Data Collection in Wireless Sensor Networks

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Aram Mohammed Rasul
Department of Computer Science
University of Leicester

September 2015

Data Collection in Wireless Sensor Networks

Abstract

This thesis is principally concerned with efficient energy consumption in wireless sensor networks from two distinct aspects from a theoretical point of view.

The thesis addresses the issue of reducing idle listening states in a restricted tree topology to minimise energy consumption by proposing an optimisation technique: the extra-bit technique. This thesis also focuses on showing lower bounds on the optimal schedule length, which are derived for some special cases of the tree, such as a single chain, balanced chains, imbalanced chains, three and four level k-ary trees and Rhizome trees. Then, we propose an algorithm which can exactly match the lower bound for a single chain, balanced chains and Rhizome trees individually and which is a few steps away from the optimal solution for imbalanced chains. Finally, we propose the use of two frequencies to further save energy and minimize latency.

Recent research has shown that significant energy improvements can be achieved in WSNs by exploiting a mobile sink for data collection via single hop communications. A mobile sink approaches the transmission range of sensors to receive their data and deposit the data at the base station. The thesis, as a second problem, focuses on the design issues of an energy efficient restricted tour construction for sink mobility. We propose two different techniques. The first one is heuristic and uses a criterion based on maximum coverage and minimum energy consumption called the "max-ratio". Although its time complexity is polynomial, this heuristic algorithm cannot always produce a good solution. As a result, we propose the second algorithm. Despite the time complexity of the second algorithm being pseudo polynomial, the optimal solution can be found if one exists. For each algorithm mentioned, two scenarios are taken into account with regard to the transmission. In the first scenario, one assumes that there is no upper bound on the transmission range while in the second setting the nodes can adjust their transmission range between 0 and the maximum range. The algorithms have been implemented and simulated in Matlab.

Acknowledgements

I am greatly indebted to my supervisor, Professor Thomas Erlebach, for all of his encouragement, guidance, dedication, direction, advice, availability and invaluable support, throughout my PhD studies at the University of Leicester. I believe that without him, none of this research would have been achieved. Thank you very much for showing tremendous patience in reading and evaluating my work. Moreover, I am grateful to Professor Erlebach for providing me with an opportunity to work with him. Working under his supervision has been a very enriching and memorable experience. I would also like to thank my second supervisor, Dr Stanley P.Y. Fung, and Dr Fer-Jan de Vries, for their discussion and feedback in the yearly thesis committee.

A special thanks to Professor Rick Thomas; his suggestions and discussions have always been of great help. I would like to express my thanks to all staff members in the department for their respectfulness and friendship, and I am grateful to my colleagues for the lovely and enjoyable time that we spent together in the department. My special thanks also go to my government for funding my studies in the UK. Of course, I am grateful to my family for their love and support at all stages of my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter addresses the motivation, design goals, challenges, contributions of the thesis and its outline.

1.1 Motivation

Wireless communication, such as cellular network, is generally based on planned infrastructure or pre-deployment of infrastructure for communication. However, due to recent diverse applications of wireless networks in certain environments (i.e., earthquake hit-places, hostile zones, battlefield, volcano prone areas) infrastructure is not available. A demand for self-arrangement, independence (infrastructure-less), adaptability and cost reduction has increased. In the aforementioned situation, An ad-hoc wireless sensor network (WSN) is the only available solution to respond to these issues such as providing connectivity among nodes in the absence of infrastructure [23]. In addition, this network yields a new promising scheme to extract and obtain data from the monitored environment. These nice properties motivated us to focus on this area.

Technological advances have led to the invention and development of small wireless devices, including sensors [72] which have the capability of sensing, processing, computing and communication. Typically, WSNs are comprised of hundreds or thousands of sensors [73], which are dispersed either randomly in an inhospitable terrain or deployed deterministically in a specific area of interest without any pre-existing infrastructure, and many objectives are associated with them. During either

deployment, sensors configure themselves to form the network in an ad-hoc fashion, which is a common mode of operation in WSNs [72], and communication happens either by means of single-hop or multi-hop dissemination, depending on the distance between the sensors [72, 12, 73]. Correspondingly, WSNs have attracted considerable attention in the research community for their own numerous applications in various areas [12], for instance, military applications, fire detection, healthcare and environmental or habitat monitoring [73, 58, 48].

Sensors are small devices that are resource-constrained as having limited power, small memory, relatively slow processors or small transceivers [73]. They are usually powered by batteries, which must be either replaced or replenished when depleted. For some applications, neither option is functional, especially when they are deployed in exotic environments wherein human intervention is not allowed or when sensors are deployed densely. Energy efficiency, consequently, becomes one of the most challenging issues in WSNs and is considered as a key factor for extending the network lifetime. Therefore, many algorithms and techniques have been proposed and designed from different perspectives to utilise this limited energy budget more efficiently, in order to improve their lifetime and operation as much as possible [12].

Data collection is one of the fundamental operations in WSNs via tree topology. Each of these sensors is capable of sensing the monitoring area and routing data back to a collection point called a sink or base station to achieve an application goal. Once the sink has received the data, it makes an appropriate decision based on the requirements of the application. The sink is considered as a powerful node to which all data is sent and through which the WSN interacts with the outside world. Furthermore, reporting data from sensors to the sink naturally forms a many to one traffic paradigm in WSNs.

In the existing literature, there are mainly two modes of data collections which have been mentioned. These are data collection with in-network processing (aggregation) and data collection without aggregation. In some applications, such as when the sink needs to determine the maximum temperature in a specific area, it is not important that each data packet is delivered to the sink individually. In such a case, the sink does not care about all the data; if all nodes send their data to the base station, the result is an over-consumption of energy due to the number of trans-

missions. Therefore, the aggregation strategy can be used to reduce total packet transmission by performing local calculations at each parent node and forwarding only aggregated values to the sink [58].

For instance, consider a scenario where seven nodes are deployed to monitor an agricultural area as shown in Figure 1.1. It is a data collection tree rooted at the sink (S). Consider that this set $\{37, 30, 27, 22, 19, 25, 15\}$ represents the values for each node $\{G, F, E, D, C, B, A\}$ and the sink asks for the maximum temperature. Therefore, when each parent receives packets from its children in addition to its own packet, some calculation needs to be performed to determine a candidate for the maximum value and forward it to its parent until the sink receives the maximum value. Thereby, node C receives $\{27, 30, 37\}$ values from its children in addition to its own value 19. Then, it chooses the maximum value which is 37 and forwards it to its parent A. Moreover, the node B has two values $\{22, 25\}$ after receiving the value 22 from its child D, it forwards the maximum of them which is 25 to its parent. Similarly, node A has three values $\{15, 25, 37\}$ after receiving two values $\{25, 37\}$ from its children B and C. It therefore calculates the maximum value, which is 37, and forwards it to the sink S.

On the other hand, there are many applications in which all packets are individually important. For instance, when sensors are deployed for structural health monitoring or leak detection, packets need to be collected from all sensors (collection without aggregation) to learn the conditions at each individual sensor location, otherwise the exercise could be a failure [37] or lead to catastrophes. It has been stated that the case of data collection without aggregation is a more challenging problem compared to the data collection with in-network processing (aggregation). Hence, this thesis investigates the case of the former.

In addition, there are two main methods that are used to order data collections [38]. In the first method, triggering of data collection can be performed by the base station; this process can be fulfilled by sending the query from the sink to either a sub or whole area for asking for the data. This kind of data collection has a great impact on energy conservation. This is because sensors do not need to send their data continually to the sink; instead they send data on demand. Asking the sink for the maximum temperature from the sensors is an example of this method. However,

data collection can also be performed by sensors according to a second method. For example, when sensors are used for event detection (e.g. intruder detection in military surveillance, fire detection or habitat monitoring), data is promptly forwarded to the sink without being asked by the sink. Accordingly, different objectives are associated with data collection. For instance, in the first method energy efficiency is the main priority while fast data collection is the main concern in the second method.

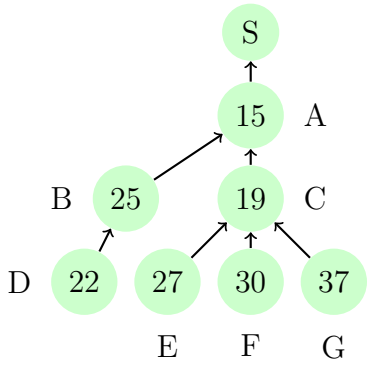


Fig. 1.1 Data collection, maximum temperature.

1.2 Challenges, Objective and Solutions

Despite an immense effort by the research community, limited energy sources remain a big challenge in WSNs, especially when they are deployed in harsh areas or scattered densely; battery replacement or replenishment is unlikely, hence their energy should be utilised efficiently to prolong their lifespan as much as possible. Another challenging issue is their topology. After their deployment and topology construction, the topology is more likely to change due to node failure. Therefore, reconstructing topology should be energy efficient. Constructing an energy saving tour for deployed static sensors is another severe challenging area in WSNs. Limited memory is an additional challenging issue in WSNs. As the sensor nodes are small devices, battery powered and having a small memory, the mechanism of their data collection should be efficient to avoid buffer overflow. The overall aim of this thesis is to design a productive energy saving strategy for WSNs. To achieve this, two main topics are investigated in this thesis. Namely, idle listening state and sink mobility with single hop communication since there is a gap for research for these

two areas as we will be explaining in the literature review.

The following are the two main strategies that should be employed to improve energy consumption in WSNs:

1. Using a power management mechanism is a very promising technique to achieve energy saving in WSNs. Since the transceivers are the most energy consuming part of the sensors, when sensors do not need to take part in the communication, most circuitry parts of the sensors can be shut down or put in a sleep mode to save energy. In other words, energy saving can be achieved by alternating between active and sleep mode. Notably, another way in which a substantial amount of energy is wasted is in idle listening state. Idle listening occurs when a node is listening to the channel to receive a packet but there is the possibility that the sending node does not have data and remains silent [9, 38, 78]. Idle state is comparable to receive state in terms of energy consumption, and the detailed explanation for that will be given in Chapter 3. The thesis therefore devotes a main part of this dissertation to dealing with this issue (Chapters 3 and 4).

2. Sink mobility with single hop communication is another method to achieve energy saving in WSNs. There are certain issues associated with static networks. Firstly, the many-to-one scheme is a more common way of constructing a network and collecting data from sensors. However, some of the nodes (especially those which are close to the base station) become a relay for others to forward their packets toward the sink, and they drain their energy very swiftly [47]. As a result, the entire network is paralysed while nodes that are far from the sink still functional but cannot forward their packets to the sink. Secondly, sometimes there are some disconnected parts that cannot communicate with each other and construct a global network. The above mentioned points are strong reasons for the allocation of the second part of this thesis (Chapter 5) to data collection with a mobile sink.

1.3 Contributions

The main contributions are as follows: The first contribution is in Chapter 3, where we improve upon a method proposed by Zhao and Tang [81, 82]. They show that in the setting when only some nodes possess data there is a restriction at which each parent node must listen to its children before making a transmission. Two main issues arise in this scenario. Firstly, there are several idle listening states which cause severe energy consumption. Secondly, the first situation leads to high latency. As a result, they proposed a heuristic algorithm to reduce idle listening states. The method by Zhao and Tang produces successive-slot schedules. It was observed that there is still room to optimise their technique. Therefore, an optimisation technique is proposed: the extra-bit technique. In our technique, each packet is associated with one extra bit (0/1) that alerts the parent node whether or not more packets are coming. Based on that extra bit, the parent node can either continue in listening or turn off its radio toward its child node. Therefore, idle listening is further reduced and this also minimises latency. Next, it is proven that the optimal number of time slots for data collection in a chain using successive-slot or extra-bit schedules is $4N - 6$, where $N \geq 3$ is the number of nodes in the network excluding the sink. Then, it is shown how to calculate the expected amount of idle listening for extra-bit schedules and successive-slot schedules in chains and trees where each node has data with a fixed probability, and it is demonstrated that the expected amount of idle listening is significantly smaller with the extra-bit technique. A version of this chapter has already been published as a conference paper [62].

Chapter 4 contains the second contribution. We derive lower bounds on the optimal schedule length for some other special cases of the tree including balanced chains (multi-chains), unbalanced chains, balanced binary tree, four level balanced k-ary tree and Rhizome tree. Moreover, the pseudo code is designed and shown for balanced multi-chains (multi-lines) and Rhizome tree. The proposed algorithms for balanced chains and Rhizome tree match the lower bound, whereas the proposed algorithm for unbalanced chains requires at most 5 steps more than optimal. Furthermore, due to the nature of communication, each node must go through three states: transmit, receive and idle state. The use of two frequencies is proposed, with multiple antennae to reduce the need for idle state among nodes. Then, the optimal

schedule is derived again for all cases except for the unbalanced chains. Part of this chapter has been submitted and accepted to a journal [61].

The last contribution is in Chapter 5, where energy saving is attempted via a different technique (i.e., sink mobility). Due to non-uniform energy depletion among the nodes in the many-to-one pattern or sparse deployment of the sensors in different zones that cannot form the complete network, sink mobility has recently encouraged the research community to remedy these problems; therefore, chapter 5 is devoted to addressing the issue of sink mobility for data collection from static sensors to enhance the life span of the network by minimising the total energy consumption. In order to achieve this, the problem is formally defined and it is shown that it is NP-complete. As a result, two different algorithms are proposed. The first algorithm is a heuristic and named max-ratio, and the second algorithm uses the dynamic programming technique. In addition, two scenarios for each algorithm are considered. In the first scenario, there is no restriction on the transmission, whereas in the second setting the maximum transmission range is determined. At the end, simulations in Matlab have been performed to show the impact of the proposed techniques on energy saving. A version of this chapter has already been published as a conference paper [63].

1.4 Thesis Outline

The thesis has been written in six chapters. Chapter 2 provides some basics about graphs, complexity theory, types of interference and a literature review. In Chapter 3, we discuss and propose an optimization technique to mitigate the idle listening problem in a constrained topology and compare our results with the other proposed techniques. Chapter 4 extends the proposed technique of Chapter 3 towards the more general structure of balanced multi-chains, unbalanced multi-chains, k-ary tree and Rhizome tree. In Chapter 5, we introduce and bring the sink mobility into a sensor network to collect data. We propose two algorithms for the mobile sink to specify its trajectory for data collection and minimize the total energy consumption. We also consider two scenarios for each algorithm. Moreover, the simulation results are presented and compared with the label covering problem which we will describe

in the literature review and also in chapter 5. Chapter 6 includes a summary of our contributions and specifies some future research directions.

Chapter 2

Preliminaries and Related Work

This chapter presents the background information that is necessary to understand some essential terminologies and notations throughout this thesis. First, we define some terminologies of graph theory; we subsequently explain the interference model, and finally complexity theory.

2.1 Graphs

Graph theory is a field in mathematics wherein many real world situations from various different fields, including communication, scheduling tasks and real-world networks, can be described, modelled, analysed and represented graphically to help us to easily understand many properties of the problems. As a result, this field gained popularity and became an attractive area for the research community.

A graph simply is a collection V of vertices (nodes) and E of edges that is represented as $G = (V, E)$. $|V|$ expresses the total number of vertices and $|E|$ indicates the total number of edges. Vertices are connected to each other by means of edges. In particular, each edge connects exactly two vertices. For instance, an edge $e = (v, u)$ is said to be incident with vertices v, u, where $v, u \in V$ and $e \in E$. This implies that they are connected (adjacent or neighbouring). A graph is called an undirected graph if its edges have no direction. In other words, the edge $e = (u, v)$ is equivalent to $e = (v, u)$, i.e., they are unordered pairs [41], meaning that they are symmetrical.

A walk $W = \{v_0, e_1, v_1, \ldots, v_{i-1}, e_i, v_i, \ldots, e_n, v_n\}$ is a sequence of vertices and

edges in a graph, such that each edge $e_i = (v_{i-1}, v_i)$ connects two vertices in the sequence, for $1 \leqslant i \leqslant n$. The number of traversed edges in the sequence indicates the length of the walk, i.e, value n is the length of the walk. A walk is called a path P if both the traversed vertices and edges are distinct (i.e., there are no repeated vertices and edges), with the exception of the possibility that the initial and final vertices are the same. Then, it is said to be a cycle. In addition, a graph G is connected if there is a walk between every pair of vertices, as shown in Figure 2.1 [16, 32, 7].

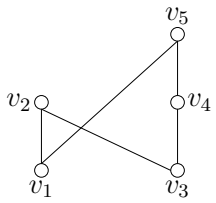


Fig. 2.1 Connected graph, 5 nodes.

A complete graph is a graph in which a unique edge connects every pair of distinct vertices, i.e., graph $G = (V, E)$ is a complete graph if for all $v, u \in V$ with $v \neq u$ there is an edge between them [41], see Figure 2.2. An acyclic graph is one with no cycles. A tree is a connected graph that has no cycle. Trees are important to study due to their common and widespread usage in diverse fields and applications.

A directed graph (or digraph) is a set of vertices and edges in which all the edges are directed (have direction), and each edge connects an ordered pair of vertices. That is, each edge points from one vertex to another. In other words, the edge $e = (u, v)$ is not equivalent to $e = (v, u)$ [42, 32]. Traffic flow, such as with airlines, trains and cars are some applications of directed graphs, in which specifying the direction is quite essential to avoiding collisions. Besides adding a direction to an edge, each edge can also be associated a weight, which often represents some kind of cost or distance depending on the application. A weight can be represented by a weight function as $w : E \longrightarrow \mathbb{R}$. This extension is a natural one when modelling real-world networks as graphs. For example, when modelling a railway network as a graph, railway stations are naturally represented by vertices, whereas two adjacent stations are connected by means of an edge. We then assign a weight to an edge representing the distance between those two stations. In our model in chapter 5, we represent the problem as a weighted graph where each edge is associated with

a value that represents the distance between two points [42]. In some applications, due to their own restrictions, we cannot pass through all the vertices and edges of the graph. Hence, some of the vertices or edges should be skipped to achieve the application goal. Our problem in chapter 5 is an example of such an application. We therefore should be familiar with another interesting term of the graph theory called a sub-graph. The graph $G' = (V', E')$ is defined as a sub-graph of $G = (V, E)$ if G' consists of a subset of vertices and edges of G. That is, $V' \subseteq V$ and $E' \subseteq E$ [42, 41, 7] as shown in Figure 2.3.

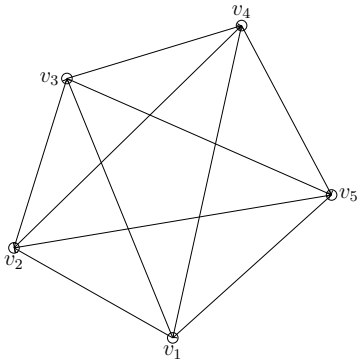


Fig. 2.2 Complete graph G, 5 nodes.

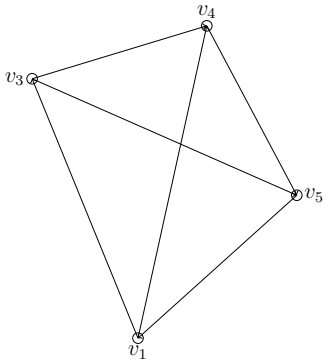


Fig. 2.3 Subgraph G' of graph G in Figure 2.2.

One version of graph called a unit disk graph (UDG) is mainly considered to study and model wireless sensor networks. A unit disk graph is a graph wherein the nodes, which are in the Euclidean plane, all have uniform unit radii (homogeneous) [16, 14]. Moreover two nodes can communicate with each other (adjacent) if the center of one disk lies within the radius (transmission range) of the second disk or vice versa [71] as shown in Figure 2.4. However this model is too idealistic and does not account for the existence of obstacles (it is not accurate).

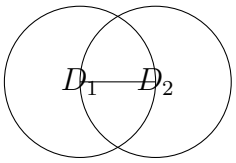


Fig. 2.4 Unit disk graph.

2.2 Interference Model

Successfully receiving and decoding a signal from the intended transmitter is a great concern and challenging task in wireless communications. To address this issue, therefore, two models of interference are widely considered in the literature: the physical interference model and the protocol interference model.

In the physical interference model, which is also called the signal-to-interference-and-noise-ratio (SINR) model, the intended receiver node j can successfully receive a signal (correctly decode it) from the intended transmitter node i, if and only if, the SINR at node j is above its predefined threshold β. When the intended transmitter node i transmits to node j, the simultaneous transmissions of other nodes whose signals reach the node j are considered as interference at node j. Hence, a transmission succeeds only if the received signal strength, divided by the total simultaneous transmissions of other nodes' strength, plus the noise (SINR), is above some predefined threshold [45], and this is mathematically expressed as follows:

$$SINR = \frac{\frac{p_i}{d(i,j)^\alpha}}{WN + \sum_{v \in V, v \neq i} \frac{p_v}{d(v,j)^\alpha}} \geq \beta \tag{2.1}$$

Here p_i is the transmission power of node i, $d(i,j)$ is the distance between node i and j; α is the path loss exponent, $2 \leqslant \alpha \leqslant 6$, depending on the environment [65]. The sum in the denominator calculates the total transmission power of the simultaneous nodes with node i and WN is the ambient noise. Due to the high complexity in the calculation of the physical interference model, researchers paid attention to a simpler model, which is called the protocol interference model. In the protocol interference model, the intended receiver node j can only receive a signal correctly from transmitter node i if and only if node j is within the transmission range of node i and outside the simultaneous interference range of other nodes. That is, two nodes can communicate only if they are within the transmission range of each other [10].

It can be observed that in the physical interference model the intended receiver can still decode the signal correctly if its received signal is above its threshold. However, in the protocol interference model, correct receiving of the signal at the intended receiver occurs only when it falls inside transmission range of the intended transmitter and outside the interference ranges of all other simultaneously transmitting nodes. Throughout this thesis, we consider the protocol interference model due to its simplicity. A detailed explanation of the interference models and signal detection are beyond the scope of this thesis.

2.3 Complexity Theory

Although the area of computational complexity is somewhat complicated and we cannot cover the exposition of it in this thesis, we try to briefly define and explain some terminologies which give an intuitive understanding of the complexity of the theory and the types of the problems which are relevant to the work presented in this thesis.

In its simplest form, an algorithm is a well-defined procedure that is composed of a finite number, and unambiguous sequence, of instructions for solving a computational problem. In other words, the algorithm is the list of computational steps that takes the set of input values of the problem, transforms them and produces the desired output [15]. Making an algorithm as efficient as possible in terms of computational resources (time, memory, bandwidth etc.) [41] is the main goal of the algorithm designers, and in certain applications such an efficient algorithm is extremely important.

There can be several algorithms for solving the same problem, however we should choose the most efficient one (a fast one) for solving the problem [15]. Namely, there are several possible solutions to a problem. When we want to measure the efficiency of two algorithms for the same problem, we compare their number of steps regardless of the hardware or software environment (machine independent model). Therefore, mathematicians and scientists designed a uniform measurement to measure the efficiency of algorithms, and use a special notation as we will explain in the next paragraph.

In general, most of the references are mainly interested in determining the time complexity (running time) to measure the efficiency of an algorithm. Hence, the time complexity of an algorithm is the number of computational steps required by the algorithm to process the input and produce the output. Strictly speaking, the efficiency of an algorithm is a measure of its run-time proportional to the number of operations as a function of the input size of the problem. Thus, the input size of the problem is a key factor to determine the efficiency of an algorithm. So when we analyse an algorithm, we mainly refer to the maximum number of steps (worst-case running time) required by the algorithm which is also called an upper bound on the time complexity and denoted using the symbol O (read as big-Oh-notation). Whereas, the minimum number of steps of the algorithm is called the best case time complexity. Furthermore, A lower bound is a function such that no algorithm can have smaller running time than specified by that function and denoted by Ω (big-Omega). It is worth pointing out that one describes the running time taken by an algorithm as a function of the input size; this is because the running time increases with the input size of the problem [15]. For instance, the running time of sorting $n = 10$ numbers is smaller than $n = 100$ numbers.

In general, we consider two different functions $f(n)$ and $g(n)$ for the same algorithm. Then, we say that $f(n)$ is $O(g(n))$, if for all sufficiently large input n ($n \longrightarrow \infty$), $f(n)$ is bounded by a constant multiple of $g(n)$. This means that the growth rate of $f(n)$ is at most $O(g(n))$. In other words, $f(n) \in O(g(n))$ if there exists some n_0, $c > 0$ such that $f(n) \leqslant c \cdot g(n)$ for all values $n \geq n_0$, where c is a constant which is independent of the input size, and n_0 is a crossing point or threshold where function $g(n)$ overtakes $f(n)$. This is called asymptotic complexity analysis.

Suppose there is an algorithm whose running time is represented by a function $f(n) = 3n^3 + 2n + 1$ on every input size n (for an array of n numbers, its input size is n). Then we say that the time complexity (running time in terms of steps) of this algorithm is $O(n^3)$. In other words, we account only for the higher order term and discard the lower order terms, because when the input size is large, the effect of the lower order terms are small and insignificant, such that they can be ignored (i.e., the higher order term overtakes the lower order terms).

We can say that the running time of an algorithm is polynomial if it requires at most $O(n^c)$ steps, where n is the input size of the problem and c some constant. In other words, a problem is polynomial time solvable if there is an algorithm that can correctly solve it in $O(n^c)$ time. Conversely, we say that the running time of an algorithm is exponential if its running time is $O(c^{f(n)})$ where c is a constant and $f(n)$ is some polynomial function of n [31]. Notably, an exponential algorithm can still be desirable for a small input size.

We should now turn our focus to some other notations. A decision problem is a problem for which there are only two possible answers, either yes or no, as an output at the end; e.g., if there is an array $A = \{a_1, a_2, ..., a_n\}$ which asks whether or not it contains the value of k; after scanning all the elements, the answer is yes if this value of k is found; otherwise it is no.

