2015

Raptorq-Based Multihop File Broadcast Protocol

Roya Lotfi
University of Mississippi

RAPTORQ-BASED MULTIHOP FILE BROADCAST PROTOCOL

A Thesis
presented in partial fulfillment of requirements
for the degree of Master
in the Engineering Science with Emphasis in
Telecommunications
The University of Mississippi

by

Roya Lotfi

Dec 2015

ABSTRACT

The objective of this thesis is to describe and implement a RaptorQ broadcast protocol application layer designed for use in a wireless multihop network. The RaptorQ broadcast protocol is a novel application layer broadcast protocol based on RaptorQ forward error correction. This protocol can deliver a file reliably to a large number of nodes in a wireless multihop network even if the links have high loss rates.

We use mixed integer programming with power balance constraints to construct broadcast trees that are suitable for implementing the RaptorQ-based broadcast protocol. The resulting broadcast tree facilitates deployment of mechanisms for verifying successful delivery.

We use the Qualcomm proprietary RaptorQ software development kit library as well as a Ruby interface to implement the protocol. During execution, each node operates in one of main modes: *source*, *transmitter*, or *leaf*. Each mode has five different phases: STARTUP, FINISHING (Poll), FINISHING (Wait), FINISHING (Extra), and COMPLETED. Three threads are utilized to implement the RaptorQ-based broadcast protocol features. Thread 1 receives messages and passes them to the receive buffer. Thread 2 evaluates the received message, which can be NORM, POLL, MORE, and DONE, and passes the response message to the send buffer. Thread 3 multicasts the content of the send buffer.

Results obtained by testing the implementation of the RaptorQ-based broadcast protocol demonstrate that efficient and reliable distribution of files over multihop wireless networks with a high link loss rates is feasible.

DEDICATION

To my family.

ACKNOWLEDGEMENTS

I owe my deepest gratitude to my my advisor Prof. John Daigle, who has made available his support in many ways throughout my graduate study at the University of Mississippi. My grateful appreciations also go to Prof. Feng Wang for the many helpful comments and suggestions on my research. I would like to thank Prof. Ramanarayanan Viswanathan for his effort and support.

I would like to thank my professors, teachers, colleagues, and family members, whom supported me in all aspects of life.

TABLE OF CONTENTS

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGEMENTS iv

LIST OF FIGURES vi

LIST OF TABLES vii

INTRODUCTION 1

MINIMUM ENERGY BROADCAST TREE FOR MULTIHOP WIRELESS BROADCAST 5

BALANCED-POWER MINIMUM ENERGY BROADCAST TREE 18

RaptorQ-based BROADCAST PROTOCOL 36

RAPTORQ-BASED BROADCAST PROTOCOL IMPLEMENTATION 42

CONCLUSION AND FUTURE WORK 60

BIBLIOGRAPHY 62

VITA 66

LIST OF FIGURES

3.1 Minimum Energy Broadcast Tree in Topology 1 27
3.2 Balanced-power Broadcast Tree, $\beta = 0.5$ in Topology 1 28
3.3 Minimum Energy Broadcast Tree using BIP in Topology 1 30
3.4 Minimum Energy Broadcast Tree in Topology 2 31
3.5 Balanced-power Broadcast Tree $\beta = 0.5$ in Topology 2 32
3.6 Minimum Energy Broadcast Tree using BIP in Topology 2 32
3.7 Minimum Energy Broadcast Tree in Topology 3 33
3.8 Balanced-power Broadcast Tree with Different β 34
3.9 Power Balance Broadcast Tree for Different Amounts of β in Topology 3 . . . 35
4.1 Finite State Machines for File Delivery. 41
5.1 Processing Flow for RaptorQ Encoder and RaptorQ Decoder 43
5.2 Raspberry Pi 2 Model B+ 53
5.3 Tree Topology for Testbed 54

LIST OF TABLES

1.1 Characteristics of RaptorQ 3

2.1 Comparison of the Spanning Tree Algorithms 13

2.2 Comparison of the Local Search Algorithms 17

3.1 Essential Instance Methods in Cplex Class 21

3.2 Essential Instance Methods in Indicator Constraints 22

3.3 The Numerical Results based on Three Different Methods in Topology 1 . . . 30

3.4 The Numerical Results based on Three Different Methods in Topology 2 . . . 33

3.5 The Numerical Results based on Different Values of β in Topology 3 34

5.1 IP Configuration in Testbed 54

5.2 The Average Required Number of Sent Symbols as a Function for Loss Rate 55

5.3 The Minimum Round for the Source Node 57

5.4 Theoretical Result for Generation Time and Completion Time 58

5.5 Measured Result for Generation Time and Completion Time in Testbed . . . 59

CHAPTER 1

INTRODUCTION

The objective of this thesis is to implement an efficient RaptorQ-based protocol for reliable file distribution over a wireless multihop network. The RaptorQ-based multihop file broadcast protocol is a novel broadcast protocol designed at the University of Mississippi based on the RaptorQ technology the objective to bring about reliable content delivery in a wireless multihop network mesh network. The RaptorQ codes are application-layer forward error correction (AL-FEC) codes. RaptorQ codes are a class of fountain codes. RaptorQ forward error correction can achieve reliable delivery of content by recovering the source from an adequate number of received symbols.

The Qualcomm RaptorQ software development kit library (RaptorQ SDK) is exploited to encode and decode symbols. The protocol is scripted in Ruby and uses Ruby interface to C functions developed over the RaptorQ library. We construct a Raspberry Pi based testbed to implement our protocol. Time delivery and reliability are computed through a series of tests performed in Raspberry Pi platforms.

Due to various reasons such as high interference and low SINR, some of the transmitted packets might be lost. The high loss rate can significantly degrade the performance of network protocols. In this study, we utilize RaptorQ to design a new protocol. By using RaptorQ, the receiver can reconstruct the file if it receives a sufficient number of encoded symbols regardless of which particular encoded symbols are received. And since RaptorQ is a fountain code, the sender can send an arbitrary number of new encoded symbols to compensate for any dropped encoded symbols. Therefore, the implementation of this protocol facilitates delivery of the file to a large number of nodes in a wireless ad hoc network even if the network has high packet loss.

The remainder of this chapter gives an introduction to RaptorQ AL-FEC. The outline of the thesis is also provided.

1.1 RaptorQ AL-FEC

RaptorQ AL-FEC is an application layer forward error correction which belongs to the Raptor code series. AL-FEC technologies solve the network packet loss issue by sending repair symbols in addition to original source symbols.

The Raptor code is also a class of fountain code. Different fountain codes differ in terms of their overhead for a given error probability and the computational efficiency of the encoding and decoding processes. In a general fountain code strategy, an original file is divided into source blocks. Each source block is partitioned into equal sized portions of data, called source symbols, that basically have the size of one packet. Let k denote the number of source symbols in the source block. Then, encoded symbols are generated from the source symbols using an encoder. The encoded symbols are generated as a linear combination of source symbols. Let n denote the number of encoded symbols. Then, receivers use decoders to decode the source symbols of the source block from any subsets of k or more linearly independent encoded symbols. The encoded symbols contain identifiers that inform receivers of the specific linear combination of source symbols that define the encoded symbol. This is accomplished by including an encoding symbol ID (ESI).

The LT code introduced by Luby [18] is the first practical fountain code. Each encoded symbol is computed as the exclusive-or (XOR) of d source symbols. The value of d is selected from the degree distribution, Ω. The expected number of XORs required to produce encoded symbols is called an encoding cost. The expected number of XORs required to decode the source symbols from the received encoded symbols is called the decoding cost. The encoding cost for LT is $O(\log(k))$ and the decoding cost is $O(k\log(k))$.

The Raptor code is an extension of the LT code introduced by Shokrollahi [19] which has linear encoding and decoding time. In fact, the Raptor code is a combination of the

LT code and the LDPC code. The Raptor code obtains linear time encoding and decoding performance by taking advantage of pre-coding technique. An appropriate binary block code $\mathcal{C}$ is used to encode source symbols to generate $n-k$ redundant symbols. The concatenation of k source symbols and $n-k$ redundant symbols is called the intermediate symbols. These intermediate symbols are then LT encoded. Raptor codes have encoding cost $O(\log(\frac{1}{\epsilon}))$ and decoding cost $O(k\log(\frac{1}{\epsilon}))$, in which ϵ is a constant overhead, outperform LT codes.

RaptorQ has better performance than the standardized Raptor code. The RaptorQ code is a systematic code. It means that all source symbols are among the encoded symbols. Thus, encoded symbols can be a combination of the original source symbols and repair symbols generated by the encoder. In addition, the RaptorQ code operates over the finite field GF(256) rather than the Galois field GF(2). The RaptorQ is predictable in terms of its failure probability as a function of overhead. The RaptorQ code also has a smaller decoding overhead compared to the standardized Raptor codes. The standardized Raptor code requires an overhead of 24 to achieve a failure probability of 10^{-6}, but RaptorQ ensures a failure probability of less than 10^{-6} with an overhead of 2. Table 1.1 [16] briefly explains the supported size and the number of blocks and encoded symbols in RaptorQ.

Item	Size
SBN	8 bits
Source Blocks	$2^8 = 256$
ESI	24 bits
Num of Encoded Symbols per Block	$2^{24} = 16777216$
Num of Source Symbols per Block	56,403
Max Symbol Size	$2^{16} = 65,536$ Bytes
File Length	$65,536 \times 56,403 \times 256 = 946270874880 \approx 1$ TByte

Table 1.1: Characteristics of RaptorQ

1.2 RaptorQ-based Broadcast Protocol

The RaptorQ-based broadcast protocol facilitates reliable content distribution in a wireless multihop network with high packet loss rate. RaptorQ encodes the content of the

transmitter node. The content is then delivered over a wireless multihop network to receiver nodes which decode the content. The RaptorQ technology used in this protocol recovers data lost, and completely reconstructs the file without using a retransmission mechanism. The RaptorQ broadcast protocol guarantees the recovery of content without needing to resend any symbols. Thus, the transport layer protocol is not required to provide reliability. Hence, the UDP transport protocol is used in our protocol.

1.3 Outline of the Thesis

The thesis is organized as follows. The next chapter presents the literature on constructing a minimum energy broadcast tree. Chapter 3 discuses the mixed integer program with the power balance objective to construct a tree which is appropriate for the RaptorQ-based broadcast protocol. Chapter 4 describes the RaptorQ-based broadcast protocol and shows finite state machines used in this protocol. Chapter 5 explains the implementation and results of the RaptorQ-based broadcast protocol and, Chapter 6 finally elaborates conclusions and future work.

CHAPTER 2

MINIMUM ENERGY BROADCAST TREE FOR MULTIHOP WIRELESS BROADCAST

In this chapter, we address the minimum-energy broadcast tree problem in a wireless mesh network. The literature has been reviewed the alternatives to find the minimum energy broadcast tree in a wireless mesh network will be discussed. In section 2.1, we introduce different methods to solve a minimum energy broadcast tree problem. In section 2.2, we discuss network models. In section 2.3, the integer programming approach for a minimum-energy broadcast tree (MEB) problem is explained. In section 2.4, different spanning tree heuristic algorithms and the comparison of their performances are elaborated. In section 2.5, local search heuristic algorithms are discussed. Finally, in section 2.6, the evolutionary search heuristic algorithm will be described.

2.1 Introduction

The wireless mesh network is a form of infrastructure-less or ad hoc network. In wireless ad hoc networks, nodes are distributed over a geographical area and communicate with each other over a multi-hop over a shared radio channel. Their connectivities are determined by the transmit power. Energy consumption is an important factor in wireless ad hoc communications because wireless nodes use a battery. The main purpose of constructing a minimum energy broadcast tree is to find a set of transmitting nodes as well as their corresponding transmission powers in order to cover all nodes in the network and minimize the total transmission energy. In the literature, the minimum energy broadcast tree problem is known as a MEB problem. Several different methods can be utilized to solve the MEB problem such as mixed integer linear programming (MILP), spanning tree heuristic algorithm, local search heuristic algorithm and, meta-heuristic algorithm.

The inherently broadcast nature of a wireless network, referred as a wireless multicast advantage (WMA) [21], permit all nodes that their received Signal-to-Noise-Ratios ratio exceeds a given threshold can receive a transmission even if they do not have direct link with transmitter node.

2.2 Network Model

The system under consideration is a general wireless mesh network that is connected to a single gateway to the internet. Let $\mathcal{N} = \{\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_{N-1}\}$ denote the collection of nodes in the network. $\mathcal{N}_0$ is a source node which is a gateway and includes a file as well. The fixed N-nodes are randomly located inside the network area. The loss matrix ℓ is general, but will be calculated in this thesis based on d^α which d is the distance between nodes and α is the loss exponent between 2 and 4. The transmission power between node i and j is equal to:

$$P_{ij} \geq \gamma_{\min} \ell_{ij} \eta \quad \forall j \in \mathcal{R}_i \tag{2.1}$$

where P_{ij} denotes the transmission power of node i, $\mathcal{R}_i$ represents target nodes of the transmitter $i, \eta_{\min}$ denotes the minimum SINR required to meet BER requirements at the rate of r_i , ℓ_{ij} is the loss between nodes i and j, $\eta = kBT$ is the noise at the receiver j, where k is Boltzmann's constant, T is the absolute temperature, and B is the bandwidth. However, in the literature the transmission power is approximately equal to the loss. Thus

$$P_{ij} \approxeq \ell_{ij} = d_{ij}^{\alpha} \tag{2.2}$$

2.3 Integer Programming Approach

An integer program is one of the methods to find the minimum energy broadcast tree in a wireless mesh network. In [15] [3], authors have proved that constructing the minimum energy broadcast tree problem in a wireless da hoc network is NP-hard.

