nodes set which has an active link (i,j), i.e., $a_{ij,c}^{s,t} = PG_{ij} - \beta F_j^{s,t}$), and negative for all other trees of interest-nodes sets (i.e., $a_{ij,c}^{s,t} = -\beta F_j^{s,t}$; these negative values add the interference caused by other active links in other trees on node j). Hence, (7.10) can be written as:

$$PG_{ij} - \beta F_j^s \geq \beta \, \eta \tag{7.11}$$

which is equivalent to (7.9). Figure 7.3 illustrates an example on how concurrent active links in two trees (solid and strip arcs) have impact on link (i,j). The figure shows only the concurrent active links in both trees. For the first tree (shown by solid arcs) $a_{ij,c}^{s,t_{solid}} = PG_{ij} - \beta P[G_{kh} + G_{ab}]$, and for the second tree (shown by strip arcs) $a_{ij,c}^{s,t_{strip}} = -\beta P[G_{eg} + G_{oq} + G_{ru}]$. Accordingly, $\sum_{t,c} a_{ij,c}^{s,t} \ Z_c^t = PG_{ij} - \beta P[G_{kh} + G_{ab} + G_{eg} + G_{oq} + G_{ru}]$, which is equal to $PG_{ij} - \beta F_j^s$.

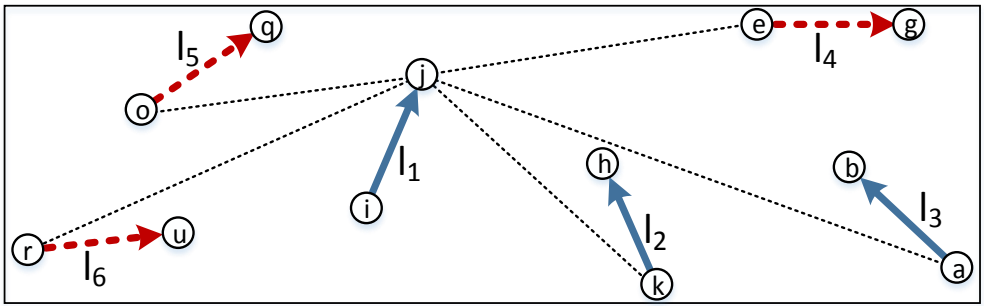


Figure 7.3: Example of interference caused by concurrent active links in two trees (solid and strip arcs) on link (i,j). The interference from the transmitter of concurrent links on the receiver of link (i,j) is shown by stipe line.

7.2.2 The Pricing Problem

During each iteration, the Pricing problem generates a new feasible column (to join the Master basis) which may improve the Master's current LP solution. The columns which improve the Master's objective value are the ones with the negative reduced costs. The Pricing sub-problem is guided towards those improving columns by the dual values passed from the Master problem. Let γ, δ, θ, μ and ν denote the dual values corresponding to constraints (7.3), (7.4), (7.5), (7.7) and (7.8) in the Master problem respectively. The Pricing problem keeps generating new columns as long as the Master problem provides the combination of dual values that allow such columns. When the best column the Pricing can generate is of reduced cost value greater than or equal to zero,

the CG algorithm stops the search process for the best combination of columns.

$$RC = \quad -\lambda \;\; \gamma + \delta - \sum_{i,s} b_i^s \;\; \theta_i^s + \sum_{ij,s} a_{ij}^s \;\; \omega_{ij}^s - \varepsilon \sum_{uv,s} d_{ij}^s \;\; \mu_{ij}^s + \sum_{ij,s} d_{ij}^s \;\; \nu_{ij}^s \qquad (7.12)$$

When the objective of the Master problem is to minimize, the standard pivoting rule of the Simplex method is to choose a new column (configuration) such that (7.12) is maximum; the column which is found is added to the basis of the Master problem. The Master problem is solved, again, with the new basis to obtain a new solution, and the dual variables are passed to the sub Pricing problems which are again solved. The Master and Pricing problems are solved iteratively until there is no off-basis column with a negative reduce cost found and therefore the LP solution is optimal. Indeed, this requires that the last simplex iteration of all individual sub Pricing problems are solved to optimality to ensure that there is no off-basis column with positive reduced cost remains unexplored in each pricing. The ILP mathematical model of the Pricing problem is derived as follows:

- **Parameters:**

The dual values (γ, δ, θ_i^s, ω_{ij}^s, μ_{ij}^s and ν_{ij}^s) from the Master problem.
The remaining parameters are listed in Table 7.2.

- **Decision Variables:**

$$x_{ij} = \begin{cases} 1, & \text{if link } (i,j) \text{ is active in a tree;} \\ 0, & \text{otherwise.} \end{cases}$$

$y_{ij} \in \mathbb{Z}:$ Indicating at what time slot link (i,j) is active ($y_{ij} \geqslant 0$).

$$d_{ij}^s = \begin{cases} 1, & \text{if link } (i,j) \text{ is active at time slot } s; \\ 0, & \text{otherwise.} \end{cases}$$

$$b_i^s = \begin{cases} 1, & \text{if node } i \text{ is busy at time slot } s; \\ 0, & \text{otherwise.} \end{cases}$$

$F_j^s \in \mathbb{R}:$ Interference caused by other links on node j at time slot s.

$$a_{ij}^s = \begin{cases} PG_{ij} - \beta F_j^s, & \text{if link } (i,j) \text{ is active at time } s; \\ -\beta F_j^s, & \text{otherwise.} \end{cases}$$

$\lambda \in \mathbb{Z}:$ Consecutive number of time slots required for scheduling.

- **Mathematical Model:**

$$Maximize \quad RC \tag{7.13}$$

Subject to:

Constraints (C.2)-(C.4) for x_{ij} in TCM (see Appendix C)

$$\mid y_{ij} - y_{ik} \mid \geq x_{ij} + x_{ik} - 1 \quad \forall (i,j) \& (i,k) \in E : j \neq k. \tag{7.14}$$

$$\mid y_{ij} - y_{jk} \mid \geq x_{ij} + x_{jk} - 1 \quad \forall (i,j) \& (j,k) \in E : i \neq k. \tag{7.15}$$

$$y_{ij} \geq \frac{x_{ij}}{B} \qquad \forall (i,j) \in E. \tag{7.16}$$

$$y_{ij} \leq B\, x_{ij} \qquad \forall (i,j) \in E. \tag{7.17}$$

$$\mid y_{ij} - y_{kj} \mid \geq x_{ij} + x_{kj} - 1 \quad \forall (i,j) \& (k,j) \in E : i \neq k. \tag{7.18}$$

$$y_{jk} + B(1 - x_{jk}) \geq y_{ij} + 1 \qquad \forall (j,k) \& (i,j) \in E. \tag{7.19}$$

$$\begin{cases} d_{ij}^s \leq 1 - \frac{|s - y_{ij}|}{B} \\ d_{ij}^s \geq 1 - (B \mid s - y_{ij} \mid) \end{cases} \qquad \forall s \in S, (i,j) \in E. \tag{7.20}$$

$$P\,G_{ij} + B(1 - d_{ij}^s) \geq \beta(\eta + \sum_{(k,h)\in E:k\&h\neq i\&j} P\,G_{kj}.d_{kh}^s) \quad \forall s \in S, (i,j) \in E. \tag{7.21}$$

$$\lambda \geq y_{ij} \qquad \forall (i,j) \in E. \tag{7.22}$$

$$\begin{cases} b_i^s \leq \sum_{j:(j,i)\in E} d_{ji}^s + \sum_{j:(i,j)\in E} d_{ij}^s \\ b_i^s \geq \sum_{j:(j,i)\in E} d_{ji}^s \qquad \forall s \in S, i \in V. \\ b_i^s \geq \sum_{j:(i,j)\in E} d_{ij}^s \end{cases} \tag{7.23}$$

$$a_{ij}^s = P\,G_{ij}d_{ij}^s - \beta \sum_{(k,h)\in E:k\&h\neq i\&j} P\,G_{kj}.d_{kh}^s) \quad \forall s \in S, (i,j) \in E. \tag{7.24}$$

It should be noted that we have m Pricing sub-problems, one for each set of interest-nodes (I_t), and the above mathematical model represents one Pricing (similar procedure is repeated for others). The objective of the Pricing is to construct a forwarding tree for a set of interest-nodes and schedule its links by considering dual values from the Master problem. The variable x_{ij} indicates the forwarding tree which is required to be scheduled in the Pricing problem. Kindly, refer to constraints (C.2)-(C.4) in Appendix C for constraints related to x_{ij}. Constraint (7.14) ensures that a node may not transmit to multiple receivers. Constraint (7.15) asserts that a node may not receive and transmit at the same time slot. Constraint (7.16) ensures that all active edges (i.e., edges which are part of a tree) have to be scheduled (i.e., when $x_{ij} = 1 \implies y_{ij} > 0$). Similarly, all edges in a graph which are not part of the tree (i.e., $x_{ij} = 0$) are not scheduled (i.e., $y_{ij} = 0$), which is enforced through constraint (7.17). Constraint (7.18) asserts that a node can not receive data from multiple transmitters simultaneously in the same time slot. Constraint

(7.19) ensures that a parent link in a tree has to be scheduled after any child link. In other words, a node in a tree can not transmit unless it receives all packets from its children. Constraints (7.20) obtain d_{ij}^s which are required for SINR measurements. Constraints (7.20) make sure $d_{ij}^s = 1$ when a link (i,j) in a tree is active (scheduled) at time slot s (i.e., $d_{ij}^s = 1$, if $x_{ij} = 1$ and $s = y_{ij}$), otherwise $d_{ij}^s = 0$. Constraint (7.21) ensures that the SINR for each active link is above the threshold β. Note that the constant B should be greater than any $\eta + \sum_{(k,h)\in E;k\neq i} P\, G_{kj}^{-\alpha}\, a_{kh}^s$. In (7.21), if link (i,j) in a tree is active in time slot s (i.e., $d_{ij}^s = 1$), then (7.21) reduces to expression (6.1). Constraint (7.22) finds the last time slot or the total number of consecutive time slots required for tree scheduling. Constraints (7.23) show whether a node is busy transmitting or receiving data (packet) at time slot s. In other words, $b_i^s = 0$, if $\sum_{(j,i)\in E} d_{ji}^s + \sum_{(i,j)\in E} d_{ij}^s = 0$, and $b_i^s = 1$, otherwise. These busy node variables (b_i^s) are needed for the Master problem. The last constraint (7.24) obtains variables a_{ij}^s (defined above in the decision variables) which are required for the master problem as well. $a_{ij}^s = PG_{ij} - \beta \sum_{(k,h)\in E:k\&h\neq i\&j} PG_{kj} d_{kh}^s$, when $d_{ij}^s = 1$; and $a_{ij}^s = -\beta \sum_{(k,h)\in E:k\&h\neq i\&j} PG_{kj} d_{kh}^s$, when $d_{ij}^s = 0$.

