DATA COLLECTION TECHNIQUES USING MULTI-CHANNEL NETWORK CODING IN LOW-POWER AND LOSSY NETWORKS

by

Mansour Abdulaziz
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Robert Simon, Dissertation Director
_____	Dr. Hakan Aydin, Committee Member
_____	Dr. Daniel Menasce, Committee Member
_____	Dr. Jill Nelson, Committee Member
_____	Dr. Sanjeev Setia, Department Chair
_____	Dr. Kenneth Ball, Dean, Volgenau School of Engineering

Date: _____ Summer Semester 2016
George Mason University
Fairfax, VA

Data Collection Techniques Using Multi-Channel Network Coding In Low-Power and Lossy Networks

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Mansour Abdulaziz
Master of Science
Missouri University of Science and Technology, MO, 2010
Bachelor of Science
Kuwait University, Kuwait, 2004

Director: Dr. Robert Simon, Professor
Department of Computer Science

Summer Semester 2016
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my patient and stimulating wife, Maryam Dashti, who endured the homesick while nursing my lovely angels: Abdullah, Fatma, Zainab and Maryam. She strengthened me with unconditional support and enlightenment. I also dedicate this dissertation to my parents and my wife's parents for their continued prayers and enthusiasms.

Acknowledgments

I would like to thank the following people who made this possible. First, I want to thank my advisor, Dr. Robert Simon for all of his supervision, support, and endurance. I also want to thank Dr. Hakan Aydin, Dr. Daniel Menasce and Dr. Jill Nelson for their feedback to enhance my dissertation.

Table of Contents

List of Tables

List of Figures

Abstract

DATA COLLECTION TECHNIQUES USING MULTI-CHANNEL NETWORK CODING IN LOW-POWER AND LOSSY NETWORKS

Mansour Abdulaziz, PhD

George Mason University, 2016

Dissertation Director: Dr. Robert Simon

The underlying motivation for my dissertation research is to investigate the combined use of wireless network coding with multi-channel communication to perform data collection in low-power and lossy networks (LLNs). Network coding allows eavesdropping on non-source, non-destination wireless nodes in order to recombine overheard data with its own data, which increases the ability of the ultimate destination to recover a high amount of information, even in the case of high bit error rates. Coordinated multi-channel communication has the potential to dramatically increase the throughput for many types of LLN applications. My work seeks to unify network coding and coordinated multi-channel communication. I consider several important classes of LLN applications, including scenarios with static data collection sinks and mobile data collection sinks.

This dissertation has three primary contributions to improve communication performance in data collection LLN systems. The first is a protocol called *MuCode*, which is designed to support single or multiple sinks that support a Convergecast (many-to-one) communication pattern. I describe a synchronized channel switching policy that takes advantage of wireless overhearing to perform network coding operations. I show that optimally solving certain aspects of this problem is quite challenging, and present a set of heuristics

for building the delivery mechanism. I have evaluated MuCode against several other schemes and my results show significant performance improvements under a variety of scenarios.

The next two contributions use the results of the MuCode approach and extend them into mobile environments. The second contribution is the protocol *MuTrans*, which is employed in environments that require mobile data collection in a way that is predictable but uncontrollable. I formally analyze the complexity of this problem in terms of how to minimize data collection latency, which shows it is a challenging problem. I present a synchronized dynamic round-robin scheduling policy for uploading data to a mobile collector that is based on assigning a method for load balancing. My evaluation of data aggregation in the presence of packet errors shows that MuTrans can significantly reduce the latency for data collection, thus providing strong support for mobile data collection.

Finally, I present the *MuCC* protocol, which is designed to support data collection when mobile data collectors motions are both controllable and predictable. After designing and evaluating a two-layer architecture, I provide algorithms on cluster head selection, cluster membership assignment, and trajectory planning. My experimental tests indicate that MuCC substantially outperforms similar approaches in hierarchical data collection.

Chapter 1: Introduction

Network coding is a mathematical technique whereby nodes combine bits from different packets to form new packets instead of relaying the packets of information they receive [1]. Network coding works by treating bit strings as elements in a Galois field and performs finite field operations over different sets of bits. Network coding methods can be used to reduce the number of packet transmissions, in the case of wired networks, or to reduce the impact of transmission errors on wireless systems. Ostovari et al. [2] has shown that by using network coding systems that experience high packet loss rates, such as wireless sensor networks (WSNs), network coding can offer tremendous benefits regarding increased throughput and reliability for successful packet transmissions received by the destination.

Another technique to improve performance in wireless systems is to use multiple communication channels. This method allows nodes to transmit packets in different frequencies without interference, as long as the gap between any adjacent frequencies is relatively large. It has also been shown that using multiple communication channels WSNs decreases end-to-end communication latency [3] and increases throughput [4]. This improvement is because using a single channel forces nodes to compete with their neighbors prior starting transmission in order to avoid collisions. By using multiple channel neighboring, nodes can simultaneously transmit packets.

Despite the advantages offered by both network coding and multi-channel transmission strategies, there has been little work done in the combination of these two techniques for wireless sensor networks. One of the reasons for this lack of attention in combining network coding and multi-channel communication is that transmitting on different frequencies eliminates an essential advantage of network coding in a wireless system: The ability to overhear neighboring transmissions. These eavesdropped packets can then be encoded with other packets to improve reliability by allowing the same information to be sent to the gateway

via different paths. One of the core ideas in my dissertation work is to explore how network coding can be used with multiple communication channels in a variety of application scenarios.

Over the last few years, WSNs have been central as an enabling technology for a broad range of Internet of Things (IoTs) applications. The IoT can be defined as networking for embedded objects in a way that allows them to communicate, interact or exchange data with other connected objects. IoT systems are growing at an explosive rate. For instance, on December 2013, Gartner [1] stated that "the internet of things installed-base will grow to 26 billion units by 2020." Here, WSN nodes are distributed and can monitor environmental conditions [5], precision agriculture [6] ,or cyber-physical systems [7]. A typical scenario in IoT system is to have the sensors periodically report their readings upwards toward a gateway or base station. This style of networking is often called *Convergecast* [8]. From a broader perspective, WSNs form an important subclass of a rapidly emerging system class called Low-power and Lossy Networks (LLNs). LLNs are embedded systems with limited resources such as processing, power, and memory [9]. LLNs may or may not sense their environment. They are lossy because they use wireless communication that has a high error rate and unpredictable reception. This situation generates a complex, unpredictable, and challenging environment. Due to the broad importance of LLNs in the IoT world, my work concentrates on this system architecture.

The primary motivation of my research combines multi-channel with network coding in both tree-based and cluster-based LLN systems, where these systems represent and support Convergecast communication style. As explained above, Convergecast is a form of networking in which source nodes transmit their data to the base station (sink) through multi-hop routing or mobile data collection. My dissertation makes three primary contributions: the MuCode protocol for static Convergecast, the MuTrans protocol for uncontrollable predictable mobile sink supporting Convergecast and MuCC protocol for controllable predictable mobile sink supporting Convergecast.

[1] The technology research and advisory corporation

The first contribution combines multi-channel coding with network coding in tree-based multi-hop routing networks through the design and implementation of the *MuCode* protocol [10]. The protocol uses a set of graph construction algorithms that support synchronized channel switching to take advantage of wireless eavesdropping for multi-path network coded packet delivery. The nodes have radios that have single transceivers and are capable of selecting multiple transmission channels. MuCode targets applications that are configured into a tree-based topology with a powerful base station, equipped with multiple radios at the root [11].

Using MuCode model for this system, I formulated an optimization problem for minimizing the maximum data load rate for the case of a Convergecast with multiple sinks. I show that this problem is quite challenging, and I present several heuristic algorithms that can be used to maximize data delivery. I then evaluate my heuristic methods with the optimal solution and other path selection strategies, using different numbers of sinks that share the same data from the source nodes.

The second and third contributions of my dissertation consider the impact of mobile data collection nodes. One of the issues with using large scale static Convergecast systems is that multi-hop routing to one or more sinks can both significantly degrade the network lifetime and increase congestion. Gathering mobile data can save nodes' battery life and diminish the number of nodes that need to be deployed. A mobile data collector can traverse the LLN network to collect data readings and relay new commands and updates [12, 13]. Such a portable scheme can reduce the number of required LLN nodes, since the network does not need to be connected. Further, because the mobile unit can directly communicate with embedded LLN nodes, the level of multi-hop routing can be reduced, thus improving reliability by reducing the number of retransmissions.

A wide range of mobile data collection application classes has been broadly investigated [14]. Data collector mobility patterns can be categorized as either *controllable* [15] or *uncontrollable* [16, 17]. On one hand, the controllable mobile collector can freely traverse to any location in the network where its trajectory is predicted. On the other hand, the

uncontrollable mobile collector can move either randomly (unpredictable) or on a fixed route (predictable).

The second part of my research applies the advantage of MuCode protocol to support applications that use uncontrollable predictable mobility collection. My research assumes that the mobile collector traverses the network in a predetermined path with an arrival schedule that is known in advance, without motion control that can be modified by my algorithms or protocols. An example of such an application: a public transportation vehicle (e.g. bus or train) in Smart Cities [18]. While the vehicle is in service, it collects data from surrounding nodes and delivers it to the appropriate base station (BS). I designed a network-coded multi-channel protocol called *MuTrans* (*Mu*lti-channel *Trans*port). In this way, MuTrans uses a cluster-based approach. There are two types of nodes in this system: head nodes and member nodes. The head nodes are the nodes that have direct communication with the mobile sink, while member nodes are unreachable by the mobile sink but are one hop away from head nodes. Multi-hop routing is not required.

Since the mobile data sink moves in a known in advance arrival, data collection latency becomes substantial in regards to the network instability, due to the dynamic changes in the network environment. I analytically formulate the uploading latency model and provide an optimization problem that is Zero-One Integer Linear Programming (ZOILP). The MuTrans protocol consists of two heuristic techniques: data load balance and weighted dynamic round robin scheduling. The first heuristic reduces the data overload at the head nodes, while the second provides a fairness uploading schedule that dynamically changes to reduce the overall uploading latency.

Under the circumstances of controlled mobility, whereby the trajectory and data transfer points can be managed, it is essential to carefully plan the actual motion of the mobile node, since the mobile node significantly affects data delivery rates and energy consumption patterns. Thus, the last part of my research presents *MuCC*, a *Mu*lti-channel Network *C*oding *C*lustering for controllable mobile data collection in the presence of high packet and bit error rates. Using available network topological information and a *centroid*-based

approach, MuCC carefully picks its communication points in a way that minimizes energy consumption and maximize reliability. MuCC efficiently develops the trajectory plan for the mobile collector in order to collect the data and store it in a data repository. The path plan is decided by finding the most efficient *polling points* to upload the data. The polling points are locations that the mobile collector needs to stop at and retrieve the data from the nearby sensors. The MuCC clustering protocol uses multiple channels for simultaneous data, which uploads to the mobile collector to avoid packet collision. Multiple channels also eliminate packet collision between clusters transmissions.

Since the polling points locations are based on the locations of the cluster heads, I developed an adaptive cluster head algorithm, called *ACH*, that determines better cluster heads in order to reduce the number of polling points while maintaining the system configuration and the durability of the network. By reducing the number polling points, ACH can significantly reduce the traveling latency that the mobile data sink has to cover in order to collect the same data. *To the best of my knowledge, this thesis is the first work to examine the use of the combined techniques, multi-channel and network coding, in order to support static and mobile data collection schemes in LLNs.*

1.1 Thesis Overview

My dissertation hypothesis can be presented as follows:

Thesis Hypothesis

Data collection in low-power and lossy networks with limited constraints require new methods for efficient communication performance. Algorithms using multi-channel and network coding techniques will significantly improve the network reliability and throughput of such systems. Both techniques can be used together to enhance static multi-hop and mobile multi-data collection systems. Data balancing among the nodes with synchronized scheduling can effectively increase the network lifetime and reduce data delivery latency. Algorithmic

techniques that exploit load balanced clustering with hierarchical structuring will also substantially improve system performance.

This dissertation describes a general communication protocol and different application models for diverse data collection LLN systems. In static Convergecast multi-sink systems, a new data load heuristic is proposed to minimize the maximum data load rate with the objective of maximizing network lifetime, throughput and reliability. Novel data collection rate allocation and dynamic uploading scheduled algorithms will be proposed to minimize the data uploading delay and utilize the network throughput in predetermined route mobility collections. A new cluster architecture scheme that appropriate multi-channel and network coding techniques will be performed in controllable mobility systems. An Innovative polling points algorithm will be offered for allocating the best uploading locations while reducing the traveling latency. These contributions will be evaluated through analytical methods and high fidelity simulation.

My work improves system performance for both static and mobile collection systems in several ways:

- An algorithm that combines network coding and multi-channel that supports LLN nodes in Convergecast applications
- The use of time or epoch based approach to synchronize data transmissions.
- A system architecture for each of the three collection models.
- A formal model and analysis of how to minimize the uploading latency in uncontrollable WSN mobility.
- An algorithm to allocate efficient polling points for latency management in controllable WSN mobility.

The outcome of this work is a set of algorithms and protocols using multi-channel and network coding, which is capable of maximizing network throughput and minimizing energy

consumption and latency in order to reach an ultimate goal of improving network resilience of inconstant environment, while substantially maximizing the network lifetime.

1.2 Dissertation Roadmap

The outline for the rest of my dissertation is as follows: first, I present background information and a literature review in Chapter 2 for WSNs and LLNs. I then define the foundation of network coding and multiple-channels techniques and offer an overview of static and mobile data collection systems. Next, I give my system architecture in Chapter 3. In Chapter 4-6, I present the technical details of my research work, including a series of multi-channel network coding algorithms for static and mobile LLN systems, and rate allocation algorithms with dynamic fairness scheduling for maximizing the concurrent uploading of WSN networks. In Chapter 7, I propose future works on using the above algorithms on different topological and system scenarios. Finally, Chapter 8 offers some observations and conclusions about my three contributions in LLN data collection systems.

Chapter 2: Background and Related Works

This chapter focuses on the characteristics of low-power and lossy networks, network coding, multi-channel networking and, types of mobile data collection for wireless sensor networks.

2.1 Low-Power and Lossy Network

My research targets the broad range of wireless sensor networks that are now generally referred to as belonging to the class of Low-Power and Lossy Networks (LLNs) [9]. LLNs consist of nodes with limited resources, such as constrained processing power, energy (battery), and memory. These nodes are interconnected by different kinds of wireless links, such as IEEE 802.15.4, Bluetooth LE, or Low Power WiFi [19]. LLNs may involve thousands of nodes that are uniformly distributed or randomly scattered, depending on their primary functionality. The constrained memory prevents nodes from maintaining complete global topological knowledge. LLNs that operate off of battery power or energy harvesting must use careful strategies to maximize network lifetime. The communication link in this system is connectionless using UDP packets, and it is lossy due to wireless impairments. Further, the link typically has a low data rate.

To support application requirements, LLN systems can be arranged in tree topologies with a base station at the root. The base station may have unlimited resources, including multiple radios while nodes periodically report their readings upwards toward the root. Example applications include security surveillance, environmental monitoring, or Smart Grid/Smart City systems. Broadcasting (one-to-many) from the base station to all the nodes is also a routine activity.

LLN nodes use radios that have single transceivers. They are capable of selecting channels among multiple available transmission channels, such that each channel is centered

on a different radio frequency (RF). One example of multi-channel technology is a CC2420 radio, which is manufactured by Chipcon. This radio is a 2.4 GHz IEEE 802.15.4-compliant technology that supports low-power and low-voltage wireless applications with a 250 kbps data rate. Its frequency region resides between 2400 to 2483.5 MHz. IEEE 802.15.4 specifies 16 channels within the 2.4 ISM GHz band, which are numbered 11 through 26 [20]. This chip can also be employed in a non-IEEE 802.15.4 system by combining the direct sequence spread spectrum (DSSS) and the frequency hopping spread spectrum (FHSS). DSSS is a modulation technique that helps share a single channel among multiple nodes transmissions, while FHSS is a radio signal transmitting method that allows switching a carrier signal among multiple frequency channels.

My work is independent of the physical layer of the IoT protocol stack and focuses on channel selection. Successful use of multi-channel communication requires that senders and receivers agree upon which channels to use at which time, which requires a channel scheduling policy. The advantage of this approach is that any pair of nodes that use different channels will not interfere with each other, and those nodes can avoid the hidden terminal problem.

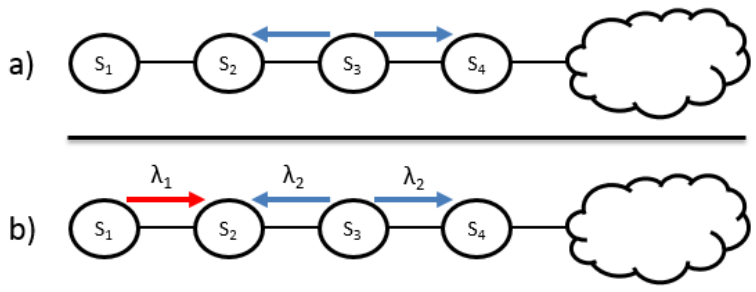


Figure 2.1: Example of Using Multi-Channel

To demonstrate the power of using multiple channels, consider Figure 2.1.(A) is a network with a single communication channel while (b) is a network with two available channels (λ_1 and λ_2). In part (a), node S_3 transmits its packets and both S_2 and S_4 receive them.

Node S_1, however, cannot send nor receive at the moment of S_3's transmission, which can cause latency in a packets delivery. Since both S_1 and S_3 interfere with S_2 in part (b), node S_3 transmits its packets to S_2 and S_4 using channel λ_2, while S_1 is transmitting its packets to S_2 simultaneously. S_2 has a single transceiver and has to tune its radio frequency to either channel λ_1 or λ_2 depending on the traffic path.

The algorithmic issue with multi-channel networking is the difficulty in channel assignment and synchronization. The nodes have to agree upon which channel to transmit at which time. Related work in tree-based multi-channel LLN networking is discussed by Wu et al. [3] who proposed a Tree-based Multi-Channel Protocol (TMCP) for multiple Convergecast, which addresses the channel assignment problem. They construct K sub-trees for K channels to eliminate inter-tree interference. They have minimized the intra-tree interference for each sub-tree.

Other examples of work in a multi-channel LLN has focused on the design of new MAC layers [21], routing protocols [3], or data dissemination [22]. Simon et al. [23] showed that by using multi-channel communication technique for reliable data dissemination, the data prorogation delay is significantly reduced. Zhou et al. [24] designed collision-free protocols by conducting some radio interference detection experiments. Zhou observed that a link with a high reliability causes the communication range to overcome the interference range, while he observed the opposite for a weak reliability link. Further, Kyasanur et al. [25] has shown that the network capacity can lose packets when the number of interferences per node is relatively smaller than the number of available channels. Meanwhile, a channel assignment algorithm is proposed in [26] to reduce the interference problem in multi-radio wireless mesh networks (WMNs). They suggested that using multi-radio routers improve the system performance, while static channel assignment can lessen the performance in WMNs. However, none of this work considers the use of network coding.

2.2 Network Coding

In this section, I introduce some background information and related research on the network coding in wireless systems. Network coding is a technique where nodes take several packets and combine them for transmission of new packets with the same size as the single packet, plus the encoding vector overhead. The combining operation can be a simple XOR or a method such as Random Linear Network Coding (RLNC). The RLNC technique uses finite field operations. The addition and multiplication operations are under a Galois field of some size 2^f. An example for f equal to 4 is to use an irreducible polynomial F_{2^z} where Z^3+Z+1 [27][28]. Figure 2.2 depicts an example of how network coding can improve reliability in a lossy network. The intermediate node (IN_2) collects and combines both packets x_1 and x_2 using algebraic linear equations to produce the encoded packet $x_1 + x_2$ and sends it to the sink node R. Now, assuming any one of the three transmissions to R is lost, R can still get the packets x_1 and x_2 by having enough linearly independent packets to decode. Hence, with a 33% probability of a packet loss at the receiver side R, the network can maintain 100% of the packets delivery (x_1 and x_2) by using network coding instead of retransmitting by using the traditional wireless transmitting technique.

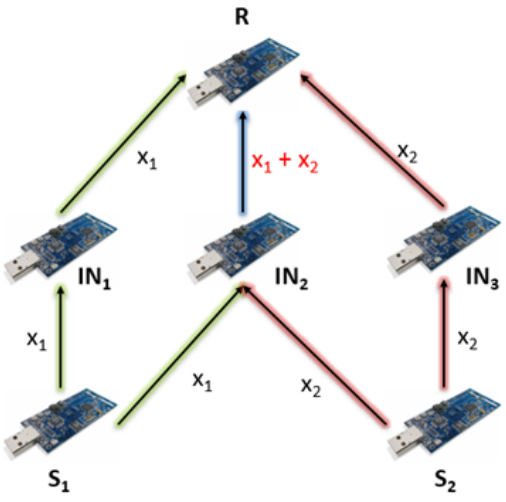


Figure 2.2: An example of network coding reliability.

The work in network coding that is related to my research includes several papers that study the impact of duty cycling or the use of an enhanced MAC protocol. [29] provided a study on a joint channel-network coding in order to examine the synergy between network coding and a convolutional code (error-correcting code). [30] proposed a communication model by combining duty cycle and network coding to improve network lifetime in bottleneck zones, where the sink is located, since it has the maximum traffic flow in Convergecast networking. Wang et al. [31] proposed a network coding-aware cooperative medium access control protocols for wireless ad hoc networks in order to increase network throughput and reduce latency. They also provided a network coding-aware utility-based relay selection strategy to select the efficient relay for distribution. Last, they have incorporated two collision-free relay selection strategies that can improve packet delivery rates and reduce latency in 802.11.

Kamal et al. [32] proposed single and multiple link failure protection methods for wired networks, without the need to detect link failure or implement rerouting for the data. They provided a single replication for each transmitted data using synchronized network coding that sends each bit of data into two opposite directions. Chandanala et al. [33] designed a network coding and duty cycling scheme in which multiple access control protocols transmit packets and implement network coding-aware algorithms in order to determine optimal duty cycling schedules that reduces energy consumption. Focusing on the encoding overhead, Jafari et al. [34] presented a new approach to design optimal relaxation for compressing the encoding vectors in order to convey the encoding coefficients efficiently, as compared to the classical approach. Alwis et al. [35] proposed a new method to minimize the coefficient encoding vector size of RLNC encoded packets by exploiting the properties of small and medium size networks.

Gnawali et al. [36] designed the first collection tree protocol in tree-based WSN networks. They described two routing protocols that improve the efficiency and robustness of the high loss and dynamic link systems. Sensecode [37] is closely related to my work in

the static Convergecast case, which demonstrates the practical use of network coding in resource constrained LLN designed to support Convergecast. Unlike all of these approaches, my work considers multi-channel networking. Further, I have designed MuCode to take advantage of eavesdropping by a conservative policy of synchronized channel switching and transmission overhearing.

2.3 Data Collection Architectures

A portion of my work falls into the category of hierarchical mobile data collection with several node tiers. This architecture has been explored in many papers. For instance, [38] construct clustering techniques by introducing a cluster head selection metric that considers both the node's residual energy and its neighbored links qualities. This metric helped in designing two distributed algorithms to construct one-hop and k-hop clusters. Zhang et al. [39] proposed a two-layered heterogeneous WSN with two types of sensors. The basic sensors are simple and cheap nodes. The cluster head nodes are powerful and energy-rich, which collect data from member nodes. They provided a collision-free polling schedule in the multi-hop cluster. Ma et al. [40] designed a mobile data collection scheme for large scale WSN. The mobile collector starts traversing the network from the static sink and collects data from sensors and uploads the data to the data sink periodically. They also introduced multiple data collectors for restricted deadlines or distance.

From an architectural perspective, cluster heads are responsible for collecting the data from their members and forwarding it to the sink. This clustering can significantly reduce packet collisions and balance the load of the data among the sensors, which typically provides more scalable network descent as compared to enhanced relay routing. However, the cluster heads face massive data traffic from both intra-cluster aggregation and inter-cluster data forwarding. Using a mobile data collector can reduce the overall congestion in cluster heads. Nevertheless, in the mobile environment, data collection latency can become excessively long.

Related work to my research includes Lee et al. [41] who proposed a data stashing

technique for wireless mesh sensor networks in which the data are routed to a relay node that is involved in the announced or predicted trajectory path of the mobile sink. The work in [42] proposes a data dissemination scheme that solves the energy consumption phase on sink nodes when diffusing the information on the network. On the other hand, Lu et al. [43] considered a mobile sink that is moving in a fixed trajectory and deploying gateways nearby. Those gateways collect data from the wireless sensors and send them to the mobile sink whenever it is within their transmission range.

Kweon et al. [44] proposes a Grid-Based Energy-Efficient Routing protocol that creates a one grid structure of an event that effectively sends the data to multiple sinks with minimized sensors energy consumption. Shon et al. [45] described a Hexagonal Path Data Dissemination (HPDD) scheme that uses straight and diagonal line paths by constructing a virtual hexagonal path. The authors in [46] present the Ring Routing (RR), a distributed energy-efficient mobile sink routing protocol. Weiwei et al. [47] consider the problem of collecting data packets from particular nodes to the sink by combining network coding with collecting data packets in a specified area of interest.

This hierarchical protocol is based on a virtual ring structure. This protocol mitigates the predicted hotspot problem and minimizes the data latency to the mobile sink. However, this protocol works only with connected and uniformly distributed wireless sensors. Some nodes can create the ring routing with low priority (such as energy), which may affect the network lifetime. For lossy networks, ring routing cannot survive with unreliable communication, since it assumes all ring exploring packets either have been delivered to neighboring nodes or there is no neighbor, which is not always the case. The enhanced relay routing – such as [48] – relays the data to the sensors. The problem with the improved relay path is that the sensors that are near the sink potentially consume more energy than the others, which inevitably reduce their battery lifetime drastically.

Another study that is closely related to my work is Zhao et al. [49], who discussed a method to reduce the collection latency from the cluster heads to the mobile sink in order to increase network lifetime. They proposed a three-layer mobile data collection

framework called Load Balanced Clustering and Dual Data Uploading (LBC-DDU). Unlike LBC-DDU, however, in my controllable predictable mobility approach, I consider the use of both multiple communication channels and network coding to improve both reliability and throughput by using MuCC, the hierarchical clustering protocol that I have designed. In the controllable predictable, the mobile collector may modify its trajectory based on some constraints (e.g. deadline or change in polling location). This mobile collector uses stop-and-wait communication whereby it traverses according to a predetermined motion plan. Once it reaches a designated sensor that holds some data, the mobile collector stops and begins collecting from that sensor [50].

In order to perform the data collection, I devised a method to determine the best locations for the mobile collector ($MuCar$) to reach. The MuCar is a multi-channel network coding mobile data collector. It has multi-radio, so it can concurrently collect data from nodes through different channels. It traverses the network freely to the designated polling points and collects data from cluster heads. It applies decoding process through network coding to retrieve the original data from the encoded received packets.

2.4 Mobility Models

Broadly speaking, there are two types of mobility trajectory management systems, controllable [51] and uncontrollable [52, 53]. The controllable mobile data collector can change its trajectory path and speed in order to collect the data from sensors and upload data to the base station, as I derived in the previous section; while the uncontrollable mobile collector can traverse the network randomly and unpredictably. Random waypoint is an example model of the random movement of a mobile collector whereby location, velocity, and speed change over time [54]. The Random Mobility Model (RMM) has been used as a synthetic model for different areas, such as mobile ad hoc and mobile mesh networks [55], because of how simple and available it was. This model can be divided into random waypoint, random walk, and random direction models [56, 57].

An example of other works in this area is Restuccia et al. [58] who proposed the Swarm-Intelligence-based Sensor Selection Algorithm (SISSA) that optimizes network lifetime by meeting predefined Quality of Service constraints for uncontrollable and random mobile data collector.

The last part of my research focuses on the uncontrollable predictable mobile data collector in WSN systems. Works related to my research include Kansal et al. [59] who demonstrated that using mobile data collection can increase network lifetime and utility in the embedded sensor nodes. The fluid infrastructure traverses the network in a predefined linear path to collect data from nearby nodes. Depending on the amount of data a node has, their paper provided an adaptive speed control technique for the mobile collector to get high-quality data.

Jea et al. [60] extended [59] research by employing multiple mobile collectors to reduce data latency. They divided a convex area into sub-areas in which a load balancing algorithm is implemented to assign each mobile collector (MULE) to evenly sub-area. With a constant speed, each MULE traverses its sub-area through a predetermined straight line path back and forth in order to collect data from single-hopped sensor nodes. Nevertheless, the scalability becomes an issue because increasing the size of the convex area and increasing the size of the sensor nodes degrades the performance with multiple sinks.