Das *et al.* [6] have proposed three different integer program (IP) models. First, they have

compared the MEB problem with a traveling salesman problem (TSP), which finds the minimum cost tour of visiting all cities in a given set of cites subjected to some constraints. By eliminating and modifying some constraints in the TSP problem, we can obtain the minimum spanning tree (MST) problem. Let C_{ij} be the cost of the edge (i, j) and X_{ij} be a binary variable which is 1 if the edge (i, j) is used in the final solution and 0, otherwise. The objective function for MST and MEB are as the following statements:

$$\text{MST} : \text{minimize} \sum_{i} \sum_{j} C_{ij} X_{ij}; i \neq j$$

$$\text{MEB} : \text{minimize} \sum_{i} \max_{j} (C_{ij} X_{ij}); i \neq j$$

The objective function that minimizes the summation of the transmission power is the same for all three models . However, IP formulation 'A' and 'B' have a large computation time and they are not practical. IP formulation 'C' is based on the flow model of the IP formulation 'B'. They have considered Y_i as a transmission power of each node, P_{ij} as a cost between node i and j, F_{ij} as a flow variable, X_{ij} as a binary variable which is 1 if there is a direct transmission between node i and j, and $\mathcal{C}_D$ as a cardinality of set D which is N in the broadcast case. The objective function of this model is given by:

$$\text{Minimize} \sum_{i=1}^{N} Y_i \tag{2.3}$$

Subjected to:

$$Y_i - P_{ij} X_{ij} \geq 0; \ \forall (i, j) \in V, \ i \neq j \tag{2.4}$$

$$\mathcal{C}_D X_{ij} - F_{ij} \geq 0; \ \forall (i, j) \in V, \ i \neq j \tag{2.5}$$

$$\sum_{j=1}^{N} F_{0j} = \mathcal{C}_D; \tag{2.6}$$

$$\sum_{j=1}^{N} F_{j0} = 0; \tag{2.7}$$

$$\sum_{j=1}^{N} F_{ji} - \sum_{j=1}^{N} F_{ij} = 1;\ \forall i \in D,\ i \neq j \tag{2.8}$$

$$\sum_{j=1}^{N} F_{ji} - \sum_{j=1}^{N} F_{ij} = 0;\ \forall i \notin D,\ i \neq j \tag{2.9}$$

$$X_{ij} \in \{0, 1\} \tag{2.10}$$

$$F_{ij} \geq 0;\ \forall (i, j) \in V, i \neq j \tag{2.11}$$

Constraint 2.4 guarantees that Y_i is at least P_{ij} if link i, j is used. Constraint 2.5 ensures that the flow out of a node cannot exceed the number of destination nodes. Constraint 2.6 implies that the summation of flow out of a source node is $\mathcal{C}_D$. Constraint 2.7 illustrates that the summation of flow into the source node is zero. Constraint 2.8 and 2.8 forces exactly one packet to be delivered to each destination node because each destination node should keep one packet and a non-destination node just forwards the packet.

Min *et al.* have presented an IP formulation, a relaxed IP formulation, and two iterative algorithms. Furthermore, they have compared the computation time of their IP formulation with Das A, B, C. Their IP formulation is based on flow constrains. They have used a variable u_{ij} which is a binary variable indicating whether or not the node i transmits at the power level of P_{ij}. By using this variable, they can have tighter LP relaxation and speed up the computation time. They also have a constraint for the destination node coverage as the following equation:

$$\sum_{i \in N} \sum_{k \in N, k \neq s, k \neq i, P_{ik} \geq P_{ij}} u_{ik} \geq 1\ \forall j \in D \tag{2.12}$$

where $D = N$ for the broadcast case. Moreover, they have bound a flow variable F_{ij} and a

transmission variable u_{ij} which are defined as:

$$|D| \sum_{k \in N, k \neq s, k \neq i, P_{ik} \geq P_{ij}} u_{ik} - F_{ij} \geq 0 \ \forall i, j \in N, j \neq s, j \neq i \tag{2.13}$$

Their flow constraint is the same as the flow constraint in Das. They obtain relaxed IP formulation by eliminating a number of constrains. They have proposed two iterative algorithms by using the relaxed IP formulation. In each iteration, they add cuts to the relaxed infeasible IP formulation to obtain a feasible solution. The original constraint guarantees the connectivity of resulting graph which is given by:

$$\sum_{i \in S} \sum_{i \notin S} u_{ij} \geq 1 \ \forall S \subset N \tag{2.14}$$

They also have used other three constraints. The first constraint is to make sure that a node is connected to the source. The second one is the repetition prevention constraint. The last one is a branch cutoff constrain based on the upper bound cut off, feasibility cut off, and no available cut off. In the second iterative algorithm, the source node's transmission power is shrunk from the maximum value to the minimum value. Then, the minimum broadcast tree for each value is computed. They have compared computation times of two iterative algorithms and their IP formulation for 20, 30, 40 and 50 nodes. Their two iterative algorithms have better performances than their IP formulation specifically in a large network. Also, the comparison between their IP formulation with Das model 'C' have illustrated that their IP formulation has a better performance.

Guo *et al.* [7] have presented another form of the MILP model based on a new concept *virtual relay*. Altinkemer *et al.* [1] have reformulated the same problem as an integer programming model of a set covering type. A multi-commodity flow model [24] has been presented for the MEB problem. The advantage of this new formulation is that both the LP relaxation and the Lagrangian relaxation of the integer program formulation obtain a good

approximation to the optimum.

2.4 Spanning Tree Heuristic Algorithm

The spanning tree heuristic algorithm maintains a tree routed at the source node. This algorithm iteratively adds new nodes to the tree based on a specific cost metric to acquire the minimum total energy. Several spanning tree algorithms for the MEB problem have been developed, e.g. the incremental power (BIP) [21], the broadcast average incremental power (BAIP) [14], the greedy perimeter broadcast efficiency (GPBE) [13], the center oriented broadcast routing algorithm (COBRA) [11].

The broadcast incremental power (BIP) heuristic algorithm exploits the wireless multicast advantage (WMA) to solve the MEB problem. The BIP is a centralized heuristic to construct the minimum energy broadcast tree. The cost metric in the BIP is defined as a minimum incremental power. As previously explained, the nodes should be equipped with omni-directional antennas to obtain WMA properties. The BIP is acquired from Prim's MST algorithm; however, the minimum shortest tree (MST) is used in the wired broadcast network. The BIP is referred as a node-based algorithm and MST as a link-based algorithm to find the minimum energy broadcast tree.

The transmission power between nodes i and j, P_{ij}, is defined as d^{α}. Initially, the tree includes the source node. By knowing the power matrix, the source node is connected to the nearest neighbor and the nearest neighbor is added to the tree. The next new node is added to the tree based on the minimum incremental power cost which is defined by:

$$P'_{ij} = P_{ij} - P(i)$$

in which $P_{ij} = r^{\alpha}$ and it is the link-based cost and $P(i)$ is the transmission power which is already assigned to the node i. The BIP algorithm measures the incremental power cost between nodes existing in the tree and nodes not existing in the tree in each iteration. This algorithm selects the minimum incremental power cost, and adds related nodes to the tree.

The BIP algorithm continues this procedure until the tree consists of all nodes.

Kang *et al.* [13] have developed a greedy perimeter broadcast efficiency (GPBE) which is associated with the broadcast efficiency metric. They have used the fundamental idea that the wireless broadcast advantage is more in the region where nodes are most densely distributed. The broadcast efficiency metric is defined as a number of newly covered nodes reached per unit transmission power. Let $\mathcal{N}_i$ denote a set of nodes can be reached by node i, and $\mathcal{C}$ denote a set of nodes currently covered by the transmission power of other nodes. The equation for the number of newly covered nodes by node i is $|\mathcal{N}_i \setminus \mathcal{C}|$. The wireless broadcast efficiency β_{ij} is defined as:

$$\beta_{ij} = \frac{|\mathcal{N}_{ij} \setminus \mathcal{C}|}{P_i} \text{ for i} \in \text{N}$$

The GPBE algorithm maintains two sets: $\mathcal{C}$ and $\mathcal{F}$. The set $\mathcal{C}$ represents nodes currently covered by other nodes. $\mathcal{F}$ represents a set of transmitting nodes such that $\mathcal{F} \subseteq \mathcal{C}$. In the beginning, $\mathcal{C} = \{\text{Source Node}\}$ and $\mathcal{F} = \emptyset$. A pair (i, j) for $i \in \mathcal{C}$ and $j \in N \setminus \mathcal{C}$ with a maximum β_{ij} can be found in each iteration. The P_{ij} is assigned to the node i. The node i is added to $\mathcal{F}$ and, $\mathcal{N}_i$ is added to $\mathcal{C}$. This procedure counties until $\mathcal{C} = N$. They have shown that the performance of MST, BIP, EWMA and GPBE in terms of total transmit power for network sizes of $20, 40, 60, 100, 150, 200, 300$. Furthermore, they have used the normalized total transmit power in 100 instants with $\alpha = 2$. Results have shown that the performance of GPBE is better than MST, but it is worse than BIP. The EWMA has obtained the best performance on all cases. Moreover, they reported that the GPBE possesses a better performance than BIP if the source node is located in the middle of the network region.

The center-oriented broadcast routing (COBRA) developed by Kang *et al.* [11] has considered that the center of a network region is the best place to take advantage of the broadcast nature. The main idea of the COBRA algorithm is that the source node sends a packet to a center node of the network region by using more efficient unicast path and the

center node broadcasts the packet. Three main concerns play important roles in this algorithm. The closest node to the center point of the network region is considered as the center node C. The shortest path tree (SPT) algorithm such as Dijkstra or distributed Bellman Ford algorithm [4] is then used for unicast path between the source node and the center node. Afterwards, different algorithms like EWMA, GPBE, BIP and MST are evaluated for the best option of the central broadcast algorithm. The ratio of the total transmit power for the random source location to the center source location is obtained for each given topology. Results illustrate that the EWMA and GPBE algorithms have the largest ratios, respectively. On the other hand, the location of the source node does not have any effects on BIP and MST algorithms. Next, they have compared several algorithms including EWMA, GPBE, BIP, MST, COBRA-GPBE, and COBRA-EWMA in the network size of 50, 100, 150, 200, 250, and 300. Results demonstrate that COBRA-EWMA has the best performance. The level of performance decreases in EWMA, BIP, COBRA-GPBE, and GPBE, respectively. In addition, there is a larger separation between COBRA-EWMA performance and the rest of the algorithm performances once the size of the network grows. Also, they have summarized that BIP, EWMA, and COBRA-EWMA reduce the total transmit power of MST about 7%, 16%, and 23%, respectively.

Kang *et al.* [9] have compared different algorithms which are not only based on the average of the total transmit power but also based on the average of the maximum and average hops and the average ratio of leaf nodes. In this study, MST, BIP, MST-Sweep, BIP-Sweep and EWMA algorithms have been considered. The results illustrate that they are ranked as EWMA, BIP-Sweep, BIP, MST-Sweep, and MST regarding to their performances. Also, they reported that the average ratio of leaf nodes to transmitter nodes in EWMA is the worst. Furthermore, the number of hops and the ratio of leaf nodes are closely related to each other. If the portion of leaf nodes are higher than transmitting nodes, it indicates that transmitting nodes transmit with a higher power. Therefore, the average and the maximum number of hops become smaller.

Table 2.1 [8] briefly explains spanning tree algorithms based on the complexity, the implementation fashion, and the approximation ratio.

	Complexity	Implementation Fashion	Approximation ratio
BIP	$O(n^3)$	Centralized	$\frac{13}{3} \leq \rho_{BIP} \leq 10.86$
BAIP	$O(n^3)$	Centralized	$\frac{4n}{\ln n} - o(1) \leq \rho_{BAIP}$
GPBE	$O(n^3)$	Centralized	Unknown

Table 2.1: Comparison of the Spanning Tree Algorithms

2.5 Local Search Heuristic Algorithm

Local search heuristic algorithms improve the solution obtained by an initial tree. The local search heuristic algorithm starts with an initial tree. The initial tree is acquired by a spanning tree heuristic algorithm such as BIP, MST, and etc. This algorithm reduces the total transmission power by assigning new powers to nodes or changing links between nodes in each step. The local search algorithm is terminated once there is no further improvement. Over the recent years, the algorithms like sweep [21], the embedded wireless multicast advantage(EWMA) [3], the r-shrink [6], the largest expanding sweep search(LESS) [10], and the iterative maximum-branch minimization(IMBM) [23] are proposed.

The sweep operation procedure [21] eliminates unnecessary transmissions and improves the performance of the BIP. The sweep procedure examines each non-leaf node in ascending order of its ID and reduces its transmission power, if its farthest children are covered by transmission of some other nodes. The algorithm stops when all nodes have been considered. The first run of this algorithm improves the performance of BIP by 5%. However, further runs show a little improvement.

Another local search algorithm containing the wireless multicat advantage (EWMA) is presented by Cagalj *et al.* [3]. The EWMA operation promotes the performance of the initial MST broadcast tree. The EWMA decision metric is a gain of transmitting nodes in the initial broadcast tree. The gain of a transmitting node, i, is defined as a decreased total energy. The gain is acquired by removing a transmitting node and increasing the transmis-

sion power of node i to cover children of a removed node. $\mathcal{C}$ denotes a set of covered nodes and $\mathcal{E}$ is a set of excluded nodes. The excluded nodes are transmitting nodes in the initial tree, but they will be discarded in the final optimal tree. Define $\mathcal{T}$ as a set of transmitting nodes in the initial tree and $\mathcal{R}_i$ is a set of the receivers of node i. The EWMA algorithm starts to construct a broadcast tree from nodes in the set $\mathcal{C} - \mathcal{F} - \mathcal{E}$ by determining their respective gains. Initially, $\mathcal{C} = \{S\}$ in which S is the source node and $\mathcal{E} = \mathcal{F} = \{\emptyset\}$. Thus, the EWMA starts from the source node and calculates its gain corresponding to other transmitting nodes as the following equations:

$$\Delta_S^i = \max_{\{j \in \mathcal{R}_i\}} \{e_{S,j}\} - e_S \quad \forall i \in \mathcal{T}$$

$$g_S^i = \sum_{k \in \mathcal{T}, e_{Sk} \leq e_{Si}} e_k - \Delta_S^i \quad \forall i \in \mathcal{T}$$

where e_{ij} is defined as the energy between node i and j in the initial MST tree. Having the gain for all nodes from $\mathcal{C} - \mathcal{F} - \mathcal{E}$, the algorithm selects a node with the highest positive gain in the set $\mathcal{F}$. The excluded nodes are added to set $\mathcal{E}$. Then, all of the covered nodes are added to $\mathcal{C}$. This procedure continues until all nodes in the network are covered. The simulated result has performed in 100 network instances with $\alpha = 2$ and $\alpha = 4$ in network sizes of 10, 30,50, and 100. Result show that the EWMA has a better performance than BIP and MST. Also, results reveal that the difference in performance of EWMA, BIP, and MST decreases as the loss constant (α) increases, because the cost of using longer link increases. Therefore, the performance of EWMA and BIP are close to MST as α increases.