7.2.3 Solution methodology

At this level, it is clear that our approach for solving the FTCS problem is through decomposing it into 1) a Master, which decides best m configurations which yield shortest schedule for data gathering and 2) a set of Pricing sub-problems, each Pricing, guided by the dual variables obtained from the Master, computes a forwarding tree for a particular set of interest nodes (and schedule its links) and returns the configuration to the Master. However, the fact that each Pricing sub-problem solves a joint tree construction and link scheduling,

renders the Pricing difficult to solve. Further, this is solved for each Pricing sub-problem, and multiple times, as the iterations between Master and Pricing evolve, until the optimal solution is found. Denote this optimal solution by x^*_{LP}, which is the LP solution of the FTCS problem. x^*_{LP} is obtained through the column generation decomposition, where only a subset of the configurations (necessary to obtain the optimal LP solution) is obtained. Let this subset of configurations be C_{CG}. Now to obtain the ILP solution of the FTCS problem, we solve the Master sub-problem one last time without relaxing its integer variables. Let the obtained solution be $\hat{x}_{ILP}$; it is clear that $\hat{x}_{ILP} \geq x^*_{ILP}$ where x^*_{ILP} is the optimal ILP solution for the FTCS problem (x^*_{ILP} is in general unknown, except for small instances where it can be obtained using [24]). This is because $\hat{x}_{ILP}$ is obtained by solving the Master sub-problem with C_{CG}. Through numerical evaluation on small networks, we observed that the gap between $\hat{x}_{ILP}$ and x^*_{ILP} can be very large and that the computation time to obtain $\hat{x}_{ILP}$ is excessive, owing to the complexities discussed earlier. To improve the gap between the LP and ILP solutions, one may need to diversify the configurations or columns in C_{CG}, e.g., by adding more columns to this basis, which could be beneficial to the ILP solution. Hence, and to overcome the above issues, we next elaborate our methodology (Figure 7.4) for solving the FTCS. First, we relax the Pricing by separating the tree construction from the link scheduling (we fix the x_{ij} variables in the Pricing); we assume forwarding trees are already constructed (as we explain below) and the Pricing solves to obtain best schedule (configuration) for the links. This substantially reduces the run time of the Pricing.

Now, to initialize the Master problem, the tree construction model (TCM) runs to construct trees, one for each set of interest nodes, and these trees are

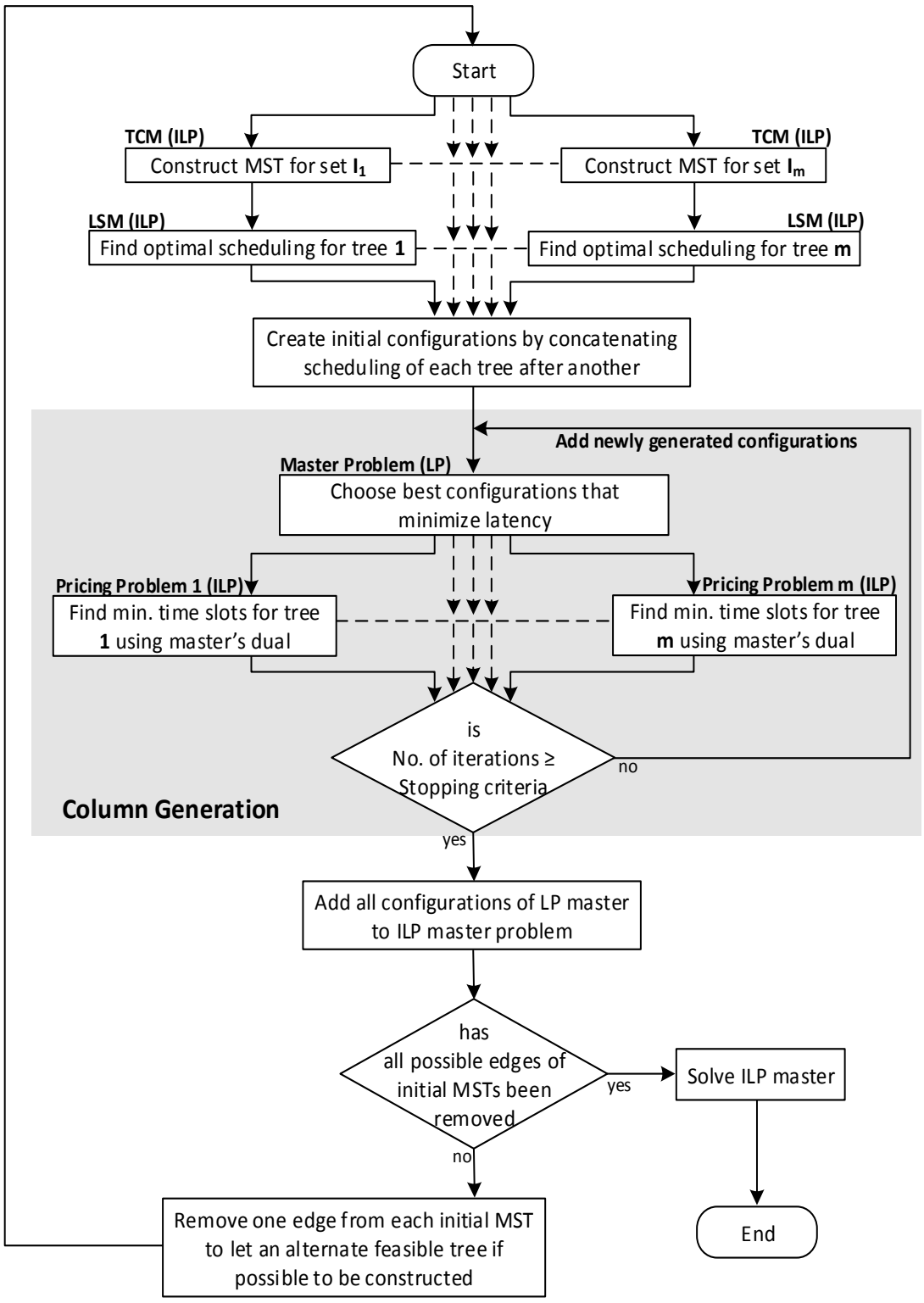


Figure 7.4: Flow chart of the decomposition method.

scheduled using LSM, and schedules are concatenated as explained earlier. Now, the Master runs with this initial basis and then the Pricing sub-problems are solved to return different configurations which improve the solution of the Master. Both Master and Pricing sub-problems keep alternating until reaching a stoping point which is defined by the user. The set of configurations used to obtain the LP solution (C_{now}) is saved into a set $C_{iter} = C_{now}$. Next, for each tree used for each set of interest nodes, we remove one edge and then construct a new tree using the TCM model; this new set of trees is used again to solve our CG model (as explained above), until optimal LP solution is obtained (using the new set of configurations C_{now}). The set of configurations C_{iter} is updated as follows $C_{iter} = C_{now} \cup C_{iter}$. This process keeps repeating (removing one edge at a time from each tree, construct new set of trees, solve CG, update the C_{iter} set) until all edges from the initial trees are removed, one after the other. Finally, the ILP instance of the Master problem is solved (using as input the set of configurations, C_{iter}). The advantages of this method over the previous one are two fold. First, the relaxed Pricing sub-problem is much faster to solve, and second the basis (C_{iter}) used to solve the ILP Master is much more diversified, containing columns which could not be enumerated through the previous decomposition. The procedural details of this method are depicted in Figure 7.4.

7.3 Numerical Results

In this section, we first evaluate the performance of our decomposition method against the optimal solution obtained using the joint ILP model [24]. We also study the performance of CG by varying the number of iterations for different forwarding tree combinations. We further analyze the performance by

either using diverse forwarding trees with different sizes or trees with minimum number of edges. Finally, we compare the performance of CG with a distributed method presented in our previous work [24]. For numerical results, we generate seven networks whose sizes vary from $n = 10$ to $n = 40$ nodes with five nodes increment; nodes are randomly distributed over a 700×700 unit square area without a single disconnected node. Further, we randomly assign each node in the network to m sets of interest-nodes where each set contains $\lceil \frac{n}{m} \rceil$ nodes; we consider $m = 20\%n$. We assume all nodes use the same normalized transmit power $P = 1$. Moreover, we assume a path loss exponent $\alpha = 3$ and SINR threshold for successful transmission $\beta = 2$. We use JAVA to simulate the operation of our CG and CPLEX to solve our optimization models. We run our program on CPU with Intel Core i7 processor, 3.6 GHz speed, 8 GB memory ram and 64-bit windows operating system.

ILP Vs. CG

We start by examining the results obtained by solving the joint FTCS problem (formulated as an ILP model) and compare it with our CG approach. Table 7.3 shows the required scheduling length (number of time slots) for gathering data, as well as the CPU run time needed to solve the FTCS problem. Note that both methods construct forwarding trees with same number of links and thus require the same number of transmissions to gather the sensory data. As it can be observed from the table, for the given network instances, the CG obtains the optimal link scheduling length after iteratively solving a certain number of iterations. However, the CG solves the FTCS problem much faster than the optimal joint ILP model for larger networks. For example, in a 15-node network, CG obtains the result in 27 seconds, whereas, the joint ILP

model took around ten hours to solve. Further, the joint ILP model failed to find solutions for 20 nodes or larger networks.

Table 7.3: ILP model Vs. CG

# Nodes	ILP		CG		
	#Slots	Time	#Slots	Time	#Iterations
n=10	6	12 sec.	6	18 sec.	13
n=15	8	10 hours	8	27 sec.	4
n=20	-	-	14	2 min.	7

Performance of CG Vs. number of iterations

At this stage, it is clear that the scheduling performance depends entirely on the number of iterations the CG performs. Figures 7.5 and 7.6 show the scheduling performance and CPU run time required to gather data for two networks (30-node and 35-node networks respectively) by varying the number of iterations. Clearly, as we increase the number of iterations, the CG performs better with respect to improving (reducing) the schedule length, however, more iterations require more CPU time to execute. In larger networks (e.g., 35-node network, Figure 7.6) the run time increases sharper than smaller networks (e.g., 30-node network, Figure 7.5). The reason for this is that in bigger networks, at each iteration, we have to solve for more trees (since we have more groups of interest-nodes) and hence more sub pricing problems are solved, and for each sub pricing, more time is required to solve the pricing ILP model since it has larger number of mapping possibilities (and larger number of links to schedule). For scheduling performance, the improvement depends on the forwarding tree combinations. The sooner a better combination occurs, the faster is the drop for the number of time slots for data gathering.