The complexity class P refers to the set of all decision problems that are polynomial time solvable (efficiently solved) under resource constraints (a deterministic machine). That is, the set of all problems that admit a polynomial time algorithm to solve it. On the other hand, the complexity class NP aims to target the set of all decision problems that are polynomial time solvable on a non-deterministic machine (an unrealistic machine) or equivalently whose solutions can be efficiently verified on a deterministic machine.

The non-deterministic machine is a powerful machine such that, whenever there are multiple choices for executing a program, it can follow all the possible choices simultaneously instead of following each one iteratively. Additionally, all class P problems can be easily solved polynomially on a non-deterministic machine. Therefore, $P \subseteq NP$, and the famous remaining unsolved question is whether or not the two classes P and NP are the same. Many scientists believe that $P \neq NP$.

There are a set of particular problems in NP, which are called NP-complete problems. They essentially cannot be solved in polynomial time (a polynomial time algorithm is not expected in general instances) unless $P = NP$. It is quite difficult to prove a problem to be NP-complete from scratch without robust knowledge of that area, therefore most of the proofs for a new problem depend on the already known NP-complete problems via a process called reduction. One of the earliest problems that was proved to be NP-complete is satisfiability by the Cook theorem

and most of the proofs for other NP-complete problems are built upon it.

We use a reduction (finding a relationship between problems) to prove that a new problem is NP-complete. For instance, given two problems A and B, we suppose that we already know that A is NP-complete and $B \in NP$. Thus, if we can show that any instance of A can be transformed polynomially to an instance of B, it is said that A is polynomially reducible to B. Accordingly, if there is a polynomial time algorithm to solve B, then it must also solve A or vice versa. Thus, B is NP-complete too [41, 28, 15, 18]. Furthermore, if there is a problem C where B is polynomially reducible to C, but we do not know how to conversely reduce C to B or if $C \in NP$, then we do not just say C is NP-complete, instead we say that it is NP-hard [41]. In other words, NP-hard problems are problems to which an NP-complete problem can be polynomially reduced, they don't need to be in NP.

There are several strategies that can be used to approximately design a polynomial time algorithm to solve and deal with NP-complete problems. Namely, we can use a more clever way to design a polynomial time algorithm without an exhaustive search (naive exponential computations). Some of these techniques are:

1. An approximation algorithm is a strategy that tells us how far the proposed solution is from an optimal solution in the worst case scenario, which is also called an approximation ratio (by which factor the proposed solution is away from the optimal).

2. A heuristic algorithm is a useful strategy that proposes a good polynomial time algorithm without any guarantee that it is the optimal one, or that it will always find a solution.

3. Additionally, the dynamic programming technique is another powerful technique that is used to deal with hard problems. It first finds the optimal solutions to sub problems and next from these sub problems computes the optimal solution to the original problem (i.e., solves the original recursively from the sub problems [41, 18]. The running time of a dynamic programming algorithm sometimes is high (depends on the problem), however it is smarter than an exhaustive search. This is the reason that the complexity of the dynamic programming algorithm is called pseudo polynomial.

We used both a heuristic algorithm and dynamic programming techniques to find the solution to our NP-hard problem.

2.4 Related Work

2.4.1 Related Work to Idle Listening and Scheduling

WSNs have penetrated into various fields for different purposes. During data collection many problems arise due to resource constraints and these pose numerous challenges. Therefore, several algorithms and techniques have been presented to tackle these challenges from different angles.

In [76] a new algorithm is proposed and identified as GAF (Geographic Adaptive fidelity). The main purpose of this algorithm is to conserve energy through load balancing. It tries to utilise redundant nodes that are equivalent for routing and alternate between them (activate/deactivate nodes) while the connectivity level is preserved. The main idea of the algorithm is basically composed of two steps. In the first step, after node deployment, the whole area is divided into square grids of side length $r = \frac{1}{\sqrt{5}}$ (normalized to half of the radio range) in which the nodes located in each grid are equivalent with regard to routing. Furthermore, nodes in adjacent grids can communicate with each other and local information such as GPS is used to determine node density and redundancy. Secondly, there is cooperation between nodes inside each grid for the state transition (sleep/awake) so as to provide load balancing for packet forwarding. The node that will be responsible for routing is elected selected through a ranked based election selection algorithm which considers the nodes' residual energy. As a result, energy conservation is achieved. One of the main drawbacks of this algorithm is the need to use GPS to identify redundant nodes, which is expensive. Moreover, the proposed algorithm is not suitable for dynamic environments in which topology frequently changes. Furthermore, the hop length is forced to match the length of the square, which is roughly half the transmission range of the actual radio which leads to high latency. Another drawback, as far as I understand, in practice, is that although GAF requires one node per cell, sometimes, sometimes some of the grids are more likely to be empty; as a result, there is no guarantee of complete connectivity. Furthermore, the orientation of the source-

destination is only considered in one direction while there are at least three directions for each cell except the corner cells (for which there are two directions).

Another direction of research aims to achieve energy conservation through load balancing by taking advantage of redundancy is Span [11]. Span is a randomized distributed algorithm that locates between MAC and network layers. It is mainly designed to dynamically choose a subset of nodes as coordinators (forwarders) from all nodes in the network to take part in the routing while the connectivity guarantee is provided. The coordinator eligibility rule is used to elect a node to be a coordinator. A node can become a coordinator only if two of its neighbours cannot either directly or through one or two coordinators communicate with each other. However, if there is a need of a coordinator there is a possibility that several nodes simultaneously decide to be a coordinator; as a consequence, collision happens among them. To resolve this issue, each node delays its coordinator announcement by random back off delay, wherein the residual energy of the node and the number of neighbours that can be connected are taken into consideration as parameters. In general, the algorithm rotates the coordinators' election among the redundant nodes in order to ensure all nodes provide connectively roughly equally. The algorithm also tries to keep the number of nodes that become coordinators as low as possible. The coordinators should withdraw when all the pairs of its neighbours can communicate either directly or indirectly. In addition, the nodes only make a local decision to join the network. As a whole, the lifetime of the network improves when the ratio of awake/sleep time and the density of the nodes increases. One of the main drawbacks of this algorithm is that neighbour and connectivity information are required to make a decision about whether the node becomes a coordinator or not. Thus, modification in the routing protocol may be required which means that it depends on the routing protocol.

In [26] energy consumption is analysed in general. They then focus on linear topology for two cases. First, for the setting where the nodes have an equal number of packets; second, where nodes have a variable number. Furthermore, both cases, of equidistance and variable distance among the nodes, have been analysed and the formula for optimal energy consumption is provided correspondingly. Further study towards GAF is performed and it is shown that by dividing the area into non-equal

lengths and using variable transmission ranges among the nodes, more energy can be saved.

In [85] a probabilistic scenario is proposed to benefit from node redundancy and achieve load balancing so as to save energy. They are interested in computing the number of involved hops that are required from source to reach the destination as a function of distance and the density of the active nodes. The basic idea of this algorithm is that when the source node has a packet, it broadcasts it to the active neighbour nodes that are within its transmission range. This packet contains its own and intender receivers' location. After broadcasting, the forwarding phase takes place, during which the focus is on the closest coverage area to the intended destination that is divided into regions. The priority is for the first slice of the region which is closest to the intended destination. One of the random nodes which has high priority in terms of closeness to the destination becomes a transmit node (relay node) for the next hop. More specifically, the coverage are of the sender is divided into a number of regions. If there is no relay node in the first slice of the region, the next slice is tested until one of which can fulfil the task. Namely, this pattern continues until the packet reaches the area wherein the destination is either within it and receive the packet directly or is one hop away from it and only needs a single hope to receive it.

It can be observed that in reality fewer hops are required to deliver the packet from source to destination, whereas, in GAF due to the restriction of the radio range to $r = \frac{1}{\sqrt{5}}$, in the worst case scenario more hops are required for transmission than GeRaF(geographic Random Forwarding) which leads to high latency.

Xin Guan et al. [33] investigated energy saving for the static network when sensors are deployed randomly in an area, where the sink being outside this area. They proposed a load balancing algorithm to save energy. Their algorithm mainly consists of two steps. In the first step, sensors are classified into layers where the closest layer to the sink is labelled as layer 0 and the furthest layer is labelled as layer d. After the step of layer construction, the second step begins. In this step, each node at layer i (for $i = 1, \ldots, d$) selects one of the parent nodes at layer $i - 1$ to forward its packet. Interestingly, their formula proposed to select parent nodes for packet forwarding, which is based on the distance between the two nodes and

the residual energy. They also proposed data aggregation to further reduce energy consumption. However data aggregation is not always possible for an application where each individual packet forwarding is indispensable.

Energy efficient data aggregation scheduling is another area of research; specifically, the energy that can be saved is via the minimization of the state transitions of the nodes in the network. In [52] the authors observed that if the consecutive time slots are assigned to the links (children) associated with the parent node, the frequent transient states are reduced substantially which lead to great energy saving and lower latency. In contiguous link scheduling, the parent node needs to start up only twice; first to receive all the data from its children consecutively, and second, to forward the received data to the upstream node. They consider only two different topologies; tree and directed acyclic graph (DAG), in their paper. The centralized and distributed algorithms are proposed to achieve contiguous link scheduling. It can be observed that in the centralized algorithm, first, the number of time slots are assigned to the nodes in decreasing order of their incident link (i.e., weight), then either recursive back-tracking, or a minimum conflicts heuristic algorithm that are given, are used to reduce the number of time slots. As they state, the recursive back-tracking algorithm is a brute force search and is not desirable due to its high complexity. As a result, the fast algorithm (minimum conflicts heuristic algorithm) is proposed to reuse time slots and minimize latency.

Further investigation has been conducted towards continuous link scheduling and expanded to heterogeneous networks [51]. They prove that the contiguous link scheduling problem in WSNs is NP-complete. The authors show that the proposed algorithm has a theoretical performance bound to the optimum in both homogeneous and heterogeneous networks. A similar issue is also examined in [84]. Based upon all aforementioned research one can observe that great importance has been paid towards idle listening and that there is still a gap for further research to be filled.

In [44], the problem of constructing an efficient data aggregation tree is investigated and termed the minimum energy cost aggregation tree (MECAT), in which the total energy consumed by sensors for transmission and reception is minimal. Two types of this problem are considered: with and without relay nodes. Both cases are shown to be NP-complete. For the case of not using relay nodes, they

find that the shortest path tree algorithm is a 2-approximation algorithm . For the case where relay nodes are considered in order to enhance the network connectivity, it is shown that the shortest path and the Steiner tree algorithms each have a bad approximation ratio in the worst cases. Then an O(1)approximation algorithm is obtained by constructing the shortest path tree on the routing structure of the capacitated network design problem.

In [8] an efficient routing of the traffic in the graph towards the sink is investigated in order to maximize the lifespan of the network. To this end, the efficient routing configuration with regard to efficient energy consumption (balancing energy consumption throughout the network) is proposed wherein packets are forwarded through multiple paths to the sink. This is achieved via determining a set of optimal vectors (a vector represents a fraction utilization of each path used to send a packet from node v to the sink) that minimize the energy consumption of the greediest sensor node in the network. Contention based MAC protocol is used in the module and the details of the energy consumption per packet per node are analysed (i.e., reception, overhearing, idle listening, transmission). One of the main drawbacks of this paper is that the authors consider unlimited retry in the transmission until the packet is successfully delivered which is an unrealistic scenario.

The Low- Energy Adaptive Clustering Hierarchy (LEACH) is proposed by Heinzelman et al [36]. The basic idea of this technique is that a set of sensors are randomly chosen to become cluster heads, to which other nodes send their data. Then, these cluster heads aggregate the received data and send it to the sink. Since the cluster heads have more responsibility, they are rotated in a randomized fashion in order to achieve evenly distributed workload between the sensors and obtain fair energy depletion among them. In the LEACH algorithm, the operation is divided into rounds and each round is composed of two stages. The first stage is the setup phase where a set of nodes based on the probabilistic formula are selected. Then the cluster heads broadcast the advertisement in order to let the other nodes join the cluster heads based on the minimum communication energy. Once the first phase is finished, the second stage, called steady state, begins. In this stage the nodes send their data to the cluster heads, then the cluster heads aggregate the data and finally transmit it to the sink. Then a new round starts, with a new cluster head formation. One

of the main drawback of this technique is that the setup phase is non-deterministic and may lead to service interruption.

Similarly, in [35], further improvements are made on LEACH and the algorithm named the Enhanced Low-Energy Adaptive Clustering Hierarchy (E-LEACH). Only two main improvements are made in the first step (setup phase). First, unlike LEACH, in E-LEACH a cluster head selection algorithm has been proposed for the setting where sensor networks have non-uniform initial energy levels. Second, the required number of cluster heads should increase by the square root of the total number of sensors so as to minimize the total energy consumption. The second phase (steady phase) of E-LEACH is the same as the second phase of LEACH. Likewise, further study has been conducted towards cluster hierarchy in [2]. Unlike LEACH and E-LEACH, in this paper cluster head selection with multi hop up to k hops within each cluster are considered. As a consequence, energy consumption is achieved by reducing communication between sensors and cluster heads. However, cluster heads may run out of energy before the other sensors.

Florens et al. [21, 22] have addressed the issue of minimum data distribution and collection times on tree networks for the scenario where each node is mainly equipped with directional and then omnidirectional antennae, each node has an arbitrary number of packets, and the node, upon receiving a packet, forwards it immediately (i.e. there is no buffering). A lower bound was derived for certain cases (i.e., a linear network, multi-line chain and tree, when the degree of the root is one), and corresponding centralized algorithms have been described. To deal with general trees, they then suggest that sub-trees should be linearised and that the proposed multi-line algorithm for the system should be applied. In addition, graphs with cycles (connected graphs) have been included in their analysis, and the performance of their own algorithms, that have an approximation ratio of two, has been compared with the optimal performance of such graphs. Their results became the starting point for many subsequent papers.

Similarly, Song et al. [69] have addressed the problem of scheduling in WSNs for a periodic traffic pattern, and the corresponding time and energy efficient algorithm was presented. This differed from the results of Florens et al. [21, 22]. They also paid attention to alleviation of energy wastage due to idle listening. Thus, a

distributed implementation algorithm was provided, to let each node determine its own duty cycle and put itself into a sleep state whenever it is not receiving a packet. However, they did not state how they mitigated interference during scheduling. Similarly, Bermond et al. [4] have also studied the delay in the data gathering process, in WSNs. They analysed and provided the optimal schedule for data collection in tree networks, for the case where the transmission and interference ranges of the sensors are the same (i.e., $d_T = d_I$). Unlike [21, 22], in their analysis there is a possibility of buffering the received packets and their later forwarding.

Similarly to Florens et al. [21, 22], the problem of a minimum completion-time for scheduling data gathering in connected graphs has been studied by Gargano et al. [29], for the setting where each node is equipped with a directional antenna. Unlike Florens et al. [21, 22], they show that there exists an algorithm that can obtain the optimal solution for any connected graph (i.e., they approach the problem by finding the optimal solution to the collision-free path-colouring problem), for the setting where the interference range D_I is equal to the transmission range D_T (i.e., $D_I = D_T$), and each node has a single packet.

Bermond et al. [3] have investigated the minimum-data gathering problem in general graphs, for the setting where $D_I, D_T \in \mathbb{R}$, such that $D_I \geq D_T > 0$, and each node has p packets (i.e., $p \geq 0$). The lower bound was then determined, before they showed that the problem is NP-hard. Finally, an algorithm with an approximation factor of four for general networks was proposed, regardless of the value of D_I, D_T. Furthermore, it has been shown that the problem still remains NP-hard for the case $D_I > D_T$, even if each node has a single packet.

Gandham et al. [24] have considered minimum latency scheduling for data collection in trees, in a setting where all nodes have data. They show that the minimum schedule length for data collection in linear networks is $3N-3$, where N is the number of nodes in the chain, excluding the sink. Then unlike previous techniques, they proposed a distributed scheduling algorithm. Their schedule is not a successive-slot schedule (it will be defined later). Furthermore, they also show an upper bound of $\max\{3n_k - 1, N\}$ on the schedule length for multi-line networks, where n_k is the length of the longest line connected to the sink, and N is the total number of nodes in the network. For tree networks, they also show an upper bound of $\max\{3n_k - 1, N\}$,

where n_k is the number of nodes in the largest one-hop sub-tree of the root. Further investigation has been performed by the same authors in [79]: firstly, for the case where a large amount of data is sparsely distributed (each node has a different amount of data); and secondly, for the case where the data is small and can be aggregated on the way to the sink. Similarly, scheduling for data collection has also been addressed by Choi et al. [13]. They show a lower bound of $3(N-2)$ time slots for collecting data in a chain, which matches the results from [24], because they take N as the number of nodes, including the sink. Moreover, they show that finding a minimum schedule for general graphs is NP-hard. As a result, a heuristic algorithm has been proposed, which tries to schedule as many interference-free segments as possible, in order to minimise the length of the schedule. The issue of the scheduling problem has also been examined by Ergen et al. [20]. They also show that the scheduling problem, in general, is NP-complete, and a heuristic algorithm has been proposed.

Another paper that has addressed the scheduling problem in a graph for both homogeneous and heterogeneous cases is [46]. The authors provide the optimal solution for a certain homogeneous graph that consists of only three layers, and where each node has only a single message to forward to the sink. Furthermore, they show that the optimal solution on three-layered graphs is NP-hard, if each node has various message sizes, and there are two approximation algorithms for three-layered graphs. They also show that the data-gathering problem for general graphs is NP-hard, and they provides an approximation ratio for it.

Dai et al. [17] also address data collection, considering a multi-sink setting for multi-lines, and using a variable interference model, compared to the constant interference distance of two hops, as in some of the above-mentioned articles. Incel et al. [37] have argued that if all interference is mitigated between nodes, then data collection can be performed in $\max\{2n_k - 1, N\}$ time slots. They then proposed an algorithm that matches the lower bound, where each node needs to buffer two packets, at most. Incel et al. have also suggested using different frequencies for data convergecast. They show that utilising two frequencies is sufficient to schedule all nodes in a tree network. Haibo et al. [77] address data collection scheduling in a model with different frequencies. Moreover, they prove a lower bound of $2N - 1$

slots for the chain.

Hang et al. [78] have demonstrated the architecture of the typical WiFi receiver for processing incoming signals. They have explained that the incoming signal is first received by the RF and then converted to a baseband signal, by a mixer. Next, the baseband signal is sampled via a digital to analogue converter (DAC), and passed to the CPU for decoding and recovering the original bits of the data frame. It has been shown that both DAC and CPU operate at the full amount of work (full clock rate) during the idle listening, similar to the receiving mode. Furthermore, it has been stated that the power consumption of the digital devices is proportional to voltage-squared and clock rate. Based on this explanation, one can observe that the energy consumption of idle listening is very similar to receiving packets. As a result, an interesting, novel technique, E-MiLi (Energy-Minimising Idle Listening), was proposed in order to reduce the energy consumption that is caused via idle listening. This is achieved by adding a special preamble to the packet, which is used to make a separation between packet detection and decoding; that is, in their technique, the clock rate of the circuitry adaptively downgrades during idle listening, and returns to full clock rate in the receiving mode. In other words, as soon as the packet is recognised as its own destination, the CPU reverts to the full clock rate, in order to decode the packet; otherwise, it remains at the lower clock rate and discards the packet. Consequently, energy can be saved. Although the proposed technique of Zhang et al. is quite fruitful, it cannot alleviate the total energy wastage of idle listening.

The most relevant article which pertain to our work is [81, 82]. Zhao and Tang [81, 82] consider data collection in trees in a setting where not all sensors have data in each round, the schedule must be independent of which sensor nodes have data, and the goal is to reduce idle listening and latency. They aim to conserve energy and extend the lifespan of the network. They present a technique called the *successive-slot technique*, in which all transmissions of a node must be made in successive slots starting from the first slot during which the node is scheduled to transmit by the schedule. In particular, parent nodes cannot transmit before their child nodes, and often parents need to listen to their child nodes one more time after their last transmission, in order to detect that no more data will come from these

child nodes. Then the parent node can be switched off for all the slots during which it is scheduled to listen to these child nodes for the remainder of the data collection.

2.4.2 Related Work to Sink Mobility

Due to several restrictions and limitations in the area of WSNs, sink mobility has been used, exploited and become the subject of many papers, with its pros and cons having been analysed. Generally speaking, mobile sinks can be classified into types: random and controlled movement. In the former one, the motion of the mobile sink is random and we cannot control its movement, however in the second form, the motion of the mobile sink is restricted to following some chosen paths or locations. We briefly summarize several energy efficient algorithms that have been proposed for mobile sinks.

In sparse networks, connectivity can be achieved either by deploying more sensors, which increases the cost, or by using multiple sinks that lead to power saving, due to the use of short transmission ranges by sensors; this approach is also cost-effective. Therefore, Shah et al. [64] have proposed a three-layer architecture. The first layer comprises sensors, while the second layer is a mobile sink (mule) that collects data when it is in the proximity of sensors, and deposits the result at the third layer, which is the access point. In their paper, the motion of mobile sink is random, which is undesirable in some applications, and it is sometimes difficult to determine latency. Further analysis was performed in [39].

Zhao et al. [83] have studied the mobility approach for data delivery in sparse networks. A set of mobile nodes message ferries (MF) was exploited in order to provide connectivity amongst the nodes. Moreover, two variations of MFs have been considered. In the first setting, MFs move along a known trajectory and carry data between nodes, whereas in the second scheme, an MF adjusts its trajectory towards the node. The idea of MF is very interesting, providing connectivity amongst disconnected or sparse networks; communication happens via a single-hop method, which is also productive in terms of energy saving. On the other hand, this technique has a high latency, which is impractical in delay-constrained applications.

For the first time, Gandham et al. [25] tried to use controlled mobility to collect data from sensors. They proposed an integer linear programming problem (ILP) to

find the locations of multiple mobile sinks per round. In their model, the locations of multiple mobile sinks are fixed in each round, and may be different in the next round. Their aim is to minimise energy consumption per node and total energy consumption per round. As the number of nodes increases, it is difficult to use their proposed integer linear programming (ILP) method; their technique is only useful for some small instances.

Kansal et al. [40] have investigated controlled sink mobility. Their design setting has multiple objectives, and is application-dependent. In some applications, energy conservation is important; therefore, the mobile sink should approach each node to use the least amount of energy in transmitting its data to the mobile sink. In contrast, latency is crucial in other applications, and fast data collection takes priority, via multi-hop forwarding. Kansal et al. have also considered the mobile sink's speed of movement (i.e., if it stops in a dense area, or slows down in a sparse area without stopping).

Scheduling and finding the optimal path for a mobile sink was examined in [67]. The main purpose is to collect data from sensors, before the buffer overflows. The authors in the proposed scenario allow the mobile sink to visit sensors multiple times, in order to avoid buffer overflows. They have shown that this scenario is NP-hard. Consequently, they proposed and analysed certain heuristic algorithms. The same authors then further investigated multiple mobile sinks, for the same setting [67].

Wang et al. [75] considered the optimisation problem of determining the sink's movement and its sojourn time at certain points in a grid network, in order to prolong the network's overall lifetime. To achieve these goals, a linear program (LP) was proposed. However, two main drawbacks were observed in their paper: firstly, it is only applicable to grid networks. Secondly, they did not include routing problems in the LP; rather, they used the shortest path to forward packets to the sink, without considering the remaining energy of the sensors. This approach was further investigated and improved by Papadimitriou et al. [55]. They addressed sink mobility and its sojourn time, in order to prolong the network lifetime. The problem was formalised as LP. However, their LP formulation is different from [75]; for example, their LP also includes routing problems, and it is applicable to general networks.

In [53], a mobile sink, named SenCar, was used for data collection. The authors have addressed the issue of load balancing and careful path planning. Furthermore, they have shown that by choosing a careful path, the lifetime of the network can be prolonged. Finally, they have proposed an algorithm based on divide and conquer in order to divide the network into two clusters and recursively apply the same algorithm to each cluster, until the desirable turning points (line segments) are obtained. Eventually, the mobile sink should follow these line segments that connect the selected turning points, in order to collect data. Park et al. [56] addressed the issue of finding a set of stop points on a fixed path with limited length the path stop point (PSP) problem in order for the mobile sink to collect data and minimise energy consumption. They have shown that selecting a set of optimal stop points is NP-hard. They then formulated the problem as LP, in order to find the optimal solution for small instances, and this became a benchmark for other heuristic algorithms. Finally, they proposed a heuristic algorithm to determine a subset of stop points, in order for the mobile sink to collect data from sensors. The selection of these stopping points is based on certain criteria such as the data rate, sensor locations and their energy consumption. The sensor nodes that are far from the stop points then send their packets in a multi-hop manner to those nodes (rendezvous points) that are a one-hop neighbour of the stopping points; finally, these nodes (rendezvous points) send the collected data to the mobile sink. Two issues can be identified with the proposed algorithm. Firstly, with an increase in the stopping points, latency increases. Secondly, it is more likely that there are some nodes that are only one hop away from the stop points, and which do not become rendezvous points; therefore, they do not need to send their data to the candidate rendezvous points. Instead, they can send data directly to the sink.

Another paper [27] addressed a similar issue and has shown mathematically that if the number of sub sinks (stopping points) is increased, the lifetime of the network can be increased. Then they proposed a heuristic algorithm to choose a sub path that has maximum sub sinks, however; they did not consider the case where the mobile sink should return to the starting point after finishing its journey.

Another interesting study that investigated sink mobility is [74]. The authors mainly focused on routing techniques to balance the network load based on the

factors of remaining energy and distance. Thus, a routing algorithm was proposed. However, the authors did not mention clearly how to choose the next sojourn location for the mobile sink.