Das *et al.* [6] have proposed the r-shrink heuristic local search algorithm to improve the performance of the initial broadcast tree algorithm such as BIP and MST. The r-shrink procedure sequentially shrinks the radii of transmitting nodes in the given broadcast tree. Let consider node i with P_{ij} as a transmitting node. Let $\{\alpha_0, \alpha_1, \ldots, \alpha_k, j\}$ denote the order

of nodes with respect to their distances from i. Therefore, node j is covered explicitly and nodes $\alpha_0, \alpha_1, \ldots,$ and α_k are covered implicitly. For $r = 1$, r-shrink reduces the transmission power of node i such that the farthest node is α_k instead of j. Now, the new parent for node j among its foster parent can be found based on an incremental and a decremental cost. The foster parents of node j is any of its non-descendants nodes, excluding the current parents. The algorithm compares the decremntal cost for current parent and the incremental cost for the new parent. The decremental cost is $P_{i,j} - P_{i,\alpha k}$. The incremental cost of adding node j to a new parent, k, with furthest node l is $P_{k,j} - P_{k,l}$. If k is a non-transmitting node, the incremental cost of adding node j is $P_{k,j}$. If value of the incremental cost is less than the value of the decremental cost, the node j chooses a new parent; otherwise, it keeps its current parent. Similarly, for $r = 2$, the r-shrink reduces the transmitting node power by 2 notches, such that the farthest node is α_{k-1}. Now, two nodes α_k and j try to find the new parent based on the incremental and decremental cost. They have evaluated 1-shrink algorithm on 10, 25, 50, 75, and 100 sizes of networks. They have used 50 network instants with $\alpha = 2$ in 5×5 area. The comparison between simulated results show the average total power for the BIP, BIP (sweep), BIP (1-shrink), MST, MST (sweep), and MST (1-shrink). BIP(1-shrink) outperforms BIP with 8.38%, 9.71%, 8.48%, 8.25%, and 9.05% for network sizes of 10, 25, 50, 75, and 100, respectively.

The large expanding sweep search (LESS) [10] is a local search heuristic algorithm which overcomes the shortage of EMWA. The LESS algorithm reduces either the transmission power or eliminates a node in each iteration. The start point in this algorithm is a node with a higher general gain. The gain in the LESS algorithm is different from the EWMA algorithm. There are some terminologies in the LESS algorithm as the following equations:

$$\Pi_{i \to S} = \{\text{all nodes in a path from i to S}\}$$

$$\mathcal{Q}_i(j) = \Pi(\mathcal{N}_i(j)) \setminus i$$

$$\mathcal{M}_i(j) = \mathcal{N}_i(j) \setminus \Pi_{i \to S}$$

in which $\delta(i)$ is a set of children of the node i in the initial tree. $\mathcal{N}_i(j)$ denotes a set of receivers of the node i. $\mathcal{Q}_i(j)$ is a set of parent nodes of $\mathcal{N}_i(j)$ except node i. This set includes all nodes which will be tested for an expanding sweep search. The sweeping gain is defined as:

$$SG_{i \to j} = \sum_{u \in \mathcal{Q}_i(j)} \left(P(u) - \max_{k \in \delta(u) \setminus \mathcal{M}_i(j)} \{P_{uk}\} \right) \tag{2.15}$$

Let $\Delta P_{i \to j} = P_{ij} - P_i$ represents the incremental power of the node i. Then, the gain is defined as:

$$G_{i \to j} = SG_{i \to j} - \Delta P_{i \to j} \tag{2.16}$$

Therefore, the gain includes the generalized sweeping gain minus the incremental power. The generalized sweeping gain consists of both eliminated and reduced transmit power of nodes. If the considered node includes children of the node i which is not in $\mathcal{M}_i(j)$ in the initial tree, the power of the node is reduced; otherwise, the node is eliminated. The (i, j) is selected based on the maximum positive gain. The P_{ij} is assigned to the node i, and it updates the parent node of the covered nodes except the path nodes $\Pi_{i \to S}$ to node i. Now, the new improved tree is considered as an input tree for the next iteration. The operation repeats until the algorithm cannot find any gain. By applying the LESS algorithm on the initial EWMA tree, there is still a significant improvement. However, applying the EWMA algorithm on the LESS initial tree has no gain.

The iterative maximum-branch minimization (IMBM) is another local search algorithm presented by Li *et al.* [23]. It starts from a basic broadcast tree in which the source node S directly transmits to all other nodes. Then, this algorithm minimizes the maximum branch for each transmitting node. After constructing a basic broadcast tree, the IMBM algorithm can find the minimum energy broadcast tree iteratively by using the maximum branch replacement (MBR) and recursive omni-directional check (ROC) operations. The

MBR operation replaces the maximum branch for a given transmitting node by a two-step less power path. This process is done by using a relay node. The relay node k can be found for the transmitting node i and a node with the maximum branch (the longest link in the tree) j such that $P_{ik} + P_{kj} < P_{ij}$. In the ROC operation, all nodes can be reached by the transmitting node which is the middle node in the two-step less power path are connected to the transmitting node. In fact, the ROC exploits the wireless broadcast advantage. The algorithm would stop if the total transmission power for the broadcast tree cannot be further reduced. Results illustrate that the IMBM outperforms BIP in $\alpha = 2$, but for a larger α IMBM does not have a better performance to BIP.

Table 2.2 [8] shows briefly the summary of local search algorithm trees based on the complexity of the improvement, the implementation fashion and the search neighborhood.

	Complexity of Improvement	Implementation Fashion	Search Neighborhood
Sweep	$O(n)$	Centralized	Tree-based
EWMA	$O(n^3)$	Centralized	Tree-based
r-Shrink	$O(n^2)$	Centralized	Power Assignment based
LESS	$O(n^3)$	Centralized	Power Assignment based
IMBM	$O(n)$	Centralized	Tree-based

Table 2.2: Comparison of the Local Search Algorithms

2.6 Meta-heuristic Algorithm

Meta-heuristic algorithms are based on evolutionary algorithms and local search algorithms. Several meta-heuristic algorithms have been proposed for the MEB problem, e.g. Genetic Algorithm (GA) [23], evolutionary local search (ELS) [22], iterated local search (ILS) [12], hybrid genetic algorithm (HGA) [20], ant colony optimization [23], particle swarm optimization [23], cluster-merged algorithm [5], and simulated annealing algorithm [17].

CHAPTER 3

BALANCED-POWER MINIMUM ENERGY BROADCAST TREE

We have implemented a mixed integer program (MIP) with the power balance objective to construct a broadcast tree in this chapter. The resulting tree is suitable to implement a RaptorQ-based protocol. In Section 3.1, characteristics of a balanced-power broadcast tree is explained. In Section 3.2, the mixed integer program will be discussed in detail. In Section 3.3, we explain the programming code in the optimization software IBM ILOG CPLEX. In Section 3.4, results obtained from MIP with the minimum energy and power balance objectives, and BIP will be compared in different network typologies.

3.1 Characteristic of Balanced-power Minimum Energy Broadcast Tree

The main objective of this thesis is to develop an efficient RaptorQ-based protocol for file distribution over a wireless mesh network. The construction of broadcast trees which are suitable for implementing a RaptorQ-based protocol is required. Characteristics of a suitable broadcast tree include energy efficiency, timely delivery, and the ability to deploy mechanisms to verify a successful delivery. We consider the ability of parent nodes opportunistically to overhear transmissions from their children as a mechanism to verify a successful delivery. Thus, such overheard transmissions serve acknowledgments and the requirement of sending acknowledgments is almost eliminated in this protocol. However, the child's transmission power often is far below the parent's transmission power in the pure minimum energy broadcast tree. Therefore, we add constraints into mixed integer program to force the transmission power of child nodes to meet SINR requirement of their parent nodes. In addition, we limit and minimize the maximum transmission power using an imbalance factor, β. Thus, there is a trade-off between the minimum-energy and the power-balanced broadcast trees.

3.2 MIP Formulation

The MIP formulation based on the balanced power objective is defined as:

$$\text{Minimize} \sum_{i=1}^{N} Z_i$$

Subject to:

$$Z_i - P_{ij} X_{ij} \geq 0;\ i \neq j \tag{3.1}$$

$$\sum_{i \in \mathcal{N}, i \neq j} X_{ij} = 1 \tag{3.2}$$

$$X_{ij} + X_{jk} \leq 1 \quad \forall i, j, k \in \mathcal{N},\ j \in \mathcal{R}_i,\ k \in \mathcal{R}_j \tag{3.3}$$

$$\frac{P_{ij} X_{ij}}{\eta + \sum_{k,l \in \mathcal{N}, k \neq i} \frac{P_{kl} X_{kl}}{\ell_{kj}}} \geq \gamma_{\min} \ell_{ij} \tag{3.4}$$

$$\sum_{i = \mathcal{N}_0, j \in \mathcal{N}, i \neq j} X_{ij} \geq 1 \tag{3.5}$$

$$Z_j - \ell_{ji} \gamma_{\min} \eta \geq 0 \quad \forall j \in \mathcal{R}_i \tag{3.6}$$

$$Z_m \leq Z_i \leq \frac{1}{\beta} Z_m \tag{3.7}$$

$$\sum_{j=1}^{N} F_{ij} = N;\ i = \mathcal{N}_0,\ i \neq j \tag{3.8}$$

$$\sum_{j=1}^{N} F_{ji} = 0;\ i = \mathcal{N}_0,\ i \neq j \tag{3.9}$$

$$\sum_{j=1}^{N} F_{ji} - \sum_{j=1}^{N} F_{ij} = 1;\ \forall\ i \neq j \tag{3.10}$$

$$X_{ij} \in \{0, 1\} \tag{3.11}$$

$$F_{ij} \geq 0;\ \forall i \neq j \tag{3.12}$$

$$\beta = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\} \tag{3.13}$$

where $\mathcal{N} = \{\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_{N-1}\}$ is a collection of nodes in the network, and $\mathcal{N}_0$ is a source node which is a gateway including a file as well. Z_i, F_{ij}, and P_{ij} are the transmission power of node i, the flow variable and the power required between nodes i and j respectively. β is an imbalance factor, and $\mathcal{R}_i$ is a set of receivers to which node i transmits. $\gamma_{\min}$ is the minimum SINR required to meet BER requirements. ℓ_{ij} and η are the loss between nodes i and j and the noise at the receiver respectively. X_{ij} is a binary variable which is 1 if node i transmit to node j, otherwise 0.

$$X_{ij} = \begin{cases} 1 & \text{if node } i \text{ transmits to node } j, \\ 0 & \text{else.} \end{cases} \tag{3.14}$$

The constraint (3.1) ensures that an appropriate power is assigned to node i. The constraint (3.2) enforces that each node receives from exactly one transmitter. The constraint (3.3) shows that a node cannot transmit and receive simultaneously. The constraint (3.4) is a power constraint and the cumulative interference term will be eliminated if any transmitting nodes do not transmit simultaneously. At least one connection from the source node is guaranteed by constraint (3.5). The constraint (3.6) enforces that the child node transmits in high enough power to reach its parent node. The constraint (3.7) limits the transmission power to within a certain interval of the maximum transmission power. The constraint (3.8) expresses that the number of flows coming out of the source node is N which is cardinality of set $\mathcal{N}$. The constraint (3.9) shows that the number of flows coming into the source node is zero. The constraint (3.10) enforces that the difference between flows into and out of the node is 1. It means that each node keeps one encoded symbol. The final set of constraints express the integerality of X_{ij} variables, non-negativity of F_{ij} variables, and a proper amount of the β variable.

3.3 Programming Code in CPLEX Python API

We use the IBM ILOG CPLEX optimization software to develop the mixed integer programming model. The CPLEX optimizer has a modeling layer called Concert that provides interfaces to C++, #.NET, Java, Matlab, and Python languages. We use the Python API of the CPLEX optimizer. There are various instance methods and the Cplex class to create, modify, and solve optimization problems in the CPLEX optimizer. Table 3.1 briefly introduces these methods.

Method	**Purpose**
init(self, *args)	Constructor of the Cplex class
read(self, filename, filetype)	Reads a problem from file
write(self, filename, filetype)	Writes a problem to file
get_problem_type(self)	Returns the problem type
set_problem_type(self, type, soln=None)	Changes the problem type
solve(self)	Solves the problem
set_results_stream(self, results_file, fn=None)	Specifies where results will be printed
set_log_stream(self, log_file, fn=None)	Specifies where the log will be printed
get_problem_name(self)	Returns the problem name

Table 3.1: Essential Instance Methods in Cplex Class

An indicator constraint interface in Cplex class is used to deliberate relationships among variables by identifying a binary variable to control whether or not a specified linear constraint is active. There are different instance methods to add and modify indicator constraints. Table 3.2 briefly introduces these methods.

add method is used to add constraints to the problem which is initiated by *Cplex.cplex* method in the Cplex class. The add method includes following arguments:

- **lin_expr=SparsePair(ind = [], val = []) :** This is a linear expression which is either a SparsePair or a list of two lists. The first one contains variable indices or names, the second one contains values.

- **Sense:** It is the sense of the constraint, may be "L" as less, "G" as greater, or "E"

as equal. A default value is "E".

- **rhs :** It is a float defining the right hand side of the constraint.
- **indvar :** It is the name or index of the variable that controls if the constraint is active.
- **complemented :**It determines whether the constraint is active when the variable indvar is equal to 0 or 1. The default value is 0.
- **name :** It is the name of the constraint.

Method	**Purpose**
add(self, lin_expr=SparsePair(ind = [], val = []), sense='E', rhs=0.0, indvar=0, complemented=0, name='')	Adds an indicator constraint to the problem
delete(self, *args)	Deletes a set of indicator constraints from the problem
get_indicator_variables(self, *args)	Returns the indicator variables of a set of indicator contraints
get_complemented(self, *args)	Returns whether a set of indicator constraints is complemented
get_rhs(self, *args)	Returns the righthand side of a set of indicator constraints
get_senses(self, *args)	Returns the sense of a set of indicator constraints
get_linear_components(self, *args)	Returns the linear constraint of a set of indicator constraints
get_names(self, *args)	Returns the names of a set of indicator constraints
set_log_stream(self, log_file, fn=None)	Specifies where the log will be printed
get_problem_name(self)	Returns the problem name

Table 3.2: Essential Instance Methods in Indicator Constraints

An arrays is initialized to store objective function multipliers, upper and lower bounds for variables, and variable types. The first N variables correspond to node variables which are the transmission powers of nodes. The initial value for the objective function multiplier, the lower bound, the upper bound, and the variable type of node variables are 1, 0, infinity,

and float, respectively. The next $N(N-2)+1$ variables are link variables. The link variable is an indicator that tells whether the link is used in the broadcast tree or not. All of the possible links are enumerated using the *equiv_index(*N, i, j*)* function. The initial value for the lower bound, the upper bound, and the variable type of link variable are 0, 1, and binary, respectively. The next $N(N-2)+1$ variables are flow variables. The flow variable determines the target nodes for each packet in each link. All links are enumerated using *flow_equiv_index (*N, i, j*)* to assign the flow variable. The initial values for the lower bound, the upper bound, and the variable type of flow variables are 0, $N-1$, and integer respectively. The total number of the variables is $N + N(N-2) + 1 + N(N-2) + 1$ up to now. The $N + N(N-2) + 1 + N(N-2) + 1 + 1^{\text{th}}$ variable is K_P which is the index of the power variable. It determines the transmission power of a node in a certain range of the maximum power. The initial value for the objective function multiplier, the lower bound, the upper bound, and the variable type of the power variable index is 1, the loss of nearest node to the source, infinity, and float. The last N variables correspond to the balanced power of transmission nodes. Thus, we have $N + N(N-2) + 1 + N(N-2) + 1 + 1 + N$ variables which is $N(N-1) + 1 + (N-1)(N-1) + N + 1$. Then, we use the *variable* interface and the *add* method in the Cplex class to add these variables to the problem.

```
prob.variables.add(obj = obj, lb = lb, ub = ub, types = "".join(ct)
```

Listing 3.1: Add Variables to the Problem

in which obj, lb, ub and types are the objective function multiplier, the lower bound, the upper bound, and the variable type.