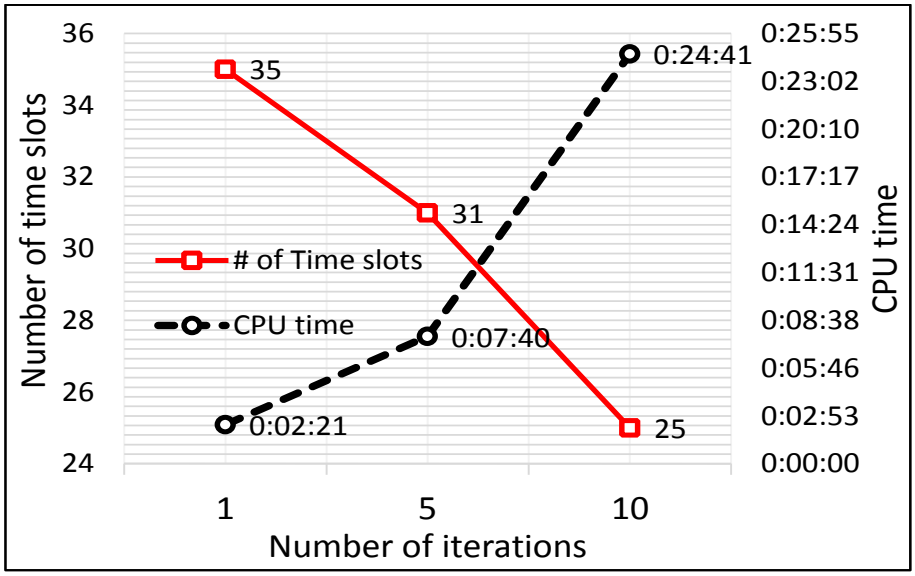


Figure 7.5: Number of time slots and CPU time Vs. number of iterations (n=30 nodes)

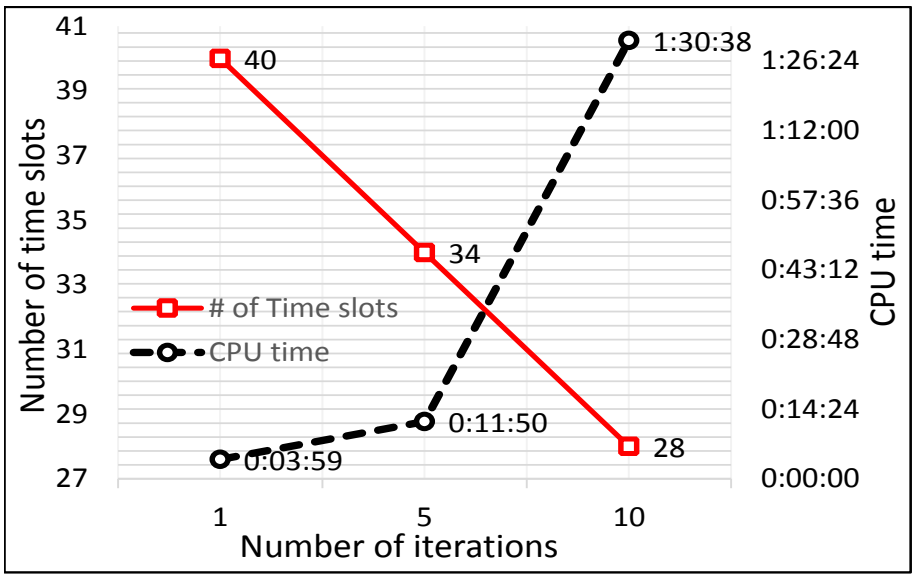


Figure 7.6: Number of time slots and CPU time Vs. number of iterations (n=35 nodes)

Performance of CG with diverse forwarding trees

As we mentioned in our solution methodology, different forwarding trees might improve the performance of the decomposition method (CG). Often, we observed, the combination of these diverse forwarding trees along with number of iterations improve the result of the CG. Table 7.4 depicts the results obtained from our decomposition method using two methods for constructing alternate trees. Namely, constructing forwarding trees with any size or/and constructing trees with minimum number of edges (i.e., minimum spanning trees (MSTs) only). In the table, starting from the initial basis, the number of time slots required for data gathering decreases as the number of iterations increases, whereas, the execution time increases as we explained earlier. When we consider the MST for our alternate trees construction, the running time of CG is significantly lower than when we consider any size forwarding trees. The reason is simply because fewer alternate trees are constructed. However, the scheduling performance is not always better when we use any size for alternate forwarding trees; as shown in the table for 40-node network at iteration 10 or 15, where the scheduling performance of MST (34 time slots) is slightly better than when we use any size tree (35 time slots). That is because, in this particular case, although the number of diverse forwarding trees is less, however, the combination of such trees have occurred in such a way that improved the scheduling performance.

Table 7.4: CG performance using any tree combinations and Minimum Spanning Tree (time is shown by h:m:s)

#Nodes	Basis	Iterations	1		2		3		4		5		10		15	
			#Slots	Time	#Slots	Time	#Slots	Time	#Slots	Time	#Slots	Time	#Slots	Time	#Slots	Time
n=10	**7**	Any tree	**7**	0:00:03	**7**	0:00:04	**7**	0:00:05	**7**	0:00:05	**7**	0:00:06	**7**	0:00:13	**6**	0:00:20
		MST	**7**	0:00:01	**7**	0:00:02	**7**	0:00:02	**7**	0:00:02	**7**	0:00:03	**7**	0:00:05	**7**	0:00:07
n=15	**17**	Any tree	**16**	0:00:11	**16**	0:00:16	**11**	0:00:21	**8**	0:00:27	**8**	0:00:34	**8**	0:01:12	**8**	0:01:52
		MST	**16**	0:00:06	**16**	0:00:08	**16**	0:00:10	**16**	0:00:13	**14**	0:00:16	**8**	0:00:32	**8**	0:00:49
n=20	**25**	Any tree	**23**	0:00:32	**17**	0:00:47	**17**	0:01:01	**17**	0:01:19	**17**	0:01:41	**14**	0:03:38	**14**	0:06:02
		MST	**23**	0:00:28	**17**	0:00:39	**17**	0:00:54	**17**	0:01:08	**17**	0:01:23	**14**	0:03:03	**14**	0:04:54
n=25	**25**	Any tree	**24**	0:00:55	**23**	0:01:21	**23**	0:01:49	**21**	0:02:27	**20**	0:02:59	**15**	0:06:25	**15**	0:11:46
		MST	**25**	0:00:44	**25**	0:01:04	**25**	0:01:25	**25**	0:01:51	**21**	0:02:19	**16**	0:04:56	**15**	0:08:20
n=30	**37**	Any tree	**35**	0:02:21	**31**	0:03:57	**31**	0:05:13	**31**	0:06:24	**31**	0:07:40	**25**	0:24:41	**25**	1:36:35
		MST	**34**	0:01:58	**34**	0:03:04	**34**	0:03:59	**34**	0:05:12	**34**	0:06:28	**30**	0:16:02	**30**	0:29:59
n=35	**40**	Any tree	**40**	0:03:59	**34**	0:05:50	**34**	0:07:36	**34**	0:09:35	**34**	0:11:50	**34**	1:30:38	**28**	7:04:18
		MST	**38**	0:02:31	**38**	0:03:37	**38**	0:04:48	**38**	0:06:20	**35**	0:07:44	**32**	0:16:59	**31**	0:43:47
n=40	**47**	Any tree	**40**	0:08:48	**39**	0:12:51	**39**	0:16:44	**35**	0:47:45	**35**	1:17:24	**35**	6:01:16	**35**	35:11:27
		MST	**40**	0:06:53	**39**	0:10:13	**39**	0:13:46	**39**	0:17:27	**39**	0:21:22	**34**	3:33:39	**34**	8:19:46

CG vs. a distributed method

To overcome the complexity of the joint FTCS problem, in our previous work [24] we designed a low complexity distributed method for solving it for large network sizes. Our distributed method constructs and schedule links completely in a distributed way. For solving a distributed scheduling problem, our method defines an interference neighborhood for each link, where a constant ϵ ($0 < \epsilon < 1$) controls the size (or radius) of the interference neighborhood. Links coordinate their transmissions through message exchanges. The smaller the value of ϵ, the larger the interference neighborhood and thus the higher the scheduling overhead. Indeed, while we were able to verify the performance of our distributed method on small instances (since we could obtain the ILP solution), it was not clear how our distributed method performed for larger instances, since we have no known ILP solutions for such instances. The purpose of this study is to bring a closure to answering this question; namely, our CG method will serve as a benchmark to compare the performance of the distributed solution of the FTCS problem. Figure 7.7 depicts the results of the comparisons where we used various setups for our distributed method: $\epsilon = 0.2$ and $\epsilon = 0.5$. Note that both methods constructed forwarding trees with same number of links. As shown in the figure, clearly, CG performs better than distributed method (service as a best lower bound); the distributed method with $\epsilon = 0.2$ shows close performance to CG with a maximum gap of 20%.

7.4 Conclusion

In this chapter, we studied the complex problem of constructing and scheduling multi-forwarding trees, each to aggregate data from a group of interest-nodes

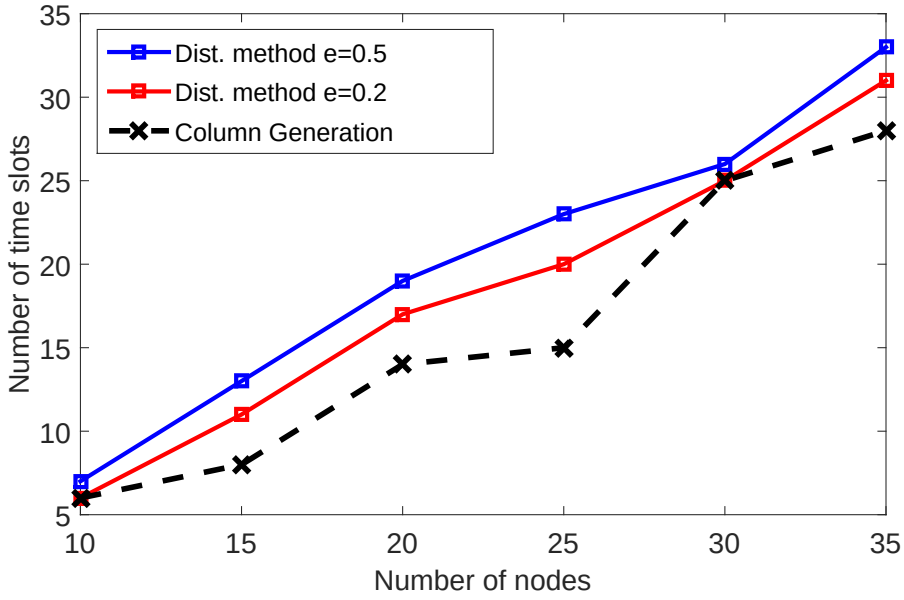


Figure 7.7: CG Vs. distributed method ($e = 0.2$ and $e = 0.5$)

to the sink in a wireless sensor network. We highlighted the complexity of the problem and proposed a primal-dual decomposition method using column generation to solve it. We investigated the performance of our proposed column generation through numerical analysis and the results showed that the problem can be solved for bigger networks with optimal or near optimal solutions. One major advantage of our primal-dual method is that it can serve as a benchmark for evaluating the performance of low complexity distributed (or other algorithmic) approaches for solving the joint FTCS problem, since no known exact solutions are available for larger networks.