The two papers most relevant to our work are [50] and [70]. In [50], the authors addressed the issue of controlled mobility to prolong the network's lifetime by finding the optimal trajectory for the mobile sink. Furthermore, they bounded the travel distance of mobile sinks per round due to petrol issues and they labelled this problem the "distance-constrained mobile sink problem". They also bounded the distance between two different candidate locations that the mobile sink can visit in order to avoid buffer overflow. Moreover, when the mobile sink moves to another location, the routing tree rooted at that location will be reconstructed and this costs energy, therefore it should stay at each location for a certain amount of time. They formulated the problem as a mixed integer linear programming problem (MILPP). When the network size grows, it is not feasible to solve it using MILPP; therefore, a three-stage heuristic was proposed that has a lower complexity and high scalability. In their proposed algorithm, sojourn times are calculated first, then based on these calculations, a feasible tour is specified for the mobile sink. Although our constraint is the same as their constraint, our objective is different and the multi-hop approach is not considered in our problem. Then Liang et al. [49] expanded the idea of [50] to multiple mobile sinks and find the optimal trajectories for them. Furthermore, unlike [50], they bounded the maximum hop count from each sensor to its nearest sink which is quite important in delay sensitive applications.

Sugihara and Gupta [70] focused on controlled mobility for data collection without multi-hop forwarding; their main purpose is to reduce latency during data collection. To this end, they identified the problem of finding the minimum tour as the "label covering problem" and proposed an algorithm in which dynamic programming was used to achieve a solution. The Travelling Salesman Problem (TSP) is the problem of finding a minimum cost tour to visit a set of given cities, starting and ending at the same city, such that each city is visited exactly once. Their algorithm consists of two main stages. Firstly, they try to achieve the TSP tour via any existing algorithm in order to find the minimum distance (and so construct a minimum tour). Secondly, they try to optimize the TSP tour by finding shortcuts

(sub paths) for the mobile sink to follow while still covering all the nodes using the dynamic programming technique. They named the problem the label covering problem because each shortcut considered must cover the skipped nodes. Again, the main objective of their paper is to minimize latency while in our work we want to minimize total energy consumption, subject to a latency constraint.

Similar to [70], He et al. [34] have discussed the issues surrounding path selection problems, for mobile elements (i.e., mobile scheduling) during data collection. Due to the difficulty of the problem, the authors have proposed a heuristic algorithm, in order to find the optimal path for mobile elements to follow and collect data from a sensor, via single-hop communications. In their proposed algorithm, the TSP is used as the first step to determine the path. Then, by taking advantage of the combining wireless communications, some of the visited points are likely to intersect with each other; then via a binary search, further tour improvement is obtained. Finally, the mobile sink follows the constructed path, in order to collect data. Via simulation, the authors also show that the algorithm outperforms the Label Cover algorithm.

2.5 Summary

In this chapter we first defined some terminologies and some fundamental points of the graph theory which are relevant to our work. Then we revised and explained complexity theory. Finally, an intensive literature review has been provided to pave the way to our area. As explained, different techniques have been used to tackle the issue of energy consumption from different perspectives and their pros and cons have been analysed. Based on that we observed the idle listening attracted attention of the research community. Therefore, we focus mainly first on the idle listening then on sink mobility in our research to further improve energy consumption.

Chapter 3

Reducing Idle Listening in Wireless Sensor Networks

This chapter, which is the main part of our research, deals with energy consumption issues in wireless sensor networks with tree topology during data collection, and provides the comprehensive explanation of our proposed technique (i.e., Extra-bit technique) to minimise energy consumption via reducing idle listening.

3.1 Introduction

In the literature, several reasons have been pointed out as the cause for wasted energy. The first one is collisions, which occur when two or more nodes try to transmit their data to the same destination node simultaneously. Indeed, the packets collide with each other and the destination node cannot receive either one correctly [43]. As a result, retransmission is needed, which leads to the use of more energy to send out the packets again. Moreover, both primary and secondary conflicts which cause collisions must be avoided [19]. Primary conflicts happen either when a node transmits and receives at the same time, or several nodes send out their packets simultaneously to the same destination. Secondary conflicts occur when a receiver is within the transmission range of its sender and other simultaneous senders at the time of collision. When this happens, the receiver cannot receive the correct packet successfully [59, 52]. The second source of energy wastage is overhearing, which happens when a node hears a packet which is destined for another node. The

third case where energy can be wasted, and the most relevant to our work, is idle listening because our work is built upon the previous proposed technique to reduce idle listening and we found out that there was still a gap to be filled. Idle listening occurs when a node is listening to the channel to receive a packet but there is the possibility that the sending node does not have data and remains silent [43, 5, 66].

Therefore the main focus of this chapter is on idle listening. We start by extending earlier work on successive-slot schedules [81, 82]. We propose an optimization technique, called the extra-bit technique, which reduces idle listening further and also minimizes latency. We prove that the optimal number of time slots for data collection in a chain using successive-slot or extra-bit schedules is $4N-6$, where $N \geq 3$ is the number of nodes in the network excluding the sink. We show how to calculate the expected amount of idle listening for extra-bit schedules and successive-slot schedules in chains and trees where each node has data with a fixed probability, and we demonstrate by graphs based on the mathematical formula that the expected amount of idle listening is significantly smaller with the extra-bit technique.

3.2 System Model, Arbitrary Schedules, Successive-Slot Schedules

3.2.1 System Model and Arbitrary Schedules

Consider a sensor network with a tree topology. The tree network is represented as a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges in the network. In other words, V represents sensors and E represents communication links between these sensors in the tree network. The sink is the root of the tree.

We assume that all nodes have a single omnidirectional transceiver, and all communication among sensors is performed over a single unique frequency channel. Furthermore, a node cannot send and receive a packet at the same time. It cannot receive a packet successfully when it hears several packets simultaneously. That is to say, both primary and secondary conflicts must be avoided for successful transmission. We consider TDMA schedules, where time is divided into a number of slots of equal length. In each slot several packets can be scheduled, but no conflicting

transmissions can be scheduled in the same time slot.

We use TDMA as a MAC layer protocol because of its advantage of avoiding collisions, idle listening and overhearing.

Furthermore, TDMA can collect data in a timely manner which can be beneficial for certain applications.

We want to collect data from the sensor nodes in a way that minimizes the total number of time slots, while also reducing idle listening as much as possible. The aim is to achieve fast data collection and conserve energy. We assume that not every sensor will have data to be collected in each round of data collection. Nevertheless, we require a schedule that is independent of which nodes have data: the same schedule must be followed no matter whether all nodes have data or only some nodes have data. If a transmission from node v to its parent p is scheduled in a particular time slot but no data is available at v to be sent to p, there will either be idle listening (i.e., p listens for a transmission from v, but v remains silent) or, if p already knows that no data will be sent from v in this time step, the transceivers of v and p can be switched off (and energy saved).

For a node $v \in V$, we denote by T_v the set of nodes of the subtree rooted at v, and by $|T_v|$ the cardinality of that set. The set of children of node v is denoted by $C(v)$.

A schedule S of length (or latency) K is a mapping of time-slots $1, 2, \ldots, K$ to sets of transmitting nodes, where $S(t)$ is the set of nodes scheduled for transmitting in time slot t. As every transmission is from a node to its parent, the schedule does not need to specify the receivers of the transmissions. The sink will never be a transmitting node. A schedule S is *feasible* if it satisfies the following conditions:

(C1) For every t, the nodes in $S(t)$ can transmit simultaneously without conflict. This means that no two nodes in $S(t)$ have distance two or less in the tree G.

(C2) Every node v (apart from the sink) is scheduled exactly $|T_v|$ many times for transmission.

(C3) For $i > 1$, if the i-th transmission of node v is scheduled in time slot t, then at least $i - 1$ transmissions of children of v must have been scheduled before time slot t.

Condition (C1) models interference constraints. Condition (C2) is required because each node v must transmit its own data and the data of all the other nodes in its subtree T_v. Condition (C3) expresses that node v cannot send more data than its own data and the data it has already received from its children.

We refer to such a feasible schedule as an *arbitrary schedule* as no further restrictions are imposed.

If not all nodes have data, the behavior of a node is as follows. Whenever the node is scheduled to transmit in the current time slot, it checks whether it has any data (either its own data or data received from a child) that has not yet been sent to the parent. If so, it uses the current time slot to forward any such data to the parent. Otherwise, it remains silent in the current time slot. We refer to this node behavior as *local greedy*.

3.2.2 Successive-Slot Schedules

As observed by [81, 82], arbitrary schedules can cause a lot of idle listening if not all nodes have data. For example, if a node has 10 time slots for transmitting data to its parent, but only 3 nodes in its subtree have data, then there will in general be 3 time slots with transmissions and 7 time slots with idle listening. The parent usually cannot turn off its transceiver to avoid idle listening as it cannot predict whether a packet will be sent by the child in the current time slot or not. Zhao and Tang therefore propose a restricted type of schedule, which they call *successive-slot schedule*. The special property of successive-slot schedules is that all transmissions from a node to its parent will happen in successive slots starting from the first slot that is assigned to the node for transmission, provided that local greedy scheduling is used in each node. A node cannot cause idle listening at the parent in between two actual transmissions from the node to its parent. Formally, if node v is scheduled to transmit to its parent in slots $t_1, t_2, \ldots, t_{|T_v|}$ and if r of the nodes in T_v have data, the transmissions from v to its parent must be made in time slots $t_1, t_2, \ldots, t_r$. The advantage of successive-slot scheduling is that as soon as the parent detects that the child is silent in a transmission slot, it knows that no further transmissions from that child will arrive. Therefore, the parent can switch off its transceiver in all remaining time slots where that child is scheduled to transmit. Similarly, the

sink will know that data collection has been completed as soon as each child v of the sink has either sent $|T_v|$ data packets or has been silent in one time slot. This means that data collection can potentially be completed earlier (before the end of the full schedule S). Therefore, successive-slot scheduling can reduce idle listening (as there can be at most one time slot with idle listening for each parent-child pair) and schedule latency.

Zhao and Tang prove in [81] that successive-slot schedules can be characterized as follows.

Lemma 1 ([81])**.** A feasible schedule S is a successive-slot schedule if and only if the following condition holds:

(C3′) For each node v and each $1 \leq i \leq |T_v|$, the i-th transmission of node v is scheduled after the i-th transmission of each child c of v with $|T_c| > i$, and after the last transmission of each child c of v with $|T_c| \leq i$.

Observe that condition (C3′) implies condition (C3).

We illustrate the two conditions of Lemma 1 for successive-slot scheduling using the simple example shown in Figure 3.1.

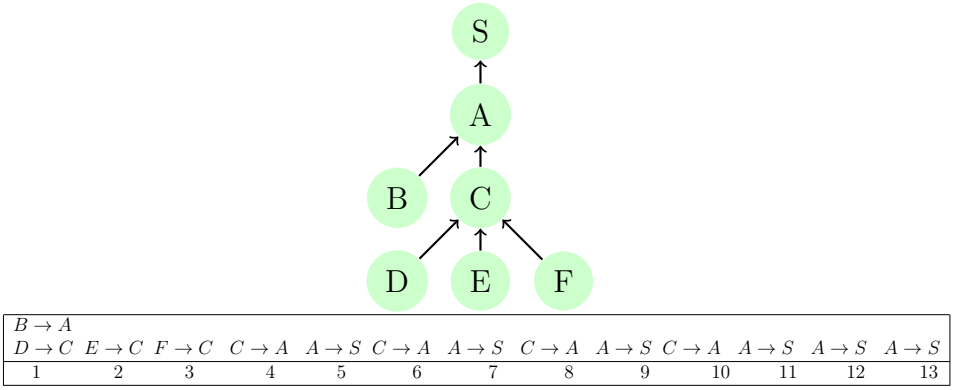


Fig. 3.1 Successive-slot schedule.

In this example node B and C are the children of node A, $|T_B| = 1$ and $|T_C| = 4$. One can observe that for node B we are in the second part of the condition of Lemma 1 and for node C we are in the first part of the condition of Lemma 1. Therefore the first, second and third transmissions of node A must happen after the first, second and third transmissions of node C (the first, second and third transmissions of node C can happen at steps 4, 6, 8, respectively). Therefore the first

second and third transmissions of node A can happen at steps 5, 7, 9, respectively). After that we are in the second part of the condition of Lemma 1, which means that $|T_C| \leq i$. Hence, the fourth, fifth and sixth transmissions of node A can happen after the last transmission of node C. In other words, the last transmission of node C happens at step 10. Therefore the rest of the transmissions of node A must come after step 10.

Successive-slot scheduling can be applied to data collection in any tree network. In more general networks, a data collection tree can be determined in a first step, and successive-slot scheduling can then be applied to that tree (but the interference constraints would be derived from the full network).

In a successive-slot schedule, the process of data collection starts from leaf nodes and proceeds towards the root node (sink). Generally speaking, a node must listen for transmissions from its child nodes until an idle transmission occurs. Receiving idle listening from any child node guarantees the end of transmission from that node, which allows the parent to turns off its transceiver for any further scheduled transmissions from that node. Each time the parent node has listened to all its child nodes and has received at least one packet, it can make one transmission to its own parent node. This continues until it has received idle listening from all of its child nodes (or the child nodes have completed all their transmissions). Finally, it will transmit the remaining packets to its own parent. This process continues for all nodes until the specified sink node has stopped listening to all its children, implying that all packets have been received.

We observe that if some node in the subtree T_v does not have data in the current round of data collection, the parent p of node v needs to listen to v one more time than the number of actual transmissions from v to p.

Zhao and Tang [81] propose a heuristic that aims at producing successive-slot schedules with minimum schedule length. They do not prove bounds on the worst-case schedule length produced by their algorithm compared to the optimum schedule length.

We illustrate successive-slot scheduling using the simple example shown in Figure 3.2. There are six nodes; the root is the sink. Suppose that only two of the leaf nodes have data, namely C and D. Firstly, B must listen to all its child nodes to

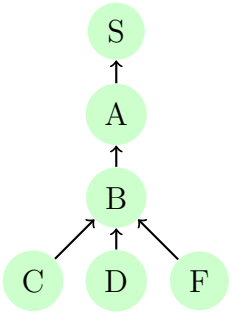


Fig. 3.2 Data collection, only C, D have data.

check whether they have data. It is obvious that B must listen three times. In the first two time slots, B can receive data from C and D, while in the last time slot no packet is available and this results in idle listening. This means that B has only two packets. After that, A is scheduled to listen to B. In the first and second such slot, A can receive data from B. Thus, A needs to listen again to B, but in the third such slot A does not receive any packet from B; therefore, A does not have to listen to B again after the third listening and turns its transceiver off for transmissions from B in the remainder of the data collection schedule. Although in B two packets are available, A will also need to listen one more time than the number of packets, to find out whether any more packets are available. Similarly, the sink needs to listen to A three times. In the first two time slots the sink can receive two packets but in the third slot it does not receive any packet. Therefore, the sink turns its transceiver off, and the data collection is completed. In an arbitrary schedule for data collection, it can be observed that the sink should listen to A five times and A should listen to B four times. Following the successive-slot technique by Zhao and Tang [81], when a parent has listened and not received any packet from the child then it does not listen again to that child. This reduces idle listening.

3.3 Extra-Bit Schedules

We propose a technique called the *extra-bit technique* that extends the successive-slot technique [81] and reduces idle listening further. In the successive-slot technique, parent nodes often have to listen to a child node one more time than the number of data packets sent by the child node, causing an idle listening slot. In the extra-bit technique, we can avoid idle listening in all cases where at least one node in the

subtree has data.

The extra-bit technique adds a single extra bit to each packet, which indicates to the receiver whether this packet is the last one being transmitted by the node in the current data collection round. The value of the bit is set to 0 if the packet is the last one, indicating that no more packets will be sent by that node. This tells the receiver that it does not need to listen any more for transmissions from this node in the current round of data collection, thus avoiding idle listening. The bit is set to 1 if the packet is not the last one, which means that more packets will be sent in later time slots.

According to this technique, when a parent has listened to its child and checked the extra bit, then it can decide whether to listen again in further time slots. The process of listening will be continued until the parent receives a packet where the extra bit is 0; the parent then stops listening to its child node and switches its radio off for the rest of the schedule regarding that node, resulting in energy conservation. The only exception is if a node has no data to transmit at all; in that case, there will be one idle listening time slot for the parent.

A successive-slot schedule in which each node always has the information required to set the extra bit correctly is called an *extra-bit schedule*.

3.3.1 Equivalence of Extra-Bit and Successive-Slot Schedules

One might expect that extra-bit schedules are more restrictive than successive-slot schedules, because whenever a node sends a packet, it needs to be able to set the extra bit to a correct value. This means that the node must know whether the packet being sent is the last one or not. Somewhat surprisingly, we can show that every successive-slot schedule is an extra-bit schedule.

Theorem 1. Every successive-slot schedule S is also an extra-bit schedule.

Proof. Let S be a successive-slot schedule. By Lemma 1, S satisfies condition (C3′). We prove by induction on the height of the nodes that each node has sufficient information to set the extra bit for each transmitted packet. The claim clearly holds for leaf nodes. Now consider a node v and assume that the claim has been

proved for all children of v. Consider the i-th transmission of node v, $1 \leq i \leq |T_v|$, and assume that v has a packet to transmit. By condition (C3′), each child c of v has already had at least i transmission slots if $|T_c| > i$, or has already had all its transmission slots if $|T_c| \leq i$. In the former case, node v knows whether c has at least $i+1$ packets or whether c has already transmitted all its packets. In the latter case, node v knows that c has already transmitted all its packets. Node v also knows how many packets it has already received but not yet forwarded to its parent. From this information, v can set the extra bit of the current packet correctly. □

For any given tree network, there can be many different data collection schedules, and also many different extra-bit schedules or successive-slot schedules. We are interested in extra-bit schedules of minimum length.

For example, consider six nodes F, E, D, C, B, A and a sink S that are arranged in a linear chain. We can schedule data collection in this linear network using the successive-slot or extra-bit technique in different ways. Two different schedules for this network, together with the idle listening that arises in the successive-slot technique and the extra-bit technique, are as follows:

1. Figure 3.4 show two different schedules for the setting when all nodes have data (it can be observed that both successive-slot schedule and extra-bit technique are the same and there is no idle listening when all nodes have data). It is worth noting that the first schedule lets each parent node receive all the packets from its respective children, then forward all to its parent accordingly. This type of successive-slot schedule in general has maximum length (latency), because there is no simultaneous transmission between the nodes. The length of the schedule is 21 time slots for the chain shown in Figure 3.3.

   On the other hand, the second type of schedule lets each parent, upon receiving data from its child node, immediately forward it to the next node. Furthermore, any node that has made one transmission and still has more data, can reschedule itself with other nodes concurrently after two steps (two hops) from its previous transmission, as shown in the Figure 3.4. The main advantage of this type of schedule is that it allows parallel transmissions between the nodes, which leads to the minimization of the schedule. In general, the first type of

the schedule is the worst case, whereas the second one is the best case (optimal schedule).

2. We show two different successive-slot schedules for the setting when only nodes F and E have data. One possible schedule is that E listens to F, then D should listen to E twice; likewise, node C should listen to node D three times, where in the first two time slots it receives data whereas in the third time slot it receives idle listening. Similarly, node B can receive packets from node C in the first two time slots, then it receives idle listening in the third time slot, which means that no packet is coming from C. In the same way, node A can receive packets from node B in the first two time slots and then it receives idle listening in the third time slot. Similarly sink node S, upon receiving idle listening in time slot 15, stops listening in the subsequent slots. It can be observed that the optimal successive-slot schedule ends at step 15. This is because where any parent node has listened to its child node and has not received data, it can stop all subsequent listening slots regarding that child node; due to its idle listening, the parent node knows that no further data will come from that child. However, if we do not follow the proposed technique of Zhao and Tang [81], listening is continuous until the end of the schedule, which is 21 time steps.

   The second possible type of schedule for this setting is similar to the second type of schedule mentioned before without applying the successive-slot schedule. One can notice that each parent, upon receiving data from its child node, immediately forwards it to the next node. As a result simultaneous transmissions can happen. Therefore in the second type of schedule the sink can conclude data collection at time step 12.

3. We also show two different extra-bit schedules for the setting when only nodes F and E have data. The first type of schedule is finished at time step 11, which finishes a few steps before the first type of successive-slot schedule. This is because there is no idle listening slots in the extra-bit technique for this setting, whereas there are 4 idle listening according to the successive-slot schedule. Similarly, the second type of schedule according to the extra-bit technique ends at time step 9. One can observe that the data collection in the

extra-bit schedule ends sooner than the successive-slot schedule for the same scenario.

Here, only two cases have been mentioned, the first being when all nodes have data, and the second being when only the last two nodes have data. In general, concluding data collection depends on which nodes have data. Note that we may have more possible forms but for simplicity only two forms are shown. Moreover, for simplicity we have explained the idea of the successive-slot schedule only for the line which is a special type of the tree, and the same idea is applicable to the tree as well.

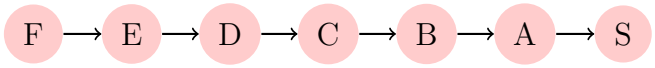


Fig. 3.3 A chain with 6 nodes.

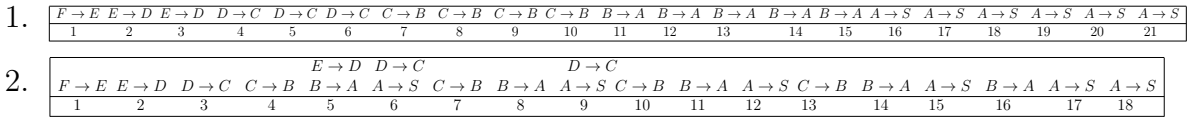


Fig. 3.4 Two different schedules for 6 nodes in Figure 3.3 according to successive-slot and extra-bit schedules for the setting when all nodes have data.

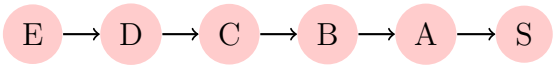
3.3.2 Optimal Extra-Bit Schedules for Linear Networks (Chains)

In this section we consider WSNs where sensors are arranged as a chain, with the sink located at one end of the chain. We let N denote the number of nodes in the chain, excluding the sink. We denote the node that is i hops away from the sink by v_i, for $1 \leq i \leq N$. We also refer to v_i as the i-th node of the chain.

In the schedule we propose, a node will first wait until it receives the first packet from its child. From this time slot onward, it will make a transmission once every three steps, and nodes that are 3, 6, 9, ... hops further away from the sink will transmit simultaneously with the node. This process continues until the only nodes that still have packets are the two nodes closest to the sink. Then, these two nodes transmit their remaining packets to the sink, with only one transmission per time slot.

As an example, a chain with $N = 5$ sensor nodes and a sink node is shown together with an extra-bit schedule of length 14 in Figure 3.5. Note that time slot 5 is the only slot in which two transmissions take place simultaneously. For illustrative purposes, we partition the schedule into five phases, where each phase ends with a time slot in which a packet is transmitted to the sink. In the first phase, five time slots are required until the sink node receives the first packet. In the second and third phase, only three time slots are needed until the sink receives the second and third packet, respectively. Two time slots are used in the fourth phase, and finally one time slot is used to finish this schedule.

Now suppose that only the last node E has data in a certain data collection round. Then the schedule shown in Figure 3.5 will complete data collection after 5 time slots at the end of the first phase (when the sink receives the packet from A with the extra-bit set to 0) and has no idle listening periods. For comparison, if the same schedule is executed as a successive slot schedule, data collection will be completed only after 8 time slots, and there will be four cases of idle listening (in steps 5 to 8).



				$D \to C$			
$E \to D$	$D \to C$	$C \to B$	$B \to A$	$A \to S$	$C \to B$	$B \to A$	$A \to S$
1	2	3	4	5	6	7	8

$C \to B$	$B \to A$	$A \to S$	$B \to A$	$A \to S$	$A \to S$						
9	10	11	12	13	14						

Fig. 3.5 Extra-bit schedule for a chain with 5 nodes.

The shortest extra-bit schedules for the cases $N = 1$ and $N = 2$ can easily be seen to have lengths 1 and 3, respectively. Next, we prove that for $N \geq 3$ the optimal length of an extra-bit schedule in the chain is $4N - 6$. We first give the lower bound, and then the upper bound.

For arbitrary schedules, it has been shown in [24, 13] that $3N - 3$ time slots are required to complete a converge-cast in the linear network. This lower bound can be shown by considering the three nodes closest to the sink. All transmissions by these three nodes must be scheduled in different time slots due to interference. The first

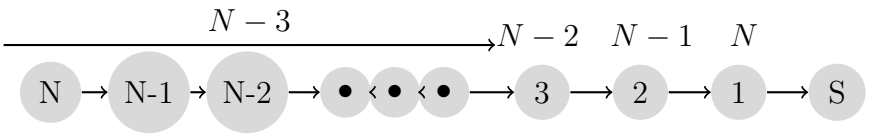


Fig. 3.6 Illustration of lower bound proof for chain with N nodes.

node in the chain must make N transmissions, the second node $N-1$ transmissions and the third node $N-2$ transmissions. As a result, at least $N+(N-1)+(N-2) = 3N-3$ time slots are required to complete data collection (and a schedule with $3N-3$ time slots actually exists). We now show that extra-bit schedules require at least $4N-6$ time slots. We remark that although extra-bit schedules are longer than the shortest arbitrary schedules, extra-bit schedules have no or substantially reduced idle listening periods if not all nodes have data.

Theorem 2. For chains with $N \geq 3$ nodes and a sink, any extra-bit schedule requires at least $4N-6$ time slots to complete the data collection.