A distributed mobile application called *CarTel* has been proposed by Hull et al. [61]. CarTel runs on a portal using a delay-tolerant continuous query processor, with opportunistic communication with other nodes. Hull analyzed Boston and Seattle metropolitan's Wi-Fi deployments, their commute time, and monitored the road traffic for a year using six cars. Somasundara et al. [62] provided an enhanced communication protocol, emphasizing the data delivery rates, which compares to [59], who used cluster formation with a mobility motion control strategy by considering node's buffer constrained and the mobile collector velocity. They analyzed the tradeoff in energy and throughput by using controlled mobile data collector with limited energy constraint.

Gao et al. [63] designed efficient member assignments to the sub sinks using a Genetic

Algorithm (GA) in order to enhance the power consumption and the data delivery to the mobile data collector, which traverses the network in fixed trajectory with multi-hop communication. Each sub-sink buffers the received data from member nodes using shortest path routing trees. On the other hand, Liang et al. [64] suggested an approximation algorithm for an NP-Hard optimization problem called Capacitated Minimum Forest (CMF) in hierarchical heterogeneous WSN architecture. The mobile sink traverses through planned trajectory and collects data from the gateways where they then temporarily aggregate data from sensors through multi-hop routing. Smeets et al. [18] implemented a platform called *Trainsense* in order to support mobile WSN applications. Essentially this platform behaves like a model train, which runs to the deployed nodes for different mobility trajectories.

None of these related works have considered fixed route and uncontrollable mobile data collectors with one-hop routing in unstable environments. By considering this mobility model, I propose two heuristic methods to minimize the uploading latency by applying the advantage of network coding and multi-channel coding in LLN networks. I use the MuCode protocol, which allows head nodes the possibility to overhear packets synchronously from their member nodes, while also eavesdropping on packets from other nodes in order to increase the chances of recovering lost data. Each head node implements network coding to the eavesdropped packets along with the already stored packets, and that combination creates encoded packets. Then the head node stores them in the transmit queue, which consists of both encoded and plain packets. Finally, each head node concurrently sends all its data to the sink through different channels in order to avoid collision with one condition, that the mobile sink has enough multi-radios to receive packets simultaneously.

Chapter 3: System Model

This chapter describes the hardware, wireless transmission channel, data, and sink model underlying my dissertation research. As described in previous chapters, I am exploring two topological configurations. The first configuration is a tree-based topological system that has a powerful base station embedded within multiple radios at the root. I assume that the network is connected and not partitioned, and based off of this assumption, multi-hop routing is used for data transmitting through relay nodes. Tree-based applications are considered general purpose LLNs that are inhabited by resource constrained wireless sensor nodes. These nodes periodically sense and process data, and then report this data back to a base station or base stations.

The second topological configuration also has the purpose to periodically sense, process and report data. This configuration does not rely on multi-hop routing but uses a mobile data collector to obtain the data. All nodes have the ability to broadcast wirelessly.

3.1 LLN Node Model

My work targets the current generation of wireless networks designed to support "Internet-of-things" applications, such as the Smart Grid or Building and Home Automation systems, which control and automate the residential applications, such as lighting, air conditioning, and appliance control. The sensors are resources constrained in the battery and memory capacity, wireless capacity, and processing capabilities. I am assuming the nodes are static placed and distributed in a two-dimensional area. LLNs may have sensors or may simply function as relays. The sensors functionality varies from sensing, dating, or reporting to the base station through relay nodes.

My communication protocol is designed to support any available network coding techniques, such as XOR [27, 65] or Random Linear Network Coding (RLNC) [28, 66], which can be implemented in the node model. In practice, RLNC is known to be an attractive option for LLNs, due to the simplicity of its implementation.

Network coding allows LLN nodes to combine packets that have been overheard from multiple senders [1] [67]. These eavesdropped packets can then be coded with other packets to improve network reliability by allowing the same information to be sent to the sink via different paths. In wireless systems, network coding is also used to allow a receiver to recover a number of packets, even when some of those packets are lost by performing mathematical operations that are based upon information they already hold. From a mathematical perspective, network coding can be performed using binary coding by simply performing bit by bit XOR operations. Another technique is to use Random Linear Network Coding (RLNC), where packets are defined by elements over a Galois field. Packets are coded randomly using finite field operations. The lost original packets are recovered using operations such as the Gaussian elimination. The Random part of RLNC means the bits are combined after they are multiplied by a random coefficient.

For a static Convergecast tree, I assume the availability of standard WSN mechanisms for routing [68], Convergecast tree construction, and inter-node time synchronization [69]. Basically, Convergecast is a many-to-one communication operation. I use mechanisms for link level fairness based upon time-slotted round-robin transmissions [11] in order to provide transmission fairness between nodes that reside on the same level, which I will discuss in more detail later in the next chapter.

For the topologies that support the mobile data collection, I define two types of nodes: member nodes and head nodes. The head nodes are within the transmission range of mobile sink (MS), while the member nodes can only be reachable by the head nodes. In uncontrollable predictable mobility, I assume that the route of the mobile collector (MuCar) is known, and those head nodes are determined using the LBC-DDU algorithm, an algorithm that I will detail in Chapter 6.

In order to continue, a working definition of a *polling sector* is in order. The *polling sector* is a range in which the MuCar can communicate with head nodes and receive data while it is on the move. The polling sector length at head node H_i can be measured by the distance between the starting point, where MuCar contacts with H_i, and the ending point before it disconnects with H_i. The location of each polling sector is known once the head nodes are selected.

In the predictable controllable systems, I will present a clustering protocol to choose the cluster head nodes and find the efficient route and locations in order to upload the data to the mobile sink. The trajectory plan is determined by finding the most efficient polling points to upload the data. The polling points are locations where the MuCar needs to stop and retrieve the data from the nearby sensors. Cluster members transmit their data to the nearby head nodes where the data is stored. Each member node picks one head node to send their data. Once MuCar initiates a contact with the cluster heads, they send their aggregated data along with their own data to the mobile collector.

My goal is to improve the efficiency of sending the data to the mobile collector, improve performance, and increase the network lifetime.

3.2 Channel Model

LLN nodes possess radios that have single transceivers and are capable of supporting multi-channel communication protocols [20]. The sink, on the other hand, has multiple radios that allow it to receive data from head nodes concurrently through different channels. This multiple radio setup is used to prevent packet collisions and reduce uploading latency. By enabling multi-channel networking, head nodes in a neighborhood can send packets simultaneously without causing collisions, as long as they use different channels (λ_i). The gap between any two frequencies is large enough so that the interference is eliminated, meaning that channel λ_i does not interfere with λ_j, where $i \neq j$. Successful use of multi-channel communication requires that senders and receivers agree upon which channels to use at which time, and their agreement requires a channel scheduling policy. The advantage

of this approach is that any pair of nodes that use different channels will not interfere with each other, thus avoiding the hidden terminal problem.

Transmitting over multiple channels decreases latency and increases throughput. In mobile systems, the head nodes must use different frequencies by a deliberate policy of synchronized channel switching that occurs while uploading data to the multi-radio mobile sink [10]. The channel switching is needed when the number of head nodes at the uploading position are larger than the number of available radios at the mobile sink.

One of the primary challenges in engineering LLN systems is network reliability for successful transmissions, especially because they have lossy communication links and low data rates. To deal with these issues, many researchers have advocated the use of a control plane and describe network management schemes that are *epoch*-based [11]. The epoch divides time into a set of discrete points during which network control decisions are made. The epoch is used to ensure stable routing trees or link transmission schedules and can be implemented by a number of schemes, including the one presented in [70]. As will be seen in the next chapter, I use an epoch based approach to coordinate channel assignments and ensure that nodes can overhear the transmissions of other nodes in static tree-based WSN networks.

3.3 Data Model

The application data, which has been processed, transmitted and stored in my research, arises from sensors that sample the environment. I am assuming a non-splitting data flow system, which means that the data from a node has to be sent to its dedicated neighbored node without dividing the data into multiple receivers.

The role of the sink is to absorb sensed data. In general, sensed data can be streamed or archived and retrieved by a sink at a later time. In static Convergecast systems, the data is streamed such that the relayed nodes take the data and send it to the next hop until it reaches the base station or the sink. Meanwhile, in mobile applications, the head nodes archive the data that they receive from member nodes until the mobile sink arrives

and upload the archived data.

In my dissertation research, I assume sufficient memory for both buffering received data and for the head nodes ability to archive data. Provisioning sufficient memory is the task of the network administrator at deployment time.

3.4 Sink Model

In the system that I have designed, there are three types of sink trajectory managements: static sink, uncontrollable predictable, and controllable predictable mobile sinks. The static sink is a powerful base station that has multiple radios to concurrently receive data from children nodes using different transmitting channels. Multiple static sinks are exploited to study the impact of increasing the number of the sinks and the number of paths to investigate the protocol performance scalability.

The uncontrollable predictable sink traverses the network in a predetermined path with a known in advance arrival schedule and without motion control. It is also fully equipped with multiple radios. Because of the uncontrollable predictable sink's trajectory, it receives data while on the move.

On the other hand, the controllable predictable sink crosses the network to collect the data from the distributed clustered nodes with the best locations in order to upload the data to where trajectory and motion are controlled by the system configuration. Like the other sink models, it has multi-radio to receive data from cluster heads.

The three following chapters present my research work. Chapter 4 proposes multi-channel network coding for single and multiple static sinks in tree-based WSN systems. Chapter 5 targets the uncontrollable predictable mobility gathering systems and provides heuristics for latency optimization. Chapter 6 frameworks the controllable predictable mobility scheme and utilizes the uploading locations.

Chapter 4: Multi-Channel Network Coding for LLN Convergecasting

This chapter presents MuCode, my Convergecast multi-channel network coded protocol. The MuCode protocol uses a set of graph construction algorithms that support synchronized channel switching to take advantage of wireless eavesdropping for multi-path network coded packet delivery. I will show how to formulate this procedure as an optimization problem in order to minimize the maximum data rate for multi-sinks Convergecast non-split network flows. At the end of the chapter, I offer an experimental evaluation of MuCode. The performance of the paper's heuristic optimization has been evaluated against the optimal shortest path *OSP* and other techniques for multi-sinks Convergecast.

4.1 MuCode Protocol

MuCode is executed in a centralized fashion by the base station before the start of each epoch. This system relies on protocols for Convergecast tree construction, such as RPL [9] and network management mechanisms [70], in order to enforce stable tree and channel selection during an entire epoch. MuCode takes advantage of some of the tree-based channel assignment algorithms presented in [3]. MuCode has five steps that need to be executed:

1. Transform the Convergecast tree to an Interference Graph.
2. Assign height metrics to each node in the Interference Graph.
3. Assign a particular channel for each node within a subtree.
4. Find and eliminates any hidden terminal problem.
5. Synchronize packet transmitting and eavesdropping opportunities.

Table 4.1: MuCode Notations Table

Notation	Meaning
V	Set of nodes on the network.
n	Total number of nodes.
SR	Total number of source nodes.
E	Set of edges in the Interference tree IG.
ζ	Set of edges in the Convergecast tree CG.
BS	Single root of the tree-based network.
K	Number of children BS has.
K_{min}	Minimum number of children a sink has.
Λ	Number of available channels.
ST_λ	Subtree that is assigned to channel λ.
Max_Height	Largest shortest number of hops between BS and a node on the network.
S_i	Node i.
R_i	Sink i.
γ	Data rate in any source node.
$\overline{\gamma_{i,j}}$	Data rate load for source node i at node j.
Ł(u)	Accumulated data rate in node u.
$\Im(u)$	Set of targeted sinks that node u has to forward the data to.
N'	Set of all nodes except the sinks.
d	Graph diameter.
pr	Available number of parents.
p-joint	Number of p sinks a node can forward the data using the shortest path.
$T(lvl, R_i)$	Set of nodes that reside at height $= lvl$ in respect to the tree constructed by R_i.
$\delta(R_i)$	Current transmitting level to the sink R_i for the multi-sink communication.
R	Encoded packet replication.

I now describe each of these steps. In step 1, I take the Convergecast tree (CG) and produce an Interference Graph (IG) using the algorithm *CTI*. IG is a graphical representation of how nodes wirelessly interfere with each other. The CG has a set of nodes V and the set of edges ζ. Initially, CG is transformed to $IG = (V, E)$ where $E \equiv \zeta$. Then, for each v node, the interfered edges that have not been discovered in CG are added to the set E with the use of the neighbor information that is provided to the base station during routing activities, such as those reported by RPL, a routing protocol for LLNs. An example of this

transformation is depicted in Figure 4.1, where the graph in (a) is transformed to the graph in (b).

Algorithm 1 Convergecast to Interference (CTI)

Input: Convergecast Graph $CG = (V, \zeta)$
$\grave{V} \leftarrow V, \quad E \leftarrow \zeta;$
while $\grave{V} \neq \phi$ **do**
  $v \leftarrow DEQUEUE(\grave{V});$
  for each $u \in NB(v)$ **do**
    if $(v, u) \notin E$ **then**
      $E \leftarrow E \cup \{(v, u)\};$
    end if
  end for
end while
Output: $IG \leftarrow (V, E);$

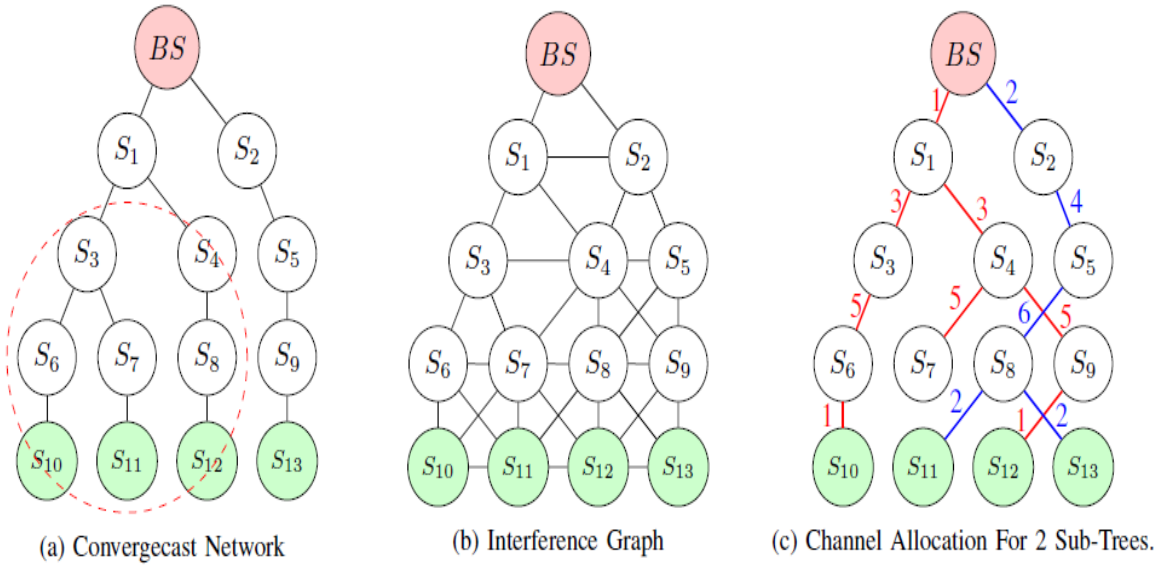


Figure 4.1: An arbitrary Convergecast network

In step 2, each node in the IG tree is assigned a particular height and a parent set, using a Breadth-First-Search Fat Tree (BFT) algorithm [3]. The BFT algorithm uses Breadth First Search (BFS) to determine a node's height in IG, which starts with the height of the

root BS, eventually being initialized to 0. A node u adjusts its height to its neighbor's (v) height plus one if, and only if, v is in a lower level than u's level. Then, u adds node v into its parent list. u's height represents the minimum number of hops from u to the root BS, while Max_Height is defined as the largest number of hops in the shortest path between node q and root BS, where q is a node that is the farthest from BS in IG graph.

Algorithm 2 Breadth-First-Search Fat Tree (BFT)

Input: Interference Graph $IG = (V, E)$
for each node $u \in IG$ **do**
  $height(u) \leftarrow MAX\_INT$;
  $parent(u) \leftarrow \phi$;
end for
$Q \leftarrow \{BS\}$, $height(BS) \leftarrow 0$;
for each node $u \in Q$ **do**
  for each $(u, v) \in IG$ **do**
    if $height(v) > height(u)$ **then**
      $height(v) \leftarrow height(u) + 1$;
      $parent(v) \leftarrow parent(v) \cup \{u\}$;
      $Q \leftarrow Q \cup \{u\}$;
    end if
  end for
end for
Output: Fat Tree IG;

As for step 3, I modify the greedy Partition Maximum Intra-Tree algorithm ($PMIT$) [3]. PMIT is an algorithm that partitions the IG fat tree into K vertex-independent trees, with minimal intra-tree interference between these trees. The bottleneck in PMIT is limited to the number of children K that the root BS has. It can be constrained as $K \leq \Lambda$ where Λ is the total number of available channels. By adding the root BS to each sub-tree, the algorithm initializes the sub-tree's construction ST_λ, where λ is the channel assigned to the root's child that belongs to the subtree ST. Then, it initializes the channel for each node to the default channel λ_0. Next, it scans the IG level by level from low to high. On each level, the nodes are sorted by their number of parents in ascending order. A condition stated in the algorithm is that the node u is added to a sub-tree ST_λ if, and only if, it keeps the tree connected and has minimum intra-tree interference with nodes within ST_λ. Last, the algorithm assigns channel λ to node u and picks a parent v for node u so that both u and

v are connected to ST_λ. The time complexity of the greedy $PMIT$ algorithm is $\mathcal{O}(d$ x K x $n^2)$, where d is the graph diameter, K is the number of children the root BS has, and n is the total number of nodes.

Algorithm 3 Greedy PMIT

Input: Fat Tree Interference Graph $IG = (V, E)$
for each channel $\lambda \in \Lambda$ **do**
  $ST_\lambda \leftarrow \{BS\}$;
end for
for each node $u \in V$ **do**
  $channel(u) \leftarrow 0$;
  $parent(u) \leftarrow \phi$;
end for
$level \leftarrow 1$;
repeat
  $node\_list \leftarrow \{u \mid height(u) = level \wedge channel(u) = 0\}$;
  Sort the $node\_list$ by the number of parents in ascending order
  for each node $u \in node\_list$ **do**
    Find ST_λ such that it keep connected after adding node u and has the minimum interference.
    $ST_\lambda \leftarrow ST_\lambda \cup \{u\}$;
    $channel(u) \leftarrow \lambda$;
    $parent(u) \leftarrow v$ where v is the parent node that connects ST_λ after adding u;
  end for
  $level \leftarrow level + 1$;
until $level > Max\_Height$;
Output: K sub-trees;

For step 4, I note that [3] proposed a Tree-based Multi-channel Protocol ($TCMP$) for the Convergecast multiple channels, which solves the channel assignment problem. They have constructed K sub-trees for K channels to eliminate the inter-tree interference. However, their focus was on minimizing the intra-tree interference for each sub-tree, whereas my focus is on minimizing the inter-tree interference within each sub-tree.

I now define the Channel Assignment Within Sub-Trees algorithm $CAST$, which considers each ST individually. $CAST$ scans the sub-tree ST_λ level by level from low to high and assigns different channels for nodes that reside on the same ST_λ. Consequently, closed-related nodes with distinct subtrees will never send data over the same channel; besides, such nodes should not worry about the hidden terminal problem. The Convergecast flow

Algorithm 4 Channel Assignment Within Sub-Trees (CAST)

1: **Input:** K sub-trees are constructed using $PMIT$, where BS is the common root. Each sub-tree is assigned to unique channel λ.
2: $level \leftarrow 2$;
3: **for** each sub-tree ST_λ **do**
4: **repeat**
5: Pick node u such that height$(u) = level$;
6: **if** $level \leq 3$ **then**
7: $channel(u) \leftarrow parent(u).channel + K$;
8: **else**
9: $channel(u) \leftarrow channel(v)$, where:
 $u, v \in ST_\lambda$ and height(u) = height(v) + 3 ;
10: **end if**
11: $level \leftarrow level + 1$;
12: **until** $level > Max\_Height$ of ST_λ;
13: **end for**
14: **Output:** Multi-Channel Tree MCT graph;

is up-linked from nodes located in the highest level to the root where the opposite down-linked flow is ignored by nodes. The children nodes of the root BS (i.e. nodes with height level = 1) do not modify their channel after they have been assigned in $CAST$. If the node is within the first three levels, then the channel is assigned to the parent's channel + K. Otherwise, the node will use the same channel that has been allocated for the grandparent of its parent (i.e. three levels upward). The first three levels decide the channel assignment pattern in the whole tree. More clearly, the channel of node u at level l in ST_λ has the same channel as all other nodes at levels $l \pm 3, l \pm 6, ...$ in ST_λ. This allocation process guarantees two-hop collision avoidance within the same ST, as well as collision avoidance between the sub-trees. The time complexity for the $CAST$ algorithm is $\mathcal{O}(d \text{ x } K \text{ x } n)$. Step 3 in the algorithm runs for K sub-trees, while step 4 takes the loop iteration along the graph diameter d. Step 5 may run at most n times, choosing nodes at a given level. Figure 4.1 shows an example of MuCode tree construction and channel allocation for Convergecast WSN.

The last step of MuCode is the algorithm shown in algorithm 5, the Synchronized Eavesdropping in Multi-channel Sub-trees ($SEMS$). This algorithm is responsible for synchronizing the sending and eavesdropping of packets in MuCode protocol. Initially, nodes

Algorithm 5 Synchronized Eavesdropping in Multi-channel Sub-trees (*SEMS*)

```
1: Input: MCT graph with allocated channels.
2: Initialization: All nodes at level = l are synchronized and scheduled in transmitting
   and receiving from each other in time slot t, using default channel λ_0.
3: h ← Max_Height , Round ← 1;
4: repeat
5:    for each node u at level h do
6:       Send a synchronized control packet to node v at level h − 1 in channel λ_0;
7:    end for
8:    h ← h − 1;
9: until h = 2;
10: for each node u do
11:    for each Round do
12:       repeat
13:          if (PTS(u, t) = true) then
14:             u sends at channel λ;
15:          else
16:             u switches to channel β to overhear packets from synchronized neighbor;
17:          end if
18:          t ← t + 1;
19:       until Next transmission;
20:    end for
21: end for
22: Output: Synchronized transmitted packets in epoch-based;
```

at level l are scheduled synchronously in transmission, using time slot basis and round-robin scheduling to assure transmitting fairness. Each node sends a number of packets per *Round* session. For example, if node u sends its first packet at the time of t_1, then node v can also transmit at the same time t_1, as long as both nodes are on the same level l, and v is three-hops away from node u. Using round-robin scheduling, node u's second transmission takes place when all other nodes that reside on the same level send their first packet.

Node u can listen to v if the height(v) $\geq$ height(u) (upward-link). I define a Permission-To-Send PTS(u, t) primitive for u at level l. PTS(u, t) is TRUE if, and only if, the following three conditions are satisfied: (1) the nodes at level $l - 1$ and $l + 1$ are not sending, (2) u's transmitting queue is not empty, and (3) it is u's turn to send at t based on round-robin scheduling mechanism. If PTS(u, t) is satisfied, u can transmit its packet otherwise it switches its receiving channel to the scheduled channel β in order to overhear packets from synchronized neighbors, which increases the chance to encode.

Here, a node u at level l switches the receiving channel to β if its neighbor v at level l or $l-1$ transmits at channel β. Steps 3-9 allow node v to understand its neighbors transmission pattern in order to maximize the opportunity of overhearing. For example, in figure 4.1 (c), node S_7 can switch its receiver channel β to 1,2,5 or 6 and transmit at channel $\lambda = 5$ based on the $CAST$ algorithm. Node S_{11} sends control packets to S_7 at the default channel λ_0 while telling S_7 what time slots S_{11} is going to transmit in what channel and what level S_{11} is actually located. For high efficiency transmission, for each transmission round, nodes at level l can send if, and only if, all nodes at levels less than l have completed transmitting at the same round. This transmission process allows nodes at levels $l,\ l-2,\ l-4, ...$ to send concurrently without collisions. This is a special case for Convergecast wireless networks. The time complexity of this algorithm is $\mathcal{O}(Round$ x $n)$. The initialization in step 2 implements round-robin scheduling for n nodes. Step 4 allows all nodes, except BS, to send control packets for synchronization between transmitting and overhearing processes where the running time is $\mathcal{O}(n)$. Step 12 runs for all of the nodes n in the graph, while each node sends $Round$ packets per session.

4.2 Multi-Sink

Section 4.1 solved the multi-channel network coding problem for single sink Convergecasting. This section applies my approach to the case of multiple sinks. In this circumstance, the system is configured to send data from shared source nodes simultaneously to the multiple sinks despite how far they are geographically located from the sources. This means that some sinks can partially share the path with others, which can significantly reduce the packet delivery latency and reduce the data load on the bottleneck nodes. Note that the simple solution of treating the system as a single multi-hop routing for each sink will increase the overall packet latency, energy consumption, and decrease the network lifetime. For instance, suppose a network with three sinks R_1, R_2 and R_3 wants to receive data from shared source nodes with priority. Using single multi-hop tree-based routing for each sink, sink R_3 has to wait until sink R_2 receives all the data, which consequently has to wait until

sink R_1 get all the data. A simple and straightforward communication is that each node broadcast its packets to the neighboring nodes. Then, the nodes that receive these packets will re-broadcast the packets to the neighboring nodes until the sinks receive all the packets. This scenario is also called flooding, which creates severe packet redundancy, collision, and network contention including impacting the network lifetime.

Therefore, by designing a new method for the multi-sink scenario, we achieve a higher level of efficiency by taking the advantage of packet transmission through shared subtrees. Some of the MuCode protocol steps need to be changed, especially those that reflect the impact of using multi-sink. The MuCode protocol for multi-sink now consists of six steps:

1. Transforms the Convergecast tree to an Interference Graph.
2. Assigns height metrics for each node within the interference graph for each sink within the BFT algorithm.
3. Construct the parent-child tree with subtrees setups.
4. Assigns a particular channel for each node.
5. Finds and eliminates any hidden terminal problem.
6. Synchronizes packet transmitting and eavesdropping opportunities.

4.2.1 Multi-Sink Tree Construction

The first step in multi-sink MuCode protocol is the same as the first step for the single sink. In step 2, I use algorithm 6 to construct the heights and the trees for multiple sinks. The goal of *MTS* is to come up with an approximation approach that minimizes the data rate load (γ) on each node on the network. I assume that each source node S_j has a data rate of γ_j. In the beginning, each nodes – except the source nodes and the sinks data rate Ł is initialized to 0. I define Ł(u) as an indication of how much data rate a node u has per epoch transmission. The source nodes also have S height variables = 0, which reflect the

node's height level corresponding to the S sinks. $\Im(u)$ is the set of targeted sinks that node u has to forward the data to.

Now, each sink R_i uses the BFT algorithm to assign heights to each node on the network. Starting from the shared sources until the sinks' children, it is necessary to select carefully which parents to use since the data flow cannot be split, and the data rate load at a parent node is always greater than or equal to its child. I define candidate parents of node u - candidate_parent(u) - as the set of nodes where u's shared heights are greater than to all the shared heights on these nodes, and each of these candidate parents shared at least one targeted sink with u. Initially, the data rate load at source node S_j is equal to the data rate load of S_j (γ_j). Intuitively, the targeted set for each source $\Im(S_j)$ includes all the S sinks – $R_1, R_2, \ldots, R_S$ – since the sources are shared between the sinks.

I define $path(u)$ as the set of sinks that node u can forward the data to using the shortest path BFT. The parent-child assignment procedure is enforced by iterations level by level, starting from the leaves up to the sinks' children. The set Q has the nodes that reside on the current iteration level.