Chapter 8

Conclusion and Future Work

8.1 Conclusions

This thesis addressed the problem of gathering data from wireless sensors, which are deployed randomly in a region, to a central unit (sink) in the most energy efficient manner to prolong the lifetime of wireless sensor networks (WSNs). Namely, we presented data gathering methods using Compressive Sensing technique in WSN, referred to as Compressive Data Gathering (CDG), to reduce the global communication cost and balance the energy load throughout the network without incurring intensive computation or transmission overhead. With CDG, rather than recieving all readings from sensors, the sink may receive few weighted/encoded sums of all the readings, from which the sink can recover the original data. Our work in this thesis highlighted the benefits of CDG and exposed the combinatorial complexity of such problems and therefore presented some methods which are very efficient and scalable to solve. Several CDG methods have been proposed in the literature, however the methods presented in this thesis outperform them since we used a novel sparse projection technique by building up in-network data aggregation tree for gathering each

weighted/encoded sum from a set of chosen sensors to the sink. Our problem aims at minimizing the number of links in the forwarding trees to minimize the number of overall transmissions in the network.

First, we proposed MSTP, a new method for data aggregation in large-scale WSN using compressive sensing and random projection. The proposed method selects random projection nodes to generate routing trees with each projection node gathering a weighted sum from all the nodes in the network whose coefficients in a random basis matrix Φ are non-zero, and in turn each projection node sends the received weighted sum to the sink. We also extended the method and presented eMSTP by joining the sink to each tree and letting the sink node gathers all the weighted sums. We showed that the time complexity of MSTP and eMSTP algorithms is $O(md\rho^2 \log \rho)$ in the best case and $O(md\rho^2 n \log n)$ in the worst case. We compared our methods with three different schemes (Non-CS, Plain-CDG and Hybrid-CDG) and the numerical results showed that our methods outperform those schemes with respect to network lifetime. Further, we showed that the performance of eMSTP depends on the selection of appropriate projection nodes, and we proposed OSPN method that finds the best projection nodes for the network. In addition, we presented PB-CDG algorithmic method which gathers the compressed data directly from sensors to the sink without relying on the projection nodes, and the mathematical optimization model has been derived for the problem. Moreover, we showed that our data gathering methods dramatically increase the lifetime of sensor networks.

Next, to overcome the communication drawback of the centralized methods, we proposed a decentralized approach (DCDG), where each sensor node in the network independently selects its parent node to whom it should send

the aggregated data. In this manner there is no requirement for a central unit to accomplish a topology discovery. We showed how DCDG can easily recover in case of node(s) failure, and further we analyzed the message overhead and derived the approximation bound of the presented method. Further, through simulations we showed that DCDG performs (in terms of transmission cost) very close to the best centralized methods but outperforms them in terms of communication overhead.

We explored the benefits of using Network Coding in projection-based compressive data gathering and proposed NC-CDG method. In this method, the existence of forwarding trees to gather compressed data from sensors to projection nodes creates opportunities for many-to-many communication patterns, which in turn gives rise to network coding operations; such operations if exploited will further reduce the number of transmissions needed to collect the sensory data. We mathematically formulated the NC-CDG problem which maximally exploits the coding opportunities on compressed data being routed on these forwarding trees. Owing to its computational complexity and to evaluate NC-CDG over larger networks, we developed both centralized and distributed algorithmic methods for solving the NC-CDG problem. Through simulations we showed that NC-CDG yields significant gains over methods that do not consider network coding.

Finally, we studied the problem of jointly constructing forwarding trees for projection-based compressive data gathering and scheduling (FTCS) under the real physical interference model. We formulated the problem as a mixed integer linear program through which we obtained optimal solutions for small size networks. We proved its NP-hardness, and then, we proposed a decentralized algorithm that can solve for large scale networks. The decentralized

approach decouples the problem into two subproblems; namely, the tree construction subproblem and the link scheduling subproblem. Further, We proved the correctness of the algorithmic method and analyzed its performance. Later, after highlighting the complexity of the FTCS problem, we presented a novel primal-dual decomposition method using column generation to solve it. We also highlighted several challenges we faced when solving the decomposition problem and presented efficient techniques for mitigating those problems. This primal-dual method can serve as a benchmark for evaluating the performance of low complexity algorithmic methods for solving the FTCS problem, where no known exact solutions can be found for larger network instances. At the end, through a large set of numerical results, we validated the efficiency and performance of our distributed FTCS method.

In general, using compressive data gathering has few disadvantages. First of all, as we mentioned in the background for CDG, this technique requires the original sensors' readings to be able to transform into a sparse domain, and without this condition, the CDG can not be used for data gathering in WSNs. Secondly, CDG requires both time and computation energy to do coding and decoding at sensors and the sink side respectively. In the NC-CDG method, additional time and energy are needed to do the in-network coding (XOR-adding) operation on top of coded data for CDG. In addition, sensors require more memory to store the matrix Φ which is needed for generating a weighted sum in CDG. However, we should note that matrix Φ may take less than 0.5 KB memory RAM, where wireless sensors have 1KB - 4MB onboard memory RAM [31]. Currently, a cheap and better storage flash memories are used in wireless sensors [27]. Therefore, owing to the development in hardware technologies, memory and data processing are not major obstacles for

data gathering in WSNs. A variety of different wireless sensor nodes along with their specifications is listed in [76].

8.2 Future Work

The work presented in this thesis provided considerable enhancements of data gathering (specifically, compressive data gathering) in wireless sensor networks. However, there remain several future research direction that may provide additional benefits.

In our network model, we assumed all sensors for simplicity have fixed and uniform transmission power regardless of their distances to neighbor nodes. Each sensor can communicate with nodes which reside within its communication radius. However, considering the power control ability of sensor nodes might significantly affect the construction of forwarding trees for our energy efficient compressive data gathering. Indeed, this power control ability makes our problem combinatorially much more complicated. Therefore, solving the optimization model of the problem becomes very challenging and hence alternate heuristic method needs to be derived to overcome the scalability of the model.

The projection-based compressive data gathering methods presented in this thesis apply compressive sensing technique on entire network which require a large number of sensors to participate in each compressed gathering even in the presence of sparse random projections. Moreover, for each projection, sensors might be located far from each other, which result in additional data transmissions and hence lead to waste lot of energy. To this end, one may divide the network into cells or clusters and apply compressive data gathering on each cluster and eventually gathers the aggregated data from the entire

cluster heads to the sink in the same scheme presented in this thesis. Accordingly, new challenges will arise, namely, how to obtain the optimal number of clusters, size of the cluster, etc.

In Chapter 5, for network-coding aware compressive data gathering problem we did not consider interference and time scheduling in our formulation, since in that chapter energy efficiency was the most important objective. Therefore, it will be interesting to study the impact of link scheduling on the construction of forwarding trees required for compressive data gathering with the presence of network coding. Furthermore, we have studied the problem of finding the best projection nodes without considering the NC in our thesis, where, with the presence of NC, this problem is considered for future work.

One of the important key issues in multi-hop data gathering in WSN is to balance the energy load throughout the network. Although, CDG is helpful in this context, however, in the methods presented in this thesis, we acknowledged that some nodes may be used more than others when they belong to multiple trees (or projections). Hence, a better metric for energy balance would be to minimize the maximum number of transmissions per node, which could be considered for future work.

Appendix A

Message Overhead Analysis for Centralized CDG

The computation analysis of message overhead of the example in Figure A.1 for centralized method is given through the following steps.

1. For discovery (refer to Figure A.1), $n = 6$ messages is required.

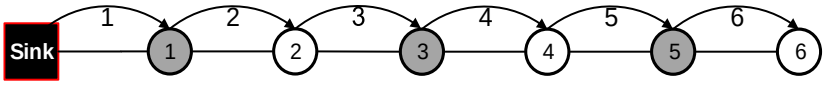


Figure A.1: Discovery message

2. For topology discovery, Node 6 sends one message to the sink (refer to Figure A.2), since it has only one neighbour node. Node 5 sends two messages to the sink (Figure A.3), since it has two neighbours. Nodes 4,3,2 send two messages each and node 1 sends one message to the sink. Therefore, in total for topology discovery, the graph needs $6+2(5)+2(4)+2(3)+2(2)+1 = 35$ messages. The overhead analysis for topology discovery of n nodes is given by (A.1).

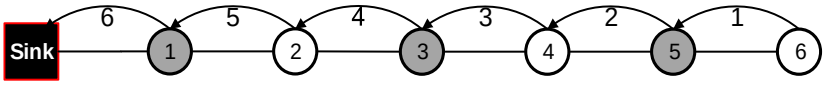


Figure A.2: Network Topology Discovery (Node 6).

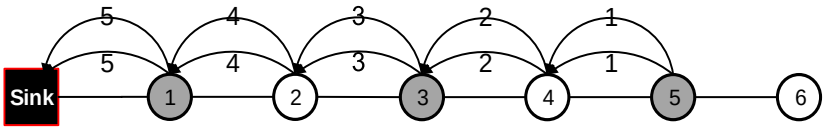


Figure A.3: Network Topology Discovery (Node 5).

$$
\begin{aligned}
& n + 2\sum_{2}^{n-1} + 1 \\
& = n + (n - 1 + 1 - 2)(n - 1 + 2) + 1 \\
& = n + (n - 2)(n + 1) + 1 \\
& = n + n^2 + n - 2n - 2 + 1 \\
& = n^2 - 1
\end{aligned}
\tag{A.1}
$$

After running the centralized algorithm at the sink, two messages are required to be sent to each node to notify it of its parent and child nodes. The message overhead for the example of linear network is as follows; a) For tree 1 (dark nodes), $2 \times (1+3+5)$ messages is required for nodes 1,3 and 5; $2 \times (2+4)$ messages needed for relay nodes 2 and 4. In total 30 messages required for tree 1. b) For tree 2 (white nodes), $2 \times (2+4+6)$ messages is needed for nodes 2,4 and 6; $2 \times (1+3+5)$ messages required for relay nodes 1,3 and 5. In total 42 messages is needed for tree 2. In total 30+42=72 messages are needed to notifying the nodes for forwarding trees. The overhead analysis for forwarding

tree notification is given by (A.2).

$$
\begin{aligned}
&2\sum_{x=1}^{m}\sum_{y=1}^{n+1-x} y = \sum_{x=1}^{m}(n+1-x)(n+2-x) \\
&= \sum_{x=1}^{m}(n^2+3n-2nx-3x+2+x^2) \\
&= (n^2+3n+2)\sum_{x=1}^{m}1-(2n+3)\sum_{x=1}^{m}x+\sum_{x=1}^{m}x^2 \\
&= (n^2+3n+2)(m)-(2n+3)\frac{(m^2+m)}{2}+\frac{m^3}{3}+\frac{m^2}{2}+\frac{m}{6} \\
&= mn^2-m^2n+2mn+\frac{m^3}{3}-m^2+\frac{2m}{3}
\end{aligned}
\tag{A.2}
$$

Therefore, the number of message overhead in the linear network of centralized method is 6+72+35=113 and the complete overhead analysis for centralized method is given in (A.3) by adding discovery message n to the equations (A.1) and (A.2).

$$
\begin{aligned}
&Total\_Centralized\_Message\_Overhead = \\
&n+n^2-1+mn^2-m^2n+2mn+\frac{m^3}{3}-m^2+\frac{2m}{3}
\end{aligned}
\tag{A.3}
$$

Appendix B

Message Overhead Analysis for Decentralized CDG

The computation analysis of message overhead of the example in Figure A.1 for decentralized method is given through the following steps.