Proof. In the extra-bit technique, a node cannot make a transmission before it has received the first packet from its child. This implies that node v_{N-i} cannot make its first transmission before time slot $i+1$, for $0 \leq i \leq N-1$. In particular, there are $N-3$ time slots before the first time slot in which the third node v_3 can make its first transmission. The third node must make $N-2$ transmissions, the second node $N-1$ transmissions, and the first node N transmissions (see Figure 3.6 for an illustration). These $3N-3$ transmissions must all be made in different time slots, and none of them can be made during the first $N-3$ time slots. Therefore, the total number of time slots must be at least $(N-3)+(3N-3) = 4N-6$. □

We now present an algorithm that produces an extra-bit schedule of length $4N-6$ for chains with N nodes and a sink. The algorithm is shown in Algorithm 1. First, it initializes the schedule's time slots $S(t)$ (representing the set of nodes to be scheduled at time t) to be empty, the number of packets on node v_i to $p(i) = 1$, and the current time slot t to 0. Then the procedure ScheduleFirstPart is used to schedule the first transmission of nodes from the last node to the fourth node. Whenever a node v_i is to be scheduled, t is incremented and the procedure call parallel(i) is used to schedule the node v_i as well as any nodes $v_{i+3}, v_{i+6}, \ldots$ that still have a packet. When ScheduleFirstPart is finished, the procedure ScheduleRest is called. As long

Algorithm 1: Extra-bit scheduling algorithm for linear network.

Input: Chain with $N \geq 3$ nodes $v_1, \ldots, v_N$ and sink s
Output: $S(t)$ for $t = 1, \ldots, 4N-6$

```
1  S(t) ← ∅ for t = 1, ..., 4N − 6;
2  p(i) ← 1 for i = 1, ..., N;
3  t ← 0;
4  Call ScheduleFirstPart();
5  Call ScheduleRest();
6  procedure ScheduleFirstPart()
7  for i ← N down to 4 do
8      t ← t + 1;
9      parallel(i);
10 procedure parallel(i)
11 for j ← i to N increment by 3 do
12     if p(j) ≠ 0 then
13         s(t) ← s(t) ∪ {v_j};
14         p(j) ← p(j) − 1;// send a packet
15         p(j − 1) ← p(j − 1) + 1;// receive a packet
16 procedure ScheduleSecondPart()
17 while p(1) ≠ 0 do
18     for i ← 3 downto 1 do
19         if p(i) ≠ 0 then
20             t ← t + 1;
21             parallel(i);
```

as the first node still has a packet, it repeatedly considers the nodes v_i for $i = 3, 2, 1$ and calls parallel(i) if node v_i still has a packet.

The state of the chain (i.e., the number of packets $p(j)$ stored at each node v_j) before the k-th time slot, $1 \leq k \leq 4N-8$, of the schedule produced by the algorithm is as follows:

- If $k = 4r+1$ for $r \geq 0$: $p(j) = 0$ for $j \geq N+1-r$, $p(j) = 2$ for $j = N-r-3m$ for $1 \leq m \leq \min\{r, (N-r)/3\}$, and $p(j) = 1$ for all other j.
- If $k = 4r+2$ for $r \geq 0$: $p(j) = 0$ for $j \geq N-r$, $p(j) = 2$ for $j = N-r-1-3m$ for $0 \leq m \leq \min\{r, (N-r-1)/3\}$, and $p(j) = 1$ for all other j.
- If $k = 4r+3$ for $r \geq 0$: $p(j) = 0$ for $j \geq N-r$, $p(j) = 2$ for $j = N-r-2-3m$ for $0 \leq m \leq \min\{r, (N-r-2)/3\}$, and $p(j) = 1$ for all other j.
- If $k = 4r+4$ for $r \geq 0$: $p(j) = 0$ for $j \geq N-r$, $p(j) = 2$ for $j = N-r-3-3m$ for $0 \leq m \leq \min\{r, (N-r-3)/3\}$, and $p(j) = 1$ for all other j.

As shown above for every value of r there are 4 values of k. In addition, for every value of k there are three conditions. These three conditions indicate the state of the chain. In other words, each node either has 0 packets, 1 packet or 2 packets. The reason for having 4 values of k with a single value of r, is that the state of the chain repeats itself after these 4 values.

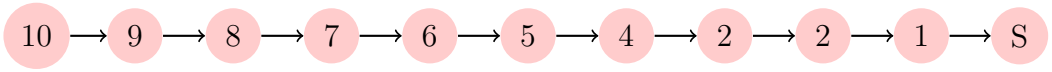
In general, the first condition indicates how many nodes have 0 packets from node j to node N, where j is the index of the starting node that has 0 packets. The second condition indicates how many nodes between node j and the sink have two packets, where nodes j is the furthest one from the sink to have 2 packets that start from node j. From this process variable m controls how many decreasing nodes will end up with 2 packets, this is subject to being 3 steps away from each other. This is the reason that value of m is multiplied by 3 and $(N-r-1)/3$ is divided by 3. The third condition (for all other j) indicates the number of nodes that have 1 packets.

These four conditions (points) show the state of the chain and they repeat as a circle until $k \leq 4N-8$ (the third node has no more packets). With each circle only one node becomes 0. At the beginning of the schedule each node has a single packet. When the algorithm starts running, each node either has 0 packet, 1 packet or 2 packets. This means that each node requires buffering for at most two packets; for further explanation refer to Figure 3.7. Furthermore, with each value of r the steps for these four values of k are as follows:

	$r=0$	$r=1$	$r=2$	...
k	1	5	9	...
k	2	6	10	...
k	3	7	11	...
k	4	8	12	...

We consider an illustrative example for 10 nodes as shown in Figure 3.7 in order to give insight into these four conditions in each step and the state of the chain. Note that at step 0 each individual node has 1 packet. Steps 1, 2, 3 and 4 indicate the first four conditions for the first value of $r = 0$. During these four steps only one node becomes 0. This process continues until the condition $k \leq 4N-8$ is satisfied.

Theorem 3. For chains with $N \geq 3$ nodes and a sink, the algorithm shown in Algorithm 1 computes an extra-bit schedule of length $4N-6$.



Conditions	Value of r	Steps/Nodes	10	9	8	7	6	5	4	3	2	1
		0	1	1	1	1	1	1	1	1	1	1
Condition 1	$r = 0$	1	0	2	1	1	1	1	1	1	1	1
Condition 2	$r = 0$	2	0	1	2	1	1	1	1	1	1	1
Condition 3	$r = 0$	3	0	1	1	2	1	1	1	1	1	1
Condition 4	$r = 0$	4	0	1	1	1	2	1	1	1	1	1
Condition 1	$r = 1$	5	0	0	2	1	1	2	1	1	1	1
Condition 2	$r = 1$	6	0	0	1	2	1	1	2	1	1	1
Condition 3	$r = 1$	7	0	0	1	1	2	1	1	2	1	1
Condition 4	$r = 1$	8	0	0	1	1	1	2	1	1	2	1
Condition 1	$r = 2$	9	0	0	0	2	1	1	2	1	1	2
Condition 2	$r = 2$	10	0	0	0	1	2	1	1	2	1	1
Condition 3	$r = 2$	11	0	0	0	1	1	2	1	1	2	1
Condition 4	$r = 2$	12	0	0	0	1	1	1	2	1	1	2
Condition 1	$r = 3$	13	0	0	0	0	2	1	1	2	1	1
Condition 2	$r = 3$	14	0	0	0	0	1	2	1	1	2	1
Condition 3	$r = 3$	15	0	0	0	0	1	1	2	1	1	2
Condition 4	$r = 3$	16	0	0	0	0	1	1	1	2	1	1
Condition 1	$r = 4$	17	0	0	0	0	0	2	1	1	2	1
Condition 2	$r = 4$	18	0	0	0	0	0	1	2	1	1	2
Condition 4	$r = 4$	19	0	0	0	0	0	1	1	2	1	1
Condition 4	$r = 4$	20	0	0	0	0	0	1	1	1	2	1
Condition 1	$r = 5$	21	0	0	0	0	0	0	2	1	1	2
Condition 2	$r = 5$	22	0	0	0	0	0	0	1	2	1	1
Condition 3	$r = 5$	23	0	0	0	0	0	0	1	1	2	1
Condition 4	$r = 5$	24	0	0	0	0	0	0	1	1	1	2
Condition 1	$r = 6$	25	0	0	0	0	0	0	0	2	1	1
Condition 2	$r = 6$	26	0	0	0	0	0	0	0	1	2	1
Condition 3	$r = 6$	27	0	0	0	0	0	0	0	1	1	2
Condition 4	$r = 6$	28	0	0	0	0	0	0	0	1	1	1
Condition 1	$r = 7$	29	0	0	0	0	0	0	0	0	2	1
Condition 2	$r = 7$	30	0	0	0	0	0	0	0	0	1	2
Condition 3	$r = 7$	31	0	0	0	0	0	0	0	0	1	1
Condition 4	$r = 7$	32	0	0	0	0	0	0	0	0	0	2
		33	0	0	0	0	0	0	0	0	0	1
		34	0	0	0	0	0	0	0	0	0	0

Fig. 3.7 Illustration of state of the chain in each step with 10 nodes.

Proof. We observe that the schedule constructed by the algorithm has the following properties:

- Every node v_i makes $N + 1 - i$ transmissions.
- Every node v_i makes its j-th transmission only after it has received j packets from its child (or all packets from the child in case $|T_{v_{i+1}}| < j$).

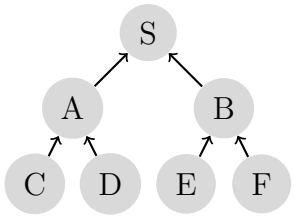
- The senders of simultaneous transmissions are at least three hops away from each other, so there is no interference between them.

Therefore, the schedule is a feasible extra-bit schedule.

The first packet reaches the sink in time slot N. Then one packet reaches the sink every three time slots, until only the first two nodes have packets left. This requires $3(N-3)$ time slots, and at that point the first node and the second node will contain one packet each. It then requires three more time slots to transmit these to the sink. The total schedule length is therefore $N+3(N-3)+3 = 4N-6$ time slots. □

3.3.3 Extra-Bit Schedules for Trees

Zhao and Tang [81] presented a heuristic algorithm for computing successive-slot schedules in trees. By Theorem 1, these schedules are also extra-bit schedules. We illustrate the benefits of extra-bit schedules for trees using the example in Figure 3.8.



C → A E → B	D → A F → B	A → S	B → S	A → S	B → S	A → S	B → S
1	2	3	4	5	6	7	8

Fig. 3.8 Schedule for data collection when only C, E have data.

In the extra-bit technique, 8 time slots are needed to complete data collection if all nodes have data. If we suppose that only C and E have data, then the beneficial impact of extra-bit scheduling can be observed: The data collection process finishes by the end of time slot 4, and there are only two occurrences of idle listening that happen when A and B listen to D and F, respectively. The sink listens to each of A and B only once and infers from receiving a packet with the extra-bit equal

to 0 that no further packets will arrive. With the successive-slot technique, data collection would finish only after six time slots, and there would be four occurrences of idle listening.

3.4 Idle Listening in Successive-Slot and Extra-Bit Schedules

In this section we compare extra-bit scheduling and successive-slot scheduling in terms of the amount of idle listening. We observe that the number of occurrences of idle listening in an extra-bit schedule does not depend on the particular choice of extra-bit schedule, and similarly for successive-slot schedules.

3.4.1 Idle Listening in Chains and Trees

We determine the number of occurrences of idle listening in chains and trees, both for the successive-slot technique and the extra-bit technique.

3.4.1.1 Chain

Consider a chain with N nodes in addition to the sink. The first node is the node closest to the sink, and the last node is the node furthest away from the sink.

With the extra-bit technique, we consider several cases for the amount of idle listening. First, if the last node has data, then there is no idle listening at all, even if some other nodes do not have data. Second, if no node has data, then there are N occurrences of idle listening. Third, if the last node has no data but some other node has data, then the amount of idle listening depends on the position of the furthest node from the sink that has data. For instance, in Figure 3.5, if node C has data and nodes D and E do not have data, then idle listening happens twice, once for a transmission from E to D and once for a transmission from D to C. For the general case, we can conclude that the number of occurrences of idle listening is equal to the number of nodes in the chain minus the position (distance from the sink) of the last node that has data. Let I denote the position of the last node that has data (and let $I = 0$ if no node has data). Then the number of occurrences of

idle listening with extra-bit scheduling is $N - I$.

With the successive-slot technique, idle listening can be analysed as follows. If the last node does not have data, the number of occurrences of idle listening is N as each of the N nodes will have an idle transmission to its parent. If J is the position of the last node that does not have data (or $J = 0$ if all nodes have data), then the amount of idle listening is J.

We observe that the amount of idle listening with extra-bit scheduling $(N - I)$ is always less than or equal to that of successive-slot scheduling (J): If $I = N$, then extra-bit scheduling has no idle listening while successive-slot scheduling may have up to $N - 1$ occurrences of idle listening. If $I < N$, then $J = N$ and therefore $J \geq N - I$. The most extreme difference between extra-bit scheduling and successive-slot scheduling occurs if only the last node has data. In that case, extra-bit scheduling has no idle listening and successive-slot scheduling has $N - 1$ occurrences of idle listening.

3.4.1.2 Tree

In the extra-bit technique, idle listening happens for a transmission from a node v to its parent if and only if none of the nodes in T_v have data. The number of occurrences of idle listening is therefore equal to the number of nodes whose subtrees have no data. In the successive-slot technique, idle listening happens for a transmission from a node v to its parent if and only if at least one node in T_v does not have data. The number of occurrences is therefore equal to the number of nodes whose subtrees contain at least one node that does not have data. It is clear that idle listening for the successive-slot technique is at least the amount of idle-listening for the extra-bit technique.

For example, consider the tree in Figure 3.8 and suppose that only A, D, B, and F have data. With the extra-bit technique there are two occurrences of idle listening, one for the transmission from C to A and one for the transmission from E to B. With the successive-slot technique, however, there are four occurrences of idle listening: the same two as for the extra-bit technique, and in addition one for a transmission from A to S and one for a transmission from B to S.

3.4.2 Expected Amount of Idle Listening

Now, we consider a probabilistic model in which each node has data with probability p (which is the same for all nodes), and show how to calculate the expected amount of idle listening for extra-bit and successive-slot scheduling.

3.4.2.1 Chain

Consider a chain with N nodes $v_1, \ldots, v_N$, indexed in order of increasing distance from the sink.

With the extra-bit technique, there is idle listening for a transmission from v_i to v_{i-1} if and only if none of the nodes v_j with $i \leq j \leq N$ have data. The probability for this event is $(1-p)^{N-i+1}$. The expected amount of idle listening is therefore:

$$\sum_{i=1}^{N}(1-p)^{N-i+1} = \sum_{i=1}^{N}(1-p)^{i} = \frac{1-p-(1-p)^{N+1}}{p}$$

For example, consider the chain with five nodes from Figure 3.5. The expected contributions of the five nodes to idle listening are as follows:

1. $1-p$ for node E
2. $(1-p)^2$ for node D
3. $(1-p)^3$ for node C
4. $(1-p)^4$ for node B
5. $(1-p)^5$ for node A

The expected amount of idle listening for the chain of five nodes is therefore:

$$1-p+(1-p)^2+(1-p)^3+(1-p)^4+(1-p)^5 = \frac{1-p-(1-p)^6}{p}$$

With the successive-slot technique, there is idle listening for a transmission from v_i to v_{i-1} if and only if at least one node v_j with $i \leq j \leq N$ does not have data. The probability that all nodes v_j with $i \leq j \leq N$ have data is p^{N-i+1}. The probability for idle listening from v_i to v_{i-1} is therefore $1-p^{N-i+1}$. The expected amount of

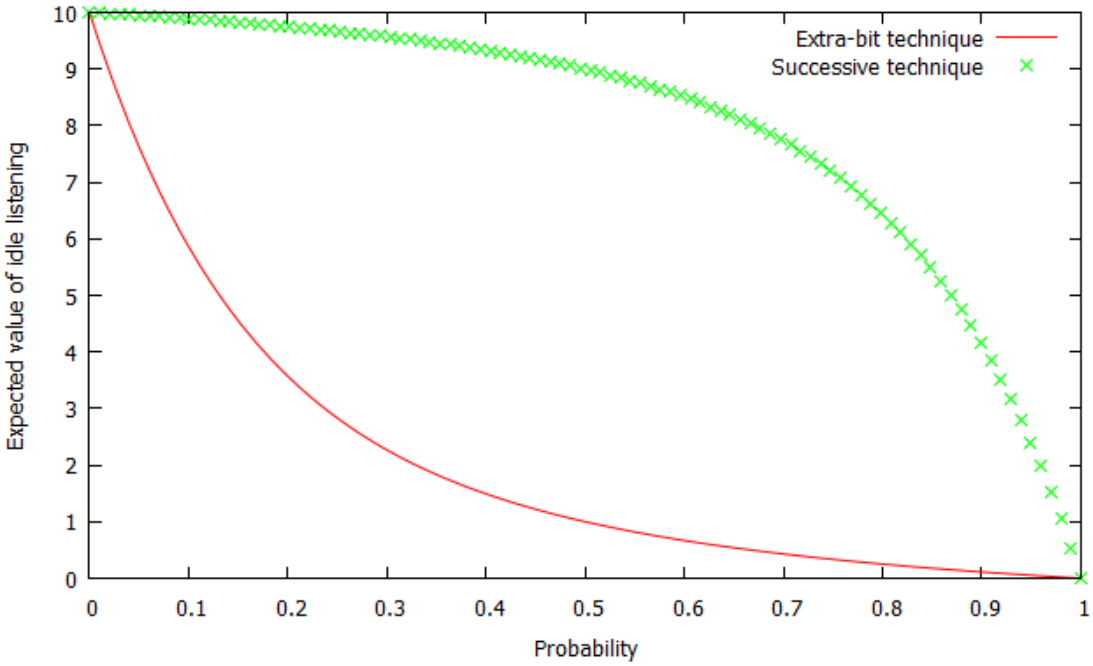


Fig. 3.9 Expected amount of idle listening for extra-bit and successive-slot technique in a chain with 10 nodes.

idle listening is then:

$$\sum_{i=1}^{N}(1-p^{N-i+1}) = N - \sum_{i=1}^{N} p^i = N - \frac{p-p^{N+1}}{1-p}$$

For the example chain with five nodes of Figure 3.5, the expected amount of idle listening for the successive-slot technique is:

$$1-p+1-p^2+1-p^3+1-p^4+1-p^5 = 5 - \frac{p-p^6}{1-p}$$

Figure 3.9 shows how the expected amount of idle listening for extra-bit and successive-slot scheduling in a chain with 10 nodes depends on the probability p. The x-axis represents the probability p that a node has data and the y-axis represents the expected amount of idle listening. We observe that the amount of idle listening for the extra-bit technique is much smaller than for the successive-slot technique, with equality happening only for the extreme cases $p = 0$ (no node has data, 10 occurrences of idle listening) and $p = 1$ (all nodes have data, no idle listening). For a wide range of p, the extra-bit technique has a significantly lower amount of idle listening. For example, when $p = 80\%$ the expected amount of idle listening is close

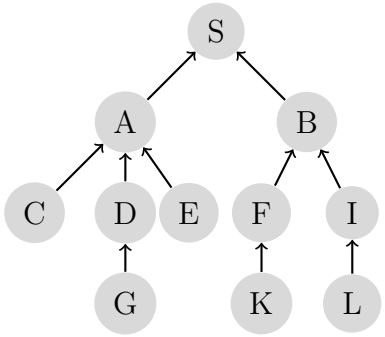


Fig. 3.10 Example tree for calculation of expected amount of idle listening

to zero for extra-bit scheduling but roughly 6.4 for the successive-slot technique. Starting at $p = 1$, the amount of idle listening increases sharply as p decreases with the successive-slot technique, but only slightly with the extra-bit technique.

3.4.2.2 Tree

Consider a tree whose set of nodes (excluding the sink) is V. With the extra-bit technique, idle listening happens for a transmission from node v to its parent if and only if none of the nodes in T_v have data, which happens with probability $(1-p)^{|T_v|}$. The expected amount of idle listening is therefore:

$$\sum_{v \in V} (1-p)^{|T_v|}$$

With the successive-slot technique, idle listening happens if at least one node in T_v does not have data, so the expected amount of idle listening is:

$$\sum_{v \in V} (1-p^{|T_v|})$$

For a concrete example, consider the tree shown in Figure 3.10. In this tree, there are five nodes with subtree size 1, three nodes with subtree size 2, and two nodes with subtree size 5. With the extra-bit technique, the expected amount of idle listening in this example is therefore $5(1-p) + 3(1-p)^2 + 2(1-p)^5$. On the other hand, the successive-slot technique has an expected amount of idle listening of $5(1-p) + 3(1-p^2) + 2(1-p^5)$.

Figure 3.11 shows how the expected amount of idle listening for both techniques

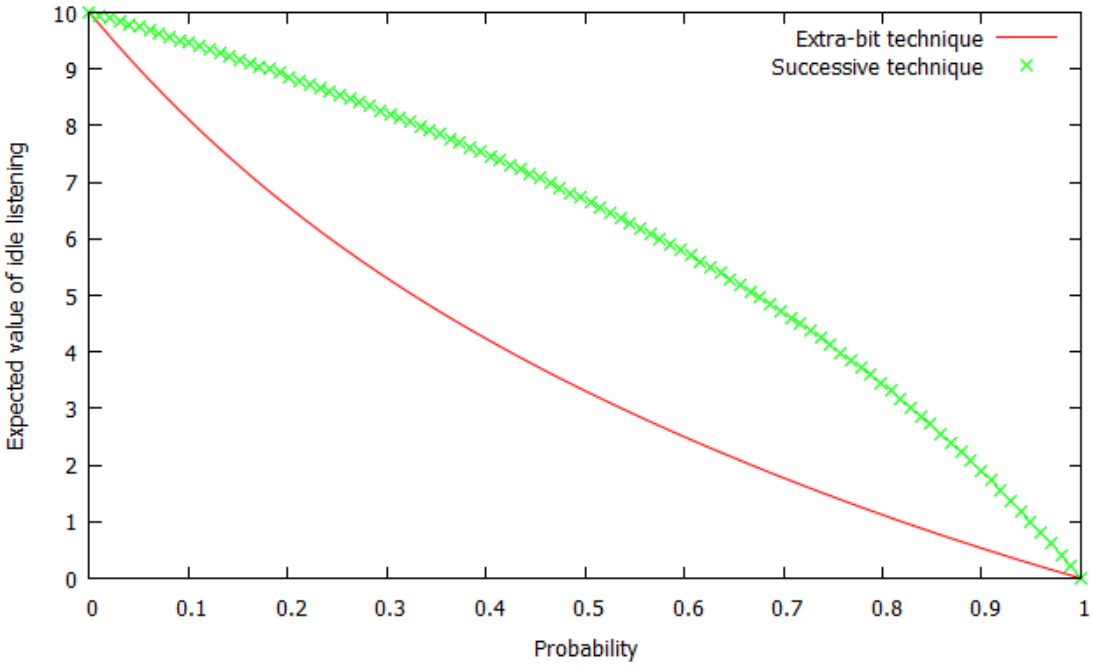


Fig. 3.11 Expected amount of idle listening for extra-bit and successive-slot technique in the tree of Figure 3.10.

for the tree of Figure 3.10 depends on p. Again, both techniques produce the same amount of idle listening only if $p = 0$ (no node has data) and $p = 1$ (all nodes have data, zero idle listening). While decreasing p from 1 to 0, the expected amount of idle listening increases more quickly for the successive-slot technique than for the extra-bit technique.

3.5 Uniform Energy Depletion

It can be observed that sensors near the sink should forward more packets than others. As a result, non-uniform energy drainage occurs; namely, the entire network is stopped while sensors far from the sink still have a greater level of energy. This point has motivated us to also address the issue of uniform energy depletion by adjusting the distance and transmission range of the nodes in order to maximise the lifespan of the entire network (i.e, all sensors' energy levels are depleted at the same time).

Consider five nodes with the sink from Figure 3.5. Assuming the maximum transmission range is used, each node has energy for sending k packets (for simplicity

$k = 5$). Moreover, assuming there is no energy consumption for receiving packets, it is concluded that the first node near the sink depletes all its energy after forwarding all the packets of the previous nodes in addition to its own packet (i.e., this network only works for one round of data collection).

Based on this, we can propose a non-uniform transmission range as shown in Figure 3.12 in order to improve the suitability of the quality of the network for taking part in k rounds of data collection (in fact, k depends on D). That is, we propose a non-uniform transmission range technique in which all nodes thereby inactivate simultaneously. Suppose that the length of the area to be covered by sensors is D and each node has an energy for forwarding k packets. We suppose that energy consumption is proportional to distance squared. Hence the distance relationship between the nodes is $d_N^2 = 2d_{N-1}^2 = 3d_{N-2}^2 = \cdots = Nd_1^2$, where d_N^2 determines the energy consumption of the transmission from the last node to the node next to it, and Nd_1^2 is the energy consumption of N the transmissions from the first node to the sink. In general, equations are needed to specify the optimal distance and transmission range for each node in order for all of them to run out of energy together. These equations can be expressed as

$$d_{N-i}^2 = \frac{d_N^2}{i+1} \Longrightarrow d_{N-i} = \frac{d_N}{\sqrt{i+1}} \tag{3.1}$$

So, all the nodes should cover the area D. Thus, $d_1 + d_2 + \cdots + d_N = D$. Hence from 3.1 we get

$$D = d_N + \frac{d_N}{\sqrt{2}} + \frac{d_N}{\sqrt{3}} + \cdots + \frac{d_N}{\sqrt{N}} \Longrightarrow d_N = \frac{D}{1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} + \cdots + \frac{1}{\sqrt{N}}} \tag{3.2}$$

From (3.2) the distance of the last node is determined. Then, when the distance of the last node is determined, the distance value of other nodes can be calculated using (3.1).