Algorithm 6 Multi-sink Tree Setup (MTS)

1: **Input:** The Interference Graph IG;
2: **Initialization:** for every node $u \in N'$, Ł$(u) \leftarrow 0$, $\Im(u) \leftarrow \phi$ and $u.height(R_i) \leftarrow 0$ for every sink R_i in the network.
3: Each R_i of S sinks uses breadth first search fat tree [3] algorithm to the shared sources.
4: **if** node u within the BFT path of sink R_i **then**
5: $\quad u.height(R_i) \leftarrow$ level of u in R_i tree;
6: **end if**
7: For each source S_j, Ł$(S_j) \leftarrow \gamma_j$ and $\Im(S_j) \leftarrow \{R_1, \ldots, R_S\}$;
8: $path(u) \leftarrow \{R_i | \forall i \in S \wedge u.height(R_i) \neq 0\}$;
9: $height \leftarrow max\_height$;
10: **repeat**
11: $\quad Q \leftarrow \{u | \forall u \in n \wedge u.max\_height = height\}$;
12: $\quad Parent\_Assignment(Q)$;
13: $\quad height \leftarrow height - 1$;
14: **until** $height = 2$;
15: **for** all nodes in level = 1 **do**
16: $\quad$ Assign all neighbored sinks as their parents;
17: **end for**
18: **Output:** Multi-Sink Tree MST graph;

The Parent_Assignment procedure (PA) is described by algorithm 7. In the beginning,

PA sorts the set Q that are based on priorities, and these priorities are the size of the candidate parents and the current data rate load in Q nodes. Iteratively, *PA* extracts the node with the highest priority and finds its parent(s) until every node in Q is assigned to his parent(s).

Algorithm 7 Parent_Assignment(Q)

1: **Input:** The set Q from Algorithm 6;
2: Sort node $q \in Q$ based on the number of its candidate parents in ascending order, then based on Ł(q) in descending order;
3: **repeat**
4: $q \leftarrow EXTRACT(Q)$
5: **if** ($\exists w \in$ candidate_parent(q) AND $path(w) \equiv \Im(q)$ AND Ł(w) is the smallest) **then**
6: q picks w as parent and $\Im(w) \leftarrow \Im(q)$;
7: Ł(w) $\leftarrow$ Ł(w) + Ł(q);
8: **else**
9: q picks set of nodes W as parents where W are candidate parents of q, $\Im(q) \subseteq path(W)$, Ł(W) is smallest possible, and $|path(W)|$ is smallest possible;
10: Sort W based on each element path size in descending order;
11: $covered \leftarrow \phi$;
12: **repeat**
13: $v' \leftarrow EXTRACT(W)$
14: $\Im(v') \leftarrow path(v') - (path(v') \cap covered)$;
15: $covered \leftarrow covered \cup \Im(v')$;
16: Ł(v') $\leftarrow$ Ł(q) + Ł(v');
17: **until** W is empty;
18: **end if**
19: **until** Q is empty;
20: **Output:** Each node is load balance assigned to parent node(s);

The criteria for selecting the parent node w of node u is based on whether w is a candidate parent of u. The path that w leads to is the same as the targeted sinks of node u, and the current data load in w is the smallest among the candidate parents of u. If the current data load is the smallest, then w copies the targeted set of $u's$ and adds the data rate load in u to its current data load.

Otherwise, u selects the set of candidate parents of W as its parents as long as the target set of u is the subset of the path that W leads to; the data load per node in W also must be as small as possible, as does the size of the path set of W. In this case, *PA* sorts W based on the size of the path from largest to smallest in order to reduce the load of the other common path nodes. Iteratively, *PA* extracts v' from W and removes the covered paths by

other extracted nodes in W and assigns the uncovered path set to the targeted set of v'. Then, it updates the covered set and adds the data rate load to v'.

The algorithm *MTS* iteratively uses procedure *PA* until reaching the sinks' children. Intuitively, these children choose their sinks as their parents. The asymptotic time for *MTS* algorithm is $\mathcal{O}(2^{pr}$ x d^2 x $n \lg n)$, where d is the graph diameter, n is the total number of nodes, and 2^{pr} is the combination of candidate parents per node. I assume the number of candidate parents of pr is small since they are chosen based on the shortest path that excludes other non-relative neighbored nodes. This is a reasonable assumption in LLNs. Otherwise, the algorithm complexity becomes exponential.

Algorithm 8 Multi-sink Subtrees (*MSS*)

```
1: Input: Multi-Sink Tree MST graph;
2: Initialization: for every node u with height ≥ 1, u.ST_list ← Φ.
3: for each node u where u.height = 1 do
4:    u.ST_list ← unique subtree {⊣_u};
5: end for
6: for each node sink R_i do
7:    for height = 2 to max_height(R_i) do
8:       if node v's height_list(R_i) = height then
9:          v.ST_list ← v.ST_list ∪ g.ST_list,
10:         where g is the parent of u in R_i's tree;
11:      end if
12:   end for
13: end for
14: Output: Multi-Sink Sub-Tree MSST graph;
```

4.2.2 Multi-Sink Subtrees

The third step in multi-sink MuCode is assigning the subtrees to each node on the network using algorithm 8. At first, each sinks' child u assigns a unique subtree $\dashv_u$, even if u has more than one parent sink. The process of assigning subtrees to the remaining nodes on the network is starting from height = 2 from each sink R_i, until the maximum height is at R_i tree $max\_height(R_i)$. Each node at that level assigns to his parent's subtree list. I define the *p-joint node* as a node that shares shortest multi-path communication for p sinks. A node may have more subtree lists than his parent's subtree list, which is the case of a p-joint node in relation to $p \geq 2$. The running time for *MSS* algorithm is $\mathcal{O}(n)$.

4.2.3 Channel Assignment

In the fourth and fifth step, each node must be assigned to channels in order to eliminate hidden terminal problems and minimize the intra-tree interference. I developed an algorithm called Multi-sink Channel Assignment (MCA) to do these steps for the multi-sink case in a centralized approach. In the initialization phase, each node has set its channel to 0. The algorithm iteration is starting from the highest leveled nodes based on their maxed level in their height_list. On each iteration, set L has all the nodes that reside on the highest level that have not been investigated. It picks the first available node u from L so that it has the highest p-joint value and high 1-hop and 2-hops density when compared to the nodes in L. To guarantee reusing the channels, $u.channel$ is assigned to channel λ if, and only if, 1-hop and 2-hop neighbored nodes have not been assigned by channel λ, and λ is the smallest possible choice from the set of available channels Λ. The iteration continues until reaching the sinks. The asymptotic time for MCA algorithm is $\mathcal{O}(d \text{ x } n \text{ x } E^2)$, where E is the number of edges in the graph.

Algorithm 9 Multi-sink Channel Assignment (MCA)

1: **Input:** $MSST$ graph.
2: **Initialization:** for every node u, $u.channel = 0$;
3: $height \leftarrow max\_height$;
4: **repeat**
5: $L \leftarrow \{u | MAX(height\_list) = height\}$;
6: Sort L based on $|height\_list|$ and the number of 1-hop and 2-hop neighbors;
7: **repeat**
8: Remove node u from L;
9: **if** $(u.channel = 0)$ **then**
10: $u.channel \leftarrow \lambda$ s.t. $\lambda \in \{1, 2, \ldots, \Lambda\}$, no 1-hop nor 2-hop neighbors of u has assigned with channel λ, and λ is smallest possible choice;
11: **end if**
12: **until** L is empty;
13: $height \leftarrow height - 1$;
14: **until** height = 0;
15: **Output:** $MSST$ graph with allocated channels.

4.2.4 Synchronized Multi-Sink Transmission

The last step of the MuCode process is synchronizing the packets that are transmitting and eavesdropping in a multi-sink network, where Algorithm 10 describes my method to implement it. For each sink R_i, the variable $\delta(R_i)$ is initialized by the maximum height at the tree of R_i ($max\_height(R_i)$). The set $T(lvl, R_i)$ is defined as the set of nodes that reside at height = lvl on the tree of the sink R_i. I define the set Q as the values that represent the maximum heights of all the sinks, with set $\mp$ as the iteration set for each level of each sink. The iteration process starts at the maximum height level in the set Q and stops at the sinks' children nodes. If all of the nodes in a tree of sink R_i, at the current level, have already transmitted their data, then its current level decreases until there are some nodes that have not been sent yet, or the current level reaches the sink. If the initial transmitting level set Q and the current transmitting level set $\mp$ are not equal, and there is no node in $C(Q)$ that has a neighbor that belongs to one node in $C(\mp)$, then the next transmission generation can be initiated. If there is no child-parent nodes in the current transmitting set, then apply the SEMS algorithm for synchronized transmission. Otherwise, the children have to transmit the rest of the nodes separately using SEMS. Once the transmission is done at that level, $\mp$ will decrease its current level by one for each sink R_i. The running time for *SEMMS* algorithm is $\mathcal{O}(d \text{ x } n \text{ x } \lg n)$.

4.3 Minimizing Data Rate Load

As previously mentioned, multi-sink Convergecast, with shared sources, creates partially shared multi-path subtrees from different source nodes. These subtrees hold the shared data flow until the path is split into two or more sub-paths and the flow is carried for each sub-path to reach its designated sink. It can be observed that relying only on the shared paths can cause traffic bottleneck, thus increasing the packet delivery latency, reducing the network lifetime, and potentially causing network congestion.

In this case, the p-joint nodes carry most of the traffic loads, where they suffer persistent

Algorithm 10 Synchronized Eavesdropping in Multi-channel Multi-Sink (*SEMMS*)

1: **Input:** $MSST$ graph with allocated channels.
2: **Initialization:** for each sink R_i, $\delta(R_i) \leftarrow max\_height(R_i)$;
3: **Initialization:** for each sink R_i, $T(lvl, R_i) \leftarrow \{u \mid u$ is a node within the path of sink R_i and $u.height = lvl\}$;
4: $Q \leftarrow \mp \leftarrow \{\delta(R_1), \ldots, \delta(R_S)\}$;
5: $C(\mp) = \bigcup T(\delta(R_i), R_i) \forall i \in \{1, \ldots, S\}$;
6: **for** ($height = max(Q)$ down to 1) **do**
7: **if** $T(\delta(R_i), R_i)$ already transmitted for each sink R_i **then**
8: **repeat**
9: $\delta(R_i) \leftarrow \delta(R_i) - 1$;
10: Update $\mp$;
11: **until** $T(\delta(R_i), R_i)$ has some sending elements or $\delta(R_i) = 0$
12: **end if**
13: **if** ($Q \neq \mp$ and nodes in $C(Q)$ have no 1-hop nodes in $C(\mp)$) **then**
14: Start new sending generation (step 6);
15: **end if**
16: **if** (no node in $C(\mp)$ is a child to another node in $C(\mp)$) **then**
17: Synchronize transmission in $C(\mp)$ using SEMS algorithm;
18: **else**
19: Synchronize transmitting the children in $C(\mp)$ using SEMS algorithm;
20: Synchronize transmitting the parents in $C(\mp)$ using SEMS algorithm;
21: **end if**
22: $\mp \leftarrow \{\delta(R_1) - 1, \ldots, \delta(R_S) - 1\}$
23: **end for**
24: **Output:** Synchronized packet transmissions to multiple sinks;

overload. The data rate transmission in p-joint nodes will be much higher compared to 1-joint nodes for $p \geq 2$. Therefore, balancing the data rate load among nodes is substantial for maximizing the network lifetime and decreasing the packet delivery latency.

4.3.1 Optimal Formulation

In order to obtain a better insight into this problem, I now describe how to solve it optimally. Let SR be the total number of source nodes, and let γ be the equal data rate of these sources for simplicity. I denote $\overline{\gamma_{i,j}}$ to be the data rate of the source i at the relay node j. I calculate the data rate load at node j as:

$$Ł(j) = \sum_{i=1}^{SR} \overline{\gamma_{i,j}} \tag{4.1}$$

To find the data rate of $\overline{\gamma_{i,j}}$ for each source, I define it in the following way:

$$\overline{\gamma_{i,j}} = \begin{cases} \gamma & \text{if } j \text{ shared the same tree with } S_i; \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

Let K_{min} be the minimum number of children for a sink, and let the set N' be the set of all nodes in the network excluding the sinks. I formulate the optimization problem of minimizing the data rate load as:

$$\min(\max Ł(j)) \qquad \forall j \in N' \tag{4.3}$$

subject to:

$$SR \,.\, \gamma \;\geq\; Ł(j) \;\geq\; 0 \tag{4.4}$$

$$\max Ł(j) \;\geq \left\lceil \frac{SR}{K_{min}} \right\rceil .\gamma \tag{4.5}$$

The minimum data rate load Ł(j) can be zero, while its maximum would never get more than all the traffic rate from the source nodes. The traffic flow cannot be split because it is only used for shared sources and because every sink needs this common data. Therefore, the traffic bottleneck resides in the sinks' children. The maximum possible data rate load in these nodes would never come below the data rate of the ratio between the number of source nodes and the minimum number of children for a sink.

Therefore, my optimization problem constraints will look like:

$$SR \,.\, \gamma \;\geq\; \min(\max \text{Ł}(j)) \;\geq\; \left\lceil \frac{SR}{K_{min}} \right\rceil .\gamma \qquad \forall j \in N' \tag{4.6}$$

Let $\overline{Y}$ be the maximum data rate load $\max$ Ł(j) on the set N'. $\overline{Y}$ is formally defined as:

$$\overline{Y} = \max \left\{ \sum_{i=1}^{SR} \overline{\gamma_{i,1}}, \ldots, \sum_{i=1}^{SR} \overline{\gamma_{i,N'}} \right\} \tag{4.7}$$

which means that:

$$\overline{Y} \;\geq\; \sum_{i=1}^{SR} \overline{\gamma_{i,1}}$$

$$\ldots$$

$$\overline{Y} \;\geq\; \sum_{i=1}^{SR} \overline{\gamma_{i,N'}}$$

Therefore,

$$\min \overline{Y} \;\geq\; \sum_{i=1}^{SR} \overline{\gamma_{i,j}} \qquad \forall j \in N \tag{4.8}$$

which gives the final optimization problem of minimizing the maximum data rate load:

$$\min \overline{Y} \;\geq\; \max_{\forall j \in N'} \sum_{i=1}^{SR} \overline{\gamma_{i,j}} \;\geq \left\lceil \frac{SR}{K_{min}} \right\rceil . \gamma \tag{4.9}$$

The complexity of this formulation is challenging because all possible permutations have to be investigated for all possible parent selections in the network in order to find the optimal solution, which is asymptotically exponential.

4.4 Heuristic

In order to balance the data rate load assignment among the nodes, my greedy heuristic *Parent_Assignment* algorithm chooses the parent node with lowest Ł(j) while simultaneously avoiding nodes with a relatively high data load rate as often as possible.

To evaluate my heuristic, I need to find and compare the optimal solution for minimizing the maximum data rate load in the network to MuCode heuristic. Cayley's formula, proved by [71], states that: "for any positive integer n, the number of all trees with vertex set[n] is $A_n = n^{n-2}$". Finding all of the permutations of Cayley's trees with a brute force approach can lead to the optimal solution. Since Cayley's formula is more general than my tree assumption, another approach to find an optimal solution for an ordered tree root is a brute force with a running complexity of $\mathcal{O}(b^n)$, where $b = \binom{pr}{S}$ and pr is the number of parents a node can have, while S is the number of sinks . Nevertheless, this computation is exponential. Therefore, my MuCode heuristic method tries to solve the problem in $\mathcal{O}(n \log n)$ running time, with the assumption that pr is small. Another competitive heuristic approach is using Breadth-First-Search for the shortest path from the root to the leaves, which runs in $\mathcal{O}(n + E)$. Another heuristic algorithm would be that each source node randomly chooses S paths to S sinks which run in $\mathcal{O}(SR \text{ x } n)$ where SR is the number of source nodes.

4.5 Evaluation

I have evaluated MuCode against Sensecode [37], one of the first network coding protocol for static Convergecast WSN with single sink and one channel. Sensecode synchronizes packets transmission, which provides nodes free-listening from neighboring transmissions without the need to perform channel switching.

4.5.1 MuCode Performance

In this section, I evaluate MuCode against Sensecode with two baseline schemes. The first scheme is No Coding without replication (called NC $R = 0$), and the second scheme is No Coding with single replication (called NC $R = 1$). No coding uses a single channel, with either one or two copies of the packet being sent. The experiments are conducted in 4X4, 6X6, and 12X12 grid networks, where the root BS is placed at the top corner and the sources are distributed at the farther corner from BS. In order to implement MuCode in these grids, the total available channels $\Lambda = 9$. I also evaluated a single chained network, the worst case topology for MuCode with 5, 8, and 12 nodes, where the source and BS are placed on opposite poles.

The reason why chained networks are the worst case is because nodes have no opportunity in overhearing packets from surroundings. Based on $CAST$, I reduced the number of channels Λ to 3 because K, the number of children of root BS, is reduced to 1. Packet Error Rate (PER) is the ratio of the number of not received packets or received in error– even after applying any error correction method– to the total number of packets that would have been received without any error. The usual PER in LLN is between 0% to 20%. While in noisy, high interference, and low Received Signal Strength Indication (RSSI) circumstances, the PER becomes relatively very high. These experiments used a variety of wireless PERs from 0% to 60%, in most cases, using a standard Random Linear Network Coding (RLNC) technique.

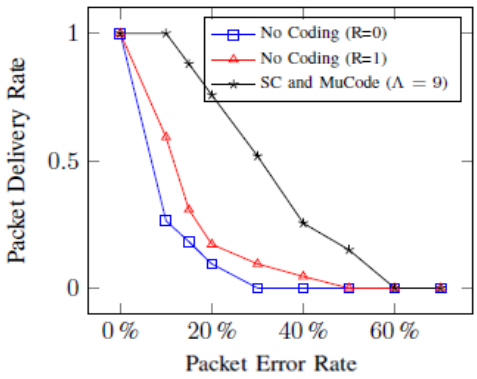
In the evaluation process, I am interested in several metrics: packet delivery rate (PDR),

throughput, latency, and *network power* [72]. Network power is defined as the ratio of network throughput and its average delay. Once the network gets higher throughput and lower delay, then its network power will increase. Network power allows architecturally independent comparisons between protocols and provides good insight to compare MuCode to the other protocols.

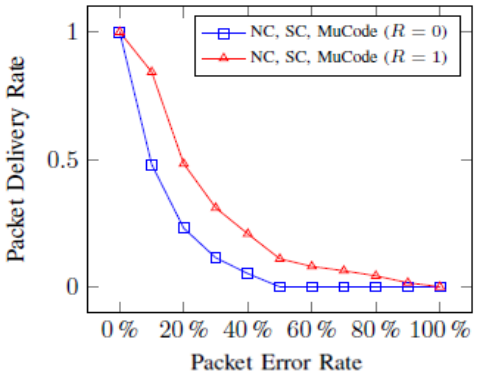
I used the Contiki operating system [73] to implement the protocols. Cooja, the standard simulation for tiny low-cost and low-power microcontrollers that runs in Contiki operating system, emulates the network nodes and its hardware platform in large-scale networks. I chose the T-Mote Sky mote because it can transmit 250 kbit/s using MSP430 microprocessor and CC2420 radio. The platform works in 10 KB of random access memory and 48 KB of program flash. I configured the radio medium to the Unit Disk Graph Medium Distance Loss with 50m transmission and interference ranges. In my simulation experiments, I assumed that each node sends a packet of size 50 bytes at time slot t, where the difference between two consecutive time slots is $\mp\, t = t_{i+1} - t_i = 100$ milliseconds per transmission. The simulation considers stochastic parameters such as generating random seeds for the same PERs and assign random data rate per node for the case of studying the multi-sink scenarios.

Packet Delivery Rate (PDR)

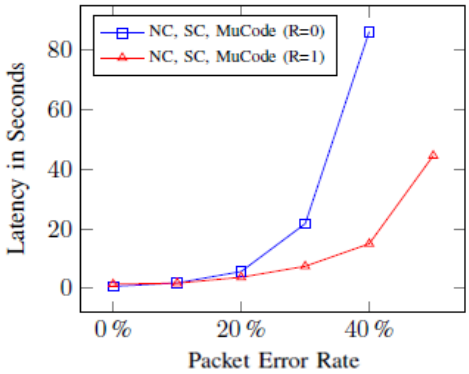
The metric Packet Delivery Rate (PDR) is the ratio of original packets that are successfully received to the sink over the total original sent packets. Figure 4.2 (a) shows the average packet delivery rate versus PER in grid networks. Sensecode and MuCode together have the same PDR since they use the same network coding technique. Both Sensecode and MuCode are more reliable in packet loss than NC $R = 0$ and NC $R = 1$ thanks to network coding. The redundant factor R leverages PDR in NC $R = 1$ comparing to NC $R = 0$. However, overhearing packets from neighbors, encoding, and retransmitting them would definitely lessen the packet loss. The nature of using network coding in reliable grid networks assumes that the total sending packets of all nodes reside at level l is less than or equal to the total sending packets of all nodes that reside at level $l - 1$.



(a) PDR Vs PER (Grid Networks)



(b) PDR Vs PER (Chained Networks)



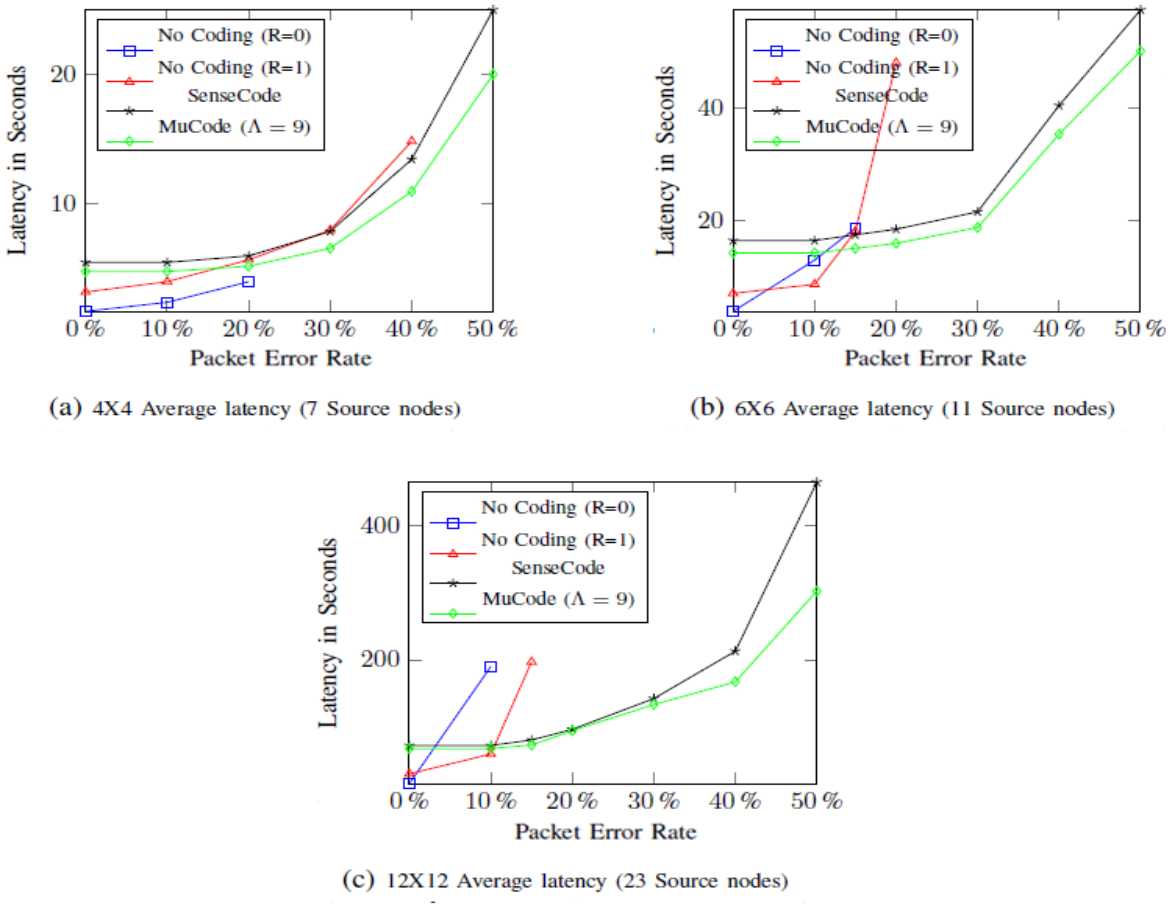
(c) Latency Vs PER (Chained Networks)

Figure 4.2: The average performance of PDR and Latency versus PER.

In a single chained WSN, the farther the distance between the source and the sink, the higher the chance for packet drops. Figure 4.2 (b) depicts the average packet delivery rate versus PER in single chained WSN with variance distances. The low packet delivery rate represents No coding, Sensecode, and MuCode with no replication. The high packet delivery represents those protocols all with single replication. Unfortunately, network coding has no advantage here because nodes have no chance to overhear packets from neighbors. Replication has the substantial influence to improve PDR in chained topologies, however.

Delivery Latency

Figure 4.2 (c) shows average packet delay versus PER in single chained WSN with variance distances. Again, nodes cannot overhear packets from neighbors but replicating the sending packets can help with PDR, as I have showed in the previous subsection. Therefore, retransmitting the lost packets would be cut down, which reduces the packets latency.



(a) 4X4 Average latency (7 Source nodes)

(b) 6X6 Average latency (11 Source nodes)

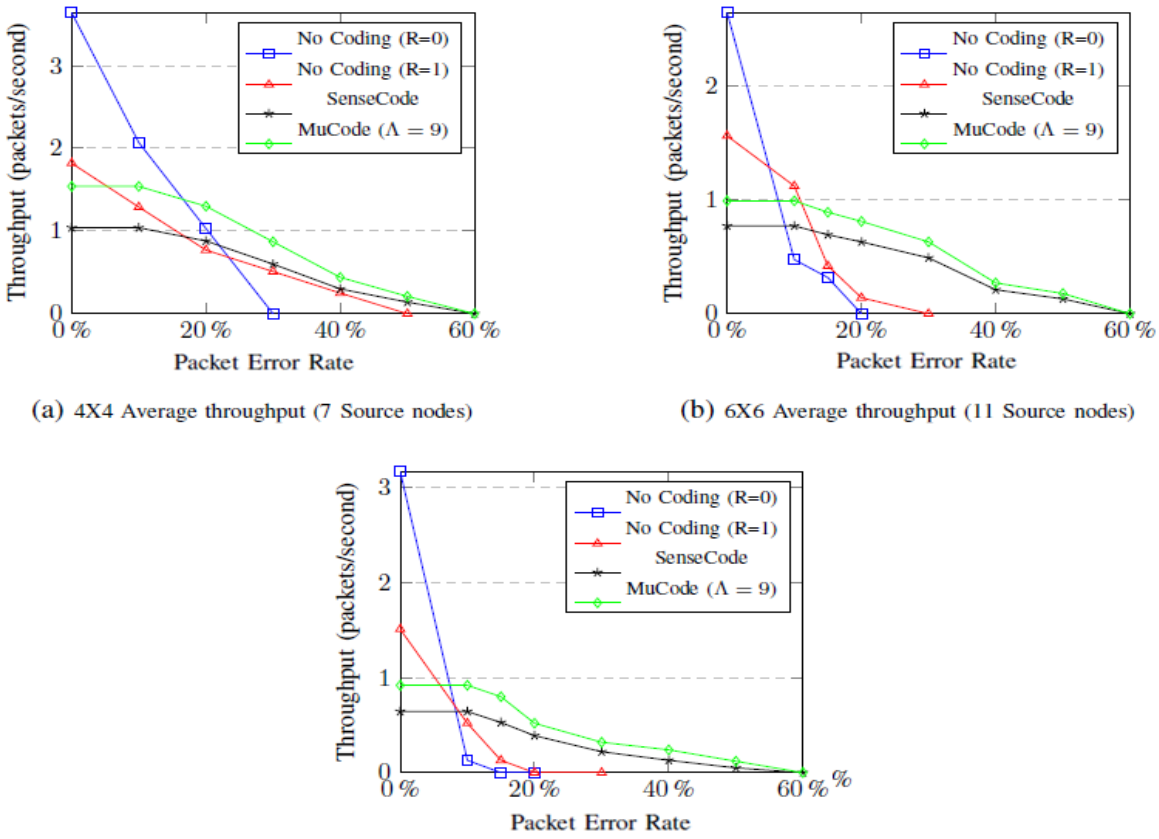
(c) 12X12 Average latency (23 Source nodes)

Figure 4.3: The average packet delay with different packet error rates in grid networks.

Figure 4.3 depicts the network delay versus PER in various densities for grid networks. In small grids, NC $R = 0$ has the lowest delay, when PER ranges between 0% and 20%. In relatively bigger grids, however, NC $R = 1$ conserves the lowest delay at PER = 10%.

Once PER gets higher in all of those cases, MuCode becomes the lowest in packet delay. Observe that the incomplete lines show that the delay is equal to ∞ seconds, which is a result of a packets loss, and they will never be delivered to the sink completely. Therefore, MuCode reduces the delay efficiently compared to Sensecode, when there is a chance of packets overhearing.