1. For discovery, we require n messages. In the example of Figure 4.7, $n = 6$.

2. Node 1 needs only one message to notify the sink.

3. Node 2 requires two message to notify the sink, since it does not have interest parent neighbour and its $h_2 - 1 = 1$. Therefore, node 2 sets the sink as its parent interest.

   The other nodes in the network to find their parent interest-node require the following messages:

   a) To find interest-nodes in radius $h_i - 1$, Node 3 requires 3 messages as shown in Figure B.1. Node 4 requires 4 messages as shown in Figure B.2. Node 5 requires 4 messages as shown in Figure B.3 and node 6 requires 5 messages.

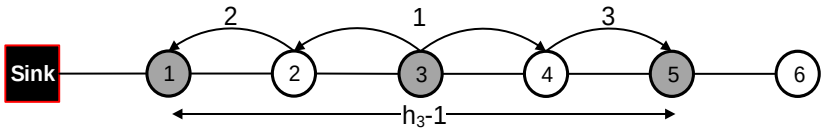


Figure B.1: Message overhead to find interest-nodes in radius $h_3 - 1$.

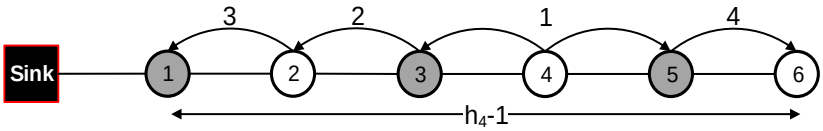


Figure B.2: Message overhead to find interest-nodes in radius $h_4 - 1$.

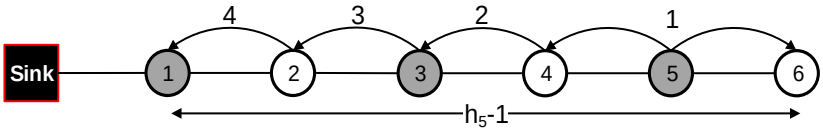


Figure B.3: Message overhead to find interest-nodes in radius $h_5 - 1$.

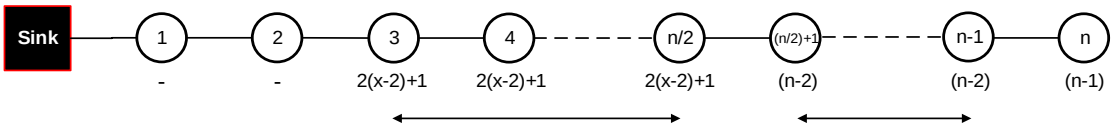


Figure B.4: Message overhead analysis to find interest-nodes in radius $h_i - 1$.

As it is clear from the example and Figure B.4, nodes from node 3 to node $\frac{n}{2}$ require $2(x-2)+1$ message overhead each, where x is the hop-distance to the sink (shown by node-ID). The total number of nodes in this range is $\lceil \frac{n}{2} \rceil - 2$. Nodes from node $\frac{n}{2}+1$ to node $n-1$ require $(n-2)$ message overhead each. The total number of nodes in this range is $\lfloor \frac{n}{2} \rfloor - 1$. The last node (node n) needs $(n-1)$ message overhead to discover its $h_n - 1$ range. The overhead analysis is given in (B.1).

$$\begin{aligned} &\sum_{x=3}^{\lceil \frac{n}{2} \rceil} (2(x-2)+1) + (n-2)(\lfloor \frac{n}{2} \rfloor - 1) + (n-1) \\ &= \sum_{x=3}^{\lceil \frac{n}{2} \rceil} (2x-3) + (n-2)(\lfloor \frac{n}{2} \rfloor - 1) + (n-1) \end{aligned} \tag{B.1}$$

b) To get information from interest-nodes in radius $h_i - 1$, node 3 requires 4 messages as it is shown in Figure B.5. Node 4 requires 4 messages (refer to Figure B.6). Node 5 requires 6 messages (Figure B.7) and as the Figure B.8 shows, node 6 requires 6 messages.

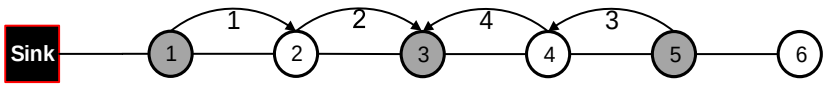


Figure B.5: Message overhead to get information from interest-nodes in radius $h_3 - 1$.

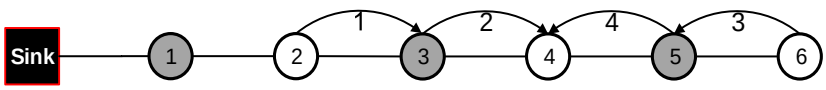


Figure B.6: Message overhead to get information from interest-nodes in radius $h_4 - 1$.

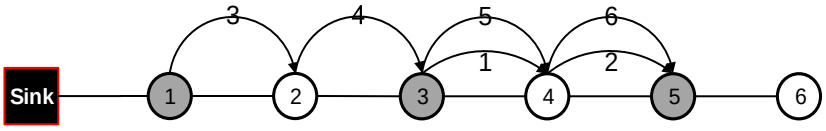


Figure B.7: Message overhead to get information from interest-nodes in radius $h_5 - 1$.

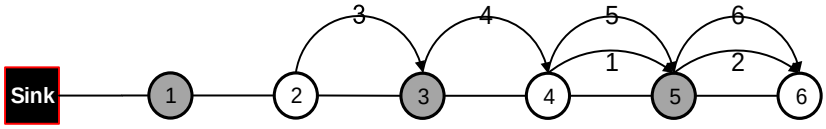


Figure B.8: Message overhead to get information from interest-nodes in radius $h_6 - 1$.

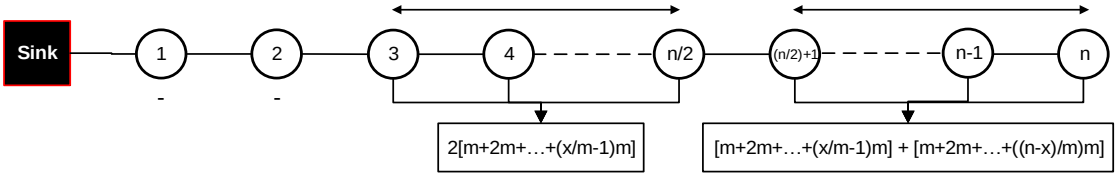


Figure B.9: Message overhead analysis to get information from interest-nodes in radius $h_i - 1$.

By observing Figure B.9, node 3 to $\frac{n}{2}$ receive equal number of messages from their interest-nodes at both sides (left and right). The distance between each two interest-nodes is m hops. Thus, the number of messages needed for a node x to receive messages from interest-nodes at both sides is $2(m + 2m + ... + (\lceil \frac{x}{2} \rceil)m)$, where node x has $\lceil \frac{x}{m} \rceil - 1$ interest-nodes at one side. Therefore, the summation of message overhead for these nodes is given by $2 \sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym$. Similarly, node $\frac{n}{2} + 1$ to last node n, each requires $m + 2m + ... + (\lceil \frac{x}{m} \rceil - 1)m$ messages for its left and $m + 2m + ... + (\lfloor \frac{n-x}{m} \rfloor)m$ messages for its right side, since their $h_i - 1$ range at right side is limited to the last node n in the graph. Therefore, the total message overhead for these nodes is $\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym + \sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \sum_{y=1}^{\lfloor \frac{n-x}{m} \rfloor} ym$. The total message

overhead for collecting information is given in (B.2).

$$2\sum_{x=3}^{\lceil \frac{n}{2} \rceil}\sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym + \sum_{\lceil \frac{n}{2} \rceil + 1}^{n}\sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym + \sum_{\lceil \frac{n}{2} \rceil + 1}^{n}\sum_{y=1}^{\lfloor \frac{n-x}{m} \rfloor} ym \tag{B.2}$$

c) To notify the chosen parent, node 3,4,5 and 6 in example shown in Fig. 4.7 requires two messages each. Each node sends a notification message to its nearest interest-node and since the nearest interest-node is m hop away from the current node, it takes only m message overhead. Therefore, the total message overhead for parent notification is given in (B.3).

$$m(n-2) \tag{B.3}$$

Now, the total decentralized message overhead for given example is $6+1+2+16+20+8=53$ compared to 113 messages in centralized method. The total decentralized message overhead for n nodes and m projections is obtained by adding the discovery message overhead n and $1+2$ messages for node 1 and 2 to the equations (B.1), (B.2) and (B.3). The total overhead is presented in (B.4).

$$\begin{aligned} & n+1+2 \\ & + \sum_{x=3}^{\lceil \frac{n}{2} \rceil}(2x-3) + (n-2)(\lfloor \frac{n}{2} \rfloor - 1) + (n-1) \\ & + m(n-2) \\ & + 2\sum_{x=3}^{\lceil \frac{n}{2} \rceil}\sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym + \sum_{\lceil \frac{n}{2} \rceil + 1}^{n}\sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym + \sum_{\lceil \frac{n}{2} \rceil + 1}^{n}\sum_{y=1}^{\lfloor \frac{n-x}{m} \rfloor} ym \end{aligned} \tag{B.4}$$

We may simplify the equation $(n-2)(\lfloor \frac{n}{2} \rfloor - 1)+(n-1)$ as in (B.5) and the summations $\sum_{x=3}^{\lceil \frac{n}{2} \rceil}(2x-3)$, $2\sum_{x=3}^{\lceil \frac{n}{2} \rceil}\sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym$, $\sum_{\lceil \frac{n}{2} \rceil + 1}^{n}\sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym$ and $\sum_{\lceil \frac{n}{2} \rceil + 1}^{n}\sum_{y=1}^{\lfloor \frac{n-x}{m} \rfloor} ym$ as in (B.6), (B.7), (B.8) and (B.9) respectively.