Now, each of constraints is set up to use the indicator interface. Listing 3.2 shows the command for the constraint (3.2) that forces the receivers to receive from exactly one transmitter.

```
for j in range(1,Number_Nodes):
      ind_set = []
      val_set = []
      for i in range(0,Number_Nodes):
          if (j != i):
              ind_set.append(equiv_index(Number_Nodes,i,j))
              val_set.append(1)
      prob.linear_constraints.add(lin_expr = \
          [[ind_set,val_set]], senses='E',rhs=[1.0])
```

Listing 3.2: The Python Code for Constraint 3.2

Listing 3.3 shows commands for the constraint (3.4). This constraint forces that the transmitters to transmit at high enough power to reach all receiver nodes. There is one constraint for each of the children nodes of each transmitter.

```
gamma = 2.0 #minimum required SINR
  for m in range(Number_Nodes):
      for j in range(1,Number_Nodes):
          if (j != m):
              k = equiv_index(Number_Nodes,m,j)
              ic_dict["lin_expr"]     = cplex.SparsePair(
              ind =[m],val = [1.0])
              ic_dict["rhs"]          = gamma*loss[m][j]
              ic_dict["sense"]        = "G"
              ic_dict["indvar"]       = k
              ic_dict["complemented"] = 0
              prob.indicator_constraints.add(**ic_dict)
```

Listing 3.3: The Python Code for Constraint 3.4

Listing 3.4 shows commands for the constraint (3.5) that forces at least one connection from the source node. The Python code refers that the sum of the *equiv_index(Number_Nodes,0,j)* values for $i = 0$ is at least 1.

```
ind_set = []
  val_set = []
  for j in range(1,Number_Nodes):
      ind_set.append(equiv_index(Number_Nodes,0,j))
      val_set.append(1)
  prob.linear_constraints.add(lin_expr = \
          [[ind_set,val_set]], senses='G',rhs=[1.0])
```

Listing 3.4: The Python Code for Constraint 3.5

Listing 3.5 shows commands for the constraint (3.6). It forces the transmitters transmit at a high enough power to reach their predecessors for ACK. Listing 3.6 shows commands for the constraint (3.7) part $Z_i \leq \frac{1}{\beta} Z_m$. This constraint forces the transmission power of all transmitters to be within a certain factor of the minimum transmission power.

```
gamma = 2.0 #minimum required SINR
  for m in range(1,Number_Nodes):
      for j in range(0,Number_Nodes):
          if (j != m):
              k = equiv_index(Number_Nodes,j,m)
              ic_dict["lin_expr"]     = cplex.SparsePair(
              ind =[m,k],val = [1.0,-gamma*loss[j][m]])
              ic_dict["rhs"]          = 0
              ic_dict["sense"]        = "G"
              ic_dict["indvar"]       =  K_P+m+1
              ic_dict["complemented"] = 0
              prob.indicator_constraints.add(**ic_dict)
```

Listing 3.5: The Python Code for Constraint 3.6

For example, if $\beta = 0.1$ the transmission power of the considering node would be less than 10 times of the power required for a minimum broadcast tree.

```
gamma = 2.0 #minimum required SINR
  for m in range(1,Number_Nodes):
      for j in range(0,Number_Nodes):
          if (j != m):
              k = equiv_index(Number_Nodes,j,m)
              ic_dict["lin_expr"]     = cplex.SparsePair(ind =
               [m,k],val = [1.0,-gamma*loss[j][m]])
              ic_dict["rhs"]          = 0
              ic_dict["sense"]        = "G"
              ic_dict["indvar"]       =  K_P+m+1
              ic_dict["complemented"] = 0
              prob.indicator_constraints.add(**ic_dict)
```

Listing 3.6: The Python Code for Constraint 3.7

Listing 3.7 shows commands for the constraint(3.8) part $Z_i \geq Z_m$ that forces the transmission power of all transmitters to be less than some maximum number.

```
or m in range(Number_Nodes):
      ic_dict["lin_expr"]     = cplex.SparsePair(
      ind = [K_P, m],val = [ef,-1])
      ic_dict["rhs"]          = 0
      ic_dict["sense"]        = "L"
      ic_dict["indvar"]       = K_P+m+1
      ic_dict["complemented"] = 0
      prob.indicator_constraints.add(**ic_dict)
```

Listing 3.7: The Python Code for Constraint 3.8

3.4 Results

The simulation results obtaining from implementation of three methods, MIP with the minimum energy objective, MIP with the power balance objective, and BIP are presented in this section. We consider two different typologies in which nodes of the second network are more evenly distributed than the first one. Also, we evaluate and compare the total energy in the pure minimum energy broadcast tree with a balanced power broadcast tree with different imbalance factors β in the third topology. 25 nodes are located randomly in a 100×100 network area. We consider $\gamma_{\min} = 2$, $\beta = 0.5$, and $\alpha = 2$ in this simulation. Figure 3.1, Figure 3.2, and Figure 3.3 show the minimum energy broadcast tree, the broadcast tree

using MIP with the power balance objective with $\beta = 0.5$, and the broadcast tree using BIP heuristic in topology 1, respectively.

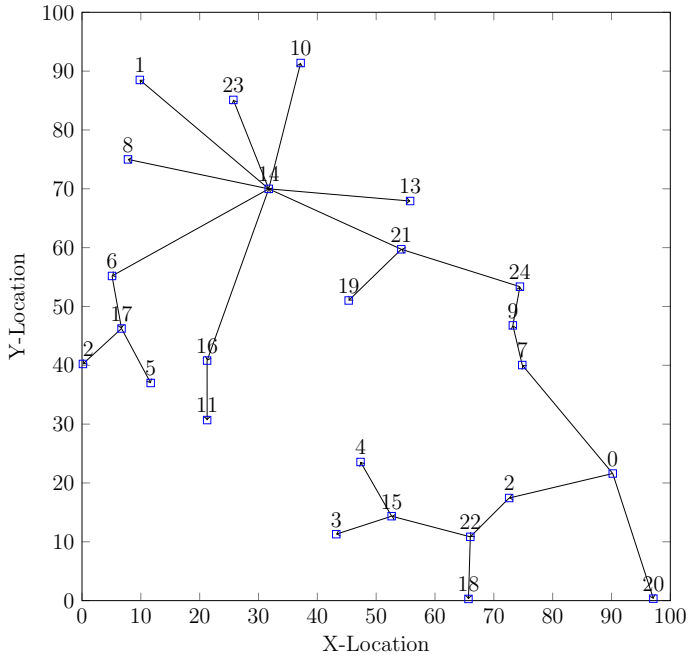


Figure 3.1: Minimum Energy Broadcast Tree in Topology 1

The summery of the minimum energy broadcast tree using MIP with the minimum energy objective in Figure 3.1 is:

- Source node 0 transmits and $\mathcal{R}_0 = \{2, 7, 20\}$
- Node 2 transmits and $\mathcal{R}_2 = \{22\}$. The parent node, node 0, can not hear it.
- Node 22 transmits and $\mathcal{R}_{22} = \{15, 18\}$. The parent node, node 2, can hear it.
- Node 15 transmits and $\mathcal{R}_{15} = \{3, 4\}$. The parent node, node 22, can not hear it.
- Node 7 transmits and $\mathcal{R}_7 = \{9\}$. The parent node, node 0, can not hear it.
- Node 9 transmits and $\mathcal{R}_9 = \{24\}$. The parent node, node 7, can hear it.
- Node 24 transmits and $\mathcal{R}_{24} = \{21\}$. The parent node, node 9, can hear it. Although node 7 is not in set of receiver, it can overhear the transmission of node 24.

- Node 21 transmits and $\mathcal{R}_{21} = \{14, 19\}$. The parent node, node 24, can hear it. Although node 13 is not in set of receiver, it can overhear the transmission of node 21.

- Node 14 transmits and $\mathcal{R}_{14} = \{1, 6, 8, 10, 13, 16, 23\}$. The parent node, node 21, can hear it. Although nodes 17 and 19 are not in set of receiver, they can overhear the transmission of node 14.

- Node 16 transmits and $\mathcal{R}_{16} = \{11\}$. The parent node, node 14, can not hear it.

- Node 6 transmits and $\mathcal{R}_6 = \{17\}$. The parent node, node 14, can not hear it.

- Node 17 transmits and $\mathcal{R}_{17} = \{2, 5\}$. The parent node, node 6, can hear it.

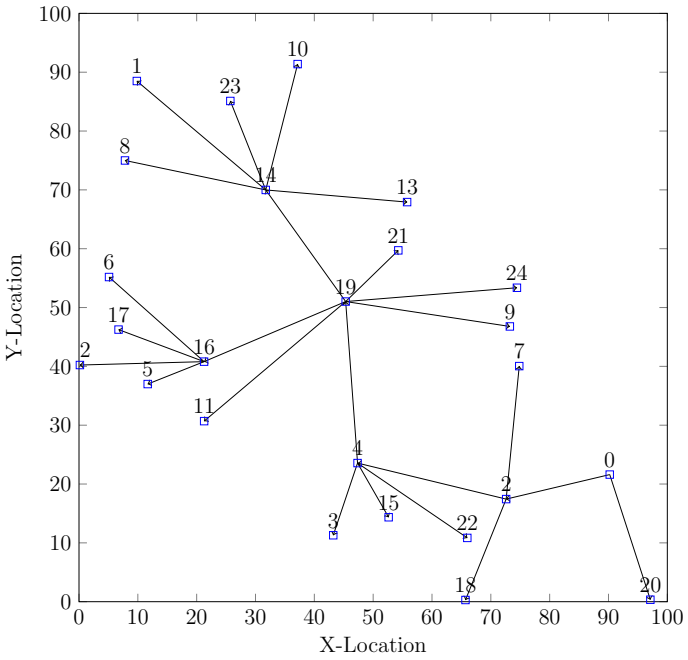


Figure 3.2: Balanced-power Broadcast Tree, $\beta = 0.5$ in Topology 1

The summery of the minimum energy broadcast tree using MIP with the power balance objective in Figure 3.2 is as follows:

- Source node 0 transmits and $\mathcal{R}_0 = \{2, 20\}$

- Node 2 transmits and $\mathcal{R}_2 = \{4, 7, 18\}$. The parent node, node 0, can hear it. Although node 15 and 22 are not in set of receiver, they can overhear the transmission of node 2.

- Node 4 transmits and $\mathcal{R}_4 = \{3, 15, 19, 22\}$. The parent node, node 2, can hear it. Al

- Node 19 transmits and $\mathcal{R}_{19} = \{9, 11, 14, 16, 21, 24\}$. The parent node, node 4, can not hear it. Although node 13 is not in set of receiver, it can overhear the transmission of node 19.

- Node 16 transmits and $\mathcal{R}_{16} = \{2, 5, 6, 17\}$. The parent node, node 19, can not hear it. Although node 11 is not in set of receiver, it can overhear the transmission of node 16.

- Node 14 transmits and $\mathcal{R}_{14} = \{1, 8, 10, 13, 23\}$. The parent node, node 19, can hear it. Although node 21 is not in set of receiver, it can overhear the transmission of node 14.

The numerical results obtained from implementation of these methods are tabulated in Table 3.3. The results show that the total energy in MIP with the balanced-power constraint is 30% higher than the minimum energy broadcast tree in topology 1. On the other hand, the number of transmitters has been decreased from 12 to 6 in MIP with the balanced-power objective method. Therefore, the delivery time declines significantly. Moreover, the potential for reducing delivery time at the expense of the modest energy increase exists. Thus, there is a trade-off between the minimum energy broadcast tree and the minimum delivery time. In addition, the number of overhearing nodes has been increased from 4 to 6 and the maximum power has been decreased by 12%. As can be seen in Table 3.3, the total energy of BIP is between the total energy of MIP with the minimum energy and the energy consumption of MIP with th balanced-power. In addition, the BIP heuristic is not an appropriate option to utilize the delivery mechanism in broadcast

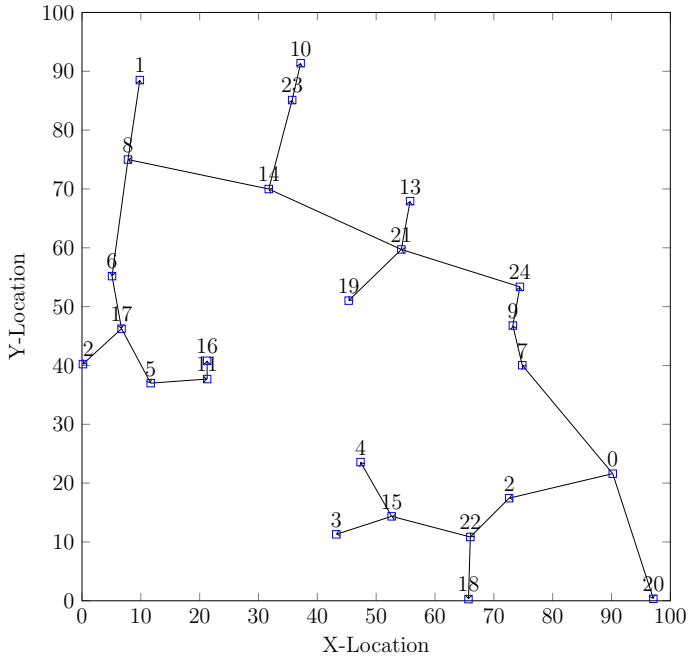


Figure 3.3: Minimum Energy Broadcast Tree using BIP in Topology 1

tree because, the transmission power of child node is blow of transmission power of parent node in most cases. Thus, parents cannot overhear the children transmission. Also, BIP is not good for the minimum delivery time, because the number of transmitters in BIP is higher than two other cases.