One of the main disadvantages of non-uniform distances is that more sensors are required to cover the area. Furthermore, sometimes a non-uniform transmission is not applicable if D is very large.

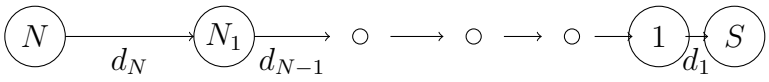


Fig. 3.12 Non-uniform transmission range for N nodes.

3.6 Summary and Discussion

We have proposed an optimized scheduling technique for data collection in sensor networks for scenarios where not all sensor nodes have data in each round of data collection. Our extra-bit technique is aimed specifically at increasing energy efficiency by reducing the amount of idle listening. Reduced idle listening helps to preserve energy and prolong network lifetime. Furthermore, we have shown how to construct extra-bit schedules for chains that have minimum schedule length. We have also shown how to compute the amount of idle listening in chain and tree networks for the previous successive-slot technique and for our new extra-bit technique. We have analysed the expected amount of idle listening in a model where each node has data with probability p, showing that the extra-bit technique reduces idle listening substantially compared to the successive-slot technique. Finally, uniform energy depletion has been addressed.

One can observe that both successive-slot and extra-bit techniques are fruitful in terms of idle listening reduction only for the setting when some nodes have data.

However, the length of the schedule compared to the arbitrary schedule is longer. Therefore, these two techniques are not helpful for the setting when fast data collection is required e.g. in oil leakage detection, fire detection, or disaster rescue operations.

Chapter 4

Towards a More General Form

In the previous chapter, we have proposed and explained the idea of our optimization technique. Despite the fact that we could determine lower and upper bound schedules for a single chain, finding the optimal schedule for a general tree is very complicated, and has not been achieved at the time of writing of this thesis. Hence, in this chapter, we continue to expand the concept of the extra-bit technique to several other special cases of the tree, such as equal length multi-chains, imbalanced multi-chains, balanced three and four-level k-ary trees and Rhizome trees. In other words, we give lower bounds on the schedule length for each of these cases. Moreover, we present algorithms for balanced chains, imbalanced chains and Rhizome trees. The proposed algorithms for balanced chains and Rhizome trees can match the lower bound whereas the algorithm for imbalanced chains is a few steps away from the optimal. Our ultimate goal is to find a solution for a general tree. As a second objective, we proceed to mitigate the idle state, which is available among the nodes due to the nature of the communication to avoid interference, to save further energy and to shorten the length of the schedule using two frequencies.

4.1 Optimal Extra-Bit Schedules for Balanced Multi-Chains

A multi-chain is the special case of a tree network topology that consists of several chains and a sink that is connected to one endpoint of each of the chains. In this section, we consider balanced multi-chains where all the chains have the same length.

We let M denote the number of chains and N the number of nodes in each chain. We assume $N \geq 3$ throughout this section, as the cases $N = 1$ and $N = 2$ are trivial. Including the sink, a balanced multi-chain has $MN + 1$ nodes. An example of a balanced multi-chain is shown in Figure 4.1. WSNs with multi-chain topology can arise in application settings where sensor data needs to be collected by a single sink from several linear structures. For example, sensors placed in a building for fire detection could form a chain on each floor, and all the chains could be connected to the sink for further processing of the sensor data.

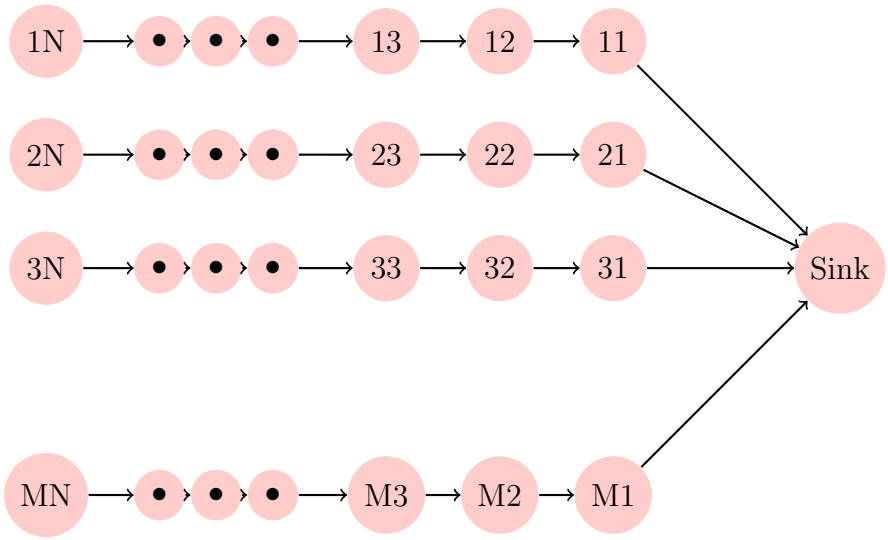


Fig. 4.1 Balanced multi-chain.

We want to determine the length of an optimal extra-bit schedule for data collection in balanced multi-chains. Since extra-bit schedules and successive-slot schedules are equivalent as shown in Chapter 3, Section 3.3.1, all results in this section also hold for successive-slot schedules. The case $M = 1$ brings us back to a single chain that requires $4N - 6$ time slots, as shown in the previous Chapter 3, Section 3.3.2. For $M \geq 2$, we will show the following:

- The optimal schedule length for $M = 2$ chains is $4N - 4$.
- The optimal schedule length for $M \geq 3$ chains is $N + MN - 1$.

Note that we cannot simply let each chain execute the optimal single-chain schedule as the sink can receive at most one packet in every time step.

First, we show the lower bounds. We need the following lemma.

Lemma 2. Consider any extra-bit schedule for a single chain with N nodes and the sink. For $1 \leq i \leq N-2$, the i-th packet received by the sink cannot be transmitted to the sink before step $N + 3(i-1)$.

Proof. For $1 \leq i \leq N$, let v_i be the node that is i hops away from the sink. Because the schedule is an extra-bit schedule, one can show that the sink can receive all packets in decreasing order of the index of their origin (i.e., first the packet from node N, then the one from node $N-1$, etc.). This can be achieved simply by each node giving preference to packets with larger index of origin when deciding which packet to forward next.

Each of the first $N-2$ packets reaching the sink must be transmitted by node 3, node 2 and node 1. In each step only one of these three nodes can transmit. None of the three nodes can make a transmission during the first $N-3$ steps, as the packet from node N cannot reach node 3 before step $N-3$. (Recall that the steps of a schedule are numbered in such a way that the first step is step 1.) For the i-th packet to reach the sink, $3i$ transmissions have to be made by the first three nodes, as each of the first i packets has to be transmitted by node 3, by node 2 and by node 1. The transmission in which the i-th packet is sent from node 1 to the sink must be the last of these $3i$ transmissions. Therefore, the i-th packet cannot be transmitted before time $N - 3 + 3i = N + 3(i-1)$. □

Theorem 4. Any extra-bit schedule for data collection in a balanced multi-chain requires at least $4N-4$ steps for $M = 2$, and at least $N + MN - 1$ steps for $M \geq 3$.

Proof. First, consider the case $M = 2$. By Lemma 2, the sink cannot receive the $(N-2)$-th packet from any chain before time step $T = N + 3(N-3) = 4N - 9$. Therefore, each of the two chains can transmit at most $N-3$ packets to the sink before time step T. Hence, at the start of time step T there are still six packets left to be sent to the sink, 3 packets from each chain. Transmitting these six packets to the sink requires at least six more steps, as the sink can receive only one packet per time step. Thus, the last packet cannot reach the sink before time step $T + 5 = 4N - 4$.

Next, consider the case $M \geq 3$. By Lemma 2, the sink cannot receive any packets before time step N. At the start of time step N, there are still MN packets that

need to be transmitted to the sink, and each of these must be sent to the sink in a different time step. Hence, the last packet cannot reach the sink before time step $N + MN - 1$. □

We remark that the lower bound for $M \geq 3$ shows that any algorithm that produces an optimal schedule must transmit a packet to the sink in each time step from time N to time $N + MN - 1$. That is, the sink must not be idle from time N until it receives the last packet.

To obtain optimal schedules for balanced multi-chains, we will combine optimal schedules for the individual chains, shifting time steps as necessary to prevent different chains from sending packets to the sink at the same time. Consider the optimal schedule for a single chain described in Chapter 3, Section 3.3.2, and observe that the sink receives packets in time steps N,$N + 3$, ..., $4N - 9$, $4N - 7$, $4N - 6$. In other words, the first $N - 2$ packets are sent to the sink starting at time N, with a gap of two time steps between consecutive time steps, finishing at time $4N - 9$. After the $(N - 2)$-th packet is sent to the sink, there is a gap of a single time step, and then the last two packets are sent to the sink in the two consecutive time steps $4N - 7$ and $4N - 6$. Denote this schedule by OPT_1. Intuitively, we use shifted versions of OPT_1 in the second and third chain in order to fill the gaps of the first chain. All remaining chains collect their packets in the node adjacent to the sink and transmit them to the sink in arbitrary order at the end of the schedule.

Theorem 5. For balanced multi-chains, there are extra-bit schedules of length $4N - 4$ if $M = 2$ and of length $N + MN - 1$ if $M \geq 3$.

Proof. First, consider the case $M = 2$. The first chain follows the optimal single-chain schedule OPT_1 of length $4N - 6$. The second chain uses a schedule S_2 that is obtained from OPT_1 by inserting an idle time slot (i.e., a time step in which no transmission is made in the chain) in the beginning and another idle time slot just before the $(N-1)$-th packet is transmitted to the sink. Note that the length of S_2 is $4N - 4$. Furthermore, we claim that the two chains can follow schedule OPT_1 and S_2, respectively, without creating a conflict at the sink. Observe that S_2 transmits packets to the sink in time steps $N + 1$, $N + 4$, ..., $4N - 8$, $4N - 5$, $4N - 4$, and therefore S_2 avoids all time steps in which OPT_1 sends packets to the sink. We have obtained a feasible extra-bit schedule of length $4N - 4$.

Now consider the case $M \geq 3$. The first chain uses a schedule S'_1 that is obtained from OPT_1 by inserting one idle time step before time $4N - 7$. The second chain uses a schedule S'_2 that is obtained from OPT_1 by inserting one idle time step at the start and two idle time steps just before the $(N - 1)$-th packet from that chain is sent to the sink. The third chain uses a schedule S'_3 that is obtained from OPT_1 by inserting two idle time slots in the beginning, and three idle time slots just before the $(N - 1)$-th packet of that chain is sent to the sink. With these schedules, the three chains send packets to the sink in time steps as follows:

S'_1: N, $N + 3$, ..., $4N - 9$, $4N - 6$, $4N - 5$
S'_2: $N + 1$, $N + 4$, ..., $4N - 8$, $4N - 4$, $4N - 3$
S'_3: $N + 2$, $N + 5$, ..., $4N - 7$, $4N - 2$, $4N - 1$

Observe that the sink receives at most one packet per time step. For $M = 3$, we have thus constructed a feasible schedule with $N + 3N - 1 = 4N - 1$ time steps, as required. For $M \geq 4$, let all the other $M - 3$ chains first execute the optimal single-chain schedule for a chain with $N - 1$ nodes, treating the node adjacent to the sink as a virtual sink. By Theorem 2, this schedule needs only $4(N - 1) - 6 = 4N - 10$ time steps if $N - 1 \geq 3$, and only $4N - 6 = 6$ time steps if $N - 1 = 2$. Hence, in each of these $M - 3$ chains all packets have been collected in the virtual sink after at most $4N - 6$ time steps. The first three chains transmit all their packets to the sink in consecutive time steps from time N to time $4N - 1$. The remaining $M - 3$ chains can then send their $N(M - 3)$ packets (which have all been collected in the virtual sinks of those chains) to the sink in arbitrary order in $N(M - 3)$ further time steps. The total length of this feasible extra-bit schedule is $4N - 1 + N(M - 3) = N + NM - 1$. □

The data collection schedules of Theorem 5 are optimal as their lengths match the lower bounds from Theorem 4.

Another interesting point that can be noticed is greater energy conservation. When the third chain finishes all of its transmissions, the other chains (chain four and upwards) have accumulated their data at their first nodes, therefore, we can let the other chains sleep until the fourth chain sends all its packets successively, then wake up another chain one after another, this process continues till the end of the last chain. As a result, the energy can be saved to some extent.

Algorithm 2: PART ONE, SCHEDULING ALGORITHM FOR M-BALANCED CHAINS IN EXTRA-BIT.

Input: $V = v_{(i,j)} \mid 1 \leq i \leq M, 1 \leq j \leq M \; with \; Sink$
Output: $S(t) \; for \; t = 1, \ldots, N + NM - 1$ for $N, M \geq 3$

```
1  procedure Main( );
2  S(t) = ∅    for t = 1, ..., N + NM − 1 ;
3  p(i, j) ← 1    for 1 ≤ i ≤ M, 1 ≤ j ≤ N;
4  p(Sink) = 0;
5  call procedure Scan-All();
6  call procedure Last-three-Nodes-of the first three chains();
7  call procedure Schedule the rest( );
8  procedure Scan-All();
9  for j ← N to 1 do
10 |  t ← t + 1
11 |  for i ← 1 to M do
12 |  |  Parallel(i,j);
13 procedure Parallel(i,j)
14 for j ← j to N  increment by 3 do
15 |  if p(i, j) ≠ 0  then
16 |  |  if (i ≠ 1) and(j == 1) then
17 |  |  |  continue;// skip first nodes of all the chains except the first chain
18 |  |  p(i, j) ← p(i, j) − 1;
19 |  |  s(t) ← s(t) ∪ {v(i, j)};
20 |  |  if (i == 1) and(j == 1)  then
21 |  |  |  Sink ← Sink + 1;
22 |  |  else
23 |  |  |  p(i, j − 1) ← p(i, j − 1) + 1;
24 procedure Schedule the rest( )
25 Schedule the remaining packets of the 3rd chain and upwards which are
   accumulated in their first nodes and did not have a chance to send in parallel
   with the main chain.
```

Algorithm 2 and 3 illustrate the implementation of the proposed schedule for balanced chains.

Generally, in this algorithm, the input is the multiple-chains with the single sink; the output is the optimal schedule.

Note: All nodes are arranged in a matrix where the farthest node of the first chain has indices $i = 1$ and $j = N$ while the first node, which is the closest node to the sink, has indices $i = 1$ and $j = 1$. For ease of description, this algorithm is divided into three steps as follows:

Step 1: In this step, which starts in line 8, all the chains are scheduled independently in parallel until the first packet of the first node reaches the sink. At the

Algorithm 3: PART TWO, COMPLEMENT OF THE FIRST PART.

```
1  procedure Last-three-Nodes-of the first three chains()
2  k = 0;
3  while p(1)(3) ≠ 0 do
4      for j ← 3 to 1 do
5          t ← t + 1
6          for i ← 1 to M do
7              switch j do
8                  case 1:
9                      if i == 1 then
10                         parallel(i,j);
11                     else if i == 2 then
12                         parallel(i,j+1);
13                     else if i == 3 then
14                         parallel(i,j+2);
15                     else
16                         call parallel(i,j+3);
17                 case 2:
18                     if (i == 3) then
19                         s(t) ← s(t) ∪ {v(i, j − 1)};
20                         packets(i, j − 1) ← packets(i, j − 1) − 1;
                           Sink ← Sink + 1, if(k > 1) parallel(i,j+2);
21                     else if i == 2 then
22                         parallel(i,j+1);
23                     else
24                         parallel(i,j);
25                 case 3:
26                     if i == 2 then
27                         s(t) ← s(t) ∪ {v(i, j − 2)};
28                         packets(i, j − 2) ← packets(i, j − 2) − 1;
                           Sink ← Sink + 1, if(k > 1) parallel(i,j+1);
                           k ← k + 1;
29                     else if (i == 3) then
30                         if(k > 1) parallel(i,j-1);
31                     else
32                         parallel(i,j);
33 schedule the last two nodes of the first chain and more parallel
   transmissions of the second chain
```

beginning, two for-loops are used to process nodes in all chains concurrently. The index of the chains is then passed to the procedure parallel in line 12. Inside of this procedure, which starts in line 13, a for-loop is used to schedule the current node and then take three steps away from the current transmission in order to perform parallel transmission of each chain in the same time slot. To perform this process,

initially the node should be checked for the packet in line 15. If the node has a packet, then there are three cases. Firstly, following N time slots, all of the chains are ready to send their first packet to the sink; however the sink is only capable of receiving one packet at a time. Therefore, only one of the chains can send its first packet to the sink; we prefer the first chain (though it could be any) in line 20. Secondly, when the sink has received the packet from the first chain, the first nodes of the other chains must refrain from sending their packet to the sink in line 16 due to the capability of the sink to receive only one packet at a time. Lastly, if they are not the first node in each of the chains, each node in each chain receives data from its child node in line 23. This is also due to concurrent transmission or reuse of time slots inside of each chain. After each operation the for-loop continues to perform parallel inside each chain. It can be noticed that in the first step N transmissions are performed in order for the sink to receive the first packet. Then it is only necessary to restart scheduling starting from the third node, which is explained in the second step.

Step 2: In this step data collection starts at the third node of the chains, at the second part of the algorithm in line 1. From that node to the sink, we need three time slots which are repeated for many rounds. In the third time slot the first chain can send the next packet to the sink, these three slots continue until the third node of the first chain is running out of packets in line 3-32. Then two and one more time slots are required for the second and first nodes of the first chain to finish their transmission respectively in line 33. It can be noticed that the sink has capacity to receive two more packets of the first and second time slots of the three slot group, which means that two slot gaps are available; therefore, the two other chains can send their packets in these two available slot gaps of the three time slots. We prefer to choose the second and third chains (it could be any others). With these three time slots, the scanning process is also performed for all the chains. The cases are used to control and regulate the three slots as follows:

1. Case 1: It refers to only the third time slot out of the three in which the next packet of the first chain reaches the sink and uses the conditional statement to check other chains with it. The first condition in line 9 means that if the node is the first node of the first chain, then it sends the packet to the sink by using

the parallel procedure. Likewise, the second condition allows the second node of the second chain to send the packet to the first node in line 11. Similarly, in the third condition the third node of the third chain can send the packet to the second node in line 13. However, the other chains can only perform their parallel transmissions in this time slot starting from their fourth nodes in line 15.

2. Case 2: It is used to regulate the second time slot of the main chain And triggers the third chain to send a packet to the sink. It checks the condition in line 18 and then initiates the first node of the third chain to send a packet to the sink. At that time, the second chain does not need to scan previous nodes because it already has checked them, therefore, we use variable k to let the node call the scan procedure in the second round and upwards in line 20. Similarly, the condition in line 21 is used to regulate the second chain with this time slot. Likewise, the last condition is used to enable the other nodes to perform their concurrent transmission in line 23.

3. Case 3: Inside this case, again three conditions are used to regulate the first time slot, the first condition in line 26 qualifies the second chain to send the packet to the sink in this time slot. At that time, the second chain does not need to scan previous nodes because it already has checked them, therefore, we use variable k to let the node call the scan procedure in the second round and upwards. Similarly, the conditional statement in line 29 is used to prohibit the third chain from transmitting in the first time slot out of the three, because chain 2 already has transmitted in this time slot and the sink can only receive one packet per time slot. Therefore, chain 3 does not perform any action in the first time slot of the first round. Likewise, the condition in line 31 allows the other chains to transmit in this time slot.

These two slot gaps have the effect of reducing the total time slots of the schedule in the network. This step is performed via the third procedure of the algorithm.

Step 3: By the end of the second step, the packets in the other chains are accumulated in their first nodes that did not have a chance to send to the sink. Thus, each remaining chain can send all its packets successively to the sink. As a

result, we need extra time slots for those chains that have remaining packets. This is performed via the procedure in line 24 in the first part.

As we expected, our proposed algorithm is optimal because it makes the sink busy from time N onward until all the chains are exhausted of their packets.

Note: We have already explained the expected amount of idle listening for a single chain in the previous chapter. For balanced chains, since each chain operates independently in the network, we calculate the amount of idle listening by multiplying the formula of the single chain by M, i.e,

$$M \sum_{i=1}^{N} (1-p)^i$$

4.2 Unbalanced Multi-Chains

In this section, we consider data collection from another special type of tree topology: unbalanced multi-chains, namely, chains of different length as shown in Figure 4.2. This topology also has its own applications in many areas such as fire detection, agriculture, etc. For unbalanced multi-chains a more sophisticated way is needed to efficiently collect data so that the number of time slots (that is, latency) is minimized.

A multi-chain is the special case of a tree network topology that consists of several chains and a sink that is connected to one endpoint of each of the chains. We consider the case that the chains can have arbitrary length, and different chains can have different length. We let M denote the number of chains. We let N_i be the number of nodes in the i-th chain and assume that the chains are indexed in order of non-increasing length, i.e., $N_1 \geq N_2 \geq \cdots \geq N_M$. We denote the i-th chain by C_i. We refer to any chain C_j for $j < i$ as a longer chain than C_i, even if it has the same length as C_i. Including the sink, a multi-chain has $1+ \sum_{i=1}^{M} N_i$ nodes.

The operation of the network discussed so far has focused on balanced networks; that is, those with chains of equal length. Where chains are of unequal length, the pattern described does not hold true.

When all chains are of equal length, priority of transmission to the sink is determined sequentially, whereas when chains are of different lengths, two factors determine the order in which those chains transmit to the sink: the order of the

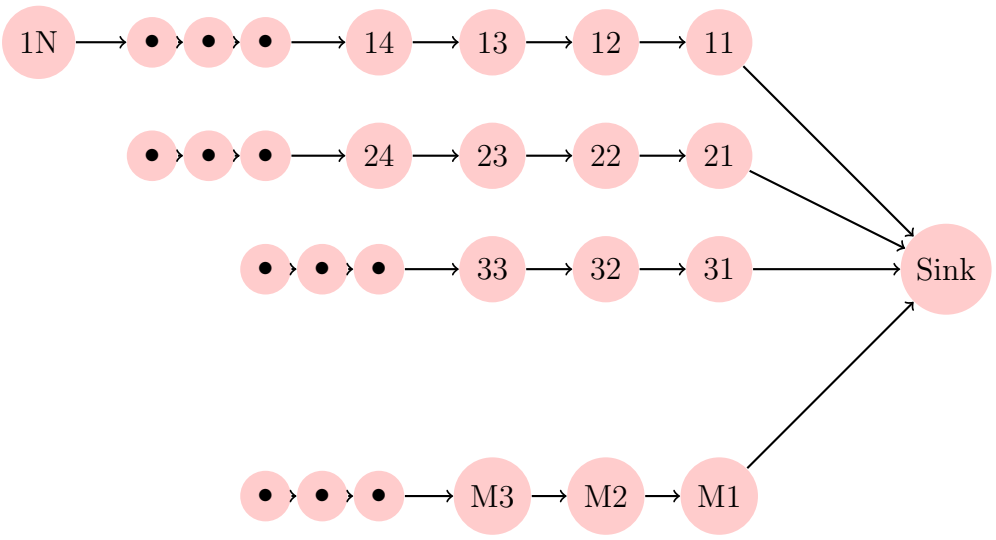


Fig. 4.2 Unbalanced multi-chain.

chain in the sequence, and the length of each chain.

Shorter chains will be ready to transmit to the sink sooner than the longest chain because they involve fewer steps between the end of the chain and the first node. Therefore, if the i-th chain for $i = 1, \ldots, M$ in the network was the shortest, it would begin transmitting to the sink first, while longer chains are still processing data. However, readiness to transmit is not the only criterion in determining which chain has priority. The length of the chain is also a factor, so that if the $(i-1)$-th chain is then ready to transmit, it will take priority and transmit in preference to the i-th chain, even if the i-th chain is also ready to transmit. In this manner, the two criteria of *length and readiness* to transmit ensure that the longest of the ready chains will have priority. Assuming no longer chains are ready to transmit, the i-th chain may then still transmit to the sink while the $(i-1)$-th chain is unable to transmit to the sink for two time slots. When longer chains, being of higher priority, are ready to transmit to the sink, they will do so in preference to shorter ready chains, which are of lower priority. The length of each chain will determine when they are ready to transmit, until the longest chain, with the consequently highest priority, is ready to transmit to the sink. At this point the shorter ones will generally no longer be able to send, unless they have an opportunity to do so in an interleaved fashion as described earlier. This point marks the end of the first step which includes only one round (i.e., N transmissions) and the sink receives one packet from the longest chain.

After N transmissions, when the longest chain has sent its first packet to the sink, the second step has begun, which consists of three time slots in each round (round means that until the sink receives a packet). This step operates many times as was previously described, with the first chain sending packets to the sink and the shorter chains transmitting to the sink while the first chain is silent for two time slots in each round (i.e., the shorter chains fill the two gap of the longest chain). Step 3 begins after the third node of the first chain sends its final packet, at that time two and one more time slots are needed for the second and first node of the longer chain to transmit their packets respectively. Step 4 begins when the longer chain has finished transmitting. Extra time slots are required for all chains to transmit their remaining packets, accumulated in their first nodes, to the sink.