Lemma B.0.1.

$$(n-2)(\lfloor\frac{n}{2}\rfloor - 1) <= \frac{n^2}{2} - 2n + 2 \tag{B.5}$$

Proof.

$(n-2)(\lfloor\frac{n}{2}\rfloor - 1) <= (n-2)(\frac{n}{2} + 1 - 1)$

$= (n-2)(\frac{n}{2})$

$= \frac{n^2}{2} - n - n + 2\frac{n^2}{2} - 2n + 2$ □

Lemma B.0.2.

$$\sum_{x=3}^{\lceil\frac{n}{2}\rceil}(2x-3) <= \frac{n^2}{4} - 1 \tag{B.6}$$

Proof.

$\sum_{x=3}^{\lceil\frac{n}{2}\rceil}(2x-3) = 2\sum_{x=3}^{\lceil\frac{n}{2}\rceil} x - 3\sum_{x=3}^{\lceil\frac{n}{2}\rceil} 1$

$= 2\frac{(\lceil\frac{n}{2}\rceil+1-3)(\lceil\frac{n}{2}\rceil+3)}{2} - 3(\lceil\frac{n}{2}\rceil + 1 - 3)$

$= (\lceil\frac{n}{2}\rceil - 2)(\lceil\frac{n}{2}\rceil + 3 - 3)$

$= (\lceil\frac{n}{2}\rceil - 2)\lceil\frac{n}{2}\rceil$

$<= (\frac{n}{2} - 1)(\frac{n}{2} - 1)$

$= \frac{n^2}{4} - 1$ □

Lemma B.0.3.

$$2\sum_{x=3}^{\lceil\frac{n}{2}\rceil}\sum_{y=1}^{\lceil\frac{x}{m}\rceil-1} ym <= \frac{n^3}{24m} + \frac{3n^2}{8m} + \frac{n^2}{8} + \frac{13n}{m} + \frac{nm}{4} + \frac{3n}{4} - \frac{4}{m} - \frac{m}{2} - 2 \tag{B.7}$$

Proof.

$2\sum_{x=3}^{\lceil\frac{n}{2}\rceil}\sum_{y=1}^{\lceil\frac{x}{m}\rceil-1} ym = 2m\sum_{x=3}^{\lceil\frac{n}{2}\rceil}(\frac{(\lceil\frac{x}{m}\rceil-1+1)(\lceil\frac{x}{m}\rceil-1+1-1)}{2})$

$= m\sum_{x=3}^{\lceil\frac{n}{2}\rceil}\lceil\frac{x}{m}\rceil^2 - m\sum_{x=3}^{\lceil\frac{n}{2}\rceil}\lceil\frac{x}{m}\rceil$ ——(a)

$\sum_{x=3}^{\lceil \frac{n}{2} \rceil} \lceil \frac{x}{m} \rceil <= \sum_{x=3}^{\frac{n}{2}+1} \frac{x}{m} + 1$

$= \frac{1}{m} \sum_{x=3}^{\frac{n}{2}+1} x + \sum_{x=3}^{\frac{n}{2}+1} 1$

$= \frac{1}{2m}(\frac{n}{2} + 1 + 1 - 3)(\frac{n}{2} + 1 + 3) + \frac{1}{2}(\frac{n}{2} + 1 + 1 - 3)$

$= \frac{1}{2m}(\frac{n}{2} - 1)(\frac{n}{2} + 4) + \frac{1}{2}(\frac{n}{2} - 1)$

$= \frac{n^2}{8m} + \frac{3n}{4m} + \frac{n}{4} - \frac{2}{m} - \frac{1}{2}$ ——(b)

$\sum_{x=3}^{\lceil \frac{n}{2} \rceil} \lceil \frac{x}{m} \rceil^2 <= \sum_{x=3}^{\frac{n}{2}+1} (\frac{x}{m} + 1)^2$

$= \sum_{x=3}^{\frac{n}{2}+1} \frac{x^2}{m^2} + \sum_{x=3}^{\frac{n}{2}+1} \frac{2x}{m} + \sum_{x=3}^{\frac{n}{2}+1} 1$

$= \frac{1}{m^2}(\sum_{x=3}^{\frac{n}{2}+1} x^2 - \sum_{x=1}^{2} x^2) + \frac{2}{m} \sum_{x=3}^{\frac{n}{2}+1} x + \sum_{x=3}^{\frac{n}{2}+1} 1$

$= \frac{1}{m^2}(\frac{(\frac{n}{2}+1)^3}{3} + \frac{(\frac{n}{2}+1)^2}{2} + \frac{(\frac{n}{2}+1)}{6} - \frac{2^3}{3} - \frac{2^2}{2} - \frac{2}{6}) + \frac{2}{3m}\frac{(\frac{n^2}{4}+2n-\frac{n}{2}-4)}{2} + \frac{n}{2} - 1$

$= \frac{n^3}{24m^2} + \frac{3n^2}{8m^2} + \frac{n^2}{4m} + \frac{13n}{12m^2} + \frac{3n}{2m} + \frac{n}{2} - \frac{4}{m^2} - \frac{4}{m} - 1$ ——(c)

(b) and (c) in (a)

$2\sum_{x=3}^{\lceil \frac{n}{2} \rceil} \sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym <=$

$\frac{n^3}{24m} + \frac{3n^2}{8m} + \frac{n^2}{4} + \frac{13n}{12m} + \frac{3n}{2} + \frac{nm}{2} + -\frac{4}{m} - 4 - m - \frac{n^2}{8} - \frac{3n}{4} - \frac{nm}{4} + 2 + \frac{m}{2}$

$= \frac{n^3}{24m} + \frac{3n^2}{8m} + \frac{n^2}{8} + \frac{13n}{m} + \frac{nm}{4} + \frac{3n}{4} - \frac{4}{m} - \frac{m}{2} - 2$ □

Lemma B.0.4.

$$\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym <= \frac{7n^3}{48m} + \frac{3n^2}{16m} + \frac{3n^2}{16} + \frac{nm}{24} + \frac{n}{8} \tag{B.8}$$

Proof.

$\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym = m \sum_{\lceil \frac{n}{2} \rceil + 1}^{n} (\frac{(\lceil \frac{x}{m} \rceil - 1 + 1 - 1)(\lceil \frac{x}{m} \rceil - 1 + 1)}{2})$

$= \frac{m}{2} \sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \lceil \frac{x}{m} \rceil^2 - \frac{m}{2} \sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \lceil \frac{x}{m} \rceil$ ——(a)

$\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \lceil \frac{x}{m} \rceil <= \sum_{\frac{n}{2}+1}^{n} (\frac{x}{m} + 1)$

$= \frac{1}{m} \sum_{\frac{n}{2}+1}^{n} x + \sum_{\frac{n}{2}+1}^{n} 1$

$= \frac{1}{2m}(n + 1 - \frac{n}{2} - 1)(n + \frac{n}{2} + 1) + n + 1 - \frac{n}{2} - 1$

$= \frac{1}{2m}(\frac{n}{2})(\frac{3n}{2}+1) + \frac{n}{2}$

$= \frac{1}{2m}(\frac{3n^2}{4} + \frac{n}{2}) + \frac{n}{2}$

$= \frac{3n^2}{8m} + \frac{n}{4m} + \frac{n}{2}$ ——(b)

$\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \lceil \frac{x}{m} \rceil^2 <= \sum_{\frac{n}{2}+1}^{n} (\frac{x}{m}+1)^2$

$= \frac{1}{m^2}\sum_{\frac{n}{2}+1}^{n} x^2 + \frac{2}{m}\sum_{\frac{n}{2}+1}^{n} x + \sum_{\frac{n}{2}+1}^{n} 1$

$= \frac{1}{m^2}(\sum_{1}^{n} x^2 - \sum_{1}^{\frac{n}{2}} x^2) + \frac{2}{m}\sum_{\frac{n}{2}+1}^{n} x + \sum_{\frac{n}{2}+1}^{n} 1$

$= \frac{1}{m^2}(\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} - \frac{(\frac{n}{2})^3}{3} - \frac{(\frac{n}{2})^2}{2} - \frac{\frac{n}{2}}{6}) + \frac{2}{m}(\frac{(n+1-\frac{n}{2}-1)(n+\frac{n}{2}+1)}{2}) + n + 1 - \frac{n}{2} - 1$

$= \frac{1}{m^2}(\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} - \frac{n^3}{24} - \frac{n^2}{8} - \frac{n}{12}) + \frac{2}{2m}(\frac{n}{2})(\frac{3n}{2}+1) + \frac{n}{2}$

$= \frac{7n^3}{24m^2} + \frac{3n^2}{8m^2} + \frac{n}{12} + \frac{3n^2}{4m} + \frac{n}{2m} + \frac{n}{2}$

$= \frac{7n^3}{24m^2} + \frac{3n^2}{8m^2} + \frac{3n^2}{4m} + \frac{n}{2m} + \frac{7n}{12}$ ——(c)

(b) and (c) in (a)

$\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \sum_{y=1}^{\lceil \frac{x}{m} \rceil - 1} ym <= \frac{7n^3}{48m} + \frac{3n^2}{16m} + \frac{3n^2}{8} + \frac{n}{4} + \frac{7nm}{24} - \frac{3n^2}{16} - \frac{n}{8} - \frac{nm}{4}$

$= \frac{7n^3}{48m} + \frac{3n^2}{16m} + \frac{3n^2}{16} + \frac{nm}{24} + \frac{n}{8}$ □

Lemma B.0.5.

$$\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \sum_{y=1}^{\lfloor \frac{n-x}{m} \rfloor} ym <= \frac{n^3}{48m} - \frac{n^2}{16m} + \frac{n^2}{16} + \frac{n}{24m} - \frac{n}{8} \tag{B.9}$$

Proof.