	Total Power	Number of Transmitters	Max Power	Min Power	Overhearing Nodes	Percentage of Increased Energy
MIP(energy)	6698	13	1923	29	6, 7, 13, 19	0
MIP(balanced)	8568	6	1705	1000	9, 11, 13, 15, 21, 22	30%
BIP	6906	14	598	25	13, 19	3%

Table 3.3: The Numerical Results based on Three Different Methods in Topology 1

Figure 3.4 illustrates pure minimum energy broadcast tree in topology 2. Figure 3.5 displays the broadcast tree using MIP with the power balance objective with $\beta = 0.5$ in topology 2. Figure 3.6 illuminates the broadcast tree using BIP heuristic in topology 2. The difference between the maximum and the minimum power in topology 2 is less than difference between the maximum and the minimum power in topology 1, and nodes are more evenly

distributed in topology 2. The numerical results provided in Table 3.4 show that the total energy has been increased by 27% in MIP (balanced) in comparison with MIP (energy). The number of transmitters has been decreased from 13 to 6 which lead to decreasing the delivery time. In addition, the number of overhearing nodes has been increased from 3 to 7 in MIP with the balanced-power objective method. The result for BIP in this topology is very similar to the previous one.

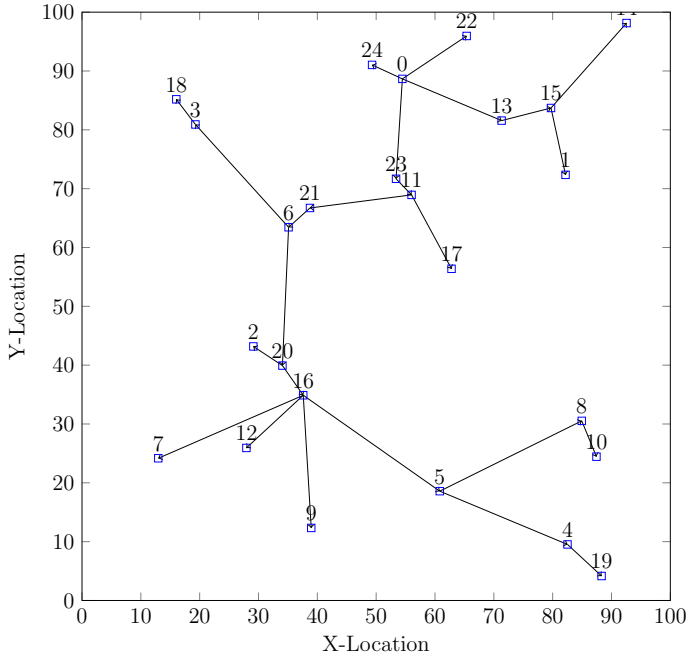


Figure 3.4: Minimum Energy Broadcast Tree in Topology 2

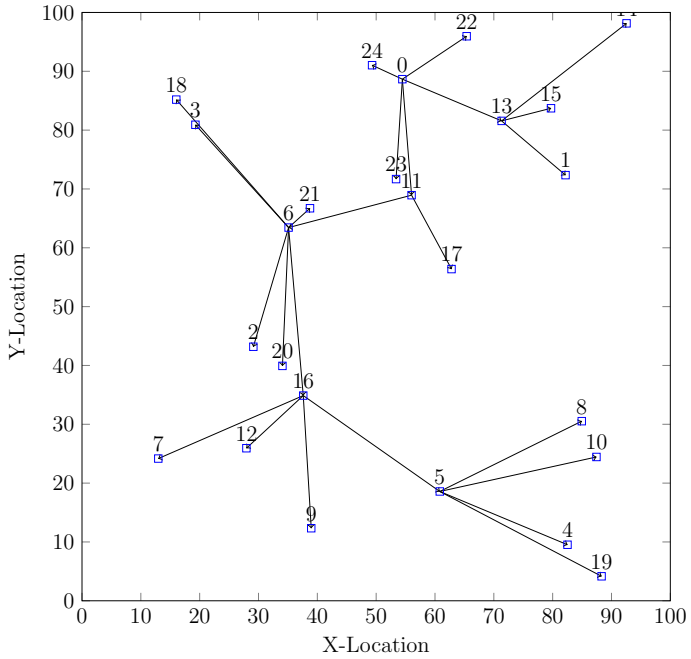


Figure 3.5: Balanced-power Broadcast Tree $\beta = 0.5$ in Topology 2

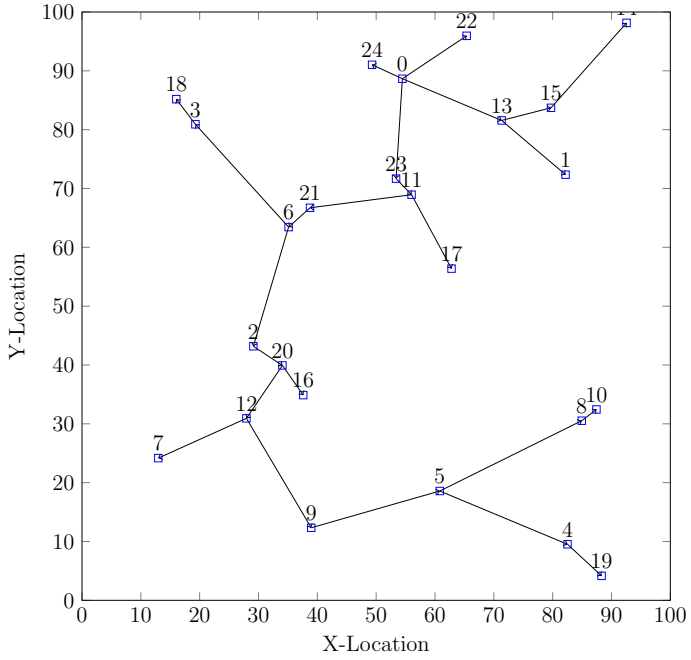


Figure 3.6: Minimum Energy Broadcast Tree using BIP in Topology 2

	Total Power	Number of Transmitters	Max Power	Min Power	Overhearing Nodes	Percentage of Increased Energy
MLP(energy)	6698	12	1608	60	13, 23, 2	0
MLP(balanced)	8627	6	1928	964	2, 9, 11, 17, 20, 21, 23	27%
BIP	7542	14	725	20	2, 9, 16, 22	13%

Table 3.4: The Numerical Results based on Three Different Methods in Topology 2

We implement MIP with the balanced-power objective for $\beta = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$ and, 1 in topology 3. Figure 3.7 displays the pure minimum energy broadcast tree and Figure 3.8 demonstrates the power balanced broadcast tree for $\beta = 0.2, 0.4, 1$.

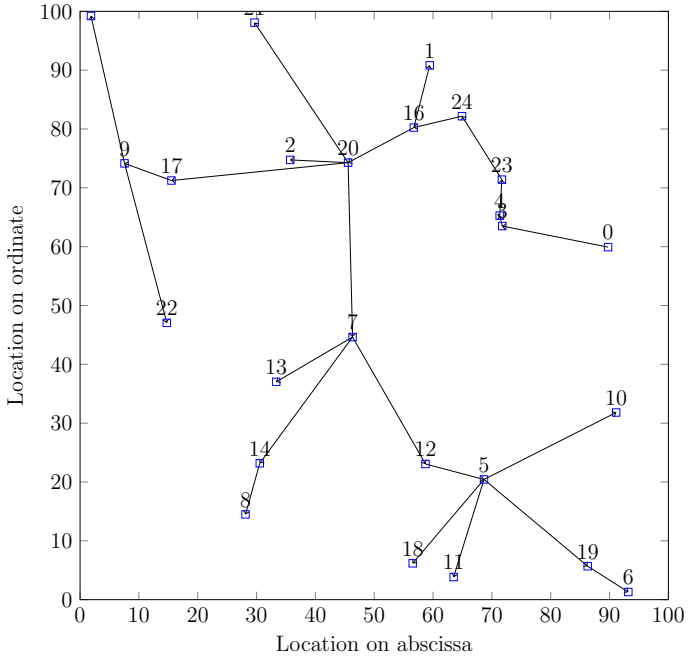
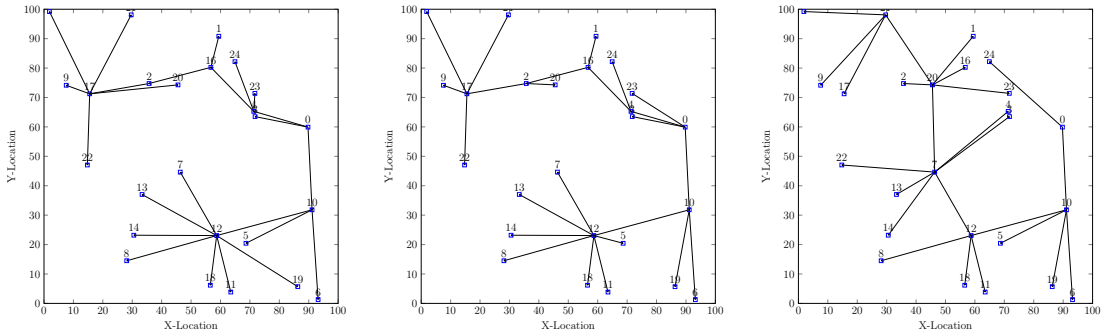


Figure 3.7: Minimum Energy Broadcast Tree in Topology 3



(a) $\beta = 0.2$ (b) $\beta = 0.4$ (c) $\beta = 1$

Figure 3.8: Balanced-power Broadcast Tree with Different β

Numerical results for various values of β for topology 3 are tabulated in Table 3.5. The total power increases by increasing the amount of β, but the power of nodes are in a smaller range. Also, Figure 3.9 illustrates the total normalized energy required in different imbalance factors. Thus, we can have a more balanced power network which is suitable for a reliable broadcast tree by increasing the value of β with respect to increasing the total power.

	Total Power	Number of Transmitters	Range of Power
MLP(energy)	827	14	75-1821
$\beta = 0.1$	10712	7	320-2254
$\beta = 0.2$	10791	7	873-2254
$\beta = 0.3$	10791	7	873-2254
$\beta = 0.4$	10819	6	901-2254
$\beta = 0.5$	11236	7	1383-2254
$\beta = 0.6$	11236	7	1383-2254
$\beta = 0.7$	11431	7	1578-2254
$\beta = 0.8$	11876	7	1803-2254
$\beta = 0.9$	12625	6	2029-2254
$\beta = 1$	13527	6	2254

Table 3.5: The Numerical Results based on Different Values of β in Topology 3

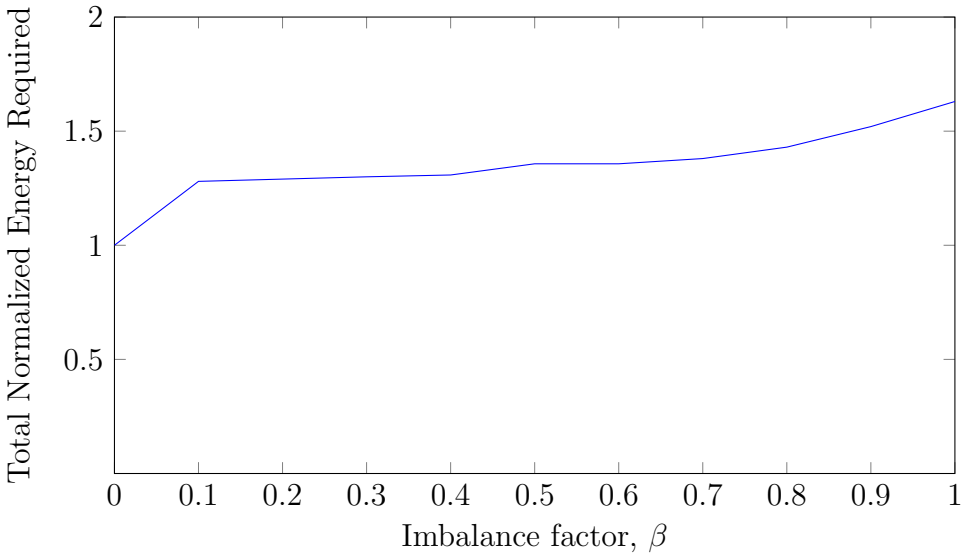


Figure 3.9: Power Balance Broadcast Tree for Different Amounts of β in Topology 3

CHAPTER 4

RaptorQ-based BROADCAST PROTOCOL

This chapter presents the RaptorQ broadcast protocol. This protocol implements a reliable energy efficient file distribution over the wireless ad hoc network by exploiting the properties of RaptorQ and wireless broadcast advantages. Section 4.1 discusses the explanation of the RaptorQ broadcast protocol. Section 4.2 explains the RaptorQ broadcast protocol algorithm which is designed by Daigle and finite state machines developed by Wang.

4.1 Description of the RaptorQ Broadcast Protocol

The main concept of this protocol is that each transmitter node is responsible for delivering the file to its children. Each transmitter node generates repair symbols with ID equal to its own ID mod N after constructing the file. Thus, different encoded symbols are distributed over the network by different transmitters expediting the file distribution, because the receiver is able to reconstruct the file after collecting enough distinct symbols. Another important concept of this protocol is that broadcast nature of the wireless link can accelerate the file delivery. It means that the reception of each node is not limited those received from its parent. That is, the node can also overhear the transmission of other multiple nodes.

We are given a fully connected mesh network of N nodes, $\mathcal{N} = \{\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_{N-1}\}$, of which one node, $\mathcal{N}_0$, has content to be delivered to all other nodes. The content is partitioned into a L fixed-length symbols, and these symbols are RaptorQ encoded prior to transmission. According to the properties of RaptorQ, we define a failure probability of less than 10^{-6} with an overhead of 2 symbols. $L + 2$ required symbols is called $k_{\min}$ in the developed protocol. The broadcast tree is obtained by applying the MIP with its balanced-power objective. Thus, we have a set of transmitter nodes and a set of their children nodes.

We define three types of nodes *source*, *transmitter*, and *leaf* in this protocol. Children nodes of a transmitter can be either a transmitter node or a leaf node. The nodes which have a file are considered as *source* nodes. The network includes only one *source* node which is $\mathcal{N}_0$ in the beginning of the process. Nodes which have a set of receivers are *transmitter* nodes. These nodes will be *source* nodes whenever they successfully decode the file. The nodes that do not have any children nodes are called *leaf* nodes. Also, each transmitter in the network is assigned a transmitter identification number which starts with 0 in the source node. In addition, two children of the same parent are not allowed to be scheduled to transmit in the same slot because of the implementation of the balanced-power broadcast tree. The protocol has five discrete phases for *source* and *transmitter* nodes, *STARTUP*, *FINISHING (Poll)*, *FINISHING (Wait)*, *FINISHING (Extra)* and *COMPLETED*. Leaf nodes include only *STARTUP* and *COMPLETED* phases.