Network Throughput



(a) 4X4 Average throughput (7 Source nodes)

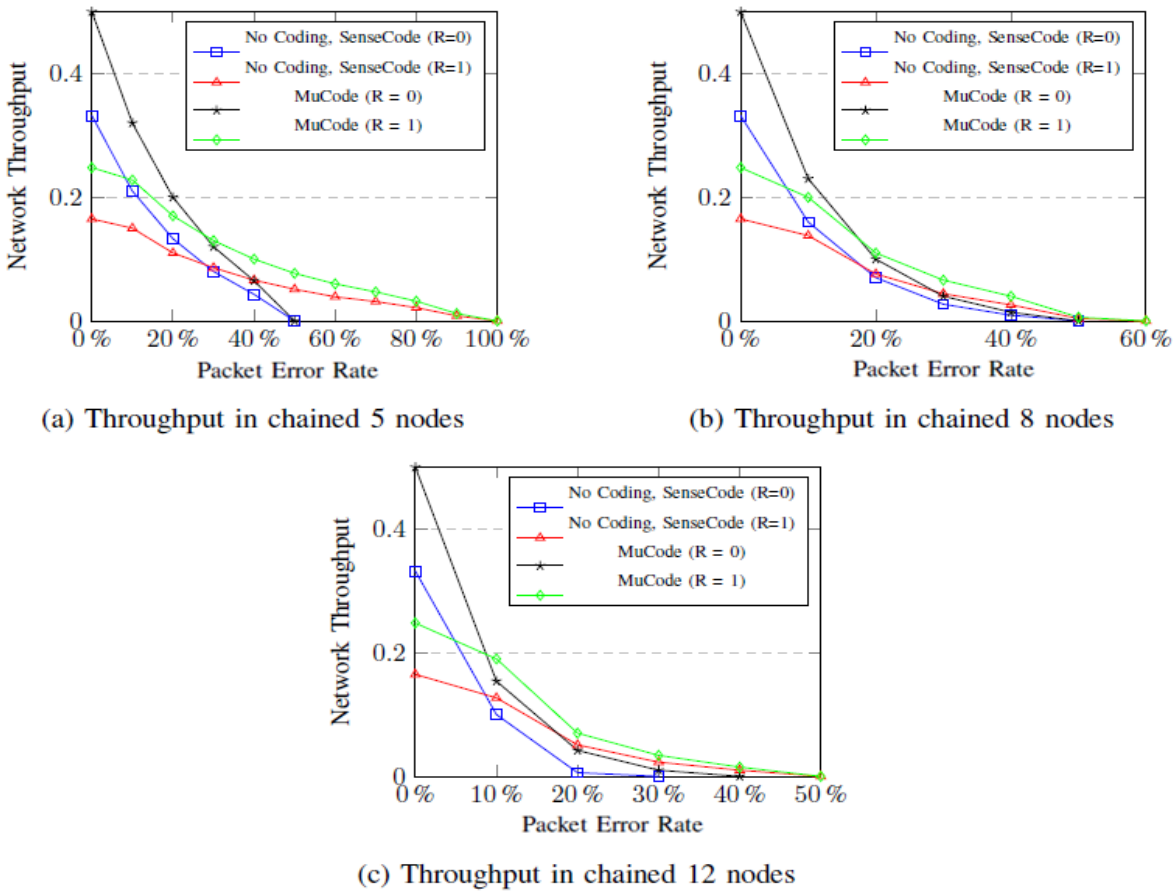
(b) 6X6 Average throughput (11 Source nodes)

(c) 12X12 Average throughput (23 Source nodes)

Figure 4.4: Average throughput with different packet error rates in grid networks.

Figure 4.4 depicts the grid network throughput versus PER in different densities. When the network error ratio is less than 20%, NC $R = 0$ is superior to the network throughput

in 4X4 grid. In more dense grid networks, NC $R = 0$ throughput would not be efficient in unreliable communications. NC $R = 1$ possess the highest network throughput in 6X6 grid at PER = 10%. Other than those cases, MuCode delivers highest network throughput comparing to all the others.



(a) Throughput in chained 5 nodes

(b) Throughput in chained 8 nodes

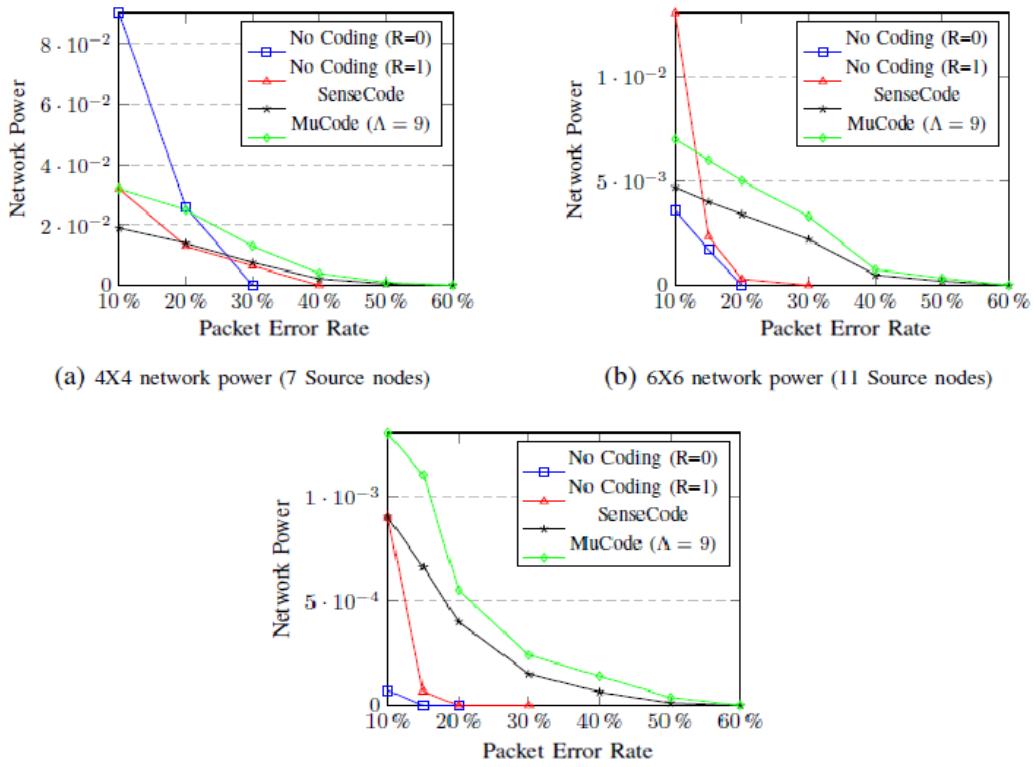
(c) Throughput in chained 12 nodes

Figure 4.5: Average throughput with different packet error rates in single chained networks.

Figure 4.5 depicts average throughput versus PER for the single chained networks in different densities. In 5 nodes dense, MuCode $R = 0$ has higher throughput when PER is from 0% to almost 30%. Once PER gets higher, MuCode $R = 1$ will have the highest throughput. Nothing is changed when the network density gets higher. However, the PER break point between MuCode $R = 0$ and MuCode $R = 1$ is reduced from 30% to 20%

and 10% in single chained 8 and 12 nodes respectively. Bottom line, MuCode outperforms Sensecode in throughput with or without end-to-end errors in chained networks.

Network Power



(a) 4X4 network power (7 Source nodes)

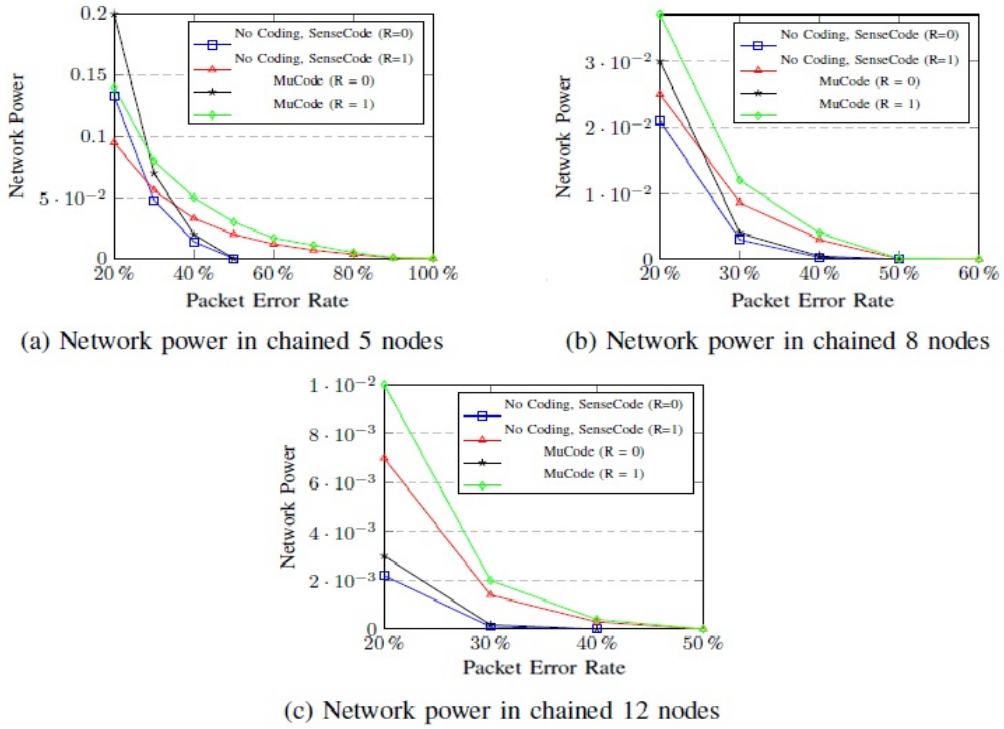
(b) 6X6 network power (11 Source nodes)

(c) 12X12 network power (23 Source nodes)

Figure 4.6: Average network power with different packet error rates in grid networks.

Figure 4.6 depicts grid network power versus PER in different densities. In small density, NC $R = 0$ has the highest network power when PER is less than 20%, which is true because the packet drops are low with high throughput. In medium density at 10% PER, NC $R = 1$ becomes the highest in network power. The reason for this superiority is because NC $R = 0$ drops huge packets compared to NC $R = 1$. Despite the fact that both Sensecode and MuCode send all the packets to the sink without retransmitting, their delay is still high

comparing to NC $R = 1$. Whereas in high density, MuCode, with nine channels, has a higher network power than the others. The chance of packet drops is very high in which Sensecode and MuCode can survive better. Nevertheless, MuCode throughput is higher, and its latency is lower compared to Sensecode.



(a) Network power in chained 5 nodes

(b) Network power in chained 8 nodes

(c) Network power in chained 12 nodes

Figure 4.7: Average network power with different packet error rates in single chained networks.

Figure 4.7 depicts single chained network power versus PER. MuCode $R = 0$ has higher network power when PER ranges between 0% to 10%. Nevertheless, the more PER there is, then the more network power there is in MuCode $R = 1$. Even though the algorithm requires a slight increase $O(d \text{ x } K)$ in computational complexity, the overall improvement in network power under certain circumstances justifies the complexity.

4.5.2 Latency Evaluation for Multi-Sink

In this evaluation section, I want to consider the impact of using MuCode heuristics for multi-sink systems by comparing it with the optimal technique and other rival methods in terms of packet delivery latency. The optimal technique, in this case, is called optimal shortest path with minimized maximum data rate method (*OSP*), which has the optimal latency. As previously shown, the optimal solution computation for multi-sink is asymptotically expensive. Hence, I am comparing it against the MuCode heuristics with relatively small size complexity by using the brute-force approach to obtain (*OSP*).

Another rival technique is the optimal minimal maximal data rate (*OMX*), which also can be found in the same process as obtaining (*OSP*). Essentially, *OMX* only considers minimizing the maximum data rate without taking into consideration the shortest path that provides the optimal solution for maximizing the network lifetime but with extra latency compared to *OSP*.

The breadth-first search (*BFS*) algorithm, where each sink runs (*BFS*) to find the source nodes, is another heuristic that can be matched against the MuCode. More specifically, each sink creates the shortest path tree to the source nodes. However, it does not count the data overload at the nodes.

Last, I examine MuCode heuristic with the Random Shortest Path (*RSP*) approach, where, at the beginning, it runs the (*BFS*) algorithm in order to find the shortest path per node, and then each node randomly chooses its parent that is within its shortest path routes.

I have conducted three major network setups with different experiments. The first setup is implemented in 3X3, 4X4, 5x5, 6x6, and 8X8 grid networks where a single root BS is placed at the top corner, and the sources are distributed at the farther corner from BS. The second setup is constructed in 3X3, 4X4, 5x5, 6x6, and 8X8 grid networks. I placed the two sinks separately in the top corners, while the source nodes are located at the bottom line. I am assuming all source nodes reside on the same level with respect to the sinks. The last network setup is conducted in 5x4, 7x5, 9x6, 11x7, and 13x8 grids, where four sinks are

placed at the four corners while the source nodes reside in the middle horizontally.

I am focusing on measuring the packet's delivery latency with different data transmitting rate per node, using MuCode protocol by comparing my heuristic to the other heuristics' optimal but exponential solutions.

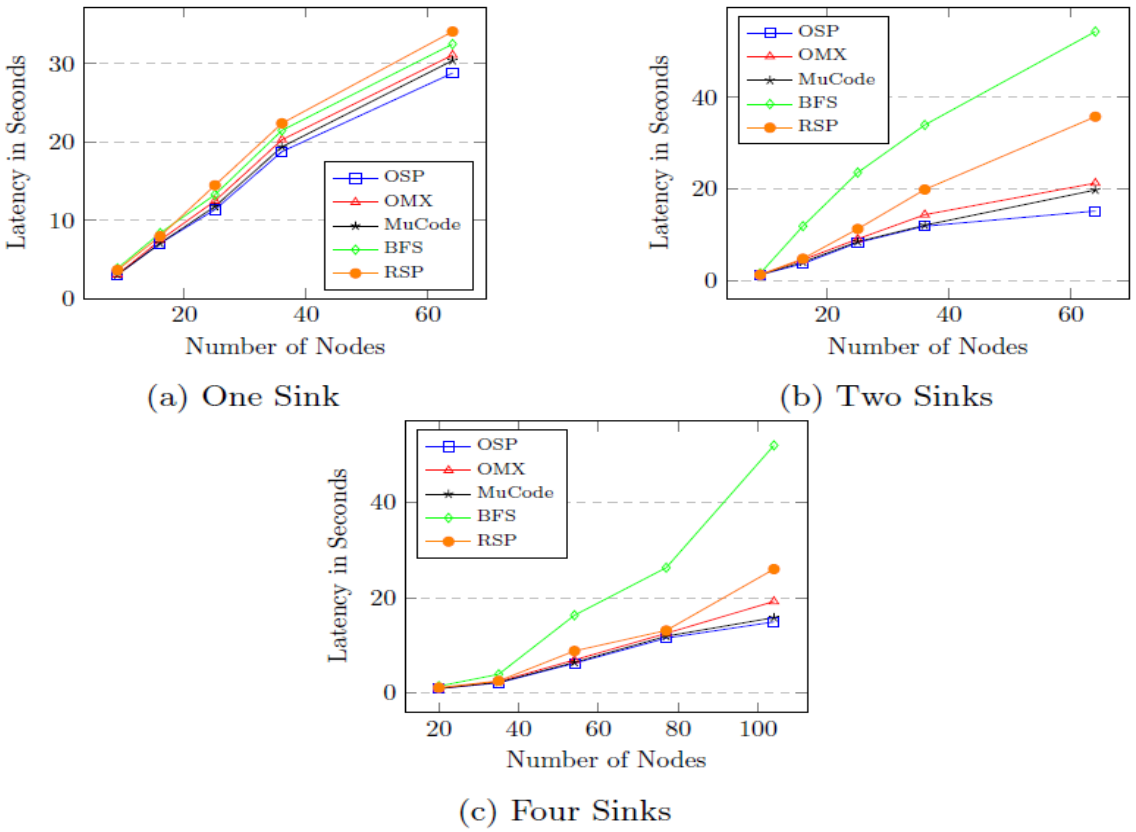


Figure 4.8: Average packet delay with different number of nodes deployed on the network with no packet errors with variety of number of sinks.

Figure 4.8 part (a) depicts the packet delivery latency for the one sink while increasing the data transmitting rates as the number of nodes increase. With a relatively small grid size, my heuristic showed an identical solution to the optimal *OSP* and *OMX*. The reason for this similarity is that my greedy heuristic found the full picture of the parent assignments for the small grid, which allows it to provide the same answer as the *OSP*. At the same

time, *OMX* has been able to balance the data rate load per level while using the shortest path. On the other hand, *BFS* does not consider selecting parents by data rate. Instead, it chooses based on local node discovery, which potentially selects more children than its next node's discovery. Finally, the node in random shortest path *RSP* chooses his parent randomly, which may lead to an unbalanced data rate load that increases the data latency, even when using multi-channel in order to maximize the overhearing possibilities. Once the network gets more depth, my heuristic will, for the most part, not provide the optimal latency compared to *OSP*.

Even though the full picture of the network cannot be obtained from the local optimality, which may end with inevitable parent selecting that is not generally optimal, the results are promising and are the nearest to *OSP*, compared to all the other heuristics including the optimal *OMX*. Even though *OMX* reduces the differences among the nodes per level in terms of the data rate load into the minimum, it does not consider the shortest path as the only solution, which provides longer paths that marginally increase the packet latency compared to *OSP* and my heuristic.

The Figure 4.8 part (b) depicts the packet delivery latency for the two sinks that aggregate the data from the same source nodes that are far equally distanced from the both sinks. With a small grid network, all the methods have the same results except for *BFS*. The reason is that *BFS* can construct a path that is longer than the shortest path, and this path can lead to both sinks as the shortest path for one and not to the other sink.

In the medium size network, the gap in packet delivery latency gets higher between *OSP* and the others, especially *BFS* and *RSP* as the network height goes deeper. In the meantime, my heuristic and *OMX* are closer to *OSP* because the *MuCode* heuristic finds more options to select the optimal local parent, while *OMX* has more chances to obtain the shortest path with minimizing the maximum data rate.

In a large-sized network, both *BFS* and *RSP* data latencies are huge compared to the others. *RSP* can construct an overloaded path that is accumulated throughout the randomness choosing per parent selecting level in the deep network. It can be noticed that

my heuristic outperforms *OMX* because it still maintains the shortest path while minimizing the local parent load assignment, whereas *OMX* creates longer paths that needs to be scheduled using *SEMSS* algorithm mostly.

The Figure 4.8 part (c) shows a packets latency to four sinks that collect data from source nodes, which are equally distanced to these sinks. My heuristic outperforms the other techniques, except *OSP* when the number of nodes become near to 40 and more. I still maintain reasonable packet delivery latencies compared to the optimal solution.

Chapter 5: MuTrans: Uncontrollable Predictable Mobility

The previous chapter demonstrated that the combination of network coding and multi-channel communication can substantially improve the performance of Convergecast LLNs. The current chapter considers the network performance in uncontrollable predictable mobile data collection systems using the MuCode protocol. Although the application class is somewhat different, given that sinks are now mobile, the performance issues are similar. In the mobile LLN environment, the wireless communication medium is unreliable. Further, mobility means that there is limited contact time for the cases where the mobile collector cannot be forced to reside until all of the data is successfully received.

I first show the basic architecture, and then I describe how to model and optimize the system. Next, I describe the load balancing and scheduling protocol and provide the details of my evaluation.

5.1 MuTrans Architecture

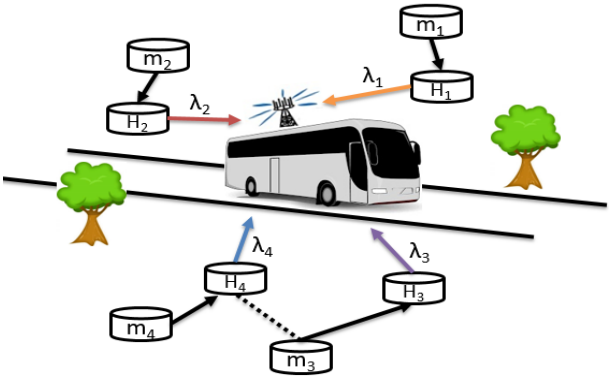


Figure 5.1: MuTrans System Model

Figure 5.1 illustrates the MuTrans system architecture. The basic idea is that there are two types of nodes – head nodes and member nodes. The Figure shows the member node m_3 sending its packets to its head node (solid line) H_3 , while another head node H_4 overhears the packets (dotted line) and uses network coding to enhance network reliability.

Recall that the definition of a polling sector is the range within which the MuCar can communicate with head nodes and receive data while on the move. The polling sector length at head node H_i can be measured by the distance between the starting point where MuCar contacts with H_i and the ending point before it disconnects with H_i. Notice that the location of each polling sector is known once the head nodes are selected. Once the mobile sink (MS) enters the polling sector, the head nodes transmit their stored data through different assigned channels λ_i.

The member node assignment to each head node becomes critical in this situation. This assignment can increase the data load at the head node, which increases the energy consumption while also increasing the uploading latency to the MS. Since the MS has uncontrollable mobility, the uploading deadline is firm and unbreakable at each polling sector. Another important consideration is when the number of concurrent uploading head nodes exceeds the number of radios available at the MS. Therefore, fair and balanced scheduling mechanisms have to be designed to overcome this problem.

5.2 System Optimization

Based off of the system architecture described in Chapter 3 and the previous sections, I now show how to model and optimize network and application performance.

5.2.1 Data Delivery Latency Analysis

This section describes the latency analysis in the worst-case scenario where all the packets are successfully received to their designated head nodes. I first define the inter-session σ as the total time the MS travels the network while receiving the data from all of the head nodes and download them to the base station in a dwell position. Without loss of generality,

I consider the MS to perform periodic data collections, so, for every σ time unit, the mobile makes a "round trip."

Let the distance PS_j represent the polling sector j and the distance the MS travels once it starts communicating with H_i until it terminates that communication. I define Δ_j as the contact duration time between MS and the head nodes at PSj. The upload time $\omega_j(\Delta_j)$ is the time MS needs to upload data from a polling point j at contact duration Δ_j. I assume the MS has $\Re$ radios, and those radios allow it to receive data simultaneously from most $\Re$ head nodes.

Let $\Psi_i(\sigma)$ represent the data that needs to be uploaded to the MS from head node H_i at inter-session σ. $\Psi_i(\sigma)$ can be formulated as:

$$\Psi_i(\sigma) = \gamma_i\sigma + \sum_{z=1}^{k_i} \gamma_{z,i}\sigma \tag{5.1}$$

where γ_i is the data rate generated by the head node H_i and $\gamma_{z,i}$ is the data rate generated by a member node $m_{z,i}$ that is assigned to the head node H_i, while k_i is the number of member nodes assigned to H_i.

By assuming the packet size is fixed, using Equation (6.4) , the number of packets that the head node H_i has at inter-session time σ can be found by the following equation:

$$G_i(\sigma) = \left\lceil \frac{\Psi_i(\sigma)}{packet\_size} \right\rceil \tag{5.2}$$

Subsequently, the uploading latency for head node H_i at inter-session σ is equal to multiplying $G_i(\sigma)$ by τ, where τ is the packet transmitting latency time.

Let Γ_j represent the number of head nodes that the MS can cover while receiving data at polling sector j. Again, without a loss of generality, and to simplify the notation, I assume that if H_i and $H_{i'}$ share the same polling sector PS_j, then they both have the same starting and ending contact time while uploading their data to the MS. Consequently, if the number

Table 5.1: MuTrans Notations Table

Notation	Meaning
MS	The uncontrollable predictable mobile sink
σ	The time interval MS takes to collect data from all polling sectors per trip.
PS_j	A polling sector j where MS upload the data from.
Δ_j	The contact duration between MS and the head nodes at PS_j.
$\overline{N}$	The total number of polling sectors.
$\omega_j(\Delta_j)$	Time MS needs to upload data from PS_j at contact duration Δ_j.
$\Re$	Number of multi-radios MS has.
H_i	head node i.
HS_j	The set of head nodes at PS_j.
m_i	Member node i.
$\Psi_i(\sigma)$	Archived data in H_i during σ of time.
γ_i	The data rate generated by H_i.
$\gamma_{z,i}$	The data rate generated by m_z assigned to H_i.
k_i	Number of member nodes assigned to H_i.
$G_i(\sigma)$	Number of packets H_i has during σ of time.
τ	Packet transmitting time.
Γ_j	Total number of head nodes at PS_j.
μ_j	Total number of member nodes at PS_j.
Π_j	Number of uploading schedules per concurrent transmitting at PS_j.
$\Omega(\sigma)$	The total uploading latency at inter-session σ.
β	Node's buffer size.
$C(m_i)$	Set of m_i's candidate head nodes.
G^i_{min}	The minimum number of packets a head node has in schedule i.
G^i_{tot}	The total number of packets in all head node at schedule i.

of head nodes are more than the number of radios available at MS, then all these nodes cannot upload to the MS simultaneously, because there will be transmitting interference between some head nodes that use the same channel. Therefore, I introduce the concept of

uploading scheduling where each number of at most $\Re$ head nodes are allowed to transmit concurrently to the MS, while the remaining are scheduled for next transmissions in the same way as the previous schedule. The number of different uploading scheduling that the MS needs at PS_j is formulated as:

$$\Pi_j = \left\lceil \frac{\Gamma_j}{\Re} \right\rceil \tag{5.3}$$

This equation shows that if $\Gamma_j \leq \Re$, then only one schedule is needed to transmit a packet per head node at PS_j. The challenge is to minimize the uploading latency. I observe that each schedule latency is bounded by the maximum number of packets a head node has per schedule. Hence, the upload time $\omega_j(\sigma)$ takes to the MS at polling sector j for inter-session time σ can be formulated as:

$$\omega_j(\Delta_j) = \sum_{c=1}^{\Pi_j} G_*^c(\sigma) \ . \ \tau \ \leq \Delta_j \tag{5.4}$$

where $G_*^c(\sigma)$ is the selected maximum number of packets that a head node has that belong to the schedule c in inter-session σ at PS_j using Equation (5.2). This uploading latency should not exceed the contact interval time Δ_j or else not all data will be received during that trip. Unlike traditional transmitting with a single transceiver at the MS, in my multi-channel scheme the uploading latency among the head nodes is only considered by the node that has the maximum total number of transmitted packet among the head nodes. The reason is that head nodes transmit concurrently to the MS using different channels that have been assigned by the MS prior the uploading process.

Assuming that the communication duration between the MS and all of the head nodes from all of the polling sectors have the same contact period Δ, the *total uploading latency* $\Omega(\sigma)$ at inter-session time σ can be formulated as:

$$\Omega(\sigma) = \sum_{j=1}^{\overline{N}} \omega_j(\Delta_j) \ \leq \overline{N} \ \Delta \tag{5.5}$$

where $\overline{N}$ is the total number of polling sectors on the network. In order to minimize $\Omega(\sigma)$, $\omega_j(\Delta_j)$ has to be minimized where $\overline{N}$ is fixed. System design goals are to minimize Equations (5.4) and (5.5).

5.2.2 Optimization

Consider the case where I want to minimize the uploading time $\omega_j(\Delta_j)$ at PS_j. Let ∇_j be the set of head nodes that are within the range of MS communication at the polling sector j. The set ∇_j is sorted by $G_i(\sigma)$ for each head node H_i in the set in descending order. Therefore, an optimization problem for minimizing the uploading latency per inter-session $\Omega(\sigma)$ ca be characterized as follows:

$$\min \ \Omega(\sigma) = \sum_{j=1}^{\overline{N}} \sum_{c=1}^{\Pi_j} G_*^c(\sigma) \ . \ \tau \ \leq \overline{N} \ \Delta \tag{5.6}$$

where G_*^c is the total packets of H_*^c placed in the $\left[\Re(c-1)+1\right]^{th}$ position of the set ∇_j. I assume the round robin scheduling is implemented for the Π_j schedules to emphasize the fairness between the head nodes. I define the fairness scheduling in Section 5.3.2.