$\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \sum_{y=1}^{\lfloor \frac{n-x}{m} \rfloor} ym = \frac{m}{2}\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \lfloor \frac{n-x}{m} \rfloor (\lfloor \frac{n-x}{m} \rfloor + 1)$

$= \frac{m}{2}\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \lfloor \frac{n-x}{m} \rfloor^2 + \frac{m}{2}\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \lfloor \frac{n-x}{m} \rfloor$ ——(a)

$\sum_{\lceil \frac{n}{2} \rceil + 1}^{n} \lfloor \frac{n-x}{m} \rfloor <= \sum_{\frac{n}{2}+1}^{n} (\frac{n-x}{m})$

$= \frac{n}{m}\sum_{\frac{n}{2}+1}^{n} 1 - \frac{1}{m}\sum_{\frac{n}{2}+1}^{n} x$

$= \frac{n}{m}(n + 1 - \frac{n}{2} - 1) - \frac{1}{2m}(n + 1 - \frac{n}{2} - 1)(n + \frac{n}{2} + 1)$

$= \frac{n}{m}(\frac{n}{2}) - \frac{1}{2m}(\frac{n}{2})(\frac{3n}{2}+1)$

$= \frac{n^2}{2m} - \frac{3n^2}{8m} - \frac{n}{4m}$

$= \frac{n^2}{8m} - \frac{n}{4m}$ ——(b)

$\sum_{\lceil \frac{n}{2} \rceil+1}^{n} \lfloor \frac{n-x}{m} \rfloor^2 <= \sum_{\frac{n}{2}+1}^{n} (\frac{n-x}{m})^2$

$= \frac{n^2}{m^2} \sum_{\frac{n}{2}+1}^{n} 1 - \frac{2n}{m^2} \sum_{\frac{n}{2}+1}^{n} x + \frac{1}{m^2} \sum_{\frac{n}{2}+1}^{n} x^2$

$= \frac{n^2}{m^2}(n+1-\frac{n}{2}-1) - \frac{n}{m^2}(n+1-\frac{n}{2}-1)(n+\frac{n}{2}+1) + \frac{1}{m^2}(\sum_1^n x^2 - \sum_1^{\frac{n}{2}} x^2)$

$= \frac{n^2}{m^2}(\frac{n}{2}) - \frac{n}{m^2}(\frac{n}{2})(\frac{3n}{2}+1) + \frac{1}{m^2}(\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} - \frac{n^3}{24} - \frac{n^2}{8} - \frac{n}{12})$

$= \frac{7n^3}{2m^2} - \frac{3n^3}{4m^2} - \frac{n^2}{2m^2} + \frac{7n^3}{24m^2} + \frac{3n^2}{8m^2} + \frac{n}{12m^2}$

$= \frac{n^3}{24m^2} - \frac{n^2}{8m^2} + \frac{n}{12m^2}$ ——(c)

(b) and (c) in (a)

$\sum_{\lceil \frac{n}{2} \rceil+1}^{n} \sum_{y=1}^{\lfloor \frac{n-x}{m} \rfloor} ym <= \frac{n^3}{48m} - \frac{n^2}{16m} + \frac{n^2}{16} + \frac{n}{24m} - \frac{n}{8}$ □

Therefore, the roughly total message overhead in decentralized algorithm is given by (B.10).

$$Total\_Decentralized\_Message\_Overhead <= \frac{5n^3}{24m} + \frac{n^2}{2m} + \frac{9n^2}{8} + \frac{31mn}{24} + \frac{313n}{24m} - \frac{3n}{4} - \frac{5m}{2} - \frac{4}{m} + 1 \quad \text{(B.10)}$$

Appendix C

TCM (Tree Construction Model)

• **Parameters:**

The parameters are listed in Table 7.2.

• **Decision Variables:**

$f_{ij} \in \mathbb{Z}$: Flow on link (i,j).

x_{ij} = $\begin{cases} 1, & \text{if link } (i,j) \text{ is active in a tree;} \\ 0, & \text{otherwise.} \end{cases}$

• **Mathematical Model:**

$$Minimize \qquad \sum_{(i,j)\in E} x_{ij} \tag{C.1}$$

Subject to:

$$\sum_{j:(i,j)\in E} f_{ij} - \sum_{j:(j,i)\in E} f_{ji} = \begin{cases} - \parallel I_t \parallel, & i = sink; \\ 1, & \forall i \in I_t; \\ 0, & otherwise. \end{cases} \tag{C.2}$$

$$\begin{cases} x_{ij} \leq f_{ij} \\ x_{ij} \geq \frac{f_{ij}}{B} \end{cases} \qquad \forall (i,j) \in E. \tag{C.3}$$

$$\sum_{j:(i,j)\in E} x_{ij} \leq 1 \qquad \forall i \in V. \tag{C.4}$$

The objective of the TCM problem is to construct a forwarding tree that connects a set of interest-nodes (I_t) in the network to a root (i.e., sink) with minimum edges (thus reducing the number of transmissions and hence energy efficiency). Constraints (C.2) are the flow conservation for routing across the network, which force the set of interest-nodes (vector set I_t) to have one data flow from each interest-node to the sink. Constraints (C.3) identify forwarding links for the tree. When there is a positive traffic flow on link (i,j) (i.e, $f_{ij} > 0$), this link is assigned for the tree (i.e, $x_{ij} = 1$). In other words, it implies that $f_{ij} = 0 \Leftrightarrow x_{ij} = 0$ and $f_{ij} > 0 \Leftrightarrow x_{ij} = 1$. Constraint (C.4) asserts that each node can have a maximum of one outgoing transmission (link) in each tree to avoid loops. Otherwise, data is not aggregated to a root (sink).

Appendix D

LSM (Link Scheduling Model)

• **Parameters:**

x_{ij}: Indicate whether link (i,j) belongs to a tree.

The remaining parameters are listed in Table 7.2.

• **Decision Variables:**

Same variables as in section 7.2.2 (pricing problem).

• **Mathematical Model:**

$$Minimize \qquad \lambda \tag{D.1}$$

Subject to: (7.17) - (7.24).

The LSM problem schedules transmission for each link in a constructed tree while it is not violating 1) the order of transmissions and 2) the interference constraints for transmissions to be successful. The objective of the model is to minimize the scheduling length (i.e., λ). Here, the parameter x_{ij} (variable in TCM) indicates the constructed tree which is required to be scheduled in LSM problem. Constraints are exactly same as the pricing sub problem; for details refer to Section 7.2.2.

Bibliography

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[2] M. Alnuaimi, K. Shuaib, K. Alnuaimi, and M. Abed-Hafez. Data gathering in wireless sensor networks with ferry nodes. In *2015 IEEE 12th International Conference on Networking, Sensing and Control (ICNSC)*, pages 221–225. IEEE, 2015.

[3] M. Andrews and M. Dinitz. Maximizing capacity in arbitrary wireless networks in the sinr model: Complexity and game theory. In *IEEE INFOCOM 2009*, pages 1332–1340. IEEE, 2009.

[4] W. Bajwa, J. Haupt, A. Sayeed, and R. Nowak. Compressive wireless sensing. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 134–142. ACM, 2006.

[5] K. C. Barr and K. Asanović. Energy-aware lossless data compression. *ACM Transactions on Computer Systems (TOCS)*, 24(3):250–291, 2006.

[6] M. Borghini, F. Cuomo, T. Melodia, U. Monaco, and F. Ricciato. Optimal data delivery in wireless sensor networks in the energy and latency domains. In *Proceedings of the First International Conference on Wireless Internet*, pages 138–145. IEEE, 2005.

[7] C. Caione, D. Brunelli, and L. Benini. Compressive sensing optimization over zigbee networks. In *2010 International Symposium on Industrial Embedded Systems (SIES)*, pages 36–44. IEEE, 2010.

[8] E. J. Candè and M. B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008.

[9] G. Cao, F. Yu, and B. Zhang. Improving network lifetime for wireless sensor network using compressive sensing. In *2011 IEEE 13th International Conference on High Performance Computing and Communications (HPCC)*, pages 448–454. IEEE, 2011.

[10] M. Cardei, J. Wu, M. Lu, and M. O. Pervaiz. Maximum network lifetime in wireless sensor networks with adjustable sensing ranges. In *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications, 2005 (WiMob'2005).*, volume 3, pages 438–445. IEEE, 2005.

[11] S. Chen and Y. Wang. Data collection capacity of random-deployed wireless sensor networks under physical models. *Tsinghua Science and Technology*, 17(5):487–498, 2012.

[12] Z. Chen, G. Yang, L. Chen, and J. Wang. An algorithm for data aggregation scheduling with long-lifetime and low-latency in wireless sensor networks. *International Journal of Future Generation Communication and Networking*, 5(4):141–152, 2012.

[13] J.-H. Chou, D. Petrovic, and K. Ramachandran. A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference*

of the IEEE Computer and Communications. IEEE Societies, volume 2, pages 1054–1062. IEEE, 2003.

[14] V. Chvatal. *Linear programming*. Macmillan, 1983.

[15] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[16] R. Cristescu, B. Beferull-Lozano, and M. Vetterli. On network correlated data gathering. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2571–2582. IEEE, 2004.

[17] R. Cristescu, B. Beferull-Lozano, M. Vetterli, and R. Wattenhofer. Network correlated data gathering with explicit communication: NP-completeness and algorithms. *IEEE/ACM Transactions on Networking*, 14(1):41–54, 2006.

[18] F. Cuomo, A. Abbagnale, and E. Cipollone. Cross-layer network formation for energy-efficient ieee 802.15. 4/zigbee wireless sensor networks. *Ad Hoc Networks*, 11(2):672–686, 2013.

[19] M. A. Davenport, M. F. Duarte, Y. C. Eldar, and G. Kutyniok. Introduction to compressed sensing. *Preprint*, 93(1):2, 2011.

[20] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[21] D. Ebrahimi and C. Assi. Optimal and efficient algorithms for projection-based compressive data gathering. *IEEE Communications Letters*, 17(8):1572–1575, 2013.

[22] D. Ebrahimi and C. Assi. Compressive data gathering using random projection for energy efficient wireless sensor networks. *Ad Hoc Networks*, 16:105–119, 2014.

[23] D. Ebrahimi and C. Assi. A distributed method for compressive data gathering in wireless sensor networks. *IEEE Communications Letters*, 18(4):624–627, 2014.

[24] D. Ebrahimi and C. Assi. Joint compressive data gathering and scheduling in wireless sensor networks under the physical interference model. In *2015 IEEE 16th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–9. IEEE, 2015.

[25] M. Enachescu, A. Goel, R. Govindan, and R. Motwani. Scale free aggregation in sensor networks. In *Algorithmic Aspects of Wireless Sensor Networks*, pages 71–84. Springer, 2004.

[26] J. Gao, L. Guibas, N. Milosavljevic, and J. Hershberger. Sparse data aggregation in sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 430–439. ACM, 2007.

[27] K. K. Gautam, N. K. Gautam, and P. Agrawal. Memory required for wireless sensor nodes on the basis of characteristics and behaviour when using TinyOS. 4(1):26–34, 2014.

[28] A. Giridhar and P. Kumar. Computing and communicating functions over sensor networks. *IEEE Journal on Selected Areas in Communications*, 23(4):755–764, 2005.

[29] D. Gong and Y. Yang. Low-latency sinr-based data gathering in wireless sensor networks. *IEEE Transactions on Wireless Communications*, 13(6):3207–3221, 2014.

[30] O. Goussevskaia, Y. A. Oswald, and R. Wattenhofer. Complexity in geometric sinr. pages 100–109. ACM MobiHoc, 2007.

[31] D. K. Gupta. A review on wireless sensor networks. *Network and Complex Systems*, 3(1):18–23, 2013.

[32] H. Gupta, V. Navda, S. Das, and V. Chowdhary. Efficient gathering of correlated data in sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4(1):4:1–4:31, 2008.