All packets transferred between nodes are in the same format. We define a packet format that consists of four parts: time, header, payload which is an encoded symbol, and tail. The time in the packet is used for scheduling. The data type part specifies the type of this packet which can be either NORM, POLL, MORE, or DONE. The payload is an encoded symbol which can be source symbols or repair symbols. The tail part is employed when the header is POLL or MORE. In the POLL header type, the tail will be the IP address of selected children, and the tail is K_{extra} which implies the number of extra symbols needed to reconstruct the file in the MORE case.

4.2 Algorithm for RaptorQ Broadcast Protocol

We discuss the details of the RaptorQ broadcast protocol in this section. The RaptorQ broadcast protocol facilitates a reliable distribution of a file over a wireless ad hoc network. We explain the basic features of this algorithm designed by Daigle for three different node types as follows:

1. *source*

- During the START-UP phase, node $\mathcal{N}_0$ transmits encoded symbols having sequential sequence numbers. The encoded symbols include source symbols and repair symbols randomly.
- During the START-UP phase, node $\mathcal{N}_0$ updates the symbol sender's state if it receives any encoded symbols.
- After transmitting $K_{\min}$ sequentially numbered encoded symbols, $\mathcal{N}_0$ changes over to the FINISHING (Poll) phase.
- During the FINISHING (Poll) phase, node $\mathcal{N}_0$ transmits a POLL message to poll the selected neighbor. Thus, non-polled children do not need to reply to the POLL message. Also, the POLL message can be another symbol with a POLL header. Node $\mathcal{N}_0$ changes over to the FINISHING (Wait) phase.
- During the FINISHING (Wait) phase, node $\mathcal{N}_0$ waits to receive the response from the selected children. This response can be either a MORE message or a DONE message. If node $\mathcal{N}_0$ receives a DONE message, it updates the status of the selected child as DONE. The node $\mathcal{N}_0$ changes over the FINISHING (Extra) phase once it receives MORE message.
- During the FINISHING (Extra) phase, node $\mathcal{N}_0$ generates K_{extra} number of repair symbols with symbol IDs that have the same number as its identifier 0 mod N and transmits them sequentially in its time slot. After sending K_{extra} new generated encoded symbols, $\mathcal{N}_0$ changes over to the FINISHING (Poll) phase.
- After coming back to the FINISHING (Poll) phase, node $\mathcal{N}_0$ transmits a POLL message to the next selected children. $\mathcal{N}_0$ changes over from the FINISHING (Poll) phase to the COMPLETED phase once all of its children marked as DONE.

2. *Transmitter*

- During the START-UP phase, all other transmitters can receive the NORM type of data which is an encoded symbol or a POLL message. If they receive a POLL

message, they schedule a MORE message with the number of K_{extra} in the next coming time slot. Otherwise, the node simply sends the newly received encoded symbols to its children in its time slot.

- Transmitter nodes after receiving $K_{\min}$ and reconstruct the file switch to the FINISHING(Poll) phase.
- During the FINISHING (Poll) phase, if the transmitter receives a POLL message that polls itself it schedules a DONE message in its coming time slot.
- During the FINISHING (Poll) phase, the transmitter nodes transmits a POLL message to poll the selected neighbor and change over to the FINISHING (Wait) phase.
- The FINISHING (Wait) phase for transmitter nodes is as same as the source node except transmitter nodes which generate K_{extra} number of repair symbols with symbol IDs that have the same number as their own identifier mod N.
- After coming back to the FINISHING(Poll) phase, transmitter nodes transmit POLL message to the next selected children. Transmitter nodes change over from the FINISHING(Poll) phase to the COMPLETED phase once all of its children marked DONE.
- All transmitters listen for symbols of all time slots and maintain a counter, K_{ack} for the number of ACK'd symbols they hear from each of their children.
- Other alternatives for marked children as *DONE* are that the parent marks the child as *DONE* whenever $K_{\text{ack}} \geq K_{\min}$. Also, if a transmitter overhears any of its children transmitting the child's own mod numbers, the child is marked as DONE.

3. *Leaf*

- During the START-UP phase, leaf nodes receive a message. If the message is a POLL message, they send a MORE message with the number of required symbols K_{extra} in the assigned coming slot of its parent. The leaf nodes switch to the COMPLETED phase after receiving $K_{\min}$ and successfully decoding the file.
- During the COMPLETED phase,the leaf node transmit the DONE message in the assigned coming slot of its parent.

The finite state machines developed by Wang for this protocol is shown in Fig4.1.

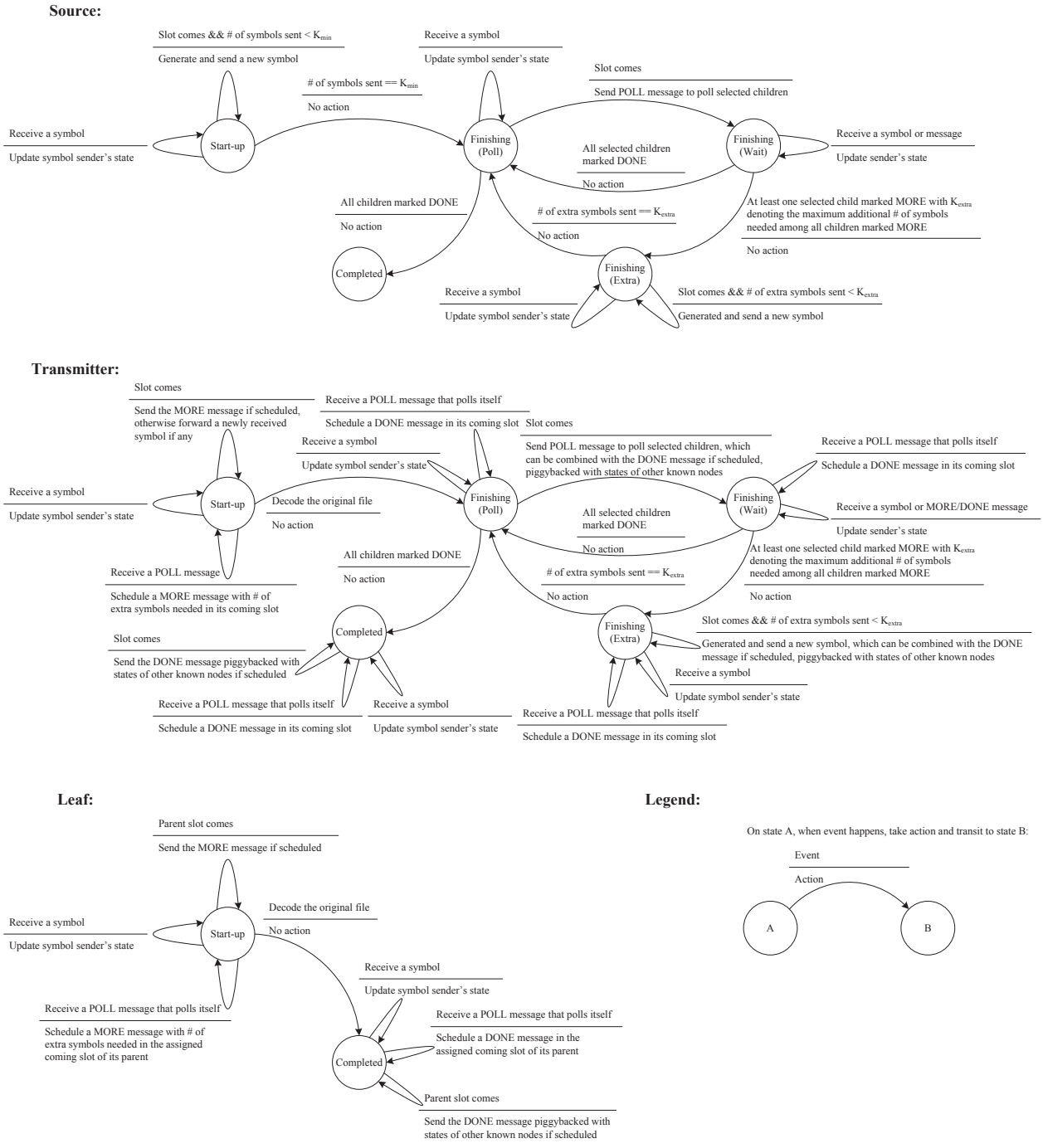


Figure 4.1: Finite State Machines for File Delivery.

CHAPTER 5

RAPTORQ-BASED BROADCAST PROTOCOL IMPLEMENTATION

This chapter discusses the implementation of the RaptorQ-based broadcast protocol in detail. The program has been developed in Ruby. Section 5.1 explains the Ruby interface for the RaptorQ library. Section 5.2 discusses implementation of protocol. Section 5.3 elaborate the message format in the protocol and Section 5.4 explains the testbed setup.

5.1 Ruby Interface for RaptorQ Library

In the implementation of the RaptorQ-based broadcast protocol, we generate desired repair symbols via the RaptorQ libraries. These libraries provide a set of C functions that enables us to build our own RaptorQ encoding and decoding applications.

The Qualcomm-proprietary RaptorQ SDK includes a RaptorQ encoder library and a RaptorQ decoder library, which provide functions to establish RaptorQ encoding and decoding applications. A typical processing flow of the RaptorQ encoder and decoder is shown in Figure 5.1 [19]. The sender application passes source blocks to the RaptorQ encoder to generate intermediate blocks, and intermediate blocks are exploited to generate repair symbols. Then, theses source and repair symbols are passed to the transport layer of a sender which is UDP. The receiver's transport layer receives a set of source and repair symbols corresponding to ESI and passes them to the RaptorQ decoder. When the RaptorQ decoder receives enough either source or repair symbols, the original source blocks can be recovered. The Qualcomm RaptorQ SDK provides a series of functions to perform the above process. The interface is responsible for wrapping these functions such that Ruby script operates the encoding or decoding process. We generate all of repair symbols with ID of transmitter own ID MOD N after the file is reconstructed successfully in the transmitter.

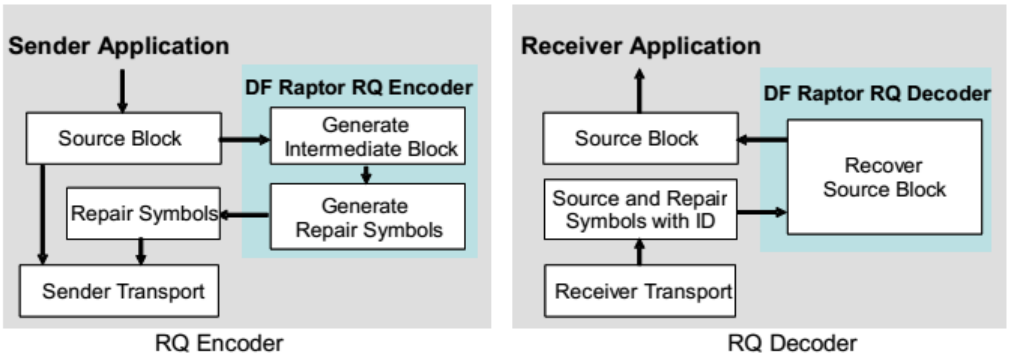


Figure 5.1: Processing Flow for RaptorQ Encoder and RaptorQ Decoder

In the encoding part, the function StringSimpleSend(), was revised as a full RaptorQ encoding function utilizing APIs. This function takes in the source data as a string and returns encoded symbols in separate strings. In addition, we can generate repair symbols with symbol IDs which are ID mod N where ID is an identifier of the current transmitter and N is the number of total transmitters. Variables for this function are as follows:

- **Symbol size**. This is an integer variable that gives the length of every symbol in bytes.
- **File size**. This integer variable specifies the length of the input string (source data) in bytes.
- **Source data pointer**. This is a pointer to the source data which would be encoded.
- **Transfer percent**. This is an integer variable more than 100. It is typically set to be 200, 300, 400, and etc. The meaning of the transfer percent is the percentage protection to be applied to the source data. For instance, a 100 transfer percent stands for a 100 percent protection where all output symbols will be source symbols and there is no repair symbols. Once the transfer percent is 200, it implies that as many repair symbols as source symbols will be generated.

- **Output file name**. This is a string variable determining the output file name. Each source symbol partitioned from the source data will be written into a output file named as filename.src.symbolID, and each generated repair symbol will be written into an output file named name.rep.symbolID.

- **Transmitter number**. This is an string variable determines the transmitter identification number.

- **Total transmitter**. This is an string variable specifies the number of transmitters in the network.

In the decoding part, the function FileSimpleDecode() is modified to achieve the RaptorQ decoding function. This function will recover the original data from a collection of source and repair symbols. We write a collection of source and repair symbols to a file. The recovered data will be returned in a file after a successful decoding, or the function will return a failure status. The variables for this function are as follows:

- **Symbol size**. This integer variable corresponds to the symbol size determined in the encoding process.

- **Input file name**. This is a string variable. All incoming source and repair symbols will be written in a file with a name as same as the input file name.

- **Output file name**. This is also a string variable indicating the output file name of the decoder. The recovered data will be stored into the output file if the decoding process succeeds.

- **File size**. This integer variable specifies the length of the input string (source data) in bytes.

- **Number of extra symbols**. This is an integer variable no less than 0, and is defined as the difference between the number of received symbols and the number of original

source symbols. If the number of source symbols is k and we receive k + 2 symbols, then the number of extra symbols is 2.

In order to execute these two functions in Ruby, we need to use the software development tool named simplified wrapper and interface generator (SWIG) [2]. SWIG is a software development tool that establishes connections between programs written in C and C++ with a variety of high-level programming languages such as Javascript, Perl, PHP, Python, Tcl, and Ruby. The process of wrapping the functions StringSimpleSend() and FileSimpleDecode() along with the RaptorQ encoder and decoder library into dynamic libraries that can be used by Ruby is as follows:

- Creating a C library that includes the RaptorQ encoder and decoder library.
- making an interface file for SWIG which has an *.i extension.
- Producing a desired dynamic library (*.so) with SWIG.

We wrap the RaptorQ library so that our modified function StringSimpleSend() can be called directly in Ruby. We run the following command in the command line:

```
$ swig   ruby StringSimpleSend.i
```

Listing 5.1: Generate a wrap file with SWIG

This will generate a StringSimpleSend_wrap.c, which can be compiled into a shared library used in Ruby. This step will also create an extconf.rb which configures a makefile to generate the extension. Listing 5.2 illustrates commands to create the extension:

```
$ ruby extconf.rb
$ make
$ sudo make install
```

Listing 5.2: Commands to Generate Dynamic Library

A file named StringSimpleSend.so is generated after a successful make. This is a dynamic library containing the StringSimpleSend() function that can be called in Ruby. An example

of using this function in Ruby to encode a olemiss.jpg file with a symbol sizes of 606 is as follows:

```
require './StringSimpleSend'

file = File.open("olemiss.jpg","rb")
StringSimpleSend::StringSimpleSend(606,file.size,file.read,"Encoded", 200)
```

Listing 5.3: The Ruby Code for StringSimpleSend()

In the above example, Ruby passes the source data inFile.read with a size of inFile.size along with a transfer percent of 200. The encoding process is then finished within the C function. Generated source symbols will be written into files Encoded.src0 to Encoded.src267 and generated repair symbols will be written into files Encoded.rep0 to Encoded.rep267 , respectively.