Figure 5.2 depicts an example of the scheduling method with the number of radios available $\Re = 3$, where the number of head nodes are 9 at polling sector j. The set ∇_j is sorted by the number of packets that each head node has, while Equation (5.3) calculates the number of different schedules (Π_j) needed which equals to $\left\lceil \frac{9}{3} \right\rceil = 3$ schedules. For c = 1, the $G_*^1 = 10$ where it is positioned in $\left[3(1-1)+1\right]^{th} = 1^{st}$ in set ∇_j. In the same way,

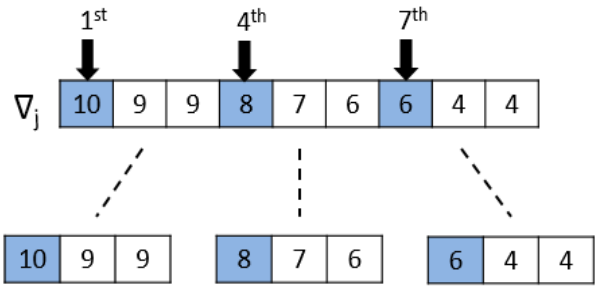


Figure 5.2: A scheduling example for $\Re = 3$ and $\Gamma_j = 9$ head nodes at polling sector j. Each number indicates the number of packets that must be uploaded to the Mobile Collector where 3 uploading schedules are required.

$G^2_* = 8$ while $G^3_* = 6$. Hence, the uploading time needed to transmit all the packets in PS_j is $(G^1_* + G^2_* + G^3_*) \,.\, \tau = 24 \,.\, \tau$ where each schedule is transmitting in round-robin scheduling for fairness perspective.

Observe that Equation (5.6) has two variables that are part of minimizing the uploading latency: the number of schedules per polling sector (Π_j) and the total number of packets the head nodes have that are positioned in $\left[\Re(c-1)+1\right]^{th}$ in the set ∇_j for $c = 1, \ldots, \Pi_j$.

The inner sum equation of minimizing $\Omega(\sigma)$ is equivalent to Equation (5.4). Basically, the uploading time $\omega_j(\Delta_j)$ at PS_j is minimized if head nodes are sorted based on their packets and balanced, and it will be ideal if $\Gamma_j = a \,.\, \Re$ where a is a positive integer. I now develop a theorem to compute the optimal assignments that minimize Equation (5.6). To do this, I first prove two lemmas.

Lemma 5.2.1. *Let G^x_* be the maximum number of packets a head node has in schedule x where $x = 1, \ldots, \Pi_j$ and all have the same packet size. For $\Re \geq 2$, ω_j is minimized if every head node with G^x_* is in the $\left[\Re(x-1)+1\right]^{th}$ position of the set ∇_j, where ∇_j is sorted by G_i for each H_i in the set in descending order.*

Proof. Since all head nodes that share the same schedule x can send packets simultaneously using different channels, schedule x will finish transmitting as long as the head node with G^x_* finishes. Given that each schedule sends packets at different times, $\omega_j(\Delta_j) = \sum_{x=1}^{\Pi_j} G^x_* \,.\, \tau$.

The uploading time for schedule x depends on the size of G_*^x on that schedule. Therefore, minimizing G_*^x per schedule x minimizes overall uploading time. To prove this by induction: $x = 1$ where schedule 1 has $\Re$ head nodes and G_*^1 is the maximum packets among all the head nodes. The base step satisfies my lemma. Now, I assume the lemma is true for $x = z$. For schedule $z + 1$, the ω_j is minimized as long as G_*^{z+1} is less than or equal to every G_i^z in schedule z, which must be true since they are sorted in descending order. □

Lemma 5.2.2. *ω_j is minimized if every G_i^x is equally balanced in a number of packets with G_*^x for the same schedule x.*

Proof. Let G_{min}^x be the minimum number of packets for a head node in schedule x and let G_{tot}^x be the total number of packets that all the head nodes have at schedule x. Since $G_*^x \geq G_{min}^x$, and the upload time at schedule x depends only on G_*^x. Therefore, reducing G_*^x while maintaining the same size of G_{tot}^x increases the size of packets in other head nodes, and it can be the same size as G_{min}^x's. Hence, balancing between G_*^x and other head nodes – especially G_{min}^x – should lead to minimizing G_*^x, otherwise G_*^x will be greater than G_{min}^x with another head node. This is true for every schedule, which eventually minimizes the overall uploading time at PP_j. □

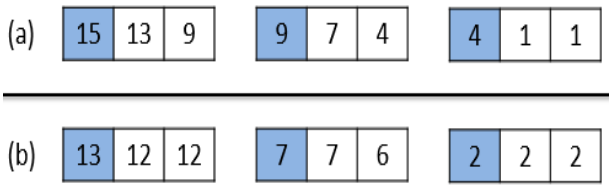


Figure 5.3: A scheduling example: (a) the schedules before using Lemma (5.2.2); (b) the schedules after using Lemma (5.2.2).

As an example, consider Figure 5.3, which shows a scheduling scenario for $\Re = 3$ where the number of head nodes is nine at polling sector j. In part (a), the schedules are sorted by the number of packets. Here, the total number of packets = 63, while the uploading

latency = 28 . τ. After applying Lemma (5.2.2), the total number of packets are the same where some head nodes are able to get packets from the maxed head nodes on the same schedule, therefore the uploading latency = 22 . τ, which is shorter in latency than the case in (a).

Theorem 5.2.3. *ω_j is minimized if $\Gamma_j = a.\Re$ where a is a positive integer and each schedule size is exactly $\Re$.*

Proof. Another way to describe this Theorem is that every schedule has exactly $\Re$ head nodes. To prove the Theorem by contradiction, let schedule x with $\aleph$ head nodes get the local minimum uploading time for G^x_{tot} total packets where $\Re > \aleph$. The remaining $\Re - \aleph$ spots in schedule x are initialized to 0 packets. Based on Lemma 5.2.2, schedule x need to be balanced to get local minimum uploading time, and that is not the case which contradicts the assumption. □

My approach to achieving this optimal result is to dimension the system, according to known parameters, such as σ and the data rates of all head and member nodes. Appendix **A.1** formulates the problem as Linear Programming Optimization which is a challenging problem. The next section describes the MuTrans protocol, which offers a practical solution.

5.3 MuTrans Protocol

The purpose of the $MuTrans$ protocol is twofold: First, to dynamically assign each member node to a head node. Second, to, as fairly as possible (in the sense of overall minimizing total upload latency), assign uploading schedules to head nodes. For each of these problems, I present polynomial time heuristics. The heuristics are meant to be run with full knowledge of system parameters, and therefore can be run by the MS. The protocol can be run continuously as the mobile traverses the system or can be run whenever system parameters change, such as head node or member node assignment, changes in data rates, or changes in mobility patterns as in LBC-DDU.

5.3.1 Data Load Balancing

The Data Load Balance (DLB) - Algorithm 11 - heuristically balances data loads among the head nodes in order to reduce the uploading duration in the mobile data collector. I define $C(m_i)$ as the set of candidate head nodes that the member node m_i can assign its data to. Those candidates are one-hop distanced to m_i. Initially, each member and head node calculates its collected data $\Psi_{m_i}(\sigma)$ and $\Psi_{H_i}(\sigma)$ respectively from Equation (6.4) during inter-session time σ.

Algorithm 11 Data Load Balance (DLB)

1: **Initialization:** each member node m_i defines $C(m_i) = \{H_i | \forall H_i \in NB(m_i)\}$;
2: **Initialization:** each member node m_i calculates $\Psi_{m_i}(\sigma) = \gamma_{m_i}.\sigma$;
3: **Initialization:** each head node H_i calculates $\Psi_{H_i}(\sigma) = \gamma_{H_i}.\sigma$;
4: $Q = \{m_i | \forall i \in \{1, \ldots, \mu_j\}\}$;
5: Sort m_i in Q based on the size of $C(m_i)$ in ascending order, then based on $\Psi_{m_i}(\sigma)$ in descending order;
6: **while** Q is not empty **do**
7: $m \leftarrow EXTRACT(Q)$;
8: m chooses head node H_i s.t. $H_i \in C(m)$ and $\Psi_{H_i}(\sigma)$ is the lowest;
9: $\Psi_{H_i}(\sigma) \leftarrow \Psi_{H_i}(\sigma) + \Psi_{m_i}(\sigma)$;
10: **end while**
11: **Output:** each member node is assigned to a head node with data load balanced;

I say that a schedule's balance is measured by the quantity $|G^x_{max} - G^x_{min}|$, and that a schedule that is "balanced" has the property $\min |G^x_{max} - G^x_{min}|$, which is also described by Lemma 5.2.2.

In steps 4-5, the set Q is defined as having all of the member nodes at the current polling sector so that it is sorted based on the number of candidate head nodes per member nodes in ascending order and then based on the local data in descending order. In this way, the least candidates' member node chooses first in order to give more precise and better options to the member nodes that have more candidate head nodes. If I do the opposite, then a member node with high candidates may choose a head node, which may be the only option for another member node – therefore increasing the load to that head node. This can be avoided if that choice was processed later.

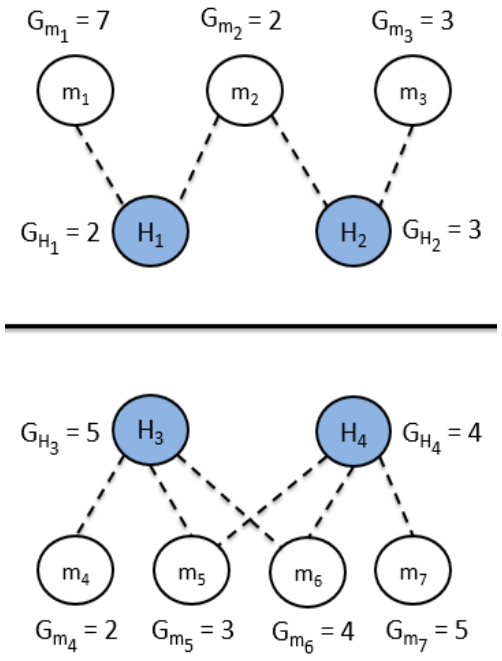


Figure 5.4: An example of the candidate head nodes per member node. Each number represents the data load.

In steps 6-10, iteratively, I pick the first member node and choose the head node with lowest data load and update its data load by including the member node's data load. This iteration is processed until the set Q is empty.

Figure 5.4 represents an example of the *DLB* algorithm where the dark circles are the head nodes, while the white circles are the member nodes. The dotted line shows the candidate head node that will be assigned by the member node, while the solid line represents the path of the MS. Each node calculates its G value during σ duration using the Equation (5.2). Because head nodes are not assigned to any member node yet, the head nodes calculate their G values using only their local data.

In this example, the set Q is equal to $\{m_1, m_7, m_3, m_4, m_6, m_5, m_2\}$, which is sorted based on the member nodes with the least options to choose a head node and with the highest number of packets. With regarding this sequence, each member node selects the head node with smallest data load. Figure 5.5 shows the final assignment with load balanced among the head nodes.

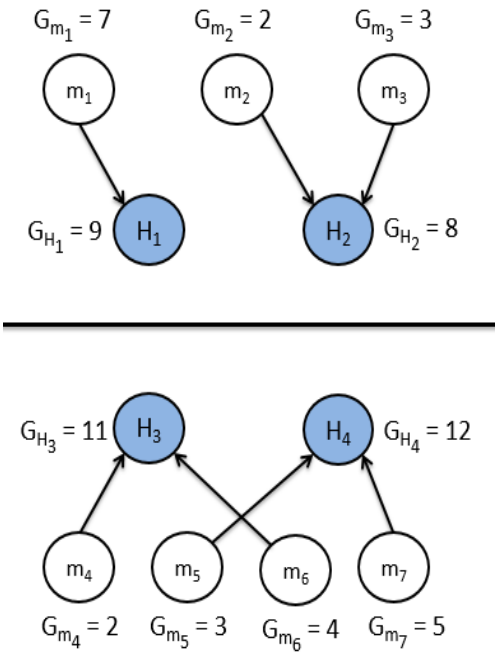


Figure 5.5: The member assignments after using *DLB* method.

The complexity time for my heuristic algorithm is $\mathcal{O}(\mu_j \text{ x } (\Gamma_j + \log \mu_j))$, where μ_j is the number of member nodes, while Γ_j is the number of head nodes, which is a feasible polynomial solution of the optimization problem.

5.3.2 Utilized Fair Scheduling

A synchronized schedule that is based on the number of packets in the head nodes has to be maintained to reduce the uploading latency. Given Γ_j, and given head nodes that want to upload their data to the mobile data collector at a given polling sector PS_j, I define Algorithm 12 , which fairly and dynamically utilizes the scheduling of the concurrent head nodes that are uploading. Let HS_j be the head node set at the polling sector PS_j.

Steps 1-2 takes the HS_j set and assigns it to set S. The iteration of the algorithm considers the remaining non-sent packets in the set S. At first, the set is sorted based on the number of packets each head node has in descending order. Then, it divides these head nodes by the number of available channels that the mobile data collector has, which creates g schedules

Algorithm 12 Dynamic Round-Robin Scheduling (DRRS)

1: **Input:** head node set HS_j;
2: $S \leftarrow HS_j$;
3: **while** S is not empty **do**
4: Sort S based on G_i in descending order;
5: Divide S into $g = \left\lceil \frac{|S|}{\Re} \right\rceil$ schedules;
6: Select the g head nodes where each has minimum G per schedule x (G^x_{min});
7: In each schedule x, head nodes concurrently transmit their data G^x_{min} times;
8: For each head node H_i at schedule x, $G^x_i \leftarrow G^x_i - G^x_{min}$;
9: Remove all head nodes with $G_i = 0$ from set S;
10: **end while**
11: **Output:** fairness dynamic schedules that maximize concurrent heads uploading;

of head nodes. Each schedule x has a head node with a minimum number of packets (called G^x_{min}), which is the indicator for how many simultaneous transmitting should be done during this iteration. I use Weighted Round Robin scheduling, in the sense that $G^1_{min} \geq G^2_{min} \geq \ldots \geq G^g_{min}$, which leads to the number of concurrent transmitting per iteration in schedule 1, which is always greater than or equal to schedule 2, and so on until schedule g. After the transmitting process is complete in this iteration, each head node at schedule x deducts G^x_{min} from its total packet, then S removes all the head nodes that have no remaining packets to send. The algorithm repeats the previous steps until the head nodes transmit all packets.

Before I define fairness, certain terminology must be introduced. Let x be a schedule, and let $|x|$ be the number of concurrent heads uploading at schedule x. A schedule x, which has the "maxed concurrent heads uploading" when it satisfies the property: $\min |\Re - |x||$, and this property is supported by Theorem 5.2.3. The fairness uploading scheduling is weighted by the G_{min} on each schedule. For each iteration, the weighted round-robin scheduling is taking place with the queuing ratio $\frac{G^1_{min}}{G^g_{min}} : \frac{G^2_{min}}{G^g_{min}} : \ldots : 1$ where for instance, in the first iteration, schedule 1 transmits G^1_{min}/G^g_{min} packets while schedule g transmits only single packet in round-robin fashion. This process is repeated until the end of the first iteration.

Figure 5.6 depicts an example of *DRRS* scheduling with two radios available at MS.

In the first iteration, the set $S = \{H_4, H_3, H_1, H_2\}$ where it is sorted based on the largest number of packets. Then, the set is divided into 2 schedules: schedule 1 = $\{H_4, H_3\}$ with $G^1_{min} = 11$ while schedule 2 = $\{H_1, H_2\}$ with $G^2_{min} = 8$. Now, in the transmitting phase, each head node on the same schedule transmits packets simultaneously using different assigned channels. However, each schedule has its own time slot that does not overlap with another schedule. The round robin scheduling is implemented to emphasize fairness among the schedules, where schedule 1 transmits 11 packets, while schedule 2 transmits eight packets during the first iteration.

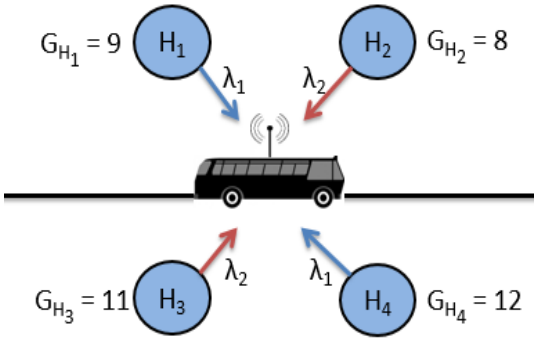


Figure 5.6: *DRRS* example using multi-channel with $\Re = 2$.

In the second iteration, the set S will be $\{H_4, H_1\}$, which produces one schedule with $G^1_{min} = 1$, where each head node has only one remaining packet. The total time slots needed to transmit 40 packets from 4 head nodes is 20, which is exactly the same as the optimal solution, and that is not always the case. The running time complexity of *DRRS* algorithm is $\mathcal{O}(\Gamma_j^2 \log \Gamma_j)$.

5.4 Evaluation and Results

Using Cooja, an LLN simulation tool, I evaluated $MuTrans$ against several other approaches. The purpose of this evaluation was to judge the effectiveness of my proposed heuristic. Since there do not exist other heuristic algorithms that are applicable in my predictable multi-channel environment, I defined two simple heuristics to solve my two subproblems: Random Member Assignment RMA and Static Round Robin Scheduling $SRRS$. RMA is an algorithm that randomly assigns member nodes to their nearby head nodes without considering data balancing, whereas $SRRS$ is a round robin scheduling that fixes the head node location to its original schedule.

These two baseline heuristics enabled us to define three new protocols. The first combination is $RMA + SRRS$, where the head node selection is randomly assigned by the member nodes, while the concurrent uploading schedules are statically assigned to the head nodes with a single iteration. The second combination is $RMA + DRRS$, where the head nodes are selected randomly by the member nodes, while the concurrent uploading schedules are dynamically assigned to the head nodes with multiple iterations as in algorithm 12. The last combination is $DLB + SRRS$. Here the head nodes are assigned by the member nodes based on algorithm 11, which balances the data rate load at the head nodes; however, the concurrent uploads are statically scheduled to these head nodes. All of these three new protocols use multi-channel network coding.

To represent a realistic uncontrollable but predictable environment, I used a topological representation of George Mason University, as shown in the Figure 5.7. I used the Contiki operating system to implement the protocols, where the red/light drop pins are the polling sectors in which head nodes upload their data to the MS, while the green/dark drop pin represent the Base Station in which MS has to dwell at and send the collected data to.

The MS is traversing the GMU campus in a fixed route with a distance of 5.3 km and a speed of 55 km/h. I installed 16 polling sectors, where the number of head nodes is 6 per polling sector. The total number of deployed nodes are 192 with $\Delta = 4.6$ seconds contact duration between the MS and the head nodes. Each member node i has its own data

transmitting rate γ_i in order to test the data balance algorithm. These member nodes are randomly scattered and are 1-hopped away from the head nodes. I evaluated my protocols with a different number of radios $\Re$ available at the MS. These experiments are conducted with a variety of wireless Packet Error Rates (PERs) using the *MuCode* network coding protocol [10].

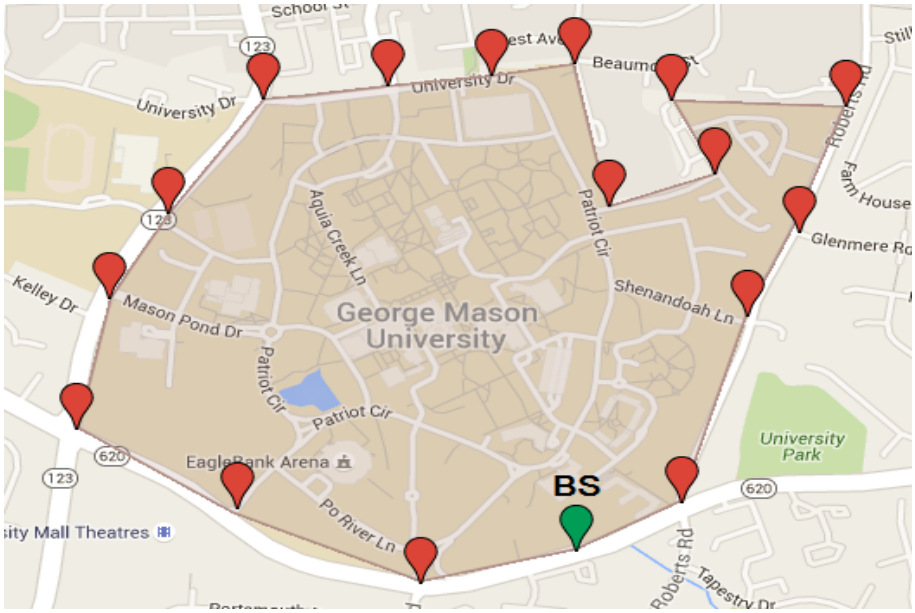


Figure 5.7: George Mason University map

5.4.1 Packet Delivery Rate (PDR)

The first experiment is to judge the Packet Delivery Rate. Figure 5.8 depicts the average packet delivery rate versus PER using two radios in MS. In a reliable communication (i.e. PER = 0%), the PDRs are maximized, and all of the four techniques are equal. At PER = 10%, however, $DLB + DRRS$ becomes the only method that maintains full PDR, with a difference up to 8.7%. Even when PER gets higher, my protocol is superior to the other techniques by, at least, 40% , 3%, and 11% in PDR, which can be compared to $RMA + SRRS$, $RMA + DRRS$, and $DLB + SRRS$.

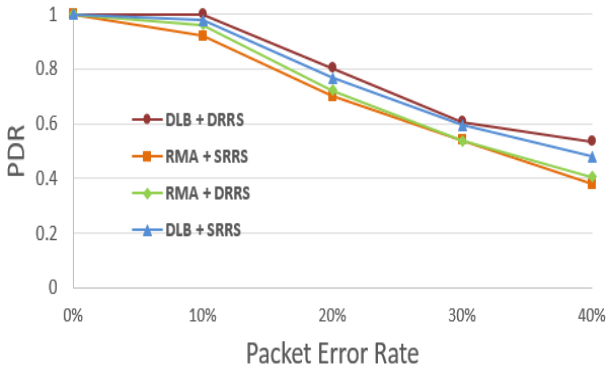


Figure 5.8: Average packet delivery rates with $\Re = 2$.

The reason for this result is that DLB increases the chance to get more overheard encoded packets from neighbors, which eventually leverages the use of network coding to decode the lost packets. Furthermore, $DRRS$ increases the total number of packets to be transmitted, compared to $SRRS$ for the same contact duration, which raises both the plain and encoded transmitted packets. It can be noticed that DLB significantly plays a greater role in packet delivery consistency when compared to $DRRS$, which emphasizes the need to consider Theorem 5.2.3 in network planning.

5.4.2 Network Throughput

Figure 5.9 shows the average network throughput with different PERs. The four protocols shown with $\Re = 2$ are the same techniques from the previous results. At PER = 0%, the throughput in $DLB + DRRS$ outperforms the other method with $\Re = 2$ from 3.7% to 12%. The reason for this superiority is that $DRRS$ allows more packets to be transmitted; however, RMA creates unbalanced data loading among the head nodes, which affects the uploading latency, as in Lemma 5.2.2. Meanwhile, DLB effectively uses this lemma in order to reduce the uploading latency, which eventually increases the network throughput with a fixed contact time.

On the other hand, $DLB + DRRS$ ($\Re = 6$) outperforms the same model when $\Re = 2$ and $\Re = 4$ in relation to network throughput in different packet error rates up to 166% and

73%, while at PER = 40%, the gap is shrunk slightly to 154% and 72%. This throughput advantage is because *DRRS* maximizes the concurrent data uploading by increasing the number of radios available in MS in order to be equal to the number of head nodes at that polling sector, which increases the total number of packets efficiently.

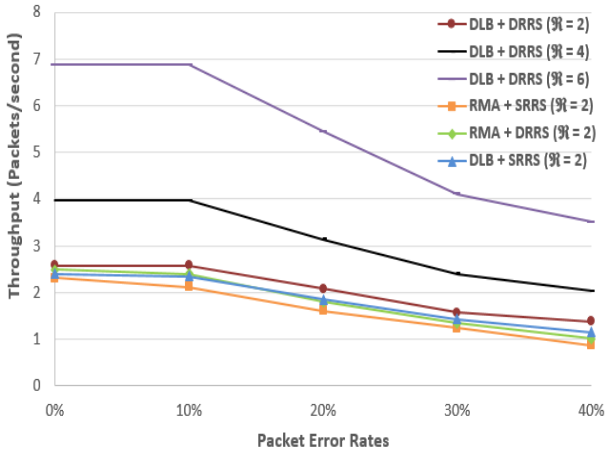


Figure 5.9: Average network throughput versus PER with different ℜ.

5.4.3 Trips Required for Data Delivery

One of my motivations in my research is to quantify the number of trips required to complete data delivery. To assess this, I ignore the impact of buffer overflow and simply count the number of required trips. Figure 5.10 depicts the average number of trips that MS ($\Re = 2$) has to accomplish in order to get the data needed by the system that has 40% of PER, with different total data packets originated by all the nodes on the network.

With a small amount of data, even with the high packet error rate, all of the four protocols can get all of the data that is originated by the nodes. There are two reasons for this successful transmission: First, network coding is implemented in all of them, which helps recover lost data packets. The second reason for the completed transmission is that the contact duration meets the deadline for uploading the needed data packets without the need for retransmissions.

Once the data packets increase to 900, the MS has to take another trip in order to get the remaining lost packets that needed to be retransmitted. Since the contact duration time is always fixed in such an application, the over amount data packets, which are not considered by the system administrator, will miss the deadline. Hence not only are the retransmission of packets needed in high packet loss rates, but also the MS has to take more trips in order to receive all of the data packets, as seen in the previous figure. Nevertheless, $DLB + DRRS$ outperforms $RMA + SRRS$, $RMA + DRRS$, and $DLB + SRRS$ by up to 57%, 36%, and 20% in data upload latency respectively.

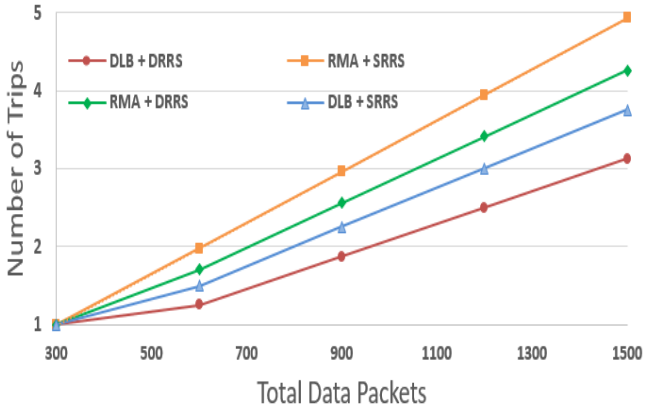


Figure 5.10: The average number of trips MS has to take to collects all the data with PER = 40% and $\Re = 2$.

5.4.4 Energy Consumption

The energy is a major factor where the transmission and the reception are using the radio to consume some amount of energy. Depending on the MAC protocol that been used, sometimes the cost of sending and receiving are roughly equal. Sometimes sending requests significantly more power. In my evaluation, I am assuming both sending and receiving are the same in terms of energy consumption because my system is fully synchronized where the nodes know when to wake up and send or receive.

I estimated the energy consumption by using the same energy model that LBC-DDU

used where the model is presented by [76]. They defined the energy consumption e_t as $e_t = (e_1 d_r^{\alpha} + e_0) l_p$ where e_1 is the loss coefficient per bit, α is the path loss exponent, e_0 is the excessive energy consumed on sending, d_r is the transmission range, and l_p is the size of the transmitted packet in bits[1].

Figure 5.11 shows the average maximum energy a head node consumes to upload all of the data to an MS with two radios and a variety of PERs. At PER = 0%, both $DLB{+}DRRS$ and $DLB + SRRS$ have the lowest energy consumption, since the data load is nearly balanced between the head nodes by using DLB, which spreads the packets transmitting fairly. Also, both $RMA + DRRS$ and $RMA + SRRS$'s maxed energy head nodes consume the same energy in the reliable environment, because they are using the same data load method.

When the PER gets higher, however, head nodes tend to retransmit the missing packets to MS, which increases the average energy consumption in many of the nodes in the network. Thus, $DLB{+}DRRS$ outperforms $RMA{+}SRRS$ in saving energy by up to 70%. It also saves energy up to 57% and 11%, which can be compared to $DLB + SRRS$ and $RMA + DRRS$.