Note that another method that can also be used to schedule unbalanced chains is as follows: An alternative procedure operates identically to the above description during step 1, so that initially the shorter chains, being ready to transmit to the sink sooner, have the opportunity to do so. Later in step 1, a bias for longer chains means that the shorter ones are, similarly to the first approach, out-competed by longer chains. However, once the first (primary chain) has transmitted its first packet to the sink, step 1 ends and step 2 begins. During step 2, a different bias is present within the selection approach for transmission to the sink. During these 3 time slot rounds, for the first two time slots, the longest chain is not ready to send a packet to the sink, and so other chains which are ready have the opportunity to do so. Under the previously detailed approach, while the longest chain is unable to transmit to the sink, the second longest and third longest chains would transmit. However, under this alternative approach, there is a bias in favour of shorter chains. This bias operates during the two time slots when the longest chain is unable to transmit to the sink, so that the shortest chains then transmit to the sink, followed by a transmission from the longest chain, in the third time slot of the round. This approach continues until the end of step 2. During step 3, the only available time slot for a chain other than the primary chain to send, is similarly utilised by the shortest chain. This bias continues in step 4, so that with all the packets in the chains having accumulated in the first node of each chain, the chains transmit their remaining packets successively in order from the shortest remaining chain to the

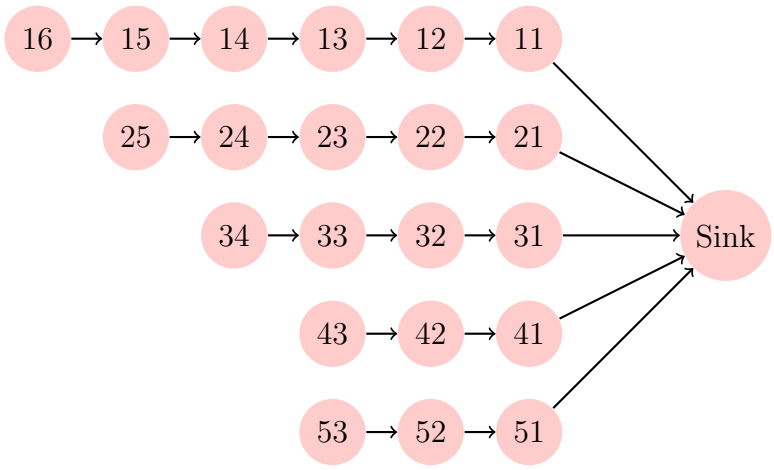
longest.

The advantage of a bias towards shorter chains is that it should conserve energy across the network. This is because a bias towards longer chains requires the shorter ones to wait until the end of all transmissions in longer chains before they themselves can transmit their final packets to the sink. Enabling the shorter chains to transmit in an interleaved fashion with the longest chain, and then favouring them in step 4, we can achieve that the shorter chains finish their transmissions to the sink before some of the longer chains so that they can then shut down, thereby conserving energy. This is the general description of how to schedule unbalanced chains. In the next part we explain the intuition behind of our proposed algorithm and show mathematically that it is at most 5 steps away from the optimal.

Due to the complexity of unbalanced multi-chains, we consider an illustrative example for 5 chains of arbitrary lengths as shown in Figure 4.3. Let c_1, c_2, c_3, c_4 and c_5 denote each chain in decreasing order (i.e., length of c_1=6, c_2=5, c_3=4, c_4=3 and c_5=3). According to the general description that we proposed for unbalanced multi-chains, in the first two steps, none of the chains is ready to send a packet to the sink, instead each one individually sends a packet from a node to its parent in the chain. It can be observed that in the third step, all of c_5 and c_4 are ready to send a packet to the sink. However, the sink can only receive one packet in each step. As explained before the priority is for c_4 (it could be c_5 as well because both c_4 and c_5 have the same length). In the fourth step, both c_5 and c_3 are ready but the priority is for c_3 according to the proposed algorithm. Similarly, in the fifth step, both c_5, c_4 and c_2 are ready but the priority is for the longer chain, which is c_2. Likewise, in the sixth step, all of c_5, c_4 and c_1 are ready but the priority is for the longer chain, which is c_1. We follow this strategy until the end of the transmission. As shown in Figure 4.3, the data collection ends at step 23 for this example.

We want to show that our proposed algorithm is near-optimal for data collection in unbalanced multi-chains. Since extra-bit schedules and successive-slot schedules are equivalent as shown before, all results in this section also hold for successive-slot schedules.

The case $M = 1$ brings us back to a single chain that can be scheduled optimally in $4N - 6$ time slots. We show how to construct a near-optimal schedule for $M \geq 2$.



c_1	$53 \to 52$	$52 \to 51$	$52 \to 51$								
c_2	$43 \to 42$	$42 \to 41$	$41 \to S$	$42 \to 41$							
c_3	$34 \to 33$	$33 \to 32$	$32 \to 31$	$31 \to S$	$33 \to 32$	$32 \to 31$	$31 \to S$	$32 \to 31$		$31 \to S$	
c_4	$25 \to 24$	$24 \to 23$	$23 \to 22$	$22 \to 21$	$21 \to S$	$24 \to 23$ $23 \to 22$	$22 \to 21$	$21 \to S$	$23 \to 22$	$22 \to 21$	$21 \to S$
c_5	$16 \to 15$	$15 \to 14$	$14 \to 13$	$13 \to 12$	$15 \to 14$ $12 \to 11$	$14 \to 13$ $11 \to S$	$13 \to 12$	$12 \to 11$	$14 \to 13$ $11 \to S$	$13 \to 12$	$12 \to 11$
steps	1	2	3	4	5	6	7	8	9	10	11

									$51 \to S$	$51 \to S$	$51 \to S$
							$41 \to S$	$41 \to S$			
				$31 \to S$							
$22 \to 21$	$21 \to S$	$21 \to S$									
$11 \to S$	$13 \to 12$	$12 \to 11$	$11 \to S$	$12 \to 11$	$11 \to S$	$11 \to S$					
12	13	14	15	16	17	18	19	20	21	22	23

Fig. 4.3 Scheduling unbalanced multi-chain with 5 chains.

By extending the proof of Lemma 2 one can also show that the $(N-1)$-*th* packet cannot be received by the sink before time $4N-7$, and the last packet before time $4N-6$. Thus, we obtain the following corollary.

Corollary 1. Consider any extra-bit schedule for a single chain with N nodes and the sink. For $1 \leq i \leq N$, the i-th packet received by the sink cannot be transmitted to the sink before step $N + 3(i-1) - 3$ in any schedule.

Note that the bound of Corollary 1 is tight for the last packet of the i-th chain, which cannot be sent before time $N_i + 3(N_i - 1) - 3 = 4N_i - 6$. To construct a schedule for a multi-chain, we will use near-optimal schedules for the individual chains, shifting time steps as necessary to prevent different chains from sending packets to the sink at the same time.

Consider the optimal schedule for a single chain of length $N \geq 3$ described before, and observe that the sink receives packets in time steps $N, N+3, \ldots, 4N-$

$9, 4N-7, 4N-6$. In other words, the first $N-2$ packets are sent to the sink starting at time N, with a gap of two time steps between consecutive time steps with transmissions, finishing at time $4N-9$. After the $(N-2)$-th packet is sent to the sink, there is a gap of a single time step, and then the last two packets are sent to the sink in the two consecutive time steps $4N-7$ and $4N-6$. The shorter gaps (i.e., a gap of two time steps and 0 time steps, respectively before each of the last tow packets) complicate the analysis, so we modify the schedule as follows: We insert one idle time step just before time $4N-7$, and two idle time steps just before the transmission of the last packet to the sink. Denote the resulting schedule for a chain of length N by $S(N)$. In the schedule $S(N)$, the sink receives packets in time steps $N+3j$ for $0 \leq j \leq N-1$. Note that such a schedule can also be found for $N=1$ and $N=2$. Note that the last packet reaches the sink in step $4N-3$, so the schedule is longer than the optimal schedule for $N \geq 2$, but at most 3 steps longer.

The idea of our algorithm is to let each chain C_i follow the schedule $S(N_i)$. If two or more chains want to transmit to the sink simultaneously, the chain with smallest index is allows to transmit, and the conflicting chains insert an idle time step into their schedules. If a chain wants to transmit a packet to the sink but cannot do so because a chain of smaller index is transmitting in the same time step, we say the chain is blocked in that time step.

More specifically, each chain C_i behaves as follows:

- Phase 0: As long as time steps N_i+3j for $j=0,1,\ldots$ are not blocked, the chain follows the schedule $S)N_i)$. If N_i+3j_0 is the first time step in which C_i is blocked, the chain enters Phase 1 at time N_i+3j_0.

- Phase 1: As long as time steps N_i+3j+1 for $j=j_0, j_0+1,\ldots$ are not blocked, the chain follows the schedule $S(N_i)$ with one idle time slot inserted just before time N_i+3j_0, i.e., it transmits to the sink in steps $N_i+3j_0+1, N_i+3(j_0+1)+1, \ldots$. If N_i+3j_1+1 is the first time step in Phase 1 in which C_i is blocked, the chain enters Phase 2.

- Phase 2: The chain inserts a second idle time slot just before step N_i+3j_1+1. As long as time steps $N_i+3j_1+2, N_i+3j_1+5, \ldots$ are not blocked, it follows the schedule and transmits to the sink in those time steps. If N_i+3j_2+2 is

the first time in phase 2 in which C_i is blocked, the chain enters Phase A.

- Phase A: The chain no longer transmits to the sink. Instead, it collects all its packets in the first node of the chain, which happens by time $4N_i - 1$.
- Phase T: From time $4N_i$ onward, the chain transmits a packet to the sink in every time step in which it is not blocked.

Note that a chain may complete its schedule in Phase 0, in Phase 1, in Phase 2, or in Phase T.

We can claim that the resulting schedule is close to optimal. We need the following lemmas.

Lemma 3. Consider a packet p that is sent to the sink in time step t. If p is sent by a chain C_i that is in Phase 0, Phase 1 or Phase 2, then p cannot be sent to the sink before time $t - 5$ in any schedule.

Proof. Let t be the j-th packet sent by the chain C_i to the sink. By Corollary 1, the packet cannot be sent to the sink before time $N_i + 3(j-1) - 3$ in any schedule. As C_i sends the packet in Phase 0, Phase 1 or Phase 2, we have $t = N_i + 3(j-1) + x$ for $x \in 0, 1, 2$, hence $t \leq N_i + 3(j-1) + 2$. The lemma follows. □

Lemma 4. A chain that is in Phase A is blocked in every time step of its Phase A.

Proof. The chain entered Phase A because it was blocked in steps N_i+3j_0, N_i+3j_1+1 and $N_i + 3j_2 + 2$. In each of these steps it was blocked by some longer chain that was in Phase 0, Phase 1 or Phase 2. If the chain was blocked by a longer chain in step $N_i + 3j_0$, it follows that it is also blocked by a longer chain in steps $N_i + 3(j_0 + 1), N_i + 3(j_0 + 2), \ldots, N_i + 3(N_i - 1)$. Similarly, it follows that the chain is blocked also in all other time steps from time $N_i + 3j_2 + 2$ to $N_i + 3(N_i - 1) + 2$ by some longer chain. □

Let t_C be the time step in which the sink receives the last packet. Let t' be the last time slot before t_C during which the sink does not receive a packet. (If there is no such time step, the schedule is clearly optimal.) Note that the sink receives a packet in every step from time $t' + 1$ to time t_C.

Lemma 5. Let p be any packet received by the sink in some time step from time $t' + 1$ to time t_C. The packet p cannot reach the sink before time $t' - 4$ in any schedule.

Proof. If p is sent by a chain that is in Phase 0, Phase 1 or Phase 2, the claim follows from Lemma 3. Hence, assume that p is sent by some chain C_i in Phase T. Note that C_i cannot have been in Phase A nor in Phase T at time t': If C_i was in Phase A, then by Lemma 4 it would be blocked in time t', a contradiction to time t' being a time step in which the sink does not receive a packet; and if C_i was in Phase T at time t', it would have transmitted a packet to the sink as it was not blocked at time t'. Therefore, we know that C_i was still in phase 0,1 or 2 at time t'. Assume that C_i had transmitted j packets to the sink before time t'. This implies $t' > N_i + 3(j-1)$ (the time when the j-th packet is sent in Phase 0) and $t' < N_i + 3j + 2$ (the time when the $(j+1)$-th packet is sent in Phase 2). Furthermore, the packets that C_i collects during Phase A and transmits during Phase T do not include the first j packets of C_i. By Corollary 1, none of the last $N_i - j$ packets of C_i can be transmitted before time $N_i + 3j - 3$. Since $t' \leqslant N_i + 3j + 1$, the packet p cannot be transmitted to the sink before time $t' - 4$ in any schedule. □

Theorem 6. The number of time steps that the algorithm needs to schedule a multi-chain is at most $OPT + 5$, where OPT is the optimal number of time steps.

Proof. If the schedule has no time step in which the sink does not receive a packet, the schedule is optimal. Otherwise, let t' and t_C be defined as above. By Lemma 5, none of the $t_C - t'$ packets transmitted to the sink from time $t' + 1$ to t_c can be transmitted to the sink before time $t' - 4$ in any schedule. Therefore, $t' - 4 + (t_C - t' - 1) = t_C - 5$ is a lower bound on the optimal schedule length, and we get $t_C \leq OPT + 5$. □

Algorithm 4 and 5 illustrate the implementation of the proposed schedule for unbalanced chains.

The description is similar to the algorithm for balanced chains with some modification. Firstly, all the chains should be arranged in descending order, with the longest chosen as a backbone chain. We already refer to the minimum time slot for that backbone chain in Chapter 3, Section 3.3.2.

Algorithm 4: PART ONE, UNBALANCED CHAINS.

Input: multi-chains $= l_1, \ldots, l_M$; $l_i = \{v_{(i,j)}, \ldots\}$, Sink, where $1 \leq i \leq M, 1 \leq j \leq length_(i)$

Output: S(t) for $t = 1, \ldots$

procedure Main( )
$flag \leftarrow 0$; $aux(i,j) \leftarrow 0$ for $1 \leq i \leq M, 1 \leq j \leq 3$;
$S(t) = \emptyset$ for $t = 1, \ldots,$ $p(i,j) \leftarrow 1$ for $1 \leq i \leq M, 1 \leq j \leq N, k = 0$;
$length(1) \geqslant length(2) \geqslant length(3), \ldots, \geqslant length(M)$, arrange all chains in descending order;
$length = \{length(1), length(2), \ldots, length_(M)\}$;
call **procedure Scan-All()**
call **procedure Procedure-three-slots()**
procedures scan-all()
for $j \leftarrow length(1)$ ***to*** 1 **do**
  $t \leftarrow t + 1$;
  for $i \leftarrow 1$ ***to*** M **do**
    if $t < length(i)$ **then**
      parallel(i,length(i)-k);
    else if $(t == length(i))\ and\ (flag == 0)$ **then**
      $s(t) \leftarrow s(t) \cup \{v(i,j)\}$;
      $p(i,j) \leftarrow p(i,j) - 1$, $Sink \leftarrow Sink + 1$;
      $flag \leftarrow flag + 1$; $aux(i,1) \leftarrow 1$;
      parallel(i,j+3);
    else if $(t \geqslant (length(i))\ and\ (flag == 1)$ **then**
      ControlThree(i,j);
    else if $(t > length(i))\ and\ (flag == 0)$ **then**
      if $(p(i,1) > 1)\ or\ (p(i,1) == 1)\ and\ (p(i,(2) == 0)$ **then**
        $s(t) \leftarrow s(t) \cup \{v(i,1)\}$;
        $p(i,1) \leftarrow p(i,1) - 1$;
        $Sink \leftarrow Sink + 1$; $flag \leftarrow flag + 1$;
        if $(aux(i,1) == 0)$ **then**
          parallel(i,4); $aux(i,1) \leftarrow 1$;
      else
        ControlThree(i,1);
  $flag \leftarrow 0, k \leftarrow k + 1$;
schedule the last three packets of the 1 and 2 chains;
Procedure-three-slots( )
while $p(1)(1) \neq 0$ **do**
  for $j \leftarrow 3$ ***to*** 1 **do**
    parallel(1,j) ;
    Concurrently other chains are used to fill the gap of two slots if they have packets.;
All the accumulated packets of the remaining chains are forwarded to the sink one after another.

Algorithm 5: PART TWO, COMPLEMENT OF THE FIRST PART.

```
1  Procedure-Parallel( i, j)
2  for j ← j to length(i) increment by 3 do
3  |  if p(i, j) ≠ 0 then
4  |  |  s(t) ← s(t) ∪ {v(i, j)};
5  |  |  p(i, j) ← p(i, j) − 1;
6  |  |  p(i, j − 1) ← p(i, j − 1) + 1;
7  Procedure-ControlThree( i, j)
8  if ((p(i, (j + 3)) ≥ 1) and (aux(i, 1) == 0)) then
9  |  j ← j + 3,  aux(i, 1) ← 1;
10 else if ((p(i, j + 2) ≥ 1) and (aux(i, 2) == 0)) then
11 |  j ← j + 2, aux(i, 2) ← 1, aux(i, 3) ← 0;
12 else if ((p(i, j + 1) ≥ 1) and (aux(i, 3) == 0)) then
13 |  aux(i, 1) ← 0, aux(i, 2) ← 0;
14 |  if (p(i, 2) ≠ 0) then
15 |  |  aux(i, 3) ← 1;// if node three and upwards still have packets
16 |  j ← j + 1;
17 parallel(i,j);
```

To simplify the explanation, this algorithm is divided into two steps:

Step1: This step includes one round which starts in line 8 and has length(1) (length(1) is the length of the longest chain) transmissions in order for the longest chain to send its first packet to the sink. At the beginning all the chains are arranged as a matrix similar to Example 4.3 and the lengths of all chains have been stored in an array (length). Then scheduling starts from the leaf nodes and the processing proceeds from the longest chain down to the shortest one in each time slot. During this process, when any chain is ready to send data to the sink they should send it to the sink (here we choose the longer chain among the ready chains, however we can choose the shorter one as well). In order to schedule as many nodes as possible with the length(1) transmissions of the longest chain, we use two for-loops (line 9,11). Inside the loops, four cases are used to control scheduling nodes as follows:

1. The first case is used to check each chain with the current step in line 12. If the length of the chain is smaller than current step it means that the chain is not ready to send data to the sink. Therefore, the node should send data to its parent.

2. The second case uses two conditions: the first is used to check whether the length of the chain is equal to the current time step (i.e., this chain is ready

to send a packet to the sink), while the second is used to ensure the sink did not receive any packets before this chain in line 14. Then the algorithm allows the first node of this chain to send data to the sink and uses variable $flag$ to remember that the sink has received data in line 17. Then the parallel procedure is called to scan and schedule previous nodes of the current chain and others with the current time slot.

3. The third case is used to handle another possibility of the chain. If the condition is satisfied in line 19, this implies that the sink has already received a packet from one of the previous chains. As a result, the chain cannot send data to the sink despite its readiness. Thus the only thing that can be done in this step is to scan previous nodes in line 20 of this chain to be scheduled in this time slot. The process of scanning previous nodes in this time slot should be used attentively, especially when the length of the chain is smaller than the current time slot. Therefore this process is performed via a control procedure ControlThree as follows:

   (a) The scanning process should start at either the fourth, third or the second node of the chain because the algorithm does not know which node has already made a transmission in the previous time slot. Moreover, the auxiliary array is utilised to control scheduling them. The first condition in line 8 checks the fourth node of the current chain, whether or not it is ready to send the packet to the next node. If the fourth node has made a transmission, its position in the auxiliary array becomes one to mark this. Then ordinarily the procedure parallel is called to scan previous nodes of this chain.

   (b) Similarly, the second condition in line 10 is used to check the third node of the chain.

   (c) The third condition is used to check the second node. It can be observed that the algorithm does not need to check the first node of this chain because one of the longer chains has already made a transmission to the sink, and due to the limited capacity of the sink to receive only one packet in each time slot, this chain cannot make a transmission to the sink. Note

that the extra condition in line 14 means that if the third node still has more packets, remark the second node that has forwarded the packet, and let the other nodes to do their turn. Otherwise let the second node to forward all its packets one after another.

Note that the extra condition in line 14 is necessary because if the third node has no more packet, the algorithm lets the second node to forward all its packets one after another.

4. Case four in line 21 means that none of the longer chains is ready to transmit a packet to the sink. As a result, the shorter chains that have already been passed should be checked to know whether there is a packet to be sent to the sink. If one of the chains has satisfied the condition in line 22, it can send a packet to the sink and procedure parallel is called to scan previous nodes of this chain. However, if this chain does not have data to send to the sink, it will call the ControlThree procedure in line 29 to check the previous nodes of this chain.

Step2: Then we come to the three time slot group of the backbone chain in line 32. These three time slots are performed many times until the first node of the backbone chain is running out of packets. In this three-time slot group, the backbone chain can send a packet to the sink at the end of this group, which is the third time slot. However, we can make the sink busy in the first and the second time slot of these three time slots by sending data from other chains. (We prefer to send data from the shorter chains to shut them down sooner than the other longer chains to save more energy).

After the end of the schedule for the backbone chain, we use another for loop to send the remaining data of the remaining chains that accumulated in their first nodes.

Note: For unbalanced chains, the expected amount of idle listening is calculated as follows:

$$\sum_{j=1}^{M}\sum_{i=1}^{N_j}(1-p)^{N_j-i+1}$$

Recall that N_j is the length of each chain and M is the total number of chains.

4.3 Balanced k-ary Tree

In this section we proceed to find the optimal extra-bit schedule for a balanced k-ary tree. Let G be a graph consisting of N nodes that are deployed and arranged as a balanced k-ary tree. A balanced k-ary tree is a tree where each single parent node including the root has exactly k children. The root of the tree is the sink to which all other nodes send their packets. The tree is divided into d levels where the level of the root node is labelled as level 0, the next one as level 1 and the last one as level d as shown in Figure 4.4. The balanced k-ary tree is also analysed in a different way in [24]; the authors explained when each level starts and ends its transmission in each round of data collection for the case when each node has data with probability p and demonstrates the expected latency, then they analysed the energy consumption for the three states of the transceiver (transmitting, receiving and idle state). In this section, we analyse and derive a lower bound for a k-ary tree, which is composed of three or four levels.

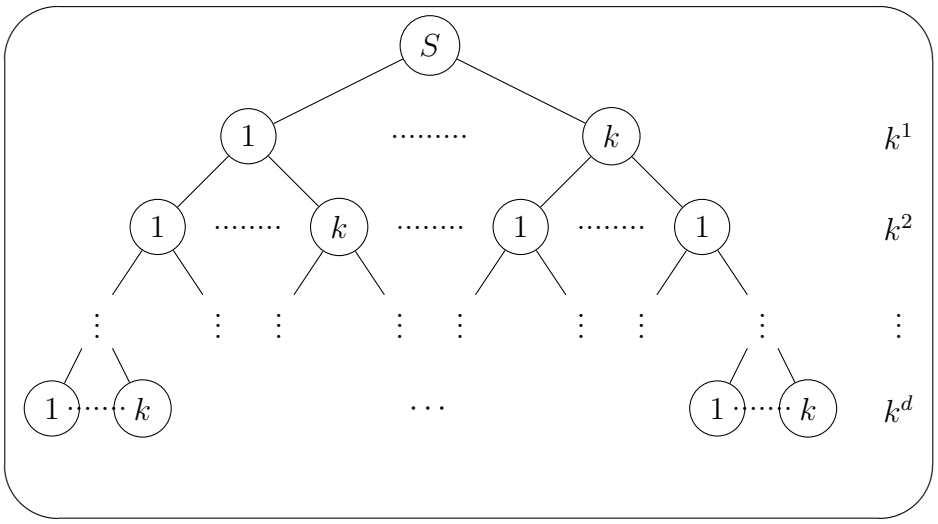


Fig. 4.4 The structure of a balanced k-ary tree.

Let T_i be the total number of nodes in the tree extending from one parent node at level i, for $i = 1, \ldots, d$ and $k \geq 2$.

As we know that

$T_d = 1$.

$T_{d-1} = T_d k + 1 = 1 + k$.

$T_{d-2} = T_{d-1} k + 1 = 1 + k + k^2$.

$\vdots$

$T_{d-i} = T_{d-(i-1)}k + 1 = 1 + k + k^2 + \cdots + k^i$.

After some simplification the general formula is

$$T_i = \frac{k^{d-(i-1)} - 1}{k - 1} \tag{4.1}$$

4.3.1 Balanced Three and Four Level k-ary Tree

In this section so far, we can confirm and show the minimum number of steps that are required for data collection in a balanced three or four-level k-ary tree. Note that we do not consider cases $d = 1$ and $d = 2$ because they are trivial.

Theorem 7. The length of any extra-bit schedule for balanced k-ary trees, $k \geq 2$, is at least $k(d + 1 + k + k^2)$ if $d = 3$ and at least $k(d + 2 + 2k + k^2 + k^3)$ if $d = 4$.

Proof. We consider first the case $d = 3$, and second, the case $d = 4$. Consider $d = 3$, meaning that there are three levels excluding the root (sink).

In the first k steps, only the nodes of level 3 can make transmissions. Assume that step $k + t$ is the first step in which a node at level 1 makes its $(k + 1)$-th transmission. Let v be that node at level 1. It follows that each of the k children of v must have made all its $k + 1$ transmissions to v before step $k + t$. Besides, v must have made k transmissions before step $k+t$. All these transmissions must have been made after step k, and no two of them can have been made in the same time step. Therefore, we have $t \geq k(k + 1) + k + 1 = 1 + 2k + k^2$.

After step $k + t$, node v still has to make k^2 further transmissions, and the $k - 1$ other nodes at level 1 still have to make at least $k^2 + 1$ transmissions each. No two transmissions by nodes at level 1 can be made in the same time step. Therefore, at least $k^2+(k-1)(k^2+1) = k^3+k-1$ steps are needed after step $k+t$. This means that the length of the schedule is at least $k+t+k^3+k-1 \geq k+(1+2k+k^2)+k^3+k-1 = 4k + k^2 + k^3$.