An example of using the FileSimpleDecode() function in Ruby to decode a olemiss.jpg file with 2 extra symbols is as follows:

breakatwhitespace

```
require './FileDecode'

FileDecode::FileSimpleDecode(606, "Output.symbols","Recov.jpg", 158902, 0, 2)
```

Listing 5.4: The Ruby Code for FileSimpleDecode

We pass a collection of source symbols and repair symbols stored in Output.symbols along with necessary variables to the FileSimpleDecode() function. When the execution of the program is finished, we get just a Received file which is exactly our original olemiss.jpg file.

5.2 Implementation of Protocol

Our program as an implementation of RaptorQ-based broadcast protocol is able to distribute a file to a number of nodes over the multihop wireless network. It consists of 3 different main modes including *source*, *transmitter* and *leaf*. The *source* and *transmitter* modes include STARTUP, FINISHING (Poll), FINISHING (Wait), FINISHING (Extra), and COMPLETED phases. The *leaf* mode consists of STARTUP and COMPLETED phases.

Program will switch between modes and corresponding phases using the case statement. In fact, the case statement implements the event and act in the protocol. Ruby does not have a built-in enum type. We create a class and define constants in it to group set of constants logically. Listing 5.5 shows MODE and PHASE class as a follows:

```
class MODE
  Source=0
  Transmitter=1
  Leaf=2
end

class PHASE
  Startup=0
  FinishingPoll=1
  FinishingWait=2
  FinishingExtra=3
  Completed=4
end
```

Listing 5.5: The Ruby Code for MODE and PHASE classes

The Class Neighbor with the Initialize method is created to classify each node instance variable like socket, IP address, port number, send buffer, receive buffer, total symbols, and children IP address. The objects of Neighbor class are accessible through all parts of the program. The types of rcvBuf and sendBuf are string. Listing 5.6 illustrates the Class Neighbor.

```
class Neighbor
   def initialize(ip)
    @ip = ip
    @multiadd = ip
    @local=ip
    @socket = socket
    @req = Array.new
    @rcvBuf = ''
    @sendBuf =''
    @sendBufUni=''
    @symbols=''
    @info=Array.new
    @port=port
  end
attr_accessor :ip, :multiadd, :state, :socket, :rcvBuf, :req, :sendBuf, :port,
    :symbols,:sendBufUni
end
```

Listing 5.6: The Ruby Code for the Class Neighbor

We use Neighbor.new method for each node. In our test implementation, we have 1 source node, 4 transmitter nodes and, 5 leaf nodes. Listing 5.7 shows a code to generate transmitter1Obj for transmitter1 using Neighbor class.

```
transmitter1='130.74.118.60'
portT=6000
s=UDPSocket.new
transmitter1Obj=Neighbor.new(source, portT)
```

Listing 5.7: The Ruby Code for Neighbor Class for Transmitter 1.

Each transmitter node has two types of multicast socket sets, one multicast address for receiving the symbol from its parent and one multicast address for sending the symbol to its children. The code for setting the IP address to send multicast is as follows:

```
socket = UDPSocket.open
socket.setsockopt(Socket::IPPROTO_IP, Socket::IP_TTL, [1].pack('i'))
```

Listing 5.8: The Ruby Code for Send Multicast IP Setting

The code for setting the IP address to receive a multicast message is as follows:

```
Local='0.0.0.0'
MULTIRCV='224.0.0.1'
portS=5000
s=UDPSocket.new
s.bind(local,portS)
ip=IPAddr.new(MULTIRCV).hton + IPAddr.new("0.0.0.0").hton
s.setsockopt(Socket::IPPROTO_IP, Socket::IP_ADD_MEMBERSHIP, ip)
```

Listing 5.9: The Ruby Code for Receive Multicast IP Setting

Three threads are utilized to implement the RaptorQ-based broadcast protocol features. Thread 1 is a receive thread which receives the message through the multicast receive socket and passes it to rcvBuff which is string defined in Neighbor object. For example, a receive thread for the transmitter node 1 is as follows:

```
recieve=Thread.new{
loop  do
          sleep(0.01)
          puts "=============================================="
          puts "Receiving thread is reading from socket."
          msg, sender = s.recvfrom(1024)
          neighbor1Obj.rcvBuf=''
          neighbor1Obj.rcvBuf << msg
end
}
```

Listing 5.10: The Ruby Code for Receive Multicast Thread

The receive thread receives the message from the receiving multicast socket defined in **??**. A msg variable includes the received message and the sender variable consists of the IP address of a sender. We pass the message to the rcvBuf of the transmitter node. Also, we push the IP address of the sender to the children array determined in Neighbor class.

This rcvBuf is evaluated in the main program and different actions occur based on the type of the message and the current state of program. After assessing the receive buffer and determining the action, one message is generated and passed to the send buffer. In the send buffer we determine what to send the reply based on the type of message. If the message is NORM or POLL, the reply is multicast to its children. If the message is MORE or DONE, the reply is unicast to its parent. The send thread for the transmitter 1 is as follows:

```
send=Thread.new{
loop do
        sleep(0.015)
        puts "=============================================="
        puts "Sending thread is writing to socket."
        puts  "Send Thread is #{neighbor1Obj.sendBuf}"
        time,header,tail,symbol= neighbor1Obj.sendBuf.split(/,/)
        if header.to_s=='NORM'
              socket.send(neighbor1Obj.sendBuf , 0, multiadd2, portT3)
        elsif header.to_s=='POLL' then
               socket.send(neighbor1Obj.sendBuf , 0, multiadd2, portT3)
        else
               sUni.send(neighbor1Obj.sendBuf, 0, source, portS)
        end
end
}
```

Listing 5.11: The Ruby Code for Send Multicast Thread

The case statement is used in the main program to implement an action and an event in the protocol. The STARTUP phase for the *source* mode and the *transmitter* mode is different. The source node generates encoded symbols and multicasts them in its time slot. Listing 5.12illustrate the STARTUP phase for *source* mode .

```
file = File.open("olemiss.jpg","rb")
StringSimpleSend::StringSimpleSend(608,file.size,file.read,"RubyOut", 200)
mode=MODE::Source
status=PHASE:: Startup
case mode
     when 0
     case status
     #Startup
          when 0
               if ksent < kmin
                    File.open('/home/pi/RaptorQBroadcast/FileSend/src2/RubyOut.
                        src'+srcSymbol.to_s,'r') do |file|
                    data=file.read(606)
                    info =Time.at(Time.now+1).to_s
                    info += ',' +"NORM"
                    info += ',' +"empty"
                    info += ',' +data
                    srcSymbol+=1
                    end
                    sourceObj.sendBuf=''
                    sourceObj.sendBuf  << info
                    ksent+=1
                    puts ksent
                    mode=MODE::Source
                    status=PHASE::Startup
               else

                     mode=MODE::Source
                     status=PHASE::FinishingPoll
               end
```

Listing 5.12: The Ruby Code for *source* Mode in STARTUP Phase

As it can seen in Listing 5.12, if number of sent symbols, K_{sent}, is less than K_{min}; then, the program reads the source symbol file stored in Raspberry Pi. These source symbol files are already generated by calling the StringSimpleSend function from the RaptorQ library. The program adds a time and a header to the symbol and passes it in the sendBuf. Once the number of K_{sent} is equal to K_{min}, the program switches to the FINISHING (Poll) phase. The *transmitter* mode receives encoded symbols, copies them to the total symbols string, and pass it to the send buffer. We drop the received message according to the loss rate in a receiver

part of each node to simulate an error in the symbol. The loss rate sets independently in each node.

```
case mode
#All for Transmitter
when 1
    case status
        #Startup
        when 0
              if krecv < kmin
                  time,header,tail,symbol= neighbor1Obj.rcvBuf.split(/,/)
                  event=EVENT.class_eval( header.to_s)
                  case event
                  #NORM message
                  when 0
                      randomloss=rand(0.1)
                      if    randomloss >= loss
                            neighbor1Obj.rcvBuf=''
                        else
                            neighbor1Obj.symbols << symbol
                            krecv=krecv+1
                            puts krecv
                            info=Time.at(Time.now+10).to_s
                            info+= comma +"NORM"
                            info+= comma +"empty"
                            info+= comma +symbol
                            neighbor1Obj.sendBuf=''
                            neighbor1Obj.sendBuf << info
                        end
                   when 1
                        time,header,tail,symbol= neighbor1Obj.rcvBuf .split
                            (/,/)
                        neighbor1Obj.symbols << symbol
                        krecv+=1
                        if tail ==neighbor1.to_s
                                kextra=kmin-krecv
                                puts "kextra is #{kextra.to_s}"
                                if kextra > 0
                                      info=Time.at(Time.now+10).to_s
                                      info+= comma +"MORE"
                                      info += comma +kextra.to_s
                                      info+=comma +symbol
                                      neighbor1Obj.sendBuf=''
                                      neighbor1Obj.sendBuf<< info
                                 else

                                 end
                        else
                                 mode=MODE::Transmitter
                                 status=STATUS::Startup
                        end
                        mode=MODE::Transmitter
                        status=STATUS::Startup
                else
                      puts "Generate File"
                      t2=Time.now
                      t=t2-t1
```

```
                puts " time is #{t}"
                fname=fileName+'.symbols'
                f=File.open(fname,"w")
                f.puts  neighbor1Obj.symbols
                f.close
                info =Time.at(Time.now+10).to_s
                info += comma +"DONE"
                info += comma +neighbor1.to_s
                neighbor1Obj.sendBuf=''
                neighbor1Obj.sendBuf << info
                mode=MODE::Transmitter
                status=STATUS::FinishingPoll
        end
```

Listing 5.13: The Ruby Code for *transmitter* MODE in STARTUP phase.

In the example of 5.13, the loss is equal to 0.2.

The source node sends a POLL message in the FINISHING (Poll) phase and switches to the FINISHING (Wait) phase. In this phase, the source node evaluates the received message which can be MORE or DONE. If the message is MORE, the source node switches to the FINISHING (Extra) and transmits K_{extra} a new generated encoded symbols.

5.3 Message Format

All packets transferred between nodes should have the same format. We define a packet format that consists of four parts: header, time, symbol, and tail. The header part is 4 bytes, the time part is 25 bytes, the symbol part is 606 bytes for this example, and the tail is 13 bytes. The header can be NORM, POLL, MORE and DONE. Whenever header is POLL, the tail should be the IP address of the selected children. Also, the tail is the number of the extra needed symbols K_{extra} once the header is MORE. The time in the message is for the scheduling purpose.

5.4 Testbed Setup

The time elapsed to completely deliver the file to all of the nodes was quantified through a series of tests performed on Raspberry Pi platforms, a single-board computer. Two models of a Raspberry Pi are used in our testbed. The model of five Raspberry Pis out of 10 is B+ with

a 900MHz quad-core ARM Cortex-A7 CPU and 1GB RAM and the model of the remaining Raspberry Pis is B with ARMv6 and 512 MB RAM. Figure5.3 shows the Raspberry Pi model 2 B+ released in February 2012. We choose the Raspbian operating system, which is a free operating system based on Debian . These 10 Raspberry Pis are interconnected via Ethernet switch. Our testbed includes 1 source node, 4 number of transmitter nodes and 5 leaf nodes. We use an Ethernet switch instead of wireless dongles for interconnection. Because wireless dongles multicast in the minimum rate which is 2 Mb/s in 802.11n. We simulate the high packet loss characteristic of the wireless multihop network with dropping the symbols based on the loss rate after the symbols received.

Figure 5.2: Raspberry Pi 2 Model B+

Table 5.1 illustrates the IP configuration for all of the nodes in our testbed.

	Recive Multicast IP	Send Multicast IP	IP
S	-	225.0.0.1	130.74.118.225
T_1	225.0.0.1	224.0.0.1	130.74.118.60
T_2	225.0.0.1	224.0.0.2	130.74.118.225
T_3	224.0.0.1	224.0.0.3	130.74.117.150
T_4	224.0.0.1	224.0.0.3	130.74.118.240
T_5	224.0.0.2	-	130.74.119.12
T_6	224.0.0.1	-	130.74.118.202
T_7	224.0.0.3	-	130.74.118.234
T_8	224.0.0.4	-	130.74.118.254
T_9	224.0.0.4	-	130.74.118.69

Table 5.1: IP Configuration in Testbed

Figure 5.3 shows the topology in our testbed.

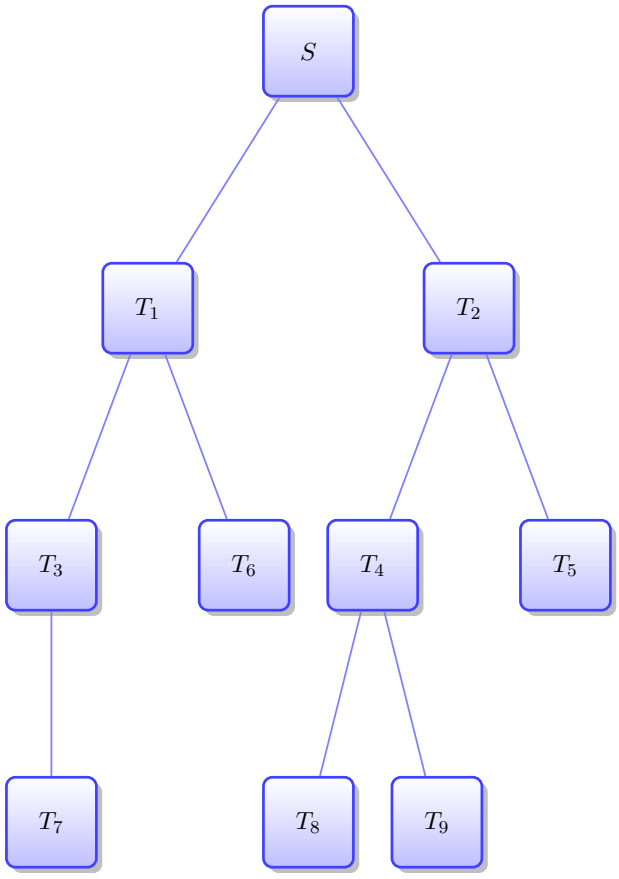


Figure 5.3: Tree Topology for Testbed

The size of the file, size of the encoded symbols and number of the source symbols are 158 kB, 606 byte and, 268 respectively. The loss rate is set independently in each node in our test.