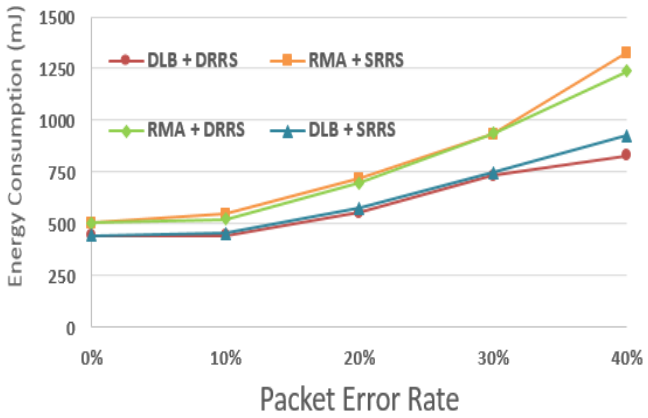


Figure 5.11: The average of the maximum head node energy consumption versus packet error rates with $\Re = 2$.

[1] $d_r = 50m, e_0 = 45$ x $10^{-9} J/bit, e_1 = 10$ x $10^{-9} J/bit, \alpha = 2$

Chapter 6: MuCC: Controllable Predictable Mobility

Chapter 4 proposes the multi-channel network coding technique to improve the communication performance for single and multiple static sinks in tree-based LLN systems. Chapter 5 addresses uncontrollable predictable mobility gathering systems and provides heuristics for latency optimization while simultaneously collecting data in a predetermined path and fixed velocity. This chapter focuses on applications that are supported by a data collector whose mobility pattern is controllable predictable. More specifically, I will describe the design of a cluster-based system architecture and utilize the uploading latency by formulating the optimization problem and provide heuristics to find a feasible solution in polynomial time. This chapter also designs a method to reduce the uploading locations in order to reduce the overall traveling latency.

I will first describe my proposed architecture and then discuss the multi-phase protocol. Next, I will present how motion planning can be performed on the controllable mobile data sink. Following that application, I will provide a performance analysis of the system and discuss an adaptive way to select cluster heads. Finally, I show the details of my performance analysis.

6.1 MuCC Architecture

I propose a two-layer clustering framework called Multi-channel Network Coding Clustering (MuCC) Protocol for Low-power and Lossy Networks (LLNs). The mobile data collector (MuCar) collects aggregated data from the cluster heads that are archived using *MuCode* communication protocol as described in Chapter 4. Cluster heads have single transceivers that are capable of choosing a unique channel from the set of available system channels Λ. MuCar is a multi-radio mobile collector. It can concurrently receive data from those cluster

heads through different channels to prevent packet collisions.

The cluster heads aggregate the data from their cluster members including their own data. The MuCar needs to visit polling points (PP) in order to collect the data from these cluster heads. I assume that MuCar knows the location of all of the sensors. Clusters are formed based on the maximum number of cluster heads per cluster (M). Figure 6.1 illustrates my system model, which shows that MuCar stops at a polling point and concurrently collects data from cluster heads (dark hexagon), where they then aggregate the data from their cluster members (white triangle). Then, MuCar traverses to the next polling point through its planned trajectory path, which I will explain in more detail in Section 6.5. Finally, MuCar returns back to store the collected data in the data repository.

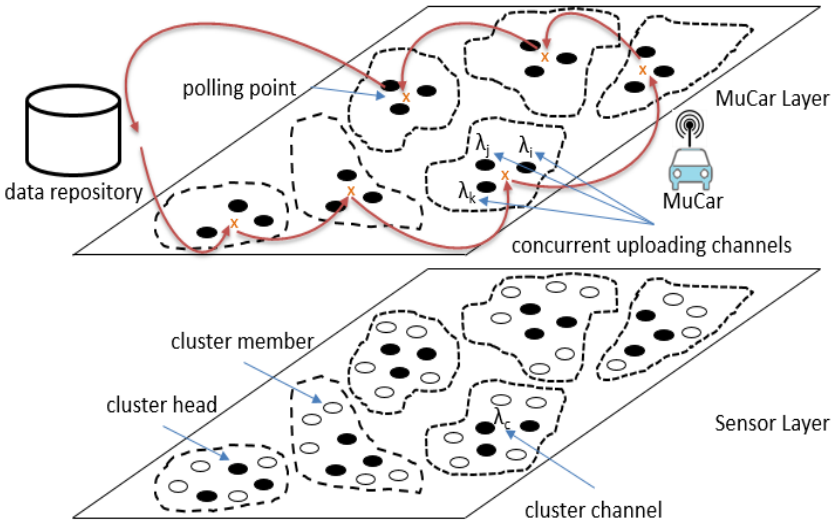


Figure 6.1: The MuCC two-layer framework.

For each cluster, when $M \geq 2$, MuCar can visit from 1 up to $M/2$ polling points. These M cluster heads are all connected, and I will explain the reason in $MuCC$ protocol section. MuCar can retrieve data from at least two cluster heads simultaneously. In Figure 6.2, assume that a cluster has $M = 4$ cluster heads (CH_1, CH_2, CH_3 and CH_4). For a single-radio mobile collector, it has to implement four uploading scheduling (M) in order to collect

the data from the cluster heads. In comparison, a protocol like SenCar [49] - the LBC-DDU mobile collector - has to perform two different uploading scheduling ($M/2$), since it uses the Dual Data Uploading antennas. SenCar reduces the uploading latency by half compared to the traditional mobile data collector. However, MuCar can reduce its uploading latency by combining the transmitting scheduling for each cluster head CH_1, CH_2,CH_3, and CH_4 into one schedule using multiple radios, where each CH_i transmits its data using a unique channel concurrently without packet collision.

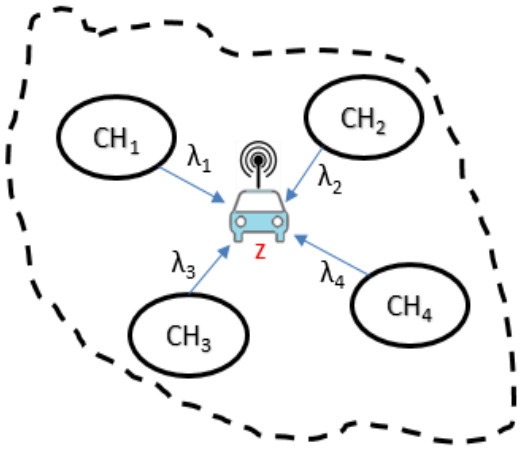


Figure 6.2: MuCar starts collecting data from the cluster heads CH_1, CH_2, CH_3, and CH_4 concurrently at the polling point z through different channels $\lambda_1, \lambda_2, \lambda_3$ and λ_4.

My approach uses the network coding technique to enhance system reliability in Low-power and Lossy Networks (LLNs), where packet loss and bit errors are considered. MuCC forms cluster heads when each head has its members that are one-hop away. Each cluster member assigns its self to one of its neighboring cluster heads that have the minimum number of members. Load balancing is an important consideration in this process of network lifetime. Assigning cluster members to a cluster head, which already has a minimum amount of cluster members, can effectively balance the energy consumption among the cluster heads, as the transmitted packets are fairly distributed.

Members and cluster heads can overhear packets. Moreover, some members can encode

those overheard packets and transmit them along with their original packets. Figure 6.3 depicts an example of the network coding technique named Random Linear Network Coding RLNC technique [74] for a cluster of $M = 2$ (CH_1 and CH_2) and three members (m_1, m_2 and m_3). Here, the member node m_2 overhears packets x_1 and x_3, encodes them using RLNC, and transmits the encoded packets $\alpha_1 x_1 \oplus \alpha_3 x_3$ and its original x_2, where α_1 and α_3 are the random coefficient values for the packets x_1 and x_3, and $\oplus$ is the encoding operation for both packets in a Galois finite field. These coefficients are randomly generated by the sensors in order to create linearly independent packets, which eases the decoding process in cluster head sides. Otherwise, the packets cannot be decoded and, hence, need to be dumped. The cluster heads receive packets where some are encoded. In the example of network coding, each cluster head has three total packets. If there is a chance of a packet loss, MuCar can still receive all of the packets. For instance, if packets x_1, x_2, and the encoded packet $\alpha_1 x_1 \oplus \alpha_3 x_3$ are lost, MuCar receives x_3, x_2, and the encoded $\alpha_1 x_1 \oplus \alpha_3 x_3$ from the cluster heads and decodes $\alpha_1 x_1 \oplus \alpha_3 x_3$ using the knowledge of having x_3.

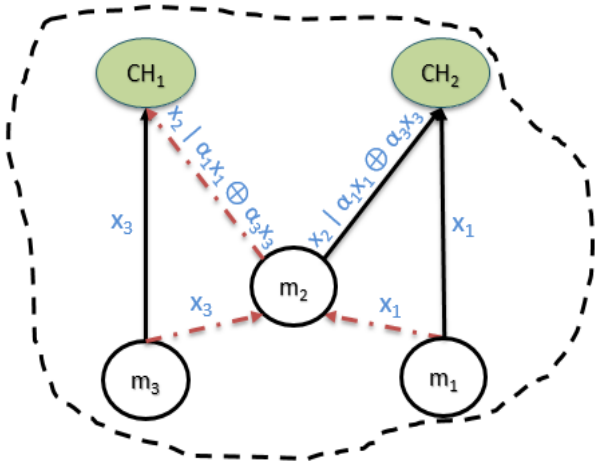


Figure 6.3: An example for MuCode communication with $M = 2$ (CH_1 and CH_2). Solid lines represent direct transmission from cluster member to its cluster head, while dashed lines represent overheard transmissions. Cluster member m_2 overhears neighboring packets from m_1 and m_3. m_2 transmits the encoded packet $\alpha_1 x_1 \oplus \alpha_3 x_3$ to cluster heads along with its own packet x_2.

On the other hand, in traditional transmitting LLNs, data replication may reduce packet loss. Back in Figure 6.3, without network coding, the cluster head CH_1 will receive packets x_1 and x_2 twice, while CH_2 will receive twice the amount of x_3. If there is a high hot spot within CH_1 transmission range, however, all of the CH_1 packets are lost. Therefore, the mobile data collector will only receive the packet x_3 twice and will never receive the lost packet under this circumstance. While, in my approach, even if the CH_1 packets are all lost, MuCar can still receive all of the packets from CH_2, which uploads x_2, $\alpha_1 x_1 \oplus \alpha_3 x_3$, and x_3. MuCar can decode $\alpha_1 x_1 \oplus \alpha_3 x_3$ using linear algebraic equations by having x_3 in the Galois field.

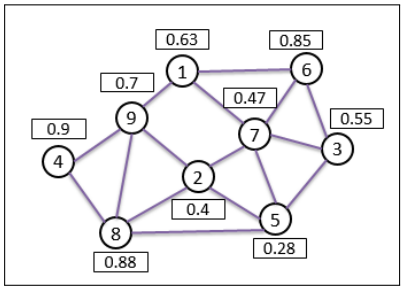
6.1.1 Sensor Layer: Distributed and Scalable Encoded Clustering

This layer constructs an energy-balanced sensor clustering by selecting cluster head nodes. The residual energy mainly contributes on selecting these cluster heads. Each cluster has, at most, M cluster heads. The other nodes become cluster members in such a way that each associates with a single neighbored cluster head, which would produce a distributed data balanced clustering. In each cluster, nodes create an encoding vector that index all of the possible encountered source nodes in the intra-cluster transmission phase.

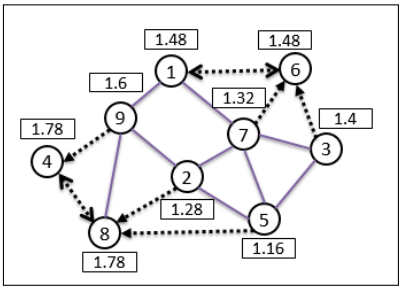
6.1.2 MuCar Layer: Trajectory Decision

The mobile collector (MuCar) approaches the selected polling points where the transmission ranges of all of the cluster heads are covered by those points. MuCar has a multi-radio capability that enables it to simultaneously receives all of the packets that have been sent from the cluster heads. For each polling point, MuCar broadcasts a control message to the local cluster heads, giving each a unique channel for uploading the transmission phase. The local cluster heads transmit their data with their dedicated channels concurrently without interference. Some of the received packets are encoded. MuCar can decode those packets after receiving all of the packets from the cluster heads in that polling point by having enough linearly independent packets for decoding.

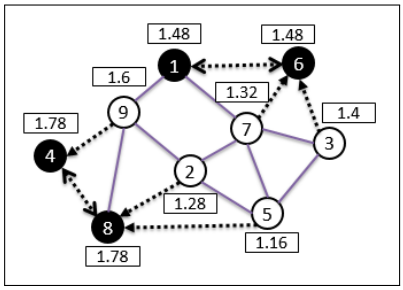
The MuCar trajectory decision is based on two steps: The first step is finding the most efficient polling points where the MuCar should go and collect. Finding these points is implemented by the Cluster Polling Points (CPP) algorithm 14, which I will discuss later. The second step is finding the minimum route cost that passes through each polling point before it returns to the base. In the next section, I will show how to find the MuCar trajectory path.



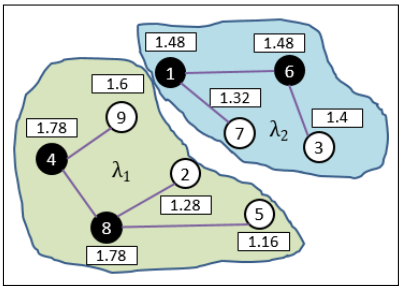
(a) Network configuration.



(b) Initialization.



(c) Status claim: dark circles are cluster heads.



(d) Cluster construction and channel assignment.

Figure 6.4: An example of network clustering for the first four phases with $M = 2$.

6.2 MuCC Protocol

MuCC cluster-based protocol consists of six phases:

Initialization Phase

Each sensor node broadcasts its battery status - a value indicates its residual energy - to its single-hopped neighbors. Then, each node sums up the battery status from the highest $M-1$ of the neighbors, including its result value, which is called *priority*. This technique is similar to LBC-DDU initialization phase. Figure 6.4a depicts an example of a network with 9 sensors and $M = 2$. Each sensor has its battery status between 0 and 1, where 1 indicates a full battery and 0 indicates a dead battery, which means the node is no longer available. The edges denote the single-hop communication on each pair of nodes. In the initialization phase, Figure 6.4b shows the *priority* value in each node. For each node, the dotted arrows indicate its $M-1$ cluster head lists. In the example, node 2's cluster head list has only node 8.

Status Claim Phase

The system defines two status thresholds: τ_h for a node to be become a cluster head, and τ_m for a node to be become a member node. These thresholds enable a node to declare itself to either status based on its *priority* value before reaching its maximum number of iterations. A sensor broadcasts its current status when its battery value satisfies one of these thresholds; otherwise, iteratively, this sensor finds the potential and the candidate cluster heads list. Whenever there is an update on each list, the node broadcasts the updates until some nodes announce their status claim, which might reflect their claiming process. For each member assignment, the cluster head broadcasts a message that tells its neighbors the content of its member list. Figure 6.4c depicts the cluster heads with dark circles, while the remaining nodes are the cluster members.

Cluster Construction Phase

Each sensor finalizes its potential cluster head list if the sensor has not already been claimed as a cluster head. If that list was empty, then the sensor will claim to be a cluster head. Otherwise, the sensor becomes a member and picks its cluster head with the minimum

members, which it already knows from the member lists that have been broadcast by the cluster heads. Figure 6.4d shows two clusters where each member is assigned to its cluster head. Observe that node 7 chooses node 1 to be its cluster head; it does not choose node 6, because node 6 already has a member, while node 1 has no member.

Channel Assignment Phase

MuCar scans the network to find the cluster heads and creates a list, a Cluster Head Location (CHL). Each cluster is assigned by a unique channel λ_i. Consider an undirected graph $G = (V, E)$, where V is the set of vertices, and each vertex is a cluster, and E is the set of edges. An edge connects two vertices, v_i and v_j, if, and only if, the node u exists in cluster v_i and neighbors the node w in cluster v_j. Let $|E_v|$ be the total number of edges for the vertex v. Algorithm 13 shows the cluster channel assignment. Initially, all the cluster channels are set to null and each cluster has a sorted channel candidate list that contains all of the possible channel assignments from 1 to Λ, where Λ is the maximum available channel number a system can have. In steps 6-13, MuCC investigates all the clusters. It picks the cluster with maximum neighbors and assigns it to the first available channel from the cluster's local channel candidate list. After that, it removes that channel from all of the cluster's neighbors channel candidate lists, where $NB(v_i)$ is the set of nodes that neighbors node v_i. Then, it eliminates that cluster from investigating. MuCC repeats this iteration until all clusters are assigned to their channels. This process can significantly eliminate data collisions for the inter-cluster communications. Back to Figure 6.4d, the left cluster communication channel is assigned to channel λ_1, while the right cluster communication channel is assigned to channel λ_2.

Scalable Encoding Vector Phase

In order to implement network coding, the encoding vector has to be appended to the packet's header. The encoding vector represents the coefficient values of all of the source nodes, which, however, may not be scalable for a large network, and that large network can

Algorithm 13 Cluster Channel Assignment

Input: The undirected graph $G = (V, E)$.
Output: Channel assignment per cluster.
for all $v_i \in V$ **do**
  $v_i.channel = 0$;
  $v_i.candidate\_channels = \{\lambda_i | \lambda_i = \{1, \ldots \Lambda\}\}$;
end for
while $(V \neq \phi)$ **do**
  Pick a vertex v such that $\forall v_i \in V$, $\max(|E_{v_i}|) = v$;
  $v.channel \leftarrow \lambda_i$ where λ_i is the first available channel in $v_i.candidate\_channels$;
  for all $v_j \in NB(v_i)$ **do**
    Remove λ_i from $v_j.candidate\_channels$;
  end for
  Remove v_i from V;
end while

lead to insufficient data communication and collection. This phase significantly reduces the packet encoding vector overhead and shortens it from n to N_C, where n is the total number of nodes on the network, and N_C is the total number of nodes in cluster C. Initially, each cluster head creates an encoding vector list that contains its indexed cluster member list including itself. Iteratively, the cluster head broadcasts a control message that includes its encoding vector list to all other cluster heads in the same cluster. Then, it re-indexes its encoding vector list including the other cluster head list until the M rounds of transmissions are satisfied. For each cluster head, all of the updated encoding vector lists have the same list of indexed cluster members and cluster heads.

Data Transmission Phase

Each cluster member stores its own packets in both the storage queue and transmission queue. Then, it transmits packets in the transmission queue to its cluster head. Another cluster member can overhear and store them in its storage queue. Whenever its storage queue is not empty, it applies random linear network coding among the storage packets and inserts the new encoded packet to its transmission queue to transmit. Eventually, a cluster head receives all packets whether it is from its cluster member or not.

6.3 MuCar Motion Planning

In order for MuCar to collect the data from all of the cluster heads, the MuCar trajectory decision has to be made, which is because the system has a latency requirement that makes the MuCar trajectory decision and data uploading scheduling quite significant. There are two steps for the MuCar trajectory decision: The first step is to find the most utilized polling points that reduce the uploading scheduling latency. I provide the cluster polling points (CPP) algorithm 14, which finds the polling points efficiently, where each polling point is chosen based on the nearest and mostly covered cluster heads.

MuCar starts at the nearest possible location for establishing communication with another node on the network. In the CPP algorithm, MuCar is initially located at the starting point p_0, where it is the location of the closest cluster head on CHL. In step 3, the goal of MuCar is to cover all of the cluster heads in CHL list. I use the Euclidean metric to calculate the distance in two dimensions plane. In steps 5-9, MuCar eliminates all of the cluster heads that are located twice as farther than the sensor transmission range R_T. The reason for this elimination is that there is no chance to find a new position with radius R_T, which can cover those far cluster heads. Calculate_Centroid(P) is a function that returns the centroid position $p_c = (x_c, y_c)$ of the points in the set P where $x_c = \frac{x_1+x_2+...+x_k}{k}$, and $y_c = \frac{y_1+y_2+...+y_k}{k}$ where k is the number of positions in P. D_{p_c,p_i} represents the Euclidean distance between the two points p_c and p_i. In steps 11-18, MuCar keeps eliminating the cluster heads that are not reachable from the candidate polling point p_c and keeps updating the p_c for each elimination process until all of the remaining cluster heads can be reached from the updated p_c position. In steps 19-21, MuCar adds p_c to the polling point set PP, eliminates the cluster heads that have already been covered, and assigns a new starting position from the Cluster Head List CHL in such a way that it has the shortest Euclidean distance from p_c. Finally, MuCar repeats the steps until all cluster heads are covered. The result is the set of polling points PP.

The second step is to find the MuCar route in order to collect data from the cluster

heads with the minimum trajectory cost. Finding the MuCar route is quite challenging. This problem is known as the Traveling Salesman Problem TSP, which is an NP-Hard problem. The TSP tries to find the best route with a minimum cost in order to visit each city only once before returning back to the starting city. Let $\widetilde{n}$ represent the number of polling points. A Held-Karp algorithm solves the TSP in a running time $\mathcal{O}(\widetilde{n}^2 2^{\widetilde{n}})$ using dynamic programming and requires $\mathcal{O}(\widetilde{n}2^{\widetilde{n}})$ of memory space. For large $\widetilde{n}$, however, this algorithm is hard to implement. Therefore, I need to use some heuristics in order to proximate the optimized solution in a reasonable running time. The most simplest and straight forward tour construction heuristic for TSP is the $Nearest\ Neighbor$ heuristic. At first, MuCar selects the base station as its starting point(PP_0). Then, it finds the nearest polling point (PP_1) from PP_0. MuCar finds the next unselected polling point in the same way until it covers all the polling points. After it covers all the polling points, it returns to the starting point PP_0. This algorithm runs in $\mathcal{O}(\widetilde{n}^2)$.

Now, the tour improvement heuristic is implemented in order to create MuCar tour near, or equivalent, to the optimized solution as in Held-Karp. The most common optimization technique is the 2-opt local search algorithm, which is proposed by Croes [1958]. The idea of this algorithm is that it removes two edges from the constructed tour and reconnects the two paths by selecting two shorter edges. For example, 2-opt selects an edge (p_1, p_2), finds another edge (p_3, p_4) and performs the complete move if, and only if, $D_{p_1,p_2} + D_{p_3,p_4} > D_{p_1,p_4} + D_{p_2,p_3}$. The running time of 2-opt algorithm is $\mathcal{O}(\widetilde{n}^2)$. Usually, the 2-opt heuristic will create a tour with a cost less than 5% above the Held-Karp bound [75].

6.4 System Analysis

Each node is integrated with a GPS receiver that allows it to transmit its location to its neighbors during the initialization phase. I am assuming that each node is surrounded by B neighbors. Implementing network coding requires overheads in both packets and a number

Algorithm 14 Cluster Polling Points (CPP)

Initialization: MuCar is located at position (x_0, y_0) that is the closest location to a cluster head.
The set of Polling Points $PP \leftarrow \Phi$.
Input: The set of cluster head locations CHL.
while $(CHL \neq \phi)$ **do**
  The current set of cluster head positions $P \leftarrow CHL$;
  for all $p_i \in P$ **do**
    if $(D_{p_0,p_i} > 2R_T)$ **then**
      There is no chance for the MuCar to collect both cluster heads in p_0 and p_i.
      $P = P - \{p_i\}$;
    end if
  end for
  $P_c \leftarrow Calculate\_Centroid(P)$;
  for all $p_i \in P$ **do**
    if $(D_{p_c,p_i} > R_T)$ **then**
      repeat
        $P = P - \{p_j\}$ s.t. $D_{p_0,p_j} = Max(D_{p_0,p_i})$ for $i, j \leq k$
        $P_c \leftarrow Calculate\_Centroid(P)$;
      until $\forall p_i \in P$: $(D_{p_c,p_i} \leq R_T)$
    end if
  end for
  $PP \leftarrow PP \cup \{p_c\}$
  $CHL \leftarrow CHL - P$
  $p'_0 \leftarrow p_j$ where $p_j = Min(D_{p_c,p_j})\ \forall p_j \in CHL$
end while
Output: PP the set of polling points.

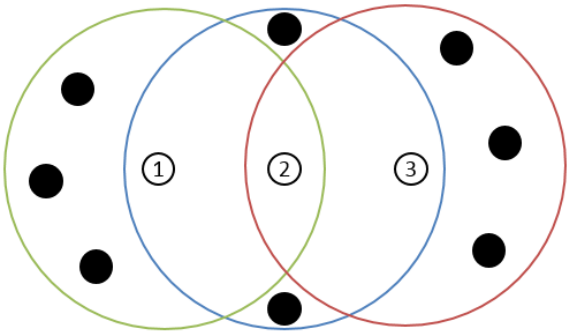


Figure 6.5: An example of a cluster that has three cluster heads (1,2 and 3) and 8 scattered cluster members (dark circles). Each circle represents the transmission range for each cluster head. Each cluster head has exactly $B = 4$ neighbored nodes.

of transmissions. For encoding, nodes that belong to the same cluster are synchronized in order to provide a higher chance for a node to overhear packets from different nodes, encode these packets, and transmit them in each transmission session. The encoding vector - which is embedded in the packet's header - represents the coefficient values α_i for each encoded data. For simplicity, I am assuming each node has a single packet to send per transmission session. Eventually, the only factor deciding the size of the encoding vector (CV_Size) is the total number of nodes N_C for the cluster C. In the worst case, a cluster member may overhear packets from all of the other cluster members from the same cluster. To find N_C, each cluster head broadcasts its local member list in M iterations and updates the list to include the list from all of the other cluster heads. Each update is considered an iteration.

Ideally, enlarging M produces larger N_C, which inevitably increases CV_Size. This enlargement is because the encoding vector is indexed with nodes encoded with coefficient values. With this consideration, each coefficient α_i in the encoding vector needs β Bytes to be represented in the packet's header. Therefore,

$$CV\_Size = \beta.N_C \tag{6.1}$$

Lemma 6.4.1. *Let N_C be the total number of nodes in cluster C, and each node has at most B neighbors. Let M be the maximum number of cluster heads. Then, $N_C \leq (B-1)M + 2$ for $M \geq 2$.*

Proof. I proof this lemma by induction. Assume that $f(M)$ is a function that returns the maximum number of nodes in a cluster with the input M. This means that the lemma can be transformed into $f(M) = M(B-1)+2$. Consider the distributed nodes in Figure 6.5. Each node has, at most, $B = 4$ neighbors. For $M = 2$, CV_Size needs 8 Bytes of packet overhead, which includes the cluster heads 1 and 2 and their six neighbors by assuming that the coefficient size $\beta = 1$ Byte. The base case in the proof is for $M = 2$. I can observe that $f(2) = (4-1)2+2 = 8$, which is equal to what I have just shown. For M >2, the worst case scenario exists if, and only if, the following three conditions are satisfied:

1. Each cluster head has exactly B neighbors
2. Only two cluster heads have exactly one neighbored cluster head
3. All the remaining cluster heads have exactly two neighbored cluster heads

Figure 6.5 depicts an example of the worst case scenario for $M = 3$, where each cluster head has exactly $B = 4$ neighbors, and there are only two cluster heads (1 and 3) that have one neighbored cluster head (2). Also, the remaining cluster head (2) has exactly two neighbored cluster heads (1 and 3). To find the increment factor of the number of cluster heads M to the total number of nodes in the cluster N_C, consider the change of M from 2 to 3. Once the third cluster head is added to the cluster, it adds, at most, three more nodes $(B-1)$. The reason this increment happens is that the cluster member itself is already in the cluster before turning into a cluster head. Furthermore, the maximum number of neighbors that this new cluster head can have to bring into the new shaped cluster is $B-1$, because one of its B neighbors is the connected cluster head that allows it to join the cluster. Otherwise, it cannot join this cluster and has to create or join different cluster based on the *Cluster Construction Phase*.

The next part of the proof is the inductive step. Let $f(k) = (B-1)k+2$ be true for $k \geq 2$. The goal is to proof that $f(k+1) = (B-1)(k+1)+2$ is also true. Based on what I have shown in incrementing the size of M, $f(k+1) = f(k)+(B-1)$ is true where $B-1$ are the new member nodes that I add to the cluster by increasing the number of cluster heads

by one. This implies $f(k+1) = (B-1)k+2+(B-1)$, which is equal to $(B-1)(k+1)+2$ and that concludes the proof. □

6.4.1 Packet Size Boundary

With the presence of Packet Error Rates (PERs), it is reasonable to consider the Lemma 6.4.1 to provide a reliable communication using network coding. On the other side, the packet overhead may become too expensive in data transmission. This tradeoff can be observed by considering factors such as M, Packet Delivery Rate (PDR), packet replication (R), and the packet size (P_size).

With network coding, the packet size (P_size) can be formulated as:

$$P\_size \geq CV\_Size + data + \partial \tag{6.2}$$

where ∂ is the constant packet overhead.