Now consider $d = 4$, at least $(d - 1)k$ time slots are required to enable the first level to make the first k transmissions. We can argue that during the $(d - 1)k$ transmissions, at most, $k - 1$ parallel transmissions can happen at the third level, among the k children of a node at the second level; that is, there is exactly one

node among the children of each node of level 2 at level 3 that cannot make the parallel transmission with the second level. This is due to the fact that a child in the third level cannot make a transmission when its parent at the second level makes a transmission. In addition, with k transmissions of the first level, at most $k-2$ concurrent transmissions can happen at the second level. This is because one transmission of the third level, which is left, needs to be completed. Therefore, each parent node at the second level is delayed by two (i.e., each parent node can only receive from $k-2$ children). Hence, two extra time slots for T_3-T_4 times are required to enable the first level to continue until the third level is running out of packets. When the third level has finished, there are at least T_2-T_3 packets left at level two and only one of them can be schedule per time slot, therefore one extra time slot, for T_2-T_3 times (i.e., after T_2-T_3 times, the second level has no more packets), is required until the second level is finished. In other words, at most $k-1$ transmissions can happen at the second level with k transmissions of the first level whenever there is no restriction on the second level by the third level. Finally, the first level can make all the transmissions. Hence $(d-1)k+(T_3-T_4)2+T_2-T_3+T_1k=k(d+2+2k+k^2+k^3)$ is the lower bound. □

Note that we do not construct schedules to match the lower bounds for $d = 3$ and $d = 4$, because they are very straightforward.

4.4 Extra-Bit Schedules with Two Frequencies

In this section, we consider a setting where two frequencies (channels) are available and transmissions made on different frequencies do not interfere with each other. We assume that each sensor node has a single radio that can be tuned dynamically to either of the two frequencies. The sink is assumed to have two radios, so that it can receive two packets sent on different frequencies simultaneously.

The availability of two frequencies increases the potential for simultaneous transmissions. For example, consider a chain network. With just one frequency, simultaneous transmission from node v_{i+1} to node v_i and from node v_{i+3} to v_{i+2} is not possible because the transmission from v_{i+1} to v_i creates interference at v_{i+2}. With two frequencies, these two transmissions can be made simultaneously provided they

use different frequencies. Therefore, it is no longer necessary for node v_{i+2} to remain idle while v_{i+1} transmits to v_i. In the following we show how to exploit this capability for chains and balanced multi-chains.

In other words, based on the above explanation, it can be observed that when a single frequency is used each node passes through three states: transmission(T), reception(R) and Idle(I), as shown in Figure 4.5. This implies that each node, after passing through these three states, repeats this cycle again and again until it runs out of packets. In this example there are five nodes, and transmission initially starts at the furthest node from the sink, which is node 5, to the node next to it according to the extra-bit condition and others are in the inactive state. In step one, node 5 transmits to node 4, this implies that node 5 is in transmit state and node 4 is in receive state. In step 2, node 5 is in the idle state while both node 4 and 3 are in the transmit and receive states together. In step 3, node 5 turns to receive state but it cannot receive any packet because it is a leaf node while node 4 is in the idle state to avoid interference, whereas node 3 and 2 are in the transmit and receive states together. This process continues until the sink receives all packets. Furthermore, each node repeats this cycle after three steps.

One can notice that using two frequencies is sufficient to reduce three states to two states i.e., transmission(T), reception(R) as shown in Figure 4.6. Consequently, the length of the schedule is reduced substantially and no idle state exists any more.

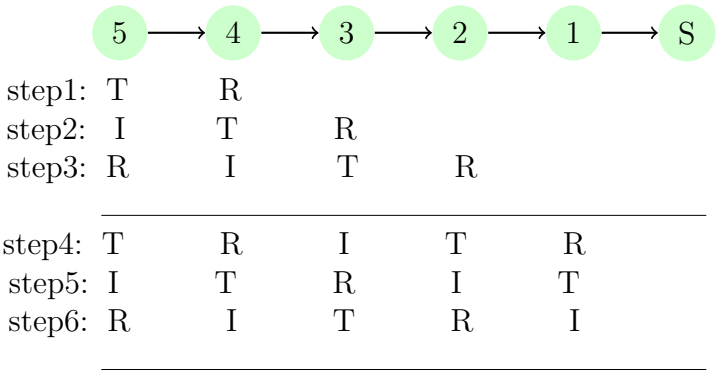


Fig. 4.5 States of the nodes until the sink receives the first packet in linear network with single frequency.

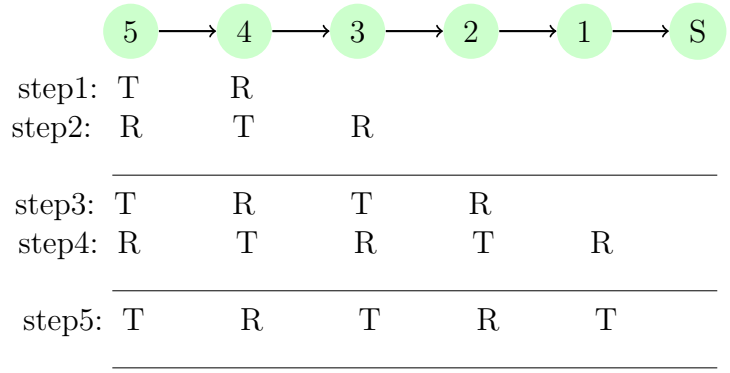


Fig. 4.6 States of the nodes until the sink receives the first packet in linear network, with two frequencies.

4.4.1 Single Chain

In the case of a single frequency, we know from Theorem 2 that the optimal length of an extra-bit schedule for a chain with N nodes is $4N - 6$. We now show that the use of two frequencies reduces the optimal schedule length to $3N - 3$ time slots. First, we show the lower bound.

Theorem 8. Consider a chain with N nodes and the sink. Any extra-bit schedule with two frequencies requires at least $3N - 3$ time steps.

Proof. For $1 \leq i \leq N$, let v_i be the node of the chain that is i hops away from the sink. The extra-bit technique requires that every node v_i for $1 \leq i \leq N - 1$ must receive a packet from v_{i+1} before it can make its first transmission. Therefore, node v_2 cannot receive its first packet from v_3 before time step $N - 2$, and neither node v_2 nor v_1 can make any transmission before time $N - 1$. After time step $N - 2$, node v_2 must make $N - 1$ transmissions and node v_1 must make N transmissions, and no two of these transmissions can be made simultaneously as v_1 has only one radio. Therefore, $2N - 1$ further time steps are required after the first $N - 2$ time steps, giving a lower bound of $3N - 3$ time steps overall. □

Theorem 9. For chains with $N \geq 2$ nodes and the sink and two available frequencies, there is a feasible extra-bit schedule of length $3N - 3$.

Proof. We present an algorithm that produces a schedule of length $3N - 3$. The pseudo-code of the algorithm is shown in Algorithm 6. The algorithm is similar to the algorithm for the single-frequency case, but nodes making simultaneous transmissions only have a distance of 2 instead of 3. Nodes need to alternate frequencies

Algorithm 6: Optimal extra-bit scheduling algorithm for chains using two frequencies.

Input: Chain with N nodes $v_1, \ldots, v_N$ and the sink v_0
Output: S(t) for $t = 1, 2, ..., 3N - 3$

```
1  S(t) ← ∅    for t = 1, 2, ..., 3N − 3 ;
2  p(i) ← 1    for i = 1, 2, ..., N; // packets at v_i
3  t ← 0;
4  call ScheduleFirstPart();
5  call ScheduleSecondPart();
6  procedure ScheduleFirstPart()
7  for i ← N down to 1 do
8      t ← t + 1;
9      parallel(i);
10 procedure parallel(i)
11 for j ← i to N increment by 2 do
12     if p(j) ≠ 0 then
13         p(j) ← p(j) − 1;// send a packet
14         p(j − 1) ← p(j − 1) + 1;// receive a packet
15         s(t) ← s(t) ∪ {v_j};
16 procedure ScheduleSecondPart()
17 while p(1) ≠ 0 do
18     for i ← 2 down to 1 do
19         if p(i) ≠ 0 then
20             t ← t + 1;
21             parallel(i);
```

to avoid interference. Observe that the algorithm never schedules transmissions of node v_i and v_{i+1} simultaneously for any i, hence there is no interference between simultaneous transmissions. The sink receives the first packet in time step N, then $N - 2$ further packets with a gap of one time step before each reception. The last packet is then received without a gap. The total number of time steps is $N + 2(N - 2) + 1 = 3N - 3$. □

4.4.2 Balanced Multi-Chains

For extending our algorithm for two frequencies from a single chain to balanced multi-chains, we follow similar ideas as in Section 4.1, but the sink can now receive two packets simultaneously on different frequencies. Let N be the number of nodes in each chain, and M the number of chains.

Let OPT_2 denote the optimal schedule with two frequencies constructed by Al-

gorithm 6 for a single chain with N nodes. The length of OPT_2 is $3N-3$. Observe that the node adjacent to the sink sends $N-1$ packets to the sink in time steps N, $N+2$, ..., $N+2(N-2) = 3N-4$, and then the last packet in time step $3N-3$.

Theorem 10. For a balanced multi-chain with $M = 2$ chains and N nodes per chain, the optimal length of an extra-bit schedule with two frequencies is $3N-3$.

Proof. The lower bound of $3N-3$ follows because a multi-chain with two chains of length N cannot admit a shorter schedule than a single chain of length N. For the upper bound, simply let both chains use schedule OPT_2, but exchange the two frequencies in the second chain. Both chains will complete their schedule in $3N-3$ time steps. The two chains will send their packets to the sink simultaneously but on different frequencies, so the schedule is feasible. □

Similarly to Lemma 2, we can prove the following.

Lemma 6. Consider any extra-bit schedule with two frequencies for a single chain with N nodes and the sink. For $1 \leq i \leq N-1$, the i-th packet received by the sink cannot be transmitted to the sink before step $N+2(i-1)$.

Theorem 11. For a balanced multi-chain with $M = 3$ chains and N nodes per chain, the optimal length of an extra-bit schedule with two frequencies is $3N-2$.

Proof. First, we prove the lower bound. By Lemma 6, the $(N-1)$-th packet of each chain cannot be transmitted to the sink before time $N+2(N-2) = 3N-4$. Hence, at the start of time step $3N-4$, at least six packets (two from each chain) have not yet been transmitted to the sink, and at least three further time steps are needed because at most two packets can be sent to the sink in each step. Hence, the length of any feasible extra-bit schedule is at least $3N-2$.

Now, we construct a schedule of length $3N-2$. The first chain follows schedule OPT_2. The second chain uses a schedule S_2 that is obtained from OPT_2 by exchanging the two frequencies and inserting an idle time step just before the last packet is sent to the sink. The third chain uses a schedule S_3 that is obtained from OPT_2 by inserting an idle time step at the start and by using frequency 2 instead of 1 for the transmission from the first node in the chain to the sink in time step $3N-3$ (to avoid a collision with the simultaneous transmission from the first chain to the sink).

The three chains make their transmissions to the sink at the following times:

OPT_2: N, $N+2$, ..., $3N-6$, $3N-4$, $3N-3$
S_2: N, $N+2$, ..., $3N-6$, $3N-4$, $3N-2$
S_3: $N+1$, $N+3$, ..., $3N-5$, $3N-3$, $3N-2$

We have a feasible schedule of length $3N-2$. □

Theorem 12. For a balanced multi-chain with $M \geq 4$ chains and N nodes per chain, the optimal length of an extra-bit schedule with two frequencies is $N-1+\lceil MN/2 \rceil$.

Proof. First, we prove the lower bound. By Lemma 6, the first packet of each chain cannot be transmitted to the sink before time N. The total number of packets that must be transmitted to the sink from time N onward is MN, and at most two packets can be sent to the sink in each time step. Therefore, the total number of time steps for any extra-bit schedule is at least $N-1+\lceil MN/2 \rceil$.

Now, we construct a schedule whose length matches the lower bound. If M is even, the following schedule is used: The first two chains use a schedule S' that is obtained from OPT_2 by inserting an idle time step just before the last packet is sent to the sink. The third and fourth chain use a schedule S'' that is obtained from S' by adding one idle time step at the start. The second and fourth chain exchange the two frequencies, compared to the first and third chain, respectively. The sink receives two packets from the first two chains in steps N, $N+2$, ..., $3N-4$, $3N-2$, and two packets from the third and fourth chain in steps $N+1$, $N+3$, ..., $3N-3$, $3N-1$. The remaining $M-4$ chains collect all their packets in their first nodes in the first $3N-3$ steps. The $M-4$ chains are grouped into pairs. From time $3N$ onward, in each time step two chains from one pair of chains each transmit one packet to the sink, requiring $(M-4)N/2$ further time steps. The total schedule length is $3N-1+(M-4)N/2 = N-1+MN/2 = N-1+\lceil MN/2 \rceil$, as required.

Finally, consider the case that $M \geq 5$ is odd. The first two chains follow the schedule S' of length $3N-2$ defined above. The next three chains, denoted by C_3, C_4, C_5, follow the schedule S'' defined above, but in every time step where all three chains want to send a packet to the sink, one of them does not transmit to the sink and instead keeps the packet at the first node of the chain. The two chains among C_3, C_4, C_5 that are selected for transmitting to the sink are chosen

by selecting C_3, C_4 the first time, C_4, C_5 the second time, C_3, C_5 the third time, and then repeating the pattern. This ensures that the number of remaining packets always differs by at most 1 between the three chains C_3, C_4, C_5. Whenever two chains send a packet to the sink in the same time step, the two frequencies can be exchanged for one of the two chains for that time step to avoid a collision at the sink. The remaining $M - 5$ chains, if any, use the first $3N - 3$ time steps to collect all their packets in their first nodes. By the end of time step $3N - 1$, the situation is as follows:

(1) The first two chains have transmitted all their packets to the sink.

(2) The three chains C_3, C_4, C_5 have together transmitted $2N$ packets to the sink, and each of the three chains has $\lfloor N/3 \rfloor$ or $\lceil N/3 \rceil$ remaining packets that have all been collected at the first node of the chain.

(3) In each of the remaining $M - 5$ chains, all N packets have been collected in the first node of the chain.

Furthermore, the sink has received two packets in every time step from time N to time $3N - 1$. The rest of the schedule is now determined as follows: In the next $\lceil N/2 \rceil$ time steps the chains C_3, C_4, C_5 transmit their N remaining packets to the sink, continuing the pattern of selecting C_3, C_4 for transmission to the sink in one step, C_4, C_5 in the next, and C_3, C_5 in the next. After that, the remaining $M - 5$ chains are paired, and in each step the two chains of a pair transmit one packet each to the sink. This requires $N(M - 5)/2$ time steps.

The total length of the schedule is $3N - 1 + \lceil N/2 \rceil + N(M - 5)/2 = N/2 - 1 + \lceil N/2 \rceil + MN/2$. If N is even, this is equal to $N - 1 + MN/2 = N - 1 + \lceil MN/2 \rceil$. If N is odd, this is equal to $N/2 - 1 + (N + 1)/2 + MN/2 = N - 1 + (MN + 1)/2 = N - 1 + \lceil MN/2 \rceil$. In either case, the length of the constructed schedule is $N - 1 + \lceil MN/2 \rceil$. □

4.5 Rhizome Tree

When we are dealing with a network that comprises more than just a linear chain, different considerations must be taken into account. Such an example would be where there is a linear topology, with each node in the main chain having arbitrary

numbers of leaf nodes (neighbours). Such a topology could be called a Rhizome, due to its particular structure, with sub-chains branching off a main chain. Furthermore, this Rhizome tree can be represented as a graph $G = (V, E)$, where $V = V_{mc} \cup V_l$. V_{mc}=$\{v_0, v_1, v_2, ..., v_N\}$ represents nodes in the main chain. These are arranged with a linear topology (chain). Moreover, these V_{mc} nodes are numbered from 0 to N, where v_0 is the sink. Furthermore, v_N is the furthest node from the sink and node v_1 is the closest node to the sink. V_l=$\{V(i,j)|1 \leqslant i \leqslant N, 1 \leqslant j \leqslant d_i\}$ where each node v_i has an arbitrary number of leaf nodes d_i.

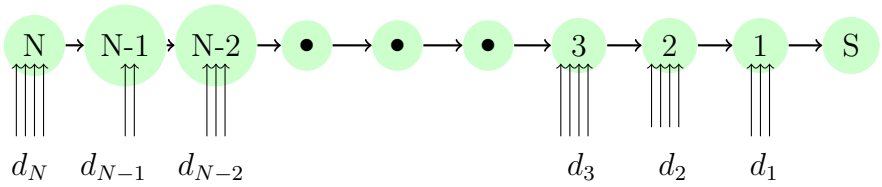


Fig. 4.7 General structure of the Rhizome tree.

Figure 4.7 illustrates the topology under consideration. The activity of each node in the backbone chain should be informed by 3 items of information: The degree (d), capacity (c) and remainder (r) for each node in the main chain. Information about the latter two can be derived from the degree. The degree is the number of neighbours or children of each main-chain node and is given by d_i. The capacity is the ability of a node to receive data from its children in parallel with transmissions from the previous nodes in the chain, without requiring extra time slots. Where the capacity is insufficient, the remainder is a number indicating how many extra time slots are required for data from all the node's children to be collected in addition to its child node in the backbone chain (i.e., v_i to v_{i-1}), triggering the main-chain node to begin transmitting data to its parent.

Definition 1: The capacity of each node is calculated from the capacity of the previous node in the chain, plus any extra time slots required (the remainder of the previous node), plus one further time slot, to account for the transfer of data from the previous node in the main chain.

We can specify the capacity of each node by this formula: $c_i = c_{i+1} + r_{i+1} + 1$, where $i = 1, 2, 3, ..., N-1$.

Definition 2: The remainder can be calculated from the degree of a node, minus its capacity and can be expressed as $r_i = \max\{d_i - c_i, 0\}$. When the remainder $\leqslant 0$,

it means that no extra time slots are required.

Note: We assume that there are N main nodes in the network in addition to the sink. Furthermore, all formulas in this section are correct when $N \geq 3$.

There are two exceptions to this rule; the furthest node from the sink is v_N, and the node next to the final node is v_{N-1}. Being the final node, v_N does not need to adjust itself to transmissions of previous nodes in the chain. Therefore, the remainder of v_N is equal to its degree ($r_N = d_N$) and its capacity is zero ($c_N = 0$). In other words, v_N has already received all the packets from the previous nodes and none of its children had a chance to be scheduled. Correspondingly, its children must be scheduled in separate time slots. Secondly, the capacity of v_{N-1} is equal to the degree of the final node ($c_{N-1} = r_N = d_N$). Once the final node has received all the transmissions of its children, it begins transmitting without delay.

The above paragraphs provide a general description of the Rhizome tree. However, there are two scenarios where this approach can be used and the formula applied is different in each. Scenario 1 is where the remainder for the first node in the chain, nearest to the sink, is zero. Scenario 2 is where this node has a remainder greater than zero, where the formula of the first scenario is redefined. At the end of each scenario an example is given to provide more intuition how the definitions and rules work.

4.5.1 How the Algorithm Works

Before explaining the formula for each scenario, we illustrate how the basic idea of the algorithm works and how the scheduling of data collection can be performed optimally in the Rhizome tree. Then, for further understanding at the end of this section the pseudocode is given.

1. First of all, the data collection starts from the last node of the main chain and proceeds towards the sink. Namely, for $i = N, N-1, N-2, ..., 1$, where v_i transmits to v_{i-1}. Furthermore, each node v_i must listen to its remaining leaf nodes in distinct, successive time slots in order to receive data from them in addition to its previous node v_{i+1} in the main chain; then it can pass a packet to the next node in the main chain. In other words, at the beginning, for each node v_i there are two constraints in order to send a transmission: firstly, the

node must listen to its previous node in the main chain, and secondly it must listen to all of its remaining leaf nodes (neighbours). This node generally can make another transmission after three time slots following its previous transmission without listening to its neighbours again because it has collected data from all of them before making its first transmission. However, it is possible to make the next transmission at exactly three time slots following its previous transmission when node v_{i-2} is busy with its leaf nodes as is explained in the description of the backward scanning process provided in the next paragraph.

Ordinarily, during each listening period, the algorithm must scan the network in both directions (forwards and backwards) from the current node v_i in order to enable concurrent transmissions, which leads to a reduction in the length of the schedule for the whole network. These two scanning processes should be used carefully as follows:

(a) **Forward scan:** It is performed to help subsequent nodes in the main chain to collect data from their leaf neighbours in parallel with node v_i. There are two cases in which forward scanning can be used.

   i. Case 1: When node v_i listens to its leaf nodes, the forward scanning processes nodes in the order $v_{i-1}, ..., v_1$.

   ii. Case 2: Conversely, when the node v_i transmits data to node v_{i-1} in the main chain, forward scanning processes the nodes from $v_{i-2}, ..., v_1$ to avoid interference from the other nodes.

(b) Similarly, **a backward scan** is performed to help previous nodes in the main chain to send parallel transmissions with node v_i and must be used attentively. Each node should have two extra variables in order to perform backward scanning successfully. The first one is called **re** which determines the number of packets that have been received by each node v_i from the previous node v_{i+1}. It is initially set to zero for all nodes ($re(i) = 0$, for $i = 1, \ldots, N-1$) except the last one, which is initially set to the number of obtainable packets available at the node v_N (i.e., $re(N) = d_N + 1$). This variable is used to control the backward scanning,

i.e., where to start. Moreover, it increases when its node receives a packet from the previous node and decreases when its node sends a packet.

The second variable is **st**. Its value is initially set to one for all nodes ($st(i) = 1,\ \text{for}\ \ i = 1, \ldots, N-1$) except for the last node ($st(N) = 0$). This implies that only the previous node of the last one has finished transmitting. This variable points out whether the previous node has finished transmitting, and it also helps stopping further backward scanning.

When each node v_i has forwarded the last packet to v_{i-1}, v_{i-1} must adjust its re to $re = re + (d_{i-1} + 1)$ and its st to 0. The number of received packets in v_{i-1} is thus equal to the number of received packets from node v_i, in addition to the number of available packets at the node itself. Notably, before v_i transmits a packet to v_{i-1}, its re and st are checked. Accordingly, the node v_{i-1} knows whether to adjust its variables (i.e., $re(i-1)$ and $st(i-1)$).

The above description provides a general idea about backward scanning, and again there are two cases in which backward scanning can be performed:

i. Case 1: When the current node v_i is busy with its leaf nodes, the backward scanning process should start at v_{i+2}. First the algorithm should check the number of available packets that have been received by v_{i+2}. For this purpose, re and st are used and there are four conditions.

   A. If both of v_{i+2}'s variables re and st have zero values ($re = 0$ and $st = 0$), then the algorithm should cease backward scanning. This is because v_{i+2} itself, and all previous nodes in the chain, have finished transmitting.

   B. When variable re is not zero and st has value zero ($re \neq 0$ and $st = 0$) for v_{i+2}, the backward scanning process starts from this node and there is no need to check other nodes, meaning that all the previous nodes have already forwarded their packets.

   C. If ($re \geqslant 1$ and $st = 1$) for node v_{i+2}, this implies that this node has a packet to forward to the next node. And then the algo-

rithm should repeat the same process of checking three hops away from this node in order to maximize the parallel transmissions of the previous nodes with the current leaf of v_i , which means that the algorithm must start checking three steps further back. The purpose of going back three steps further is to avoid interference. If the algorithm has focused three steps further back and the node cannot satisfy conditions (i.e., it has not received a packet), then the algorithm should greedily check another node. Notably, checking three nodes further (i.e., v_{i+2}, v_{i+3} and v_{i+4}) from the activity of the current one is sufficient, because when v_{i+2} forwards a packet to v_{i+1}, at the same time, the node v_{i+4} can receive a packet from v_{i+5}. The same argument is true for the other two nodes v_{i+3} and v_{i+4}. If these three nodes have no data, this implies that the other nodes have already transmitted their packets.

D. Similarly, if $re = 0$ and $st = 1$, this means that the node has not received a packet, and therefore, the algorithm should check another previous node, which means v_{i+3}, for the aforementioned conditions (i.e., re and st) and again apply the same process. Similarly, if the node v_{i+3} has not satisfied the aforementioned conditions on re and st, then the algorithm has to check v_{i+4}.

There are two reasons for using backward scanning in general: first, the algorithm should aim to forward more packets from the node as soon as possible so as to turn its radio off sooner, thereby conserving energy. Secondly, forwarding more packets leads to a reduction in the length of the schedule, especially in the second scenario.

ii. Case 2: When v_i transmits data to the next node in the main chain v_{i-1}, the backward scanning thus starts three hops further (i.e., v_{i+3}, v_{i+6},...,v_N) from the activity of the current node v_i, in order to avoid interference. However, if v_{i+3} has already forwarded its received packet, then the algorithm can check another previous node greedily to maximize parallel transmissions.

2. After receipt of the first packet by the sink, then only the first three nodes v_3, v_2 and v_1 , which are close to the sink, are required to be scheduled separately many times until they finish transmitting all the remaining packets in the network. Furthermore, these three nodes can never be scheduled together due to interference and all the remaining previous nodes are scheduled with these three nodes concurrently until they finish transmitting all the packets as well. In other words, when the sink receives another packet the third node is ready to start transmitting again.

   Note: When we are in the second scenario, sometimes due to the remainder of the first node, the third node may forward more received packets to the second node, at that time. When the first three nodes are scheduled, scheduling may start from the second node for many times. This is the main difference between the first and the second scenario.

4.5.2 Scenario 1

Scenario 1 is where the remainder for the first node in the chain, nearest to the sink, is zero (i.e., when $r_1 = 0$).