We calculate the number of encode symbols that need to be sent in order to have sufficient symbols to decode the file successfully. Let K_{sent} denote the average number of sent encoded symbols. We consider ℓ as a loss probability. Let K_{min} denote the minimum number of received symbols needed to decode the file. According to RaptorQ properties decoding failure probability of under 10^{-6} is achieved with the 2 extra symbols. Our test file includes 268 source symbols. Thus, the receiver is able to decode the file by receiving $K_{\text{min}} = 270$. We can calculate K_{sent} based on K_{min} and loss probability ℓ as a follow:

$$K_{\text{sent}}\,(1-\ell) = K_{\text{min}} \tag{5.1}$$

Thus for our case K_{sent} is

$$K_{\text{sent}} = \frac{270}{(1-\ell)} \tag{5.2}$$

Table 5.2 illustrates the K_{sent} in different loss probability.

Loss Rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
K_{sent}	300	338	386	450	540	675	900	1350	2700

Table 5.2: The Average Required Number of Sent Symbols as a Function for Loss Rate

5.5 Results

We use the Time.new and sleep methods in Ruby to achieve time synchronization and scheduling. We define the start time using the Time.new method in Ruby. We set the same start time for all of the nodes in the testbed to have the synchronized network. A time slot is allocated for the source node and each of the transmitter nodes using the *Time.parse* and the *sleep* method in Ruby. Also, we set the buffer times for each transmitter node. The source node transmits each 0.01 second. Each transmitter parses its received message time and

adds the buffer time to it. Then, the transmitter node sleeps for start time − recive time + buffer time. The buffer time is 0.1 second for transmitter T_1 and T_2. This means that time slot for T_1 and T_2 is 100 ms after receiving the message from source node. The buffer time for T_3, T_4 is 0.2 second.

The source node transmits 268 source symbols in each 0.01 seconds. The transmitters T_1 and T_2 drop 20% and 30% of the source symbols, respectively. Then, the source node transmits a POLL message to poll transmitter T_1, and the transmitter T_1 sends back a MORE message with a specified K_{extra}. Thus, two time slots are used to send the POLL message and to receives the MORE message. The source node transmits K_{extra} repair symbols with ID symbol of its own ID MOD 5. Again, the transmitter T_1 and T_2 receive the repair symbols based on their loss probabilities. In this step, the source node again sends a POLL message to poll transmitter T_1 and transmitter T_1 sends back a MORE message. This process continues till transmitter T_1 receives 270 the encoded symbols and is marked as DONE. After, the source node sends a POLL message to transmitter T_2. If transmitter T_2 is already DONE, it will send a DONE message to the source node and the source node will be in the COMPLETED phase. Otherwise, the transmitter T_2 sends a MORE message and this process is continued until the transmitter T_2 is marked as DONE. Table 5.3 shows that number of rounds it takes the last symbols to get to transmitters T_1 and T_2 and the source node is marked as COMPLETED.

Number of Sent Symbols in Source Node	Number of Received Symbols in T_1 (Loss Rate=0.2)	Number of Received Symbols in T_2 (Loss Rate=0.3)
268	214	187
1 (POLL T_1)	0	0
0	0 ($K_{\text{extra}} = 56$)	0
56	44	39
1 (POLL T_1)	0	0
0	0 ($K_{\text{extra}} = 12$)	0
12	9	8
1 (POLL T_1)	0	0
0	0 ($K_{\text{extra}} = 4$)	0
4	3	2
1 (POLL T_1)	0	0
0	0 ($K_{\text{extra}} = 1$)	0
1	1 (DONE)	1
1 (POLL T_2)	0	0
0	0	0 ($K_{\text{extra}} = 33$)
33	0	23
1 (POLL T_2)	0	0
0	0	0 ($K_{\text{extra}} = 10$)
10	0	7
1 (POLL T_2)	0	0
0	0	0 ($K_{\text{extra}} = 3$)
3	0	2
1 (POLL T_2)	0	0
0	0	0 ($K_{\text{extra}} = 1$)
1	0	1 (DONE)
COMPLETED		

Table 5.3: The Minimum Round for the Source Node

According to Table 5.3, the total number of time slots required for the source node to deliver the file to transmitters T_1 and T_2 equals to 396. The source node sends each 0.01 second. Also, we have 9 MORE messages and 2 DONE messages from T_1 and T_2. Transmitters T_1 and T_2 also send each 0.01 second and they have 0.1 send buffer time. Thus, the amount of the time elapsed for the last symbol to get to transmitter T_2 and transition the source node to COMPLETED is $(396)(0.01) + (0.1 + (11)(0.01) = 4.17$ second. According to the same process, the completion time for T_1, T_2, T_3 and T_4 is 4.31 s, 4.42 s, 4.49 s and 4.57 s from the start time. Thus, the completion time to reliably deliver the file to all 10 nodes is 4.57 second. Table 5.4 shows the time elapsed to generate the file for each node and

completion time for source and transmitter nodes. We can see that the theoretical result is close to measured one. The transmitter nodes are completed a few millisecond after their parent is completed. For example, the transmitter T_1 and T_2 are marked as completed at 25 ms and 14 ms after the source node is completed. Also, the completion time of T_1 is sooner than T_2 because the loss probability is lower in T_1. Table 5.4 briefly presents the theoretical results for the generation and completion times.

Node	Loss Rate	Generation Time (second)	Completion Time (second)
S	0.15	-	4.17
T_1	0.20	3.38	4.31
T_2	0.30	4.16	4.42
T_3	0.15	3.84	4.49
T_4	0.30	4.29	4.57
T_5	0.20	4.09	-
T_6	0.35	4.20	-
T_7	0.20	4.20	-
T_8	0.15	4.14	-
T_9	0.30	4.30	-

Table 5.4: Theoretical Result for Generation Time and Completion Time

We tested the protocol implementation in the Raspberry Pi testbed. Table 5.5presents the measured results for the generation and completion time. We can see that the theoretical result is so close to practical one. The transmitter nodes are completed a few milliseconds after their parent's completion. For example, transmitters T_1 and T_2 are marked as completed 13 ms and 45 ms after the source node is completed. Also, the completion of T_1 is sooner than T_2 because the loss probability is the lower in T_1.

Node	Loss Rate	Generation Time (second)	Completion Time (second)
S	0.15	-	4.201
T_1	0.20	3.7655	4.4312
T_2	0.30	3.8902	4.654
T_3	0.15	3.9876	4.899
T_4	0.30	4.0011	4.3244
T_5	0.20	4.004	-
T_6	0.35	4.125	-
T_7	0.20	4.32	-
T_8	0.15	4.11	-
T_9	0.30	4.214	-

Table 5.5: Measured Result for Generation Time and Completion Time in Testbed

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis we explained and implemented a RaptorQ-based broadcast protocol for a wireless multihop network. This protocol is a novel protocol designed at the University of Mississippi to facilitate distribution of files over wireless ad hoc networks. The implementation of a RaptorQ-based broadcast protocol is tested in the Raspberry Pi-based testbed with 10 nodes. The results demonstrate that reliable and efficient file distribution is doable in multihop wireless network.

We construct the minimum energy broadcast tree using mixed integer programming, a spanning tree heuristic and a local search heuristic. Moreover, a suitable broadcast tree is derived with a view towards facilitating reliable broadcast using RaptorQ. The resulting tree is implemented using the mixed integer programming with power balance constraints. The numerical results illustrate that there is trade-off between the minimum energy broadcast tree and the balanced-power broadcast tree. The RaptorQ-based broadcast protocol takes advantage of a wireless broadcast nature and the RaptorQ property. According to the RaptorQ characteristic the receiver is able to recover the original file when a sufficient number of distinct symbols is received. Thus, the reliable delivery of the file only depends on the number of received symbols no matter which encoded symbols are. This characteristic of the RaptorQ is very useful in a network with high loss.

The algorithm of the RaptorQ-based broadcast protocol is designed to consecutively distribute the file. Thus, there are not any simultaneous transmissions in our implementation. In future work, we plan to consider simultaneous transmissions. The concurrent transmission will clearly reduce the completion time of the file distribution. We need to consider the

cumulative interference effect of simultaneous transmissions, which increase transmission power.

We have changed the RaptorQ SDK interface to generate all of the source and repair symbols in one pass. This requires substantial memory, and it is not an appropriate in limited resource systems. A more efficient interface from Ruby to the SDK is needed to allow specific symbols be generated by passing the repair symbol ID from the protocol written in Ruby to the RaptorQ library written in C.

Opportunistic reception means that parent node can overhear its child's transmission. Opportunistic reception is able to reduce file delivery time. In fact, the opportunistic reception can also serve as acknowledgment. As future work, we plan to apply this mechanism in our protocol. Once the parent node overhears a child's transmission, it interprets this transmission as a acknowledgment for the corresponding symbol. The potential to reduce or eliminate some phases like Finishing (Poll) exists if the opportunistic reception mechanism is used.

The other concern is high packet loss in a wireless multihop network. The control messages like POLL, MORE and DONE are transmitted in the link with high loss rate. Thus, these messages may be dropped, and a parent or a child node may not receive them. We need to add a mechanism in our protocol to overcome this issue.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Altinkemer, K., F. Salman, and P. Bellur (2004), Solving the minimum energy broadcasting problem in ad hoc wireless networks by integer programming, in *Proceedings of the Second Workshop on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks, WiOpt04.*, pp. 48–54.

[2] Beazley, D. M., et al. (2005), Swig-1.3 documentation, *Tech. rep.*, Technical Report, University of Chicago.

[3] Cagalj, M., J.-P. Hubaux, and C. Enz (2002), Minimum-Energy Broadcast in All-Wireless Networks: NP-Completeness and Distribution Issues, doi: http://doi.acm.org/10.1145/570645.570667.

[4] Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2001), *Introduction to Algorithms*, MIT Press.

[5] Das, A. K., R. J. Marks, M. El-Sharkawi, P. Arabshahi, and A. Gray (2003), A cluster-merge algorithm for solving the minimum power broadcast problem in large scale wireless networks, in *Proceedings of Military Communications Conference, MILCOM'03.*, vol. 1, pp. 416–421, IEEE.

[6] Das, A. K., R. J. Marks, M. El-Sharkawi, P. Arabshahi, and A. Gray (2003), Minimum power broadcast trees for wireless networks: integer programming formulations, in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 2, pp. 1001–1010, IEEE.

[7] Guo, S., and O. Yang (2003), Minimum-energy broadcast routing in wireless multi-hop networks, in *Proceedings of Performance, Computing, and Communications Conference, IEEE 2003*, pp. 273–280, IEEE.

[8] Guo, S., and O. W. Yang (2007), Energy-aware multicasting in wireless ad hoc networks: A survey and discussion, *Computer Communications*, 30(9), 2129–2148, doi: 10.1016/j.comcom.2007.04.006.

[9] Kang, I., and R. Poovendran (2003), A comparison of power-efficient broadcast routing algorithms, doi:10.1109/GLOCOM.2003.1258267.

[10] Kang, I., and R. Poovendran (2004), Broadcast with heterogeneous node capability [wireless ad hoc or sensor networks], in *Proceedings of Global Telecommunications Conference, GLOBECOM'04.*, vol. 6, pp. 4114–4119.

[11] Kang, I., and R. Poovendran (2004), Cobra: Center-oriented broadcast routing algorithms for wireless ad hoc networks, *Tech. rep.*, DTIC Document.

[12] Kang, I., and R. Poovendran (2005), Iterated local optimization for minimum energy broadcast, in *Proceedings of Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, WIOPT 2005.*, pp. 332–341, IEEE.

[13] Kang, I. K. I., and R. Poovendran (2003), A novel power-efficient broadcast routing algorithm exploiting broadcast efficiency, *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*, 5, doi:10.1109/VETECF.2003.1286159.

[14] Li, D., X. Jia, and H. Liu (2004), Energy efficient broadcast routing in static ad hoc wireless networks, *IEEE Transactions on Mobile Computing*, 3(2), 144–151, doi:10.1109/TMC.2004.10.

[15] Liang, W. (2002), Constructing minimum-energy broadcast trees in wireless ad hoc networks, in *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pp. 112–122, ACM.

[16] Luby, M., A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder (2011), Raptorq forward error correction scheme for object delivery (rfc 6330), *IETF Request For Comments*.

[17] Montemanni, R., L. M. Gambardella, and A. K. Das (2005), The minimum power broadcast problem in wireless networks: a simulated annealing approach, in *Wireless Communications and Networking Conference*, vol. 4, pp. 2057–2062, doi:10.1109/WCNC.2005.1424835.

[18] Puducheri, S., J. Kliewer, and T. E. Fuja (2007), The design and performance of distributed LT codes, *IEEE Transactions on Information Theory*, 53(10), 3740–3754, doi:10.1109/TIT.2007.904982.

[19] Shokrollahi, A. (2006), Raptor codes, *IEEE Transactions on Information Theory*, 52(6), 2551–2567, doi:10.1109/TIT.2006.874390.

[20] Singh, A., and W. N. Bhukya (2011), A hybrid genetic algorithm for the minimum energy broadcast problem in wireless ad hoc networks, *Applied Soft Computing*, 11(1), 667–674, doi:10.1016/j.asoc.2009.12.027.

[21] Wieselthier, J. E., G. D. Nguyen, and A. Ephremides (2000), On the construction of energy-efficient broadcast and multicast trees in wireless networks, in *Proceedings of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2000*, vol. 2, pp. 585–594, IEEE.

[22] Wolf, S., and P. Merz (2008), Evolutionary local search for the minimum energy broadcast problem, in *Evolutionary Computation in Combinatorial Optimization*, pp. 61–72, Springer.

[23] Wu, X., X. Wang, and R. Liu (2008), Solving minimum power broadcast problem in wireless ad-hoc networks using genetic algorithm, in *Proceedings of Communication Networks and Services Research Conference, CNSR 2008.*, pp. 203–207, IEEE.

[24] Yuan, D. (2005), Computing Optimal or Near-Optimal Trees for Minimum-Energy Broadcasting in Wireless Networks, in *Proceedings of IEEE WiOpt '05: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pp. 223–331, doi: 10.1109/WIOPT.2005.16.

VITA

Roya Lotfi received her Bachelor of Engineering degree in Electrical Engineering in 2003 at Azad University, Bushehr, Iran. In August 2012, she joined the Department of Electrical Engineering at the University of Mississippi as a graduate student emphasizing in Telecommunications. Her research interest includes reliable file delivery protocols in wireless networks, optimization of constructing broadcast trees, wireless mesh network and network programming. Since August 2013, she has been a teaching assistant in control system and computer network class at the University of Mississippi.