[33] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[34] S. Hariharan and N. B. Shroff. Maximizing aggregated information in sensor networks under deadline constraints. *IEEE Transactions on Automatic Control*, 56(10):2369–2380, 2011.

[35] J. Haupt, W. U. Bajwa, M. Rabbat, and R. Nowak. Compressed sensing for networked data. *IEEE Signal Processing Magazine*, 25(2):92–101, 2008.

[36] S. He, J. Chen, D. K. Yau, and Y. Sun. Cross-layer optimization of correlated data gathering in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 11(11):1678–1691, 2012.

[37] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 10–pp. IEEE, 2000.

[38] T.-H. Hsu and P.-Y. Yen. Adaptive time division multiple access-based medium access control protocol for energy conserving and data transmission in wireless sensor networks. *IET Communications*, 5(18):2662–2672, 2011.

[39] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*. Elsevier, 1992.

[40] O. D. Incel and B. Krishnamachari. Enhancing the data collection rate of tree-based aggregation in wireless sensor networks. In *SECON'08. 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 569–577. IEEE, 2008.

[41] B. Jang, J. B. Lim, and M. L. Sichitiu. An asynchronous scheduled mac protocol for wireless sensor networks. *Computer Networks*, 57(1):85–98, 2013.

[42] S. Ji, R. Beyah, and Z. Cai. Snapshot and continuous data collection in probabilistic wireless sensor networks. *IEEE Transactions on Mobile Computing*, 13(3):626–637, 2014.

[43] S. Ji and Z. Cai. Distributed data collection in large-scale asynchronous wireless sensor networks under the generalized physical interference model. *IEEE/ACM Transactions on Networking (ToN)*, 21(4):1270–1283, 2013.

[44] C. Joo, J.-G. Choi, and N. B. Shroff. Delay performance of scheduling with data aggregation in wireless sensor networks. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[45] S. Katti, S. Gollakota, and D. Katabi. Embracing wireless interference: analog network coding. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 397–408. ACM, 2007.

[46] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard. Symbol-level network coding for wireless mesh networks. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 401–412. ACM, 2008.

[47] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. Xors in the air: practical wireless network coding. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 243–254. ACM, 2006.

[48] L. Keller, E. Atsan, K. Argyraki, and C. Fragouli. Sensecode: Network coding for reliable sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 9(2):25:1–25:20, 2013.

[49] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta informatica*, 15(2):141–145, 1981.

[50] T.-W. Kuo and M.-J. Tsai. On the construction of data aggregation tree with minimum energy cost in wireless sensor networks: NP-completeness and approximation algorithms. In *2012 Proceedings IEEE INFOCOM*, pages 2591–2595. IEEE, 2012.

[51] L. B. Le, E. Modiano, C. Joo, and N. B. Shroff. Longest-queue-first scheduling under sinr interference model. In *Proceedings of the eleventh ACM international symposium on Mobile ad hoc networking and computing*, pages 41–50. ACM, 2010.

[52] S. Lee, S. Pattem, M. Sathiamoorthy, B. Krishnamachari, and A. Ortega. Compressed sensing and routing in multi-hop networks. *University of Southern California CENG Technical Report*, 2009.

[53] H. Li, Q. S. Hua, C. Wu, and F. C. M. Lau. Minimum-latency aggregation scheduling in wireless sensor networks under physical interference model. In *Proceedings of the 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems*, pages 360–367. ACM, 2010.

[54] J. Li, A. Deshpande, and S. Khuller. On computing compression trees for data collection in wireless sensor networks. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[55] S. Lindsey, C. Raghavendra, and K. M. Sivalingam. Data gathering algorithms in sensor networks using energy metrics. *IEEE Transactions on Parallel and Distributed Systems*, 13(9):924–935, 2002.

[56] C. Liu and G. Cao. Distributed monitoring and aggregation in wireless sensor networks. In *Proceedings IEEE INFOCOM, 2010*, pages 1–9. IEEE, 2010.

[57] C. Liu, K. Wu, and J. Pei. An energy-efficient data collection framework for wireless sensor networks by exploiting spatiotemporal correlation. *IEEE Transactions on Parallel and Distributed Systems*, 18(7):1010–1023, 2007.

[58] L. Lu, T. Wang, S. C. Liew, and S. Zhang. Implementation of physical-layer network coding. *Physical Communication*, 6:74–87, 2013.

[59] C. Luo, F. Wu, J. Sun, and C. W. Chen. Compressive data gathering for large-scale wireless sensor networks. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, pages 145–156. ACM, 2009.

[60] D. Luo, X. Zhu, X. Wu, and G. Chen. Maximizing lifetime for the shortest path aggregation tree in wireless sensor networks. In *Proceedings IEEE INFOCOM, 2011*, pages 1566–1574. IEEE, 2011.

[61] J. Luo, L. Xiang, and C. Rosenberg. Does compressed sensing improve the throughput of wireless sensor networks? In *2010 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2010.

[62] S. Mehrjoo, J. Shanbehzadeh, and M. M. Pedram. A novel intelligent energy-efficient delay-aware routing in wsn, based on compressive sensing. In *2010 5th International Symposium on Telecommunications (IST)*, pages 415–420. IEEE, 2010.

[63] N. Naderializadeh and A. S. Avestimehr. ITLinQ: A new approach for spectrum sharing in device-to-device communication systems. *IEEE Journal on Selected Areas in Communications*, 32(6):1139–1151, 2014.

[64] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[65] M. Safar, H. Al-Hamadi, and D. Ebrahimi. Peca: power efficient clustering algorithm for wireless sensor networks. *International Journal of Information Technology and Web Engineering (IJITWE)*, 6(1):49–58, 2011.

[66] M. Sartipi and R. Fletcher. Energy-efficient data acquisition in wireless sensor networks using compressed sensing. In *Data Compression Conference (DCC)*, pages 223–232. IEEE, 2011.

[67] S. Sengupta, S. Rayanchu, and S. Banerjee. Network coding-aware routing in wireless networks. *IEEE/ACM Transactions on Networking*, 18(4):1158–1170, 2010.

[68] W.-Z. Song, F. Yuan, R. LaHusen, and B. Shirazi. Time-optimum packet scheduling for many-to-one routing in wireless sensor networks. *The International Journal of Parallel, Emergent and Distributed Systems*, 22(5):355–370, 2007.

[69] R. Subramanian and F. Fekri. Sleep scheduling and lifetime maximization in sensor networks: fundamental limits and optimal solutions. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 218–225. ACM, 2006.

[70] N. Thepvilojanapong, Y. Tobe, and K. Sezaki. On the construction of efficient data gathering tree in wireless sensor networks. In *IEEE International Symposium on Circuits and Systems, 2005. ISCAS 2005*, pages 648–651. IEEE, 2005.

[71] D. Traskov et al. *Network coding for multiple unicasts: An approach based on linear optimization*. PhD thesis, Citeseer, 2006.

[72] D. Traskov, M. Heindlmaier, M. Médard, R. Koetter, and D. S. Lun. Scheduling for network coded multicast: A conflict graph formulation. In *2008 IEEE GLOBECOM Workshops*, pages 1–5. IEEE, 2008.

[73] W. Wang, M. Garofalakis, and K. Ramchandran. Distributed sparse random projections for refinable approximation. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 331–339. ACM, 2007.

[74] X. Wang, Z. Zhao, Y. Xia, and H. Zhang. Compressed sensing based random routing for multi-hop wireless sensor networks. In *2010 International Symposium on Communications and Information Technologies (ISCIT)*, pages 220–225. IEEE, 2010.

[75] Z. Wei, Y. Sun, and Y. Ji. An integrating data gathering scheme for wireless sensor networks. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1151–1156. IEEE, 2013.

[76] Wikipedia. List of wireless sensor nodes. `http://timmurphy.org/2009/07/22/line-spacing-in-latex-documents/`. Accessed February 9, 2016.

[77] X. Wu, S. Tavildar, S. Shakkottai, T. Richardson, J. Li, R. Laroia, and A. Jovicic. FlashLinQ: A synchronous distributed scheduler for peer-to-peer ad hoc networks. *IEEE/ACM Transactions on Networking (TON)*, 21(4):1215–1228, 2013.

[78] X. Wu, Y. Xiong, W. Huang, H. Shen, and M. Li. An efficient compressive data gathering routing scheme for large-scale wireless sensor networks. *Computers & Electrical Engineering*, 39(6):1935–1946, 2013.

[79] Y. Wu, S. Fahmy, and N. B. Shroff. On the construction of a maximum-lifetime data gathering tree in sensor networks: NP-completeness and

approximation algorithm. In *IEEE INFOCOM 2008. The 27th Conference on Computer Communications*. IEEE, 2008.

[80] L. Xiang, J. Luo, and C. Rosenberg. Compressed data aggregation: Energy-efficient and high-fidelity data collection. *IEEE/ACM Transactions on Networking*, 21(6):1722–1735, 2013.

[81] L. Xiang, J. Luo, and A. Vasilakos. Compressed data aggregation for energy efficient wireless sensor networks. In *8th annual IEEE communications society conference on sensor, mesh and ad hoc communications and networks (SECON), 2011*, pages 46–54. IEEE, 2011.

[82] R. Xie and X. Jia. Transmission-efficient clustering method for wireless sensor networks using compressive sensing. *IEEE Transactions on Parallel and Distributed Systems,*, 25(3):806–815, 2014.

[83] X. Xu, R. Ansari, and A. Khokhar. Power-efficient hierarchical data aggregation using compressive sensing in wsns. In *2013 IEEE International Conference on Communications (ICC)*, pages 1769–1773. IEEE, 2013.

[84] X. Xu, X.-Y. Li, and M. Song. Efficient aggregation scheduling in multihop wireless sensor networks with sinr constraints. *IEEE Transactions on Mobile Computing*, 12(12):2518–2528, 2013.

[85] B. Yu, J. Li, and Y. Li. Distributed data aggregation scheduling in wireless sensor networks. In *IEEE INFOCOM 2009*, pages 2159–2167. IEEE, 2009.

[86] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Energy-latency tradeoffs for data gathering in wireless sensor networks. In *INFOCOM 2004.*

Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, volume 1, pages 224–255. IEEE, 2004.

[87] M. Zhao, Y. Yang, and C. Wang. Mobile data gathering with load balanced clustering and dual data uploading in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 14(4):770–785, 2015.

[88] W. Zhao and X. Tang. Scheduling sensor data collection with dynamic traffic patterns. *IEEE Transactions on Parallel and Distributed Systems*, 24(4):789–802, 2013.

[89] H. Zheng, S. Xiao, X. Wang, X. Tian, and M. Guizani. Capacity and delay analysis for data gathering with compressive sensing in wireless sensor networks. *IEEE Transactions on Wireless Communications*, 12(2):917–927, 2013.

[90] G. Zhou, T. He, J. A. Stankovic, and T. Abdelzaher. Rid: radio interference detection in wireless sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 891–901. IEEE, 2005.