Theorem 6.4.2. *Let P_size be the maximum packet size and r be the network throughput where each packet is transmitted in t seconds. Let β be the number of Bytes needed to represent a single coefficient α in the encoding vector. Then,*

$$M \leq \frac{P\_size - r.t - 2\beta - \partial}{\beta(B-1)} \tag{6.3}$$

Proof. Let throughput $(r) = \frac{data}{time(t)}$. Then,

$$data = r \cdot t. \tag{6.4}$$

Starting from Equation (6.2):

$$
\begin{aligned}
P\_size &\geq CV\_Size + data + \partial \\
&\geq \beta \cdot N_C + data + \partial \qquad \text{(using Equation (6.1))} \\
&\geq \beta \cdot N_C + r \cdot t + \partial \qquad \text{(using Equation (6.4))} \\
N_C &\leq \frac{P\_size - r \cdot t - \partial}{\beta} \\
(B-1)M + 2 &\leq \frac{P\_size - r \cdot t - \partial}{\beta} \qquad \text{(from Lemma 6.4.1)} \\
M &\leq \frac{P\_size - r \cdot t - 2\beta - \partial}{\beta \cdot (B-1)}
\end{aligned}
$$

□

Once the system has monitored unbearable PDR, it increases the size of M by one. The system reinitialization phase is processed among the other phases in order to set up the communication process on the network. If PDR is still not satisfiable, the system repeats the procedure until PDR is targeted or the Equation (6.3) is not satisfied. This increment of the size of M can maintain a reasonable data size while transmitting and can provide a more reliable network coding communication. If this does not satisfy the PDR, data replication R is needed to reduce the packets error rates. Initially, $R = 1$ and the PDR are checked to maintain a reliable communication. If not, the value of R increments by one and rechecks the PDR once again. This process is repeated until the PDR is satisfied.

6.4.2 Transmission Queue

I wanted to find the transmission queue upper bound in order to estimate the data uploading latency on the MuCar. In MuCode protocol, each node has two queues. The storage queue (SQ) where all of the generated, received, and overheard packets are stored. The generated packets are the source data packets, which are generated by the sensors. The received packets are the packets that are sent from a cluster member to its cluster head. The

overheard packets are those packets that are sent from a sensor either to none of its cluster head or to a cluster member. The transmission queue (TQ) stores the packets that need to be transmitted. The TQ stores all the generated packets and the received packets. As for the overheard packets, a sensor applies a network coding technique to the storage packets in order to create the encoded packet p_e and inserts it to the TQ for each transmission session.

Theorem 6.4.3. *Let $N_m(u)$ be the number of cluster members in such a way that u is their cluster head. Assuming each node transmits a single packet per transmission session, the total transmitted packets in cluster head u is,*

$$T_{pkt}(u) \leq 2(N_m(u) + 1) \qquad \forall N_m(u) \geq 1 \tag{6.5}$$

Proof. To prove Equation (6.5) by induction, the base case is when $N_m(u) = 1$. Consider the worst base case scenario where there are y cluster heads and z cluster members in such a way that a cluster head u has only one cluster member v, which has u as its cluster head i.e. $N_m(u) = 1$. Based on the adopted transmission queue, the cluster member v transmits both the generated packet to its cluster head u and the encoded overhead packets from other members. The u's TQ has 4 packets: the two received packets from v, its own generated packet, and an encoded packet that u overheard from other cluster heads and members. Hence, $T_{pkt}(u) = 4 = 2(1 + 1)$.

Let the equation $f(k) = 2(k+1)$ be true for $k \geq 1$. The goal is to proof that $f(k+1) = 2((k + 1) + 1)$ is also true. Incrementing the size of k leads to $f(k + 1) = f(k) + 2$. The reason for this exact increment is that when the one more cluster member of u is increased, the generated and the received packets are both stored in u's TQ queue. At the same time, whenever changes in other cluster members that do not consider u as their cluster head, u still has the single encoded packet to transmit. This implies $f(k+1) = 2(k+1) + 2$, which equals $2((k + 1) + 1)$, thus concluding the proof. □

Section 6.5 describes my Adaptive Cluster Head algorithm in order to minimize the

number of polling points and cluster heads.

6.4.3 Energy Consumption

The transmission queue size proportionally reflects the sensors' energy consumption. I only focus on the energy consumption in cluster heads since they transmit a huge amount of data compared to the cluster members for each transmission session.

In reliable communications, the ideal energy saving per sensor is to have zero relay sensors because each sensor transmits its own packet to the mobile collector. However, in lossy networks, this scheme maximizes the trajectory latency since the mobile collector has to visit all the sensors to collect the data. This scheme also minimizes the Packet Delivery Rate (PDR) in high Packet Error Rates because there is no chance to encode packets and transmit them along the original packets.

Unlike this scheme, the $MuCC$ protocol decreases the energy consumption per cluster head while increasing the size of M, as long as PDR is satisfiable. If PDR is relatively small while increasing M, packets retransmissions are needed, which reduce the energy saving. Notice that the increased size of M leads to lessening the number of clusters. This inevitably increases the size of CV_Size since more nodes are included and needed to encode during data transmissions.

Considering the number of cluster members N_m per cluster head is another way to save energy. From the Equation (6.5), the total number of transmission packets $T_{pkt}(u)$ for cluster head u is proportional to $N_m(u)$. Minimizing $T_{pkt}(u)$ leads to equalizing the N_m values for all the cluster heads. This can significantly decrease the energy consumption per cluster head.

6.5 Adaptive Cluster Heads

A cluster with M cluster heads forces MuCar to retrieve data from at most $M/2$ polling points. Even with the CPP algorithm, the polling points may remain the same. Therefore,

the uploading latency from the cluster heads become undesirable. In order to minimize this latency, algorithm 15 (ACH) rearranges the cluster heads to minimize the number of polling points. The rearranging procedure considers swapping between cluster head and a cluster member or changing the cluster head into a cluster member.

The MuCar runs the ACH algorithm that runs for each polling point pp in the set of polling points PP, which was already found in algorithm 14. I define the set $NCP(pp_i)$ to be the neighbored candidate polling points, which possibly can merge with pp_j to a new polling point. These possible polling points should not be twice as far in transiting range ; otherwise, there is no chance to find a polling point that can cover more cluster heads. Let NCP be the union set of all neighbored candidate polling point sets possible with respect to each polling point.

The ACH iteration covers the NCP element, pp, that is sorted based on merging possibilities in descending order. The next iteration tries to find a candidate polling point pp_i in $NCP(pp)$ that can merge with where the set of cluster head locations P is constructed. P includes all of the cluster heads locations that are within the transmission range of one of the two polling points pp and pp_i. For example, Figure 6.6a shows the set $P = \{1, 2, 3, 4, 5, 6, 7\}$ for the cluster heads that are within the transmission range of the polling points p_1 and p_2.

Step 11, the P_c is the centroid position for all the positions in P. Let ϵ be the residual energy threshold that allows cluster member to swap its status with a cluster head. Let $N_{CH}(u)$ be the number of cluster heads that neighbors the node u. In steps 12-18, for each cluster head CH_j in P that is out of the transmission range of the centroid position P_c, a member node m temporarily swaps its status with CH_j's status if, and only if, it satisfies all of the following conditions:

1. neighbors the cluster head CH_j
2. neighbors another cluster head
3. the difference between the residual energy in m and CH_j does not exceed ϵ
4. resides within the transmission range of the centroid position P_c.

Algorithm 15 Adaptive Cluster Heads (ACH)

1: **for each** $pp_i \in PP$ **do**
2: NCP(pp_i) $\leftarrow \{pp_j |\ pp_j \in PP \wedge D_{pp_i,pp_j} \leq\ 2R_T\}$;
3: **end for**
4: NCP $\leftarrow \{\cup$ NCP(pp_i) $|\ \forall pp_i \in PP\}$;
5: Sort NCP based on the size of each element set NCP(pp_i) in descending order;
6: **repeat**
7: $NCP(pp) \leftarrow EXTRACT(NCP)$;
8: **repeat**
9: **for each** $pp_i \in NCP(pp)$ where $pp \neq pp_i$ **do**
10: $P \leftarrow \{p \mid p \in CHL \wedge (D_{p,pp}\ \leq\ R_T \vee D_{p,pp_i}\ \leq\ R_T)$;
11: $P_c \leftarrow Calculate\_Centroid(P)$;
12: **for each** cluster head $CH_j \in P$ where $D_{CH_j,P_c} > R_T$ **do**
13: **if** there exists a member node m that neighbors cluster head (CH_j), $N_{CH}(m)$ >1, $e(CH_j) - e(m) \leq \epsilon$, and $D_{m,P_c} \leq R_T$ **then**
14: Both m and CH_j swap their status temporarily;
15: **else if** for all member node m that neighbors cluster head (CH_j), $N_{CH}(m)$ >1 **then**
16: Change CH_i status from cluster head to member temporarily;
17: **end if**
18: **end for**
19: **if** $\forall CH_j$, $D_{CH_j,P_c} \leq R_T$ and $\forall m, N_{CH}(m) \geq 1$ **then**
20: Confirm the swapping/changing status;
21: Update P and recalculate P_c;
22: $PP \leftarrow PP \cup \{P_c\} - \{pp \cup pp_i\}$;
23: $pp \leftarrow P_c$;
24: For every $pp_j \in PP$, update $NCP(pp_j)$;
25: Update NCP;
26: **else**
27: Rollback the swapping/changing status;
28: $pp_{max} \leftarrow pp_k$ s.t. $pp_k \in NCP(pp)$, $pp_k \neq pp$ and $CH_x \in NB(pp_k)$ where CH_x has the maximum distance to P_c ;
29: $NCP(pp) \leftarrow NCP(pp) - \{pp_{max}\}$;
30: $NCP(pp_{max}) \leftarrow NCP(pp_{max}) - \{pp\}$;
31: **end if**
32: **end for**
33: **until** $(NCP(pp) = \{pp\})$
34: **until** (NCP is empty)
35: Apply cluster construction phase.

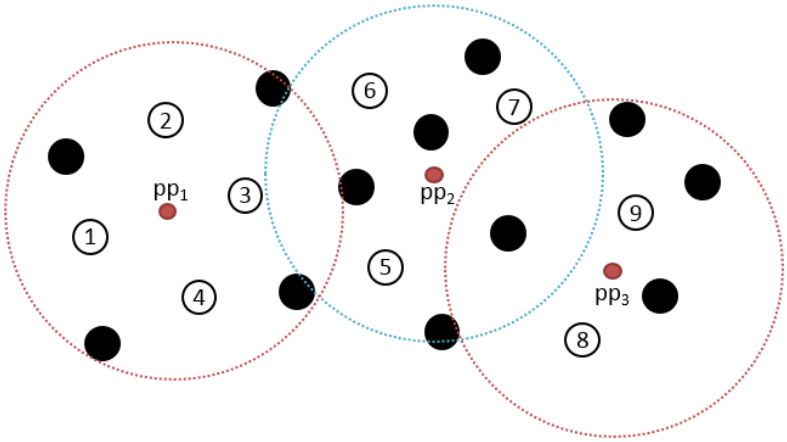
If no such m satisfies these conditions, and all members that neighbor CH_j neighbors another cluster head, then CH_ij temporarily changes its status into cluster member instead of a cluster head.

In steps 19-29, if all of the cluster heads are within the transmission range from the centroid position P_c, and if all of the members have at least one cluster head neighbor,

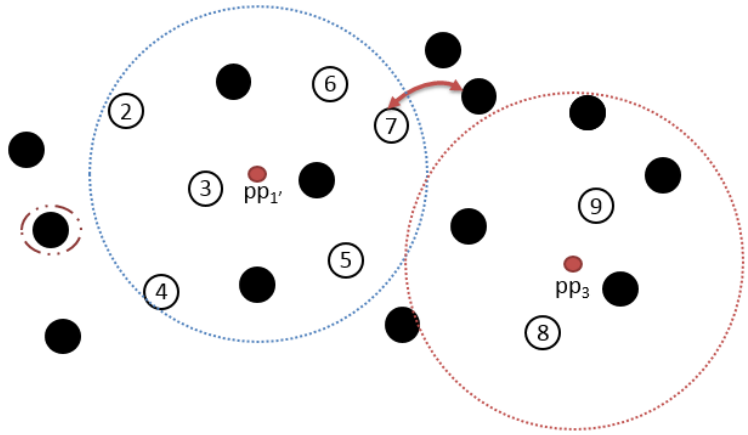
then the swapping and changing status should be confirmed, the two polling points pp and pp_i should be removed from PP, P_c should be added to PP instead , and NCP should be updated. Otherwise, the changed status should be rolled to its previous state and the polling point p_{max} should be removed from $NCP(pp)$, and vice versa when p_{max} covers the cluster head CH_x, and CH_x has the maximum distance from the centroid polling point. Once all the merging processes are finalized, the cluster construction phase should be finalized but only as far as assigning the members to their dedicated cluster head.

Figure 6.6a shows an example of a cluster with nine cluster heads and twelve members. By applying the algorithm 14, MuCar will retrieve the data from three polling points pp_1, pp_2, and pp_3. Next, I apply the algorithm 15. For the first iteration, pp equals pp_1, and the only neighbored candidate polling point is pp_2. The set $P = \{1, 2, 3, 4, 5, 6, 7\}$, which are the cluster heads that reside within pp_1 and pp_2 transmission ranges. The centroid position, pp_1, will exclude cluster heads CH_1 and CH_7 as in 6.6b. For the case of CH_1, this cluster head does not have a neighbored member that satisfies the four conditions, but all its members have another neighbored cluster heads. Therefore, CH_1 changes its status to cluster member temporarily. In the case of CH_7, this cluster head has a member that satisfies the four conditions assuming the difference between that member's residual energy and CH_7 residual energy is less than ϵ. Therefore, CH_7 swaps its status with that member temporarily. Now, since all of the cluster heads are within the $pp_{1'}$ transmission range, the swapping changes are confirmed.

For the second iteration, the set $P = \{2, 3, 4, 5, 6, 7, 8, 9\}$ for the polling points $pp_{1'}$ and pp_3. The centroid position excludes CH_2, CH_3, CH_4, CH_8, and CH_9. Since CH_9 swap its status with another member closer to the centroid position, some members will not be associated with any cluster head. Therefore, the process is rolled back. The final result in this example is the reduction of polling points from 3 to 2, which also reduces the uploading latency in MuCar. The ACH algorithm running time is $\mathcal{O}(\widetilde{n}^4\ \Gamma(\widetilde{n})\ n)$. This bi-quadratic symbol represents the number of polling points which are much smaller than the number of nodes on the network.



(a) The dark circles represent the cluster members, while the numbered circles represent the cluster heads. Each dotted circle is the transmission range for each polling point pp_i.



(b) The cluster head CH_1 changes its status to a cluster member, while CH_7 swap its status with its member for the new polling point, $pp_{1'}$. The second iteration rolled back because CH_9 leaves some of its members without any association with the other cluster heads after swapping with its member.

Figure 6.6: An example of a cluster with $M = 9$ cluster heads.

6.6 Evaluation

I evaluated MuCC against the LBC-DDU approach in two different aspects. The reason for this comparison is that, as I have mentioned previously, LBC-DDU is a cluster-based protocol for mobile data collection, and it is the most closely related to my work. The first

aspect of the experiment is conducted in two different area sizes so that (140m x 120m) is setup for a number of nodes n = 9, and the area (260m x 200m) is setup for a number of nodes n = 20. These experiments used a variety of wireless Packet Error Rates (PERs). MuCC uses the standard Random Linear Network Coding (RLNC) technique, while SenCar uses the traditional packet transmission. The second aspect's experiment considers the total distance that the $MuCar$ has to take to collect the data from all of the cluster heads with a various number of nodes that range from 30 to 130 with a different size of the network.

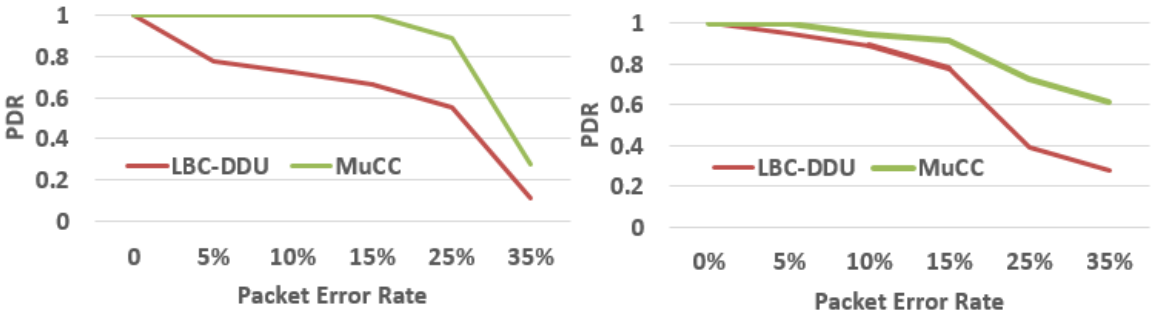
In the first set of experiments, there are 3 clusters where each has three nodes, where the size of the area is (140m x 120m). I conducted three sub-experiments by varying the number of cluster heads (M) from 1 to 3 per cluster. I am interested in several metrics for comparison: Packet Delivery Rate (PDR), energy consumption, and latency.

6.6.1 Packet Delivery Rate (PDR)

Figure 6.7 depicts the average packet delivery rate versus PER for 3 clusters and n=9 networks with different M values. In Figure 6.7-a for M =1, MuCC's PDR outperforms LBC-DDU's PDR whenever there is a PER. The reason for this superiority is that network coding is more resistant than the traditional non-coding protocol. Packets are combined by members when they overhear packets from their neighbors. This combination gives the cluster head a higher chance to decode the encoded packets by having enough linearly independent packets. On the other hand, SenCar's packets cannot be decoded, since they are not encoded. SenCar only transmits its own data, which leads to unreliable communication in lossy networks.

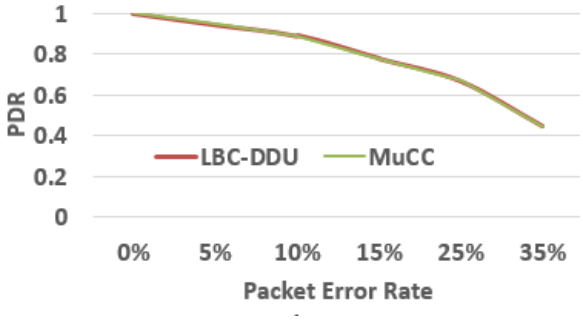
In 6.7-b for M =2, the PDR in LBC-DDU is close but not superior to - MuCCs, when PER $\leq$ 15%. The reason for this close PDR is because of network coding. Sometimes, in network coding, there is a small chance for a cluster member to overhear packets from neighbors when the number of cluster heads is increased while, at the same time, the number of members is decreased. For PER = 35% where the packet error rates are increased, the significance of network coding can be noticed by which the PDR in LBC-DDU and MuCC

are 0.11 and 0.27 respectively.



(a) PDR VS PER (M = 1)

(b) PDR VS PER (M = 2)



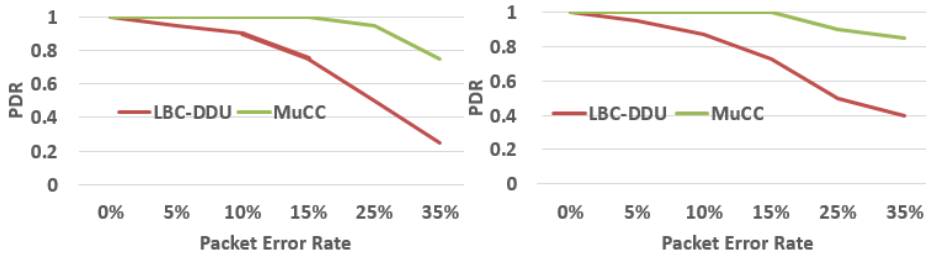
(c) PDR VS PER (M = 3)

Figure 6.7: The packet delivery rate in different packet error rates per session with a various number of cluster heads. The total number of nodes is 9.

Once all nodes become cluster heads, as you can see in Figure 6.7-c, the PDR in both MuCC and LBC-DDU become identical. This mirroring is because there is no chance for overhearing packets, therefore implementing network coding is useless.

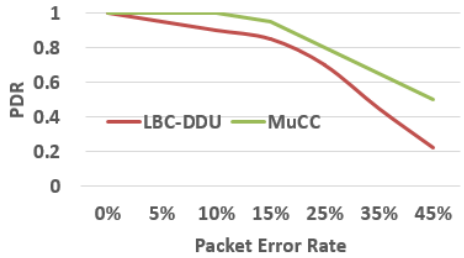
For the second set of experiments, there are 5 clusters with 3 to 5 nodes each with a total of n = 20, where the size of the area is (260m x 200m). I conducted another three sub-experiment by varying the number of cluster heads (M) = 1,2 or 4 per cluster. In Figure 6.8-a when M =1, the MuCC's PDR is very high compared to the MuCC itself in the smaller sized network. It is also evident that MuCC's PDR is superior to LBC-DDU's

PDR for the same system configuration. Like I said, the higher neighbored the members, the higher the chance for overhearing packets, thus more encoding packets are produced. This process will eventually provide a more reliable communication in lossy networks.



(a) PDR VS PER (M = 1)

(b) PDR VS PER (M = 2)



(c) PDR VS PER (M = 4)

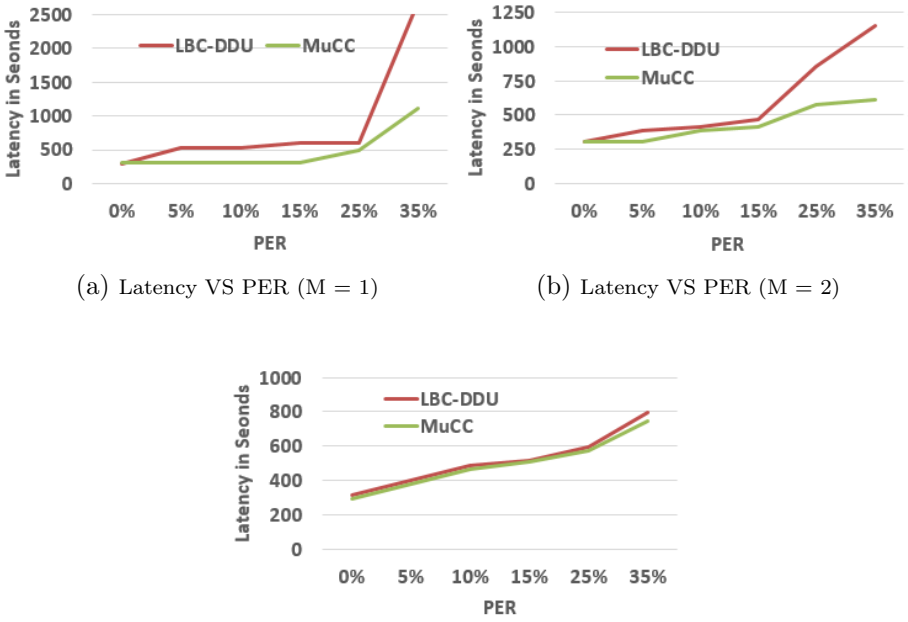
Figure 6.8: The packet delivery rate in different packet error rates per session with a various number of cluster heads. The total number of nodes is 20.

When M = 2, the PDR in both MuCC and LBC-DDU are improved when compared to the previous network configuration. Balancing between cluster members and cluster heads will reduce the number of total transmissions in members. Also, having enough encoded packets can leverage the overall PDR performance, as seen in Figure 6.8-b.

Again, whenever the total number of cluster heads become larger than the members, the chance of overhearing is significantly reduced, thus network coding will not have that high a superiority against SenCar, as seen in Figure 6.8-c for M = 4. Unbalanced distribution can lead to small PDR for high PER.

6.6.2 Packets Latency

Figure 6.9 depicts the packet delivery latency in different Packet Error Rates when n = 9. In Figure 6.9-a the MuCC's latency is about half of LBC-DDU's when PER = 15%. The difference between each is reduced when PER = 25%. Once the PER gets higher, the SenCar's latency becomes extensive because of the very high packet error rate. However, MuCC can still maintain reasonable latency in high packet loss rate.



(a) Latency VS PER (M = 1)

(b) Latency VS PER (M = 2)

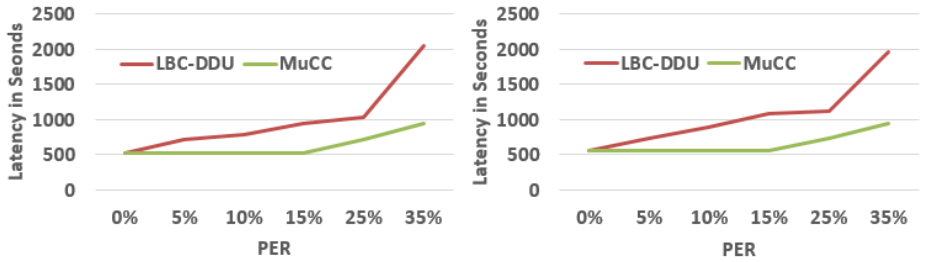
(c) Latency VS PER (M = 3)

Figure 6.9: The latency in different packet error rates per session with a various number of cluster heads. The total number of nodes is 9.

In Figure 6.9-b, the latency in both LBC-DDU and MuCC are close to each other when PER $\leq$ 15%. Here, the clusters are balanced between members and cluster heads. The latency gets higher in LBC-DDU than MuCC's when PER gets increased, whereas in Figure 6.9-c, both LBC-DDU and MuCC have the same latency because there is no chance for applying network coding.

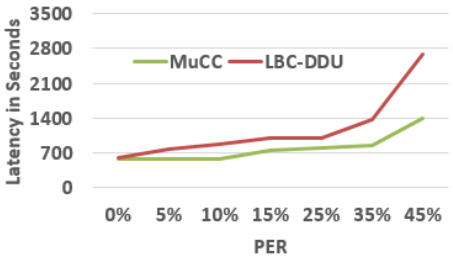
Figure 6.10 depicts the packets delivery latency in different packet error rates when n

= 20. The MuCC's latency is steady when PER $\leq$ 15% for M =1 and 2. The latency in MuCC also becomes stable when PER $\leq$ 10% for M = 4. On the other hand, LBC-DDU's latency gets higher once there is PER. The latency in LBC-DDU gets worse when PER gets higher compared to MuCC in latency.



(a) Latency VS PER (M = 1)

(b) Latency VS PER (M = 2)



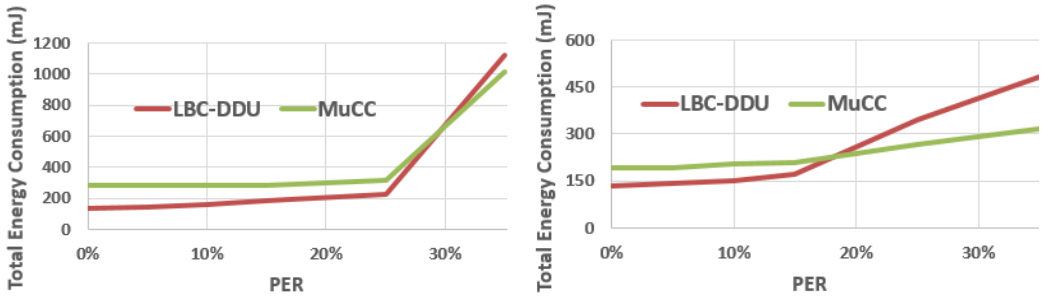
(c) Latency VS PER (M = 4)

Figure 6.10: The latency in different packet error rates per session with a various number of cluster heads. The total number of nodes is 20.

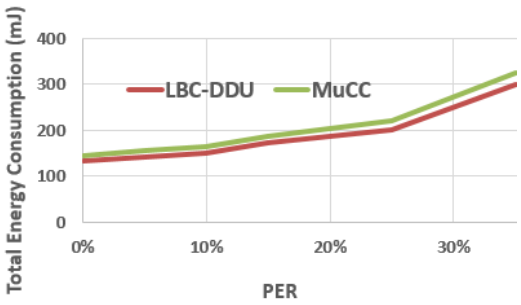
6.6.3 Energy Consumption

In this part of my evaluation, I used the same energy model as it has been described in Section 5.4.4. Figure 6.11 depicts the total nodes energy consumption in different packet error rates for n = 9. I excluded the energy consumption while setting up the members and cluster heads status claim phase. SenCar consumes more energy than my proposed MuCC when PER $\geq$ 35% and $>$ 15% for M = 1 and 2 respectively. However, the MuCC approach consumes more energy in nodes than SenCar when M =3. The reason for this high energy consumption is that there is no chance to use network coding when there is

no chance for overhearing. Furthermore, there is a packet overhead for the encoding vector when implementing network coding.



(a) Energy Consumption VS PER (M = 1) (b) Energy Consumption VS PER (M = 2)



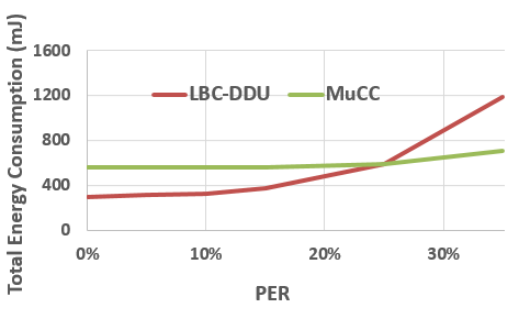
(c) Energy Consumption VS PER (M = 3)

Figure 6.11: Consumption energy unit per session in different packet error rates with a various number of cluster heads. The total number of nodes is 9.

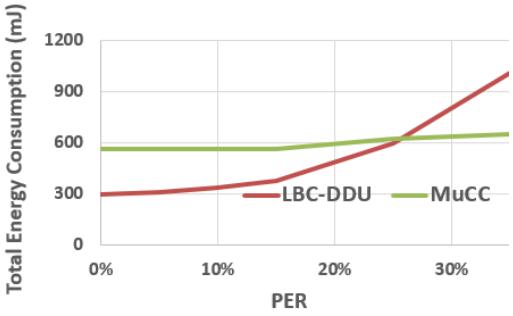
Figure 6.12 depicts the total nodes energy consumption in different packet error rates for n = 20. Again, LBC-DDU's nodes consume more energy than MuCC's when PER $\geq$ 25% and $\geq$ 35% for M = 1 and 2. MuCC approach consumes less energy in nodes as compared to the LBC-DDU approach when M = 4 for PER $\geq$ 45%.

6.6.4 Covered Distance

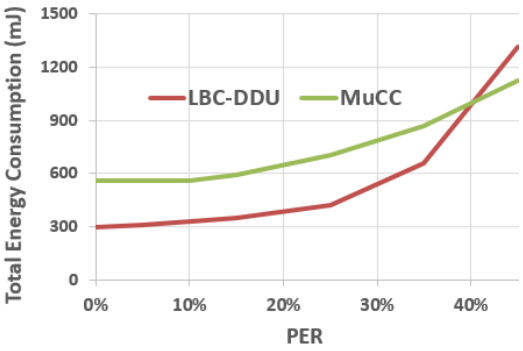
I evaluated the *ACH* algorithm against the *LBC* algorithm in relation to the total distance that the MuCar has to take in order to collect the data from all the cluster heads that have



(a) Energy Consumption VS PER (M = 1)



(b) Energy Consumption VS PER (M = 2)



(c) Energy Consumption VS PER (M = 4)

Figure 6.12: Consumption energy unit per session in different packet error rates with a various number of cluster heads. The total number of nodes is 20.

a various number of nodes ranges from 30 to 130. The nodes are randomly distributed to a network with a size that varies from 75m X 75m up to 150m X 150m.

Figure 6.13 depicts the distance amount in the meters that MuCar has traveled in order to collect the data per session. The mobile sink starts moving from the BS location, which is located outside of the network region. MuCar traverses the polling points based on the locations of the cluster heads using the TSP heuristic algorithm. These cluster heads are assigned based on either the LBC or the ACH algorithms. With the small size of the network, the ACH's distance is shorter than LBC by 26%. When the system scales in size, the difference of the magnitude between the two algorithms becomes noteworthy because, for a number of nodes = 95 and 130, the ACH's distance is shorter than LBC by 32.8% and 45% respectively.

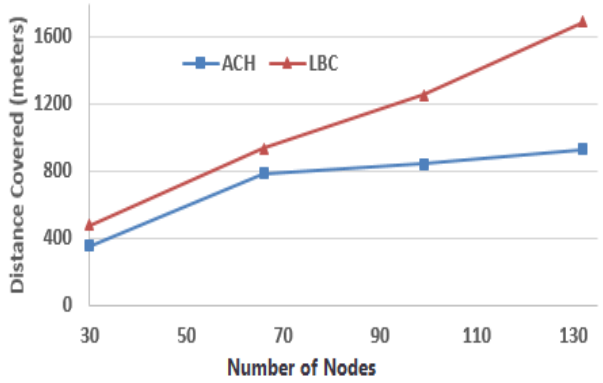


Figure 6.13: Total distance $MuCar$ travels per number of nodes on the network.

Chapter 7: Future Work

This chapter describes my future research. I plan to implement MuCode, MuTrans, and MuCC algorithms proposed in chapter 4,5, and 6 on real LLN nodes. This implementation will ultimately provide the most important evaluation of the methods I have discussed in this dissertation.

Another portion of my future research is the consideration of a non-uniformly distributed source node, along with more divergent network topologies in the scope of static Convergecast sinks. I also want to consider specific topologies – such as high-density networks – where the number of available channels is not enough to fully use MuCode protocol. An algorithm for reusing the channels is suggested in order to reduce the interference in neighborhood nodes that reside on the same level.

Further, my future research, in relation to uncontrollable predictable mobility, will consider the buffer overflow, where head nodes may not be able to hold all of the data from their member nodes during the inter-session time. Analyzing the tradeoff between archiving and streaming data to the mobile sink is needed.

In the area of controllable predictable mobility, more work can be done by finding the practical ϵ value to allow cluster heads and member nodes to exchange their status for better uploading locations. Also, comparing the network lifetime before and after the adaptive cluster head is applied to an ACH algorithm should also be considered.

Finally, uncontrollable unpredictable mobility systems have not been explored in my dissertation. Researchers have investigated this scope of work but have not considered both multi-channel and network coding. Therefore, future work can implement a MuCode protocol and a hierarchical architecture that is similar to MuCC clustering-based architecture in order to improve communication performance in LLN collection systems.

Chapter 8: Conclusion

My dissertation proposes multi-channel network coding techniques in order to improve networking performance in different data collection schemes for Low-power and Lossy networks. Network coding can significantly improve the reliability and throughput of lossy wireless sensor networks. Additionally, multi-channel protocols can significantly shorten the end-to-end delay and improve throughput.

I first introduce *MuCode*, a tree-based multi-channel WSN protocol that uses network coding. *MuCode* supports synchronized channel switching in order to take advantage of wireless eavesdropping for multi-path network coded packet delivery. I have conducted numerous simulated experiments and performance evaluation against *Sensecode* and other schemes. My results indicated that *MuCode* delivers superior performance for combined measures of throughput and latency in a tree based network and has offered clear advantages to single-channel Network Coded systems in chained networks. I formulated the optimization problem for the minimum of the maximum data rate for multi-sinks Convergecast non-split network flows. Later, I introduced a heuristic optimization and evaluated it against the optimal shortest path, the optimal minimal maximal data rate, the breadth-first search, and Random Shortest Path techniques. The results showed that my heuristic solution outperforms other methods and is closest to the optimal solution in packet latency metric.

I then presented *MuTrans*, a network-coded multi-channel protocol for uncontrollable but predictable mobile data transport. *MuTrans* balances the non-reachable node assignments to the local head nodes. It uses synchronized dynamic round robin scheduling for uploading data to the mobile data collector. I have investigated the optimization problem for minimizing the uploading latency at the polling sector. I present two heuristic algorithms Data Load Balance DLB and Dynamic Round-Robin Scheduling $DRRS$ and evaluate them

against different heuristic techniques. The results indicated that *MuTrans* outperforms the other methods in packet delivery rates, throughput, latency, and energy consumption.

Finally, I described the two-layer framework for controllable predictable mobile data gathering in WSNs: sensor layer and mobile collector layer with multi-radio multi-channel capabilities. Clusters are formed and balanced based on the remaining battery. Each cluster has multiple paths based on each cluster head to reduce the latency and increase energy savings. Network coding is implemented to maintain reliable paths, where cluster heads decode and forward the data simultaneously to the MuCar. MuCar collects data concurrently from different channels that have been assigned to each cluster head. The MuCar trajectory decision is optimized by finding the most utilized polling points of the cluster heads that collect the largest amount of cluster heads per polling point and apply the traveling salesman algorithm to visit these points with minimum trajectory cost. The simulation results show that, with moderate packet error rates, my scheme outperforms the Load Balanced Clustering with Dual Data Uploading scheme 50%, which is shorter in latency and up to 400% in energy saving. An adaptive cluster head algorithm has been designed to overcome the traveling latency by reducing the traveling distance between the polling points. The simulation shows that my ACH scheme outperforms the LBC-DDU in up to %45 shorter in traveling distance.

Appendix A: Appendix

A.1 Uploading Optimization Problem

In this appendix, I formulate the optimization problem for the uploading latency in uncontrollable predictable mobile data collection. Let A_b be Γ_b X μ_b binary matrix where the rows represent the head nodes, and the column represents the member nodes at the polling sector b. The element $a_{i,j}$ is equal to 1 when member node j neighbors head node H_i otherwise, $a_{i,j} = 0$. With buffer capacity β, the optimization problem for balancing the head nodes load for inter-session duration σ can be formulated as 0-1 integer linear programming (ZOILP) as the following:

$$\min \max_{\forall i \in \{1,\ldots,\Gamma_b\}} \Psi_{H_i}(\sigma) \tag{A.1}$$

Subject to:

$$\Psi_{H_j}(\sigma) = \gamma_{H_i}.\sigma + \partial + \sigma.\sum_{j=1}^{\mu_b} \gamma_j \; x_{i,j} \leq \beta \tag{A.2}$$

$$x_{i,j} = \begin{cases} 1 & \text{if } m_j \text{ is assigned to } H_i \text{ and } a_{i,j} = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{A.3}$$

$$\sum_{i=1}^{\Gamma_b} x_{i,j} = 1 \qquad \forall j \in \{1, \ldots, \mu_b\} \tag{A.4}$$

$$\Gamma_j \geq \sum_{j=1}^{\mu_b} x_{i,j} \geq 0 \quad \forall i \in \{1, \ldots, \Gamma_b\} \tag{A.5}$$

$$\beta \geq \gamma_e \, . \, \sigma \;\geq\; 0 \qquad \forall e \in \mu_b \cup \Gamma_b \tag{A.6}$$

The constraint (A.2) defines the data load at the head node H_i which is the sum of its current data plus ∂ which is the additional overheard encoded data with the sum of all his assigned member nodes' data during σ time that is upper bounded by the buffer size β . The condition (A.3) restricts that a member node can assign to a head node if both are single-hop neighbors to each other. In constraint (A.4), it guarantees that a member node has to be assigned to exactly one nearby head node, while constraint (A.5) makes sure that a head node can be assigned by none up to μ_b member nodes. The last condition (A.6) forces the data to be upper bounded by the buffer size β. By demonstrating that one way of formulating an optimal uploading problem is through a zero-one integer linear programming (ZOILP) which shows the desirability of developing heuristic solutions.

Bibliography

Bibliography

[1] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: an instant primer," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 63–68, 2006.

[2] P. Ostovari, J. Wu, and A. Khreishah, "Network coding techniques for wireless and sensor networks," in *The Art of Wireless Sensor Networks*. Springer, 2014, pp. 129–162.

[3] Y. Wu, J. Stankovic, T. He, and S. Lin, "Realistic and efficient multi-channel communications in wireless sensor networks," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008.

[4] S. Chieochan and E. Hossain, "Channel assignment for throughput optimization in multichannel multiradio wireless mesh networks using network coding," *Mobile Computing, IEEE Transactions on*, vol. 12, no. 1, pp. 118–135, Jan 2013.

[5] C. J. Watras, M. Morrow, K. Morrison, S. Scannell, S. Yaziciaglu, J. S. Read, Y.-H. Hu, P. C. Hanson, and T. Kratz, "Evaluation of wireless sensor networks (wsns) for remote wetland monitoring: design and initial results," *Environmental monitoring and assessment*, vol. 186, no. 2, pp. 919–934, 2014.

[6] J. Tooker and M. C. Vuran, "Mobile data harvesting in wireless underground sensor networks," in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on*, June 2012, pp. 560–568.

[7] I. Stojmenovic, "Machine-to-machine communications with in-network data aggregation, processing and actuation for large scale cyber-physical systems," 2014.

[8] V. Annamalai, S. K. Gupta, and L. Schwiebert, "On tree-based convergecasting in wireless sensor networks," in *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, vol. 3. IEEE, 2003, pp. 1942–1947.

[9] T. Watteyne, A. Molinaro, M. G. Richichi, and M. Dohler, "From manet to ietf roll standardization: A paradigm shift in wsn routing protocols," *Communications Surveys & Tutorials, IEEE*, vol. 13, no. 4, pp. 688–707, 2011.

[10] M. Abdulaziz and R. Simon, "Multi-channel network coding in tree-based wireless sensor networks," in *Computing, Networking and Communications (ICNC), 2015 International Conference on*. IEEE, 2015, pp. 924–930.

[11] P. T. A. Quang and D.-S. Kim, "Enhancing real-time delivery of gradient routing for industrial wireless sensor networks," *Industrial Informatics, IEEE Transactions on*, vol. 8, no. 1, pp. 61–68, 2012.

[12] M. Di Francesco, S. K. Das, and G. Anastasi, "Data collection in wireless sensor networks with mobile elements: A survey," *ACM Trans. Sen. Netw.*, vol. 8, no. 1, pp. 7:1–7:31, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/1993042.1993049

[13] F. Restuccia, G. Anastasi, M. Conti, and S. Das, "Analysis and optimization of a protocol for mobile element discovery in sensor networks," *Mobile Computing, IEEE Transactions on*, vol. 13, no. 9, pp. 1942–1954, Sept 2014.

[14] Y. Gu, F. Ren, Y. Ji, and J. Li, "The evolution of sink mobility management in wireless sensor networks: A survey," 2015.

[15] R. C. Shah, S. Roy, S. Jain, and W. Brunette, "Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks," *Ad Hoc Networks*, vol. 1, no. 2, pp. 215–233, 2003.

[16] A. Chakrabarti, A. Sabharwal, and B. Aazhang, "Using predictable observer mobility for power efficient design of sensor networks," in *Information Processing in Sensor Networks*. Springer, 2003, pp. 129–145.

[17] A. Hasson, "Daknet: Rethinking connectivity in developing nations," *work*, vol. 5, 2004.

[18] H. Smeets, C.-Y. Shih, M. Zuniga, T. Hagemeier, and P. J. Marrón, "Trainsense: a novel infrastructure to support mobility in wireless sensor networks," in *Wireless Sensor Networks*. Springer, 2013, pp. 18–33.

[19] M. Dohler, D. Barthel, T. Watteyne, and T. Winter, "Routing requirements for urban low-power and lossy networks," 2009.

[20] M. Sha, G. Hackmann, and C. Lu, "Real-world empirical studies on multi-channel reliability and spectrum usage for home-area sensor networks," *Network and Service Management, IEEE Transactions on*, vol. 10, no. 1, pp. 56–69, 2013.

[21] O. D. Incel, L. van Hoesel, P. Jansen, and P. Havinga, "Mc-lmac: A multi-channel mac protocol for wireless sensor networks," *Ad Hoc Networks*, vol. 9, no. 1, pp. 73–94, 2011.

[22] G. H. E. Fard, M. Yaghmaee, R. Monsefi *et al.*, "An adaptive cross-layer multichannel qos-mac protocol for cluster based wireless multimedia sensor networks," in *Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on*. IEEE, 2009, pp. 1–6.

[23] R. Simon, L. Huang, E. Farrugia, and S. Setia, "Using multiple communication channels for efficient data dissemination in wireless sensor networks," in *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, Nov 2005, pp. 10 pp.–439.

[24] G. Zhou, T. He, J. 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*, vol. 2, March 2005, pp. 891–901 vol. 2.

[25] P. Kyasanur and N. H. Vaidya, "Capacity of multi-channel wireless networks: Impact of number of channels and interfaces," in *Proceedings of the 11th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '05. New York, NY, USA: ACM, 2005, pp. 43–57.

[26] K. Ramachandran, E. Belding, K. Almeroth, and M. Buddhikot, "Interference-aware channel assignment in multi-radio wireless mesh networks," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1–12.

[27] 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*, vol. 36, no. 4. ACM, 2006, pp. 243–254.

[28] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," 2003.

[29] G. Angelopoulos, A. Paidimarri, A. Chandrakasan, and M. Medard, "Experimental study of the interplay of channel and network coding in low power sensor applications," in *Communications (ICC), 2013 IEEE International Conference on*, June 2013, pp. 5126–5130.

[30] R. R. Rout and S. K. Ghosh, "Enhancement of lifetime using duty cycle and network coding in wireless sensor networks," *Wireless Communications, IEEE Transactions on*, vol. 12, no. 2, pp. 656–667, 2013.

[31] X. Wang and J. Li, "Network coding aware cooperative mac protocol for wireless ad hoc networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 1, pp. 167–179, Jan 2014.

[32] A. E. Kamal, A. Ramamoorthy, L. Long, and S. Li, "Overlay protection against link failures using network coding," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 4, pp. 1071–1084, 2011.

[33] R. Chandanala, W. Zhang, R. Stoleru, and M. Won, "On combining network coding with duty-cycling in flood-based wireless sensor networks," *Ad Hoc Netw.*, vol. 11, no. 1, pp. 490–507, Jan. 2013.

[34] M. Jafari, L. Keller, C. Fragouli, and K. Argyraki, "Compressed network coding vectors," in *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, June 2009, pp. 109–113.

[35] C. de Alwis, H. Arachchi, A. Fernando, and A. Kondoz, "Towards minimising the coefficient vector overhead in random linear network coding," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, May 2013, pp. 5127–5131.

[36] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '09. New York, NY, USA: ACM, 2009, pp. 1–14.

[37] L. Keller, E. Atsan, K. Argyraki, and C. Fragouli, "Sensecode: Network coding for reliable sensor networks," *ACM Trans. Sen. Netw.*, vol. 9, no. 2, pp. 25:1–25:20, Apr. 2013.

[38] D. Gong, Y. Yang, and Z. Pan, "Energy-efficient clustering in lossy wireless sensor networks," *Journal of Parallel and Distributed Computing*, vol. 73, no. 9, pp. 1323–1336, 2013.

[39] Z. Zhang, M. Ma, and Y. Yang, "Energy-efficient multihop polling in clusters of two-layered heterogeneous sensor networks," *Computers, IEEE Transactions on*, vol. 57, no. 2, pp. 231–245, 2008.

[40] M. Ma, Y. Yang, and M. Zhao, "Tour planning for mobile data-gathering mechanisms in wireless sensor networks," *Vehicular Technology, IEEE Transactions on*, vol. 62, no. 4, pp. 1472–1483, 2013.

[41] H. Lee, M. Wicke, B. Kusy, O. Gnawali, and L. Guibas, "Data stashing: energy-efficient information delivery to mobile sinks through trajectory prediction," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. ACM, 2010, pp. 291–302.

[42] J.-L. Lu and F. Valois, "On the data dissemination in wsns," in *Wireless and Mobile Computing, Networking and Communications, 2007. WiMOB 2007. Third IEEE International Conference on*. IEEE, 2007, pp. 58–58.

[43] X. Xu, W. Liang, and T. Wark, "Data quality maximization in sensor networks with a mobile sink," in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*. IEEE, 2011, pp. 1–8.

[44] K. Kweon, H. Ghim, J. Hong, and H. Yoon, "Grid-based energy-efficient routing from multiple sources to multiple mobile sinks in wireless sensor networks," in *Wireless Pervasive Computing, 2009. ISWPC 2009. 4th International Symposium on*. IEEE, 2009, pp. 1–5.

[45] M. Shon, C. Kong, and H. Choo, "Hexagonal path data dissemination for energy efficiency in wireless sensor networks," in *Information Networking, 2009. ICOIN 2009. International Conference on*. IEEE, 2009, pp. 1–5.

[46] C. Tunca, M. Donmez, S. Isik, and C. Ersoy, "Ring routing: An energy-efficient routing protocol for wireless sensor networks with a mobile sink," in *Signal Processing and Communications Applications Conference (SIU), 2012 20th*. IEEE, 2012, pp. 1–4.

[47] J. Weiwei, C. Canfeng, X. Dongliang, and M. Jian, "Efficient data collection in wireless sensor networks by applying network coding," in *Broadband Network & Multimedia Technology, 2009. IC-BNMT'09. 2nd IEEE International Conference on*. IEEE, 2009, pp. 90–94.

[48] W. C. Cheng, C.-F. Chou, L. Golubchik, S. Khuller, and Y.-C. Wan, "A coordinated data collection approach: design, evaluation, and comparison," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 10, 2004.

[49] 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*, vol. 14, no. 4, pp. 770–785, 2014.

[50] G. Wang, G. Cao, T. L. Porta, and W. Zhang, "Sensor relocation in mobile sensor networks," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 4. IEEE, 2005, pp. 2302–2312.

[51] Y.-C. Tseng, Y.-C. Wang, K.-Y. Cheng, and Y.-Y. Hsieh, "imouse: an integrated mobile surveillance and wireless sensor system," *Computer*, no. 6, pp. 60–66, 2007.

[52] R. Sugihara and R. K. Gupta, "Optimal speed control of mobile node for data collection in sensor networks," *Mobile Computing, IEEE Transactions on*, vol. 9, no. 1, pp. 127–139, 2010.

[53] L. Zhou, X. Wang, W. Tu, G.-M. Muntean, and B. Geller, "Distributed scheduling scheme for video streaming over multi-channel multi-radio multi-hop wireless networks," *Selected Areas in Communications, IEEE Journal on*, vol. 28, no. 3, pp. 409–419, 2010.

[54] E. Hyytiä, P. Lassila, and J. Virtamo, "Spatial node distribution of the random waypoint mobility model with applications," *Mobile Computing, IEEE Transactions on*, vol. 5, no. 6, pp. 680–694, 2006.

[55] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile computing*. Springer, 1996, pp. 153–181.

[56] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless communications and mobile computing*, vol. 2, no. 5, pp. 483–502, 2002.

[57] J. Härri, F. Filali, and C. Bonnet, "Mobility models for vehicular ad hoc networks: a survey and taxonomy," *Communications Surveys & Tutorials, IEEE*, vol. 11, no. 4, pp. 19–41, 2009.

[58] F. Restuccia and S. K. Das, "Lifetime optimization with qos of sensor networks with uncontrollable mobile sinks," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*. IEEE, 2015, pp. 1–9.

[59] A. Kansal, A. A. Somasundara, D. D. Jea, M. B. Srivastava, and D. Estrin, "Intelligent fluid infrastructure for embedded networks," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*. ACM, 2004, pp. 111–124.

[60] D. Jea, A. Somasundara, and M. Srivastava, "Multiple controlled mobile elements (data mules) for data collection in sensor networks," in *Distributed computing in sensor systems*. Springer, 2005, pp. 244–257.

[61] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, "Cartel: a distributed mobile sensor computing system," in *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM, 2006, pp. 125–138.

[62] A. A. Somasundara, A. Kansal, D. D. Jea, D. Estrin, and M. B. Srivastava, "Controllably mobile infrastructure for low energy embedded networks," *Mobile Computing, IEEE Transactions on*, vol. 5, no. 8, pp. 958–973, 2006.

[63] S. Gao, H. Zhang, and S. K. Das, "Efficient data collection in wireless sensor networks with path-constrained mobile sinks," *Mobile Computing, IEEE Transactions on*, vol. 10, no. 4, pp. 592–608, 2011.

[64] W. Liang, P. Schweitzer, and Z. Xu, "Approximation algorithms for capacitated minimum forest problems in wireless sensor networks with a mobile sink," *Computers, IEEE Transactions on*, vol. 62, no. 10, pp. 1932–1944, 2013.

[65] S.-Y. Li, R. W. Yeung, and N. Cai, "Linear network coding," *Information Theory, IEEE Transactions on*, vol. 49, no. 2, pp. 371–381, 2003.

[66] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *Information Theory, IEEE Transactions on*, vol. 52, no. 10, pp. 4413–4430, 2006.

[67] F. Awad, O. Banimelhem, and N. Al-Rousan, "The potential of using network coding with geographical forwarding routing for wireless multimedia sensor networks," in *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*. IEEE, 2011, pp. 9–14.

[68] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," *Wireless communications, IEEE*, vol. 11, no. 6, pp. 6–28, 2004.

[69] T. Watteyne, A. Molinaro, M. G. Richichi, and M. Dohler, "From manet to ietf roll standardization: A paradigm shift in wsn routing protocols," *Communications Surveys & Tutorials, IEEE*, vol. 13, no. 4, pp. 688–707, 2011.

[70] J. Pope and R. Simon, "Crest: An epoch-oriented routing control plane for low-power and lossy networks," in *Local Computer Networks Workshops (LCN Workshops), 2013 IEEE 38th Conference on*. IEEE, 2013, pp. 128–136.

[71] M. Aigner, G. Ziegler, and P. Erdos, "Proofs from the book, vol. 274," 2010.

[72] K. Bharath-Kumar and J. Jaffe, "A new approach to performance-oriented flow control," *Communications, IEEE Transactions on*, vol. 29, no. 4, pp. 427–435, 1981.

[73] A. Dunkels, F. Österlind, and Z. He, "An adaptive communication architecture for wireless sensor networks," in *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM, 2007, pp. 335–349.

[74] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on wireless communications*, vol. 1, no. 4, pp. 660–670, 2002.

[75] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," 2003.

[76] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: A case study in local optimization," *Local search in combinatorial optimization*, vol. 1, pp. 215–310, 1997.

[77] C. M. Angelopoulos, S. Nikoletseas, D. Patroump, and C. Rapropoulos, "A new random walk for efficient data collection in sensor networks," in *Proceedings of the 9th ACM international symposium on Mobility management and wireless access.* ACM, 2011, pp. 53–60.

[78] Z. Yao and K. Gupta, "Backbone-based connectivity control for mobile networks," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.* IEEE, 2009, pp. 1133–1139.

[79] S. Duquennoy, F. Österlind, and A. Dunkels, "Lossy links, low power, high throughput," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems.* ACM, 2011, pp. 12–25.

[80] E. Ancillotti, R. Bruno, and M. Conti, "Rpl routing protocol in advanced metering infrastructures: an analysis of the unreliability problems," in *Sustainable Internet and ICT for Sustainability (SustainIT), 2012.* IEEE, 2012, pp. 1–10.

[81] ——, "The role of the rpl routing protocol for smart grid communications," *Communications Magazine, IEEE*, vol. 51, no. 1, pp. 75–83, 2013.

[82] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: an instant primer," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 63–68, 2006.

[83] M. Abdulaziz and R. Simon, "Mobile data collection using multi-channel network coding in wireless sensor networks," in *Local Computer Networks (LCN), 2015 IEEE 40th Conference on.* IEEE, 2015, pp. 205–208.

[84] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *Internet of Things Journal, IEEE*, vol. 1, no. 1, pp. 22–32, 2014.

[85] Q. Wang, K. Ren, W. Lou, and Y. Zhang, "Dependable and secure sensor data storage with dynamic integrity assurance," in *INFOCOM 2009, IEEE*, April 2009, pp. 954–962.

[86] K. Viswanatha, S. Ramaswamy, A. Saxena, and K. Rose, "Error-resilient and complexity-constrained distributed coding for large scale sensor networks," in *Proceedings of the 11th international conference on Information Processing in Sensor Networks.* ACM, 2012, pp. 293–304.

[87] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Médard, "Resilient network coding in the presence of byzantine adversaries," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE.* IEEE, 2007, pp. 616–624.

[88] L. Nutman and M. Langberg, "Adversarial models and resilient schemes for network coding," in *Information Theory, 2008. ISIT 2008. IEEE International Symposium on.* IEEE, 2008, pp. 171–175.

[89] M. Wang and B. Li, "How practical is network coding?" in *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on.* IEEE, 2006, pp. 274–278.

[90] C. Fragouli and E. Soljanin, *Network coding fundamentals.* Now Publishers Inc, 2007.

Curriculum Vitae

Mansour Abdulaziz earned his B.S. degree from Kuwait University, Kuwait; and his M.S. degree from Missouri University of Science and Technology, Rolla, Missouri, both in Computer Science. His research interests include wireless sensor networks, low-power embedded systems, and communication protocols. He has been awarded a full scholarship from Kuwait University to pursue his Ph.D. in Computer Science.