The following formula indicates the minimum number of time slots that is required to collect all the packets from the Rhizome tree.

$$\underbrace{\underbrace{N-3}_{\text{term 1}}+\underbrace{\sum_{i=3}^{N} r_i}_{\text{term 2}}}_{\text{combine 1 and 2}}+\underbrace{(N-2+\sum_{i=3}^{N} d_i)}_{\text{term 3, Third Node}}+\underbrace{(N-1+r_2+\sum_{i=2}^{N} d_i)}_{\text{term 4, Second Node}}+\underbrace{(N+\sum_{i=1}^{N} d_i)}_{\text{term 5, First Node}} \quad (4.2)$$

1. The first term is the earliest possible time slot when the first packet can reach the third node in the main chain without accounting for the degree.
2. The second term indicates the number of extra time slots that are required for the degree of the nodes starting from node v_N up to node v_3. Combined with the first term, this indicates the earliest possible time slot in which the third node receives the first packet from $_4$ and from all leaf neighbours of v_3 to v_3.
3. The third term is the number of packets that are forwarded by the third node.

4. The fourth term is the number of packets that are forwarded by the second node plus the extra time slots to collect data from the remaining children. The possibility of having r_2 extra time slots in the second node comes from the fact that this node may have a huge number of children and it could not finish listening to all of them during the previous transmissions.

5. The fifth term is the number of packets that is forwarded by the first node. This term indicates no extra time slots, like the fourth term, because the first node has finished data collection in parallel with the previous nodes.

Theorem 13. The formula(4.2) is a lower bound on the optimal number of time slots to collect data in the Rhizome tree for any extra-bit algorithm (first scenario).

Proof. For ease of understanding we break the above formula into two parts.

1. In the first part,

$$\text{Let } \; T_{v_{N+1-i}} = i + \sum_{j=N+1-i}^{N} r_j \text{ , for } \quad i = 1, \ldots, N.$$

We assume that the schedule starts at time 1.

Claim: node v_{N+1-i} cannot make its first transmission before time $T_{v_{N+1-i}}$. We can prove this part by induction.

Base case: for $i = 1$.

$$\begin{aligned} T_{v_{N+1-1}} &= 1 + \sum_{j=N+1-1}^{N} r_j \\ &= 1 + r_N \end{aligned} \tag{4.3}$$

Induction hypothesis: Suppose that this is true for $i = k$, then we get the following

$$T_{v_{N+1-k}} = k + \sum_{j=N+1-k}^{N} r_j \tag{4.4}$$

Induction step: We should prove that the formula is true for $i \to k + 1$.

We substitute each i by $k+1$, hence left hand side $= T_{v_{N+1-(k+1)}}$

Note that node $v_{N+1-(k+1)}$ needs one more step in order to receive data from the previous node v_{N+1-k} and, further, $r_{N+1-(k+1)}$ extra steps to collect data from the remaining leaf nodes before it can transmit.

these are required because it is obvious that there are T_{N+1-k} steps before node $v_{N+1-(k+1)}$ and this node can collect packets from its leaf nodes in parallel with these T_{N+1-k} steps. Furthermore, these T_{N+1-k} steps intuitively indicate how many parallel transmissions of node $v_{N+1-(k+1)}$ can be performed concomitantly. So, as stated before in definition 2 , $r_{N+1-(k+1)} = max\{d_{N+1-(k+1)} - c_{N+1-(k+1)}, 0\}$. Hence $r_{N+1-(k+1)} = max\{d_{N+1-(k+1)} - T_{N+1-k}, 0\}$. Therefore

$$T_{v_{N+1-(k+1)}} = \underbrace{T_{v_{N+1-k}}}_{\textbf{term 1}} + \underbrace{r_{N+1-(k+1)} + 1}_{\textbf{term 2}}$$

$$= k + \sum_{j=N+1-k}^{N} r_j + r_{N+1-(k+1)} + 1 \quad \text{by equation (4.4)}$$

$$= (k+1) + \sum_{j=N+1-(k+1)}^{N} r_j$$

In general, the above formula is true for all i, $1 \leq i \leq N$.

Therefore, it can be concluded that the earliest time for the first node (i.e., v_1) to make the first transmission or the earliest possible time for the sink to receive the first packet is equal to

$$T_{v_1} = \sum_{j=1}^{N} r_j + N$$

2. After receipt of the first packet by the sink, the Rhizome tree becomes a single chain with the possibility of having more than one packet per node (i.e., an arbitrary number of packets). Similar to a single chain as it has been shown before, only the first three nodes v_3, v_2 and v_1, which are close to the sink, have to be scheduled individually in three distinct time slots until they finish their

packets. This is because these three nodes can never be scheduled together due to interference, and the previous nodes are scheduled with these three nodes in parallel.

It is obvious that after one packet arriving at the sink at T_{v_1}, v_3 is responsible for forwarding $N-3$ packets in the main chain. Likewise, v_2 should forward $N-2$ packets of the main nodes. v_1 similarly forwards $N-1$ packets. Therefore $3N-6$ time slots are needed for the main chain without degree and extra time slots for the degree of the nodes. Arguably, if an algorithm sends the first packet to the sink after T_{v_1}, it cannot have a shorter schedule.

The third, second and first nodes require extra time slots to forward the extra packets both of itself and the previous nodes; therefore, v_3 must pass $\sum_{i=3}^{N} d_i$ extra packets while v_2 and v_1 must forward $\sum_{i=2}^{N} d_i$ and $\sum_{i=1}^{N} d_i$ extra packets, respectively. These packets come from the secondary chains of each node, and are then transmitted along the Rhizome tree. Hence $T_{v_1} + 3N - 6 + \sum_{i=3}^{N} d_i + \sum_{i=2}^{N} d_i + \sum_{i=1}^{N} d_i$ is a lower bound

□

Note: Any packet from the third node and beyond by Lemma 2 needs three time slots until it reaches the sink. Similarly all the degrees of the second and first nodes need two and one time slots until they arrive at the sink.

Theorem 14. The proposed algorithm for Rhizome trees is optimal for $N \geq 3$.

Proof. The constructed schedule proposed by the algorithm has the following properties:

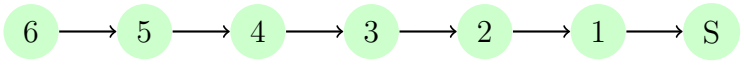
1. Every node v_i makes $(N+1-i) + \sum_{k=i}^{N} d_k$ transmissions.
2. Every node v_i, after collecting all the packets from its children, makes its j-th transmission only after it has received the j-th packet from its child in the main chain (i.e., v_{i+1}) or after receiving all packets from its child in case $|T_{v_{i+1}}| < j$.
3. The senders of simultaneous transmissions of the main chain are at least three hops away from each other. Therefore there is no inference between them.

It is obvious that the algorithm sends packets in the way considered in the induction in the lower bound proof without extra time steps. Namely, the sink receives the first packet at time

$$\sum_{i=1}^{N} r_i + N$$

When the first packet has arrived at the sink, this implies that all the packets from leaf neighbours of the main nodes have been collected. Thus the algorithm has to only schedule v_3, v_2 and v_1, in three distinct time slots due to interference respectively similar to the single chain. The previous nodes can be scheduled in parallel with them. Therefore the sink receives one packet in every three time slots according to the algorithm until the third node has no more packets left. This implies that v_3 must pass $N-3+\sum_{i=3}^{N} d_i$ packets and each one requires three time slots to reach the sink. Similarly, the sink receives one packet in every two time slots as shown in the algorithm until the second node has no more packets left, thus v_2 must forward $1+d_2$ packets and each one requires two time slots. Finally, v_1 must forward $1+d_1$ packets and each one requires one time slot. The total schedule length therefore is $(\sum_{i=1}^{N} r_i + N) + 3(N-3+\sum_{i=3}^{N} d_i) + 2(1+d_2) + 1 + d_1 = (\sum_{i=1}^{N} r_i) + 3\sum_{i=3}^{N} d_i + 2d_2 + d_1 + 4N - 6$. Hence the proposed algorithm is optimal. □

Example:



d= 2	1	4	7	0	0
c= 0	2	3	5	8	9
r= 2	0	1	2	0	0

Fig. 4.8 Rizhome tree for 6 nodes when $r_1 = 0$.

To illustrate how this works in practice, consider Figure 4.8 and 4.9. The main chain has 6 nodes, with varying numbers of children, as shown. Node 6 requires 2 extra time slots to listen to the transmissions of its 2 children (i.e., d(6,1) and d(6,2)), as previously explained. After this, it is ready to transmit to node 5. Node 5 has 1 child (i.e., d(5,1)), and its capacity is equal to the degree of the final node, therefore it does not require any extra time slots, which means that the remainder is 0 and it

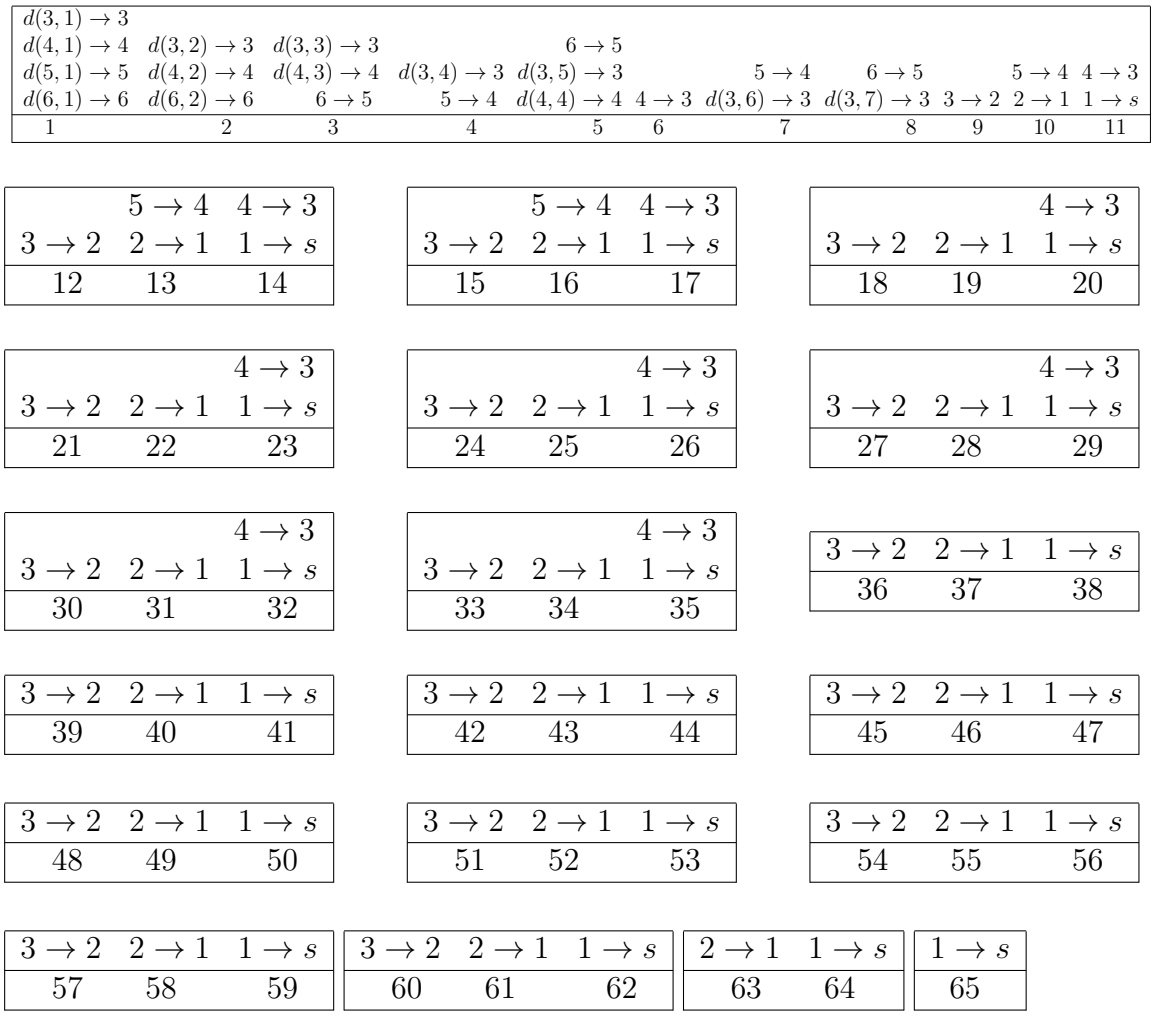


Fig. 4.9 Schedule for Figure 4.8.

immediately forwards the packet to the next node in the main chain. Node 4 has 4 children (i.e., d(4,1), d(4,2), d(4,3) and d(4,4)), however its capacity is equal to the capacity of the previous (i.e., node 4), plus 1, plus the remainder ($r_4 = 0$), totalling 3. The capacity means that node 4 can listen to 3 children in parallel with the transmissions of the previous nodes, without extra time slots. However, because it has 4 children, and not 3, it requires 1 extra time slot. The third node has 7 children (i.e., d(3,1), d(3,2), d(3,3), d(3,4), d(3,5), d(3,6) and d(3,7)) and its capacity is 5, therefore it requires 2 extra time slots, after which it can start transmitting to the next node. The second node has no children, and as such requires no extra time slots to transmit. Finally, the first node also has no children and so requires no extra time slots to transmit. When the sink has received the first packet, the third node should subsequently forward 17 packets, the second node will forward 18 packets and the first one will forward 19 packets. At this time, the process follows the pattern

described for a linear chain.

We summarize the scheduling steps as follows:

1. As shown in the schedule table of Figure 4.9, when the last node listens to its first child, nodes 5, 4 and 3 also concurrently listen to their first child nodes to receive a packet during the same time slot, and this is performed through forward scanning in the algorithm.
2. In the second time slot, when the last node listens to its second child node, simultaneously nodes 4 and 5 also receive packets from their second children. Likewise, in the third time slot, all these three nodes listen to their child nodes to receive data.
3. It can be seen that in the third time slot the last node has no more children, thus it is ready to transmit the first packet to the next node (i.e., 5), at the same time that nodes 3 and 4 listen to their third child nodes.
4. Again, node 5 forwards a packet to node 4, and node 3 listens to its fourth child node with it.
5. It can be noticed that when it is time for node 4 in the main chain to transmit, it cannot forward a packet at once to node 3, thus it must listen to its remaining child node(s). With it the third node can listen to its fifth child node and this is performed through forward scanning by the algorithm, and the last node, which is node 6, can make another transmission to the next node in the main chain, this is achieved via the backward scan by the algorithm. This process continues until the sink receives the first packet. Subsequently, only the first three nodes close to the sink need to be scheduled in three distinctive time slots to finish the rest of the schedule.

4.5.3 Scenario 2

Scenario 2 is where the remainder for the first node in the chain, nearest to the sink, is greater than zero (i.e., for $r_1>0$).

Formula (4.5) below indicates the minimum number of time slots that is required to collect all the packets from the Rhizome tree.

In that scenario, the third node ordinarily can forward $N-2$ packets across the main chain in addition to the degree of itself and the nodes downstream from it (i.e., $N-2+\sum_{i=3}^{N} d_i$). However, this definition is no longer correct when $r_1 > 0$, therefore the algorithm must detect the number of packets that have reached the second node concurrently with the remainder of the first node. This is performed through variable re of the third node re_3, of the fourth node re_4, and variable p respectively. Here variable p indicates all the remaining packets from the fifth node onwards that have not been passed through node v_5 when the first node starts data collection from its children. The reason for specifying only these three variables is that, when v_3 forwards a packet to v_2, node v_5 can receive a packet from its previous node in parallel. Therefore, the algorithm does not need to check further nodes.

$$T = N - 3 + \sum_{i=1}^{N} r_i + y + (N - 1 + \sum_{i=2}^{N} d_i) + (N + \sum_{i=1}^{N} d_i) \tag{4.5}$$

$$y = \begin{cases} (N-2+\sum_{i=3}^{N} d_i) - \underbrace{(re_3 + \lfloor \frac{r_1 - re_3}{2} \rfloor)}_{x} & \text{if } re_3 < r_1 \leq 2re_4 + re_3 \\ 0 & \text{if } r_1 \geqslant 3p + 2re_4 + re_3 \\ (N-2+\sum_{i=3}^{N} d_i) - \underbrace{(re_3 + re_4 + \lfloor \frac{r_1 - (2re_4 + re_3)}{3} \rfloor)}_{x} & \text{if } 2re_4 + re_3 \leq r_1 < 3p + 2re_4 + re_3 \\ (N-2+\sum_{i=3}^{N} d_i) - \underbrace{r_1}_{x} & \text{if } \quad r_1 \leqslant re_3 \end{cases}$$

The second scenario is almost the same as the first one and the same argument of the first scenario is true for the second one; the only modification is regarding the third node. When the remainder r_1 of the first node is greater than zero, the packets that can be forwarded by the third node while v_1 collects data from its children are significant.

After the first node finishes collecting data from its remaining children, the length of the subsequent data transmission schedule depends upon the number of packets remaining at the third node and downstream from it. Therefore we can perform that calculation according to the conditions that have been shown in the second scenario.

Note: The x variable indicates the extra number of packets that have arrived at the second node when the first node has finished data collection from its children. This implies that the responsibility of the third node decreases by x.

1. The first condition means that when the remainder is in the specified range, the algorithm can forward each packet of the third node concurrently with each transmission of the remainder of the first node according to the condition of the extra-bit technique. When all the packets of the third node have been forwarded then it is the turn of the fourth node to forward each packet during two transmissions of the remainder, because each packet needs two time slots to reach the second node. So, in the formula, after forwarding all the packets of the third node, re_3 is subtracted from r_1 and the result divided by 2 in order to determine how many more packets can be forwarded to the second node. As a result, the number of packets that have been forwarded to the second node can be subtracted from the overall transmissions of the third node.
2. The second condition explains that when the remainder of the first node is quite large and sufficient in order for the third and downstream nodes to forward all their packets simultaneously with it, then we need only schedule the first and the second node respectively to pass all the packets to the sink and no more packets are left in the third and downstream nodes. That is, the responsibility of the third node becomes zero.
3. The third condition illustrates that when the remainder of the first node is greater than the number of the transmissions that are required by the third and fourth nodes to hand over all their packets to the second node, then the algorithm must detect how many more packets can be forwarded with the rest of the remainder. In this case, each packet needs three time slots until it can reach the second node, therefore, we have to divide the rest of the remainder by 3.
4. The last condition means that when the $r_1 < re_3$, we know that only r_1 packets of the third node can be forwarded with r_1. Then it is the responsibility of the third node to forward the remaining packets of itself and downstream nodes.

The proof for the second scenario is almost identical to the proof of the first scenario and we only need to know the responsibility of the third node. The rest of the proof is the same as the proof of the first scenario for both lower and upper bounds. Therefore all arguments of the first scenario are correct except the responsibility of the third node. Example:

To illustrate how the second scenario operates, an example is provided in Figure 4.10 and 4.11. Suppose that a network consists of 6 main nodes in the Rhizome as well as the

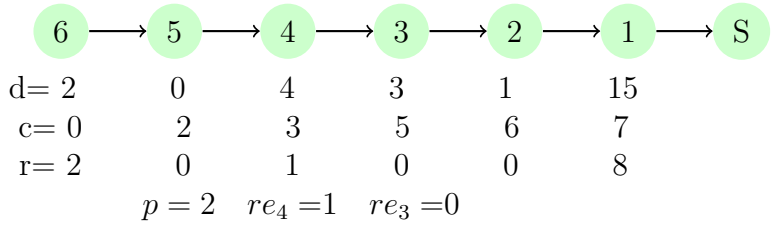


Fig. 4.10 Rizhome tree for 6 nodes when $r_1 > 0$.

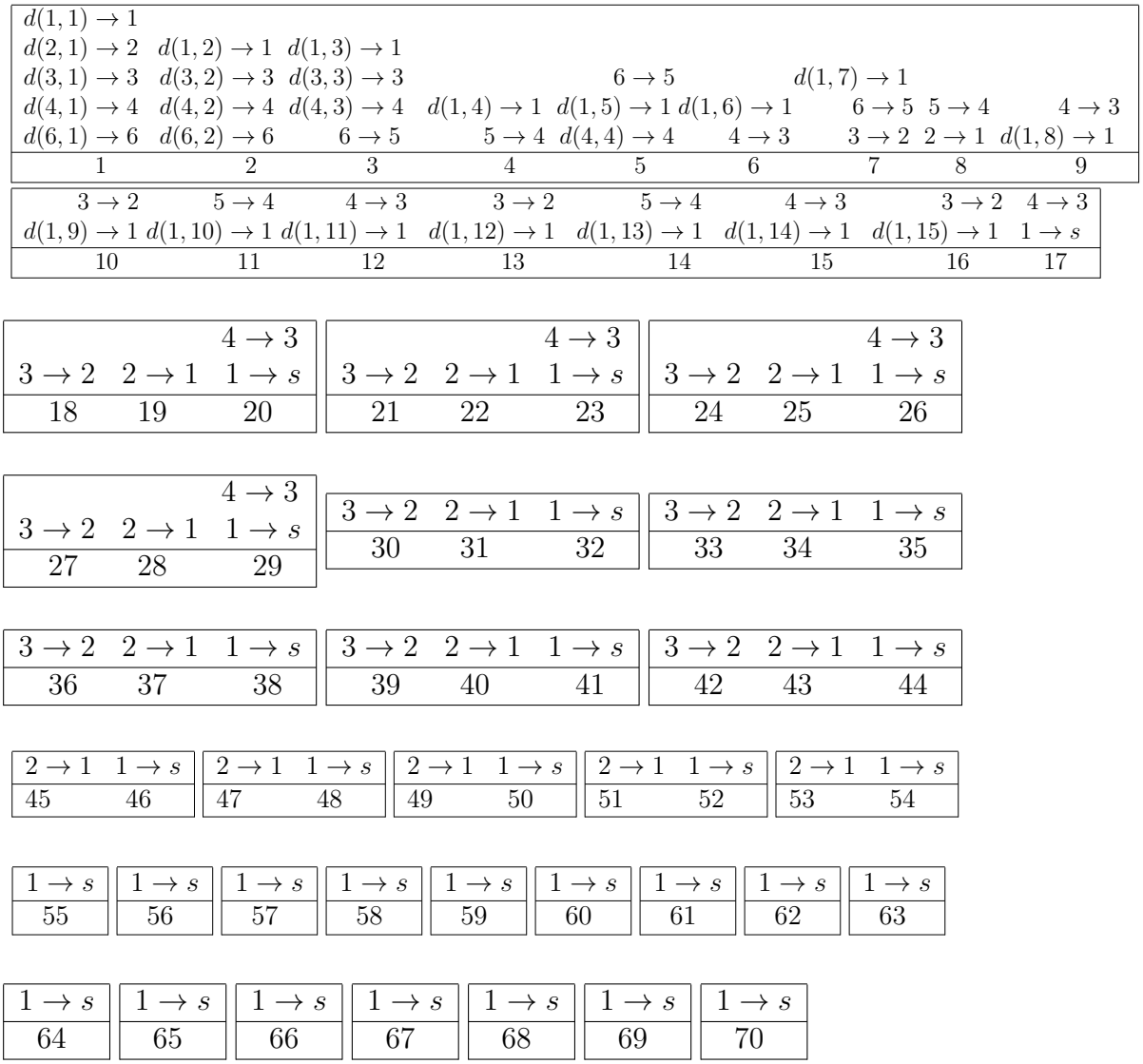


Fig. 4.11 Schedule for Figure 4.10.

sink. The three pieces of information are shown in the figure. As shown, node 6 requires 2 extra time slots to listen to the transmissions of its children. Following this, it is ready to transmit to node 5. Node 5 has 0 children; therefore, it does not require any extra time slots and it immediately forwards the packet to the next node in the main chain. Node 4 has 4 children; however, its capacity is 3, its remainder is 1. Thus, it requires 1 extra time slot; with this extra time slot, one more packet of the last node is forwarded

to node 5. Then, the fourth node can forward a packet to the third node. The third node has 3 children and its capacity is 5 and its remainder is 0; therefore, it does not require any extra time slots and it immediately starts transmitting a packet to the next node and concurrently the last packet of the last node is forwarded to node 5. This implies that $re_5 = 2$. Then the second node has 1 child and requires no extra time slots to transmit, and node 5 forwards the second packet with it (i.e., $re_5 = 1$ and $re_4 = 1$). Finally, the first node has 15 children and its capacity is 7 while its remainder is 8; therefore it requires 8 extra time slots to collect data from its remaining children. With the first 2 time slots, the received packet at v_4 arrives at node 2 (i.e., $re_4 = 0$). Similarly, in the next 3 time slots during the remainder of the first node, the received packet of node 5 is forwarded to node 2. Finally with the last 3 time slots of the remainder, the last available packet of the fifth node reaches the second node. In total, 3 more packets can be forwarded during these 8 time slots corresponding to the remainder of the first node. Then, the first node is ready to forward the first packet to the sink. It is observed that, according to the first scenario for this example, the third node has the responsibility of forwarding 13 packets; but due to the remainder of the first node (i.e., second scenario), three more packets can reach the second node from the previous nodes. Therefore the responsibility of the third decreases from 13 to 10. In total, 70 time slots are required to collect data in this example.

4.5.4 Implementation of the Algorithm

Algorithm 7 and 8 show the scheduling of the Rhizome tree. Firstly, the capacity and remainder of each node is calculated based on the explanation previously provided. Then, five procedures are utilised to finish the process of scheduling as follows:

1. Nodes in the main chain should be processed, checked and scheduled. This is performed via the procedure (line 6) and consists of two main steps. The algorithm starts from the last node in the first step. First, it has to check each node's remainder, and if it has any packets (line 9), the algorithm has to schedule the remainder of the node (line 10-12). At the same time, the other main nodes should be checked via both forward (line 13) and backward scanning (line 14) in order to ensure more parallel scheduled transmissions.

   In the second step (line 15-18), the transmissions of the main nodes are scheduled (line 16) and both forward and backward scanning are used (line 17-18).

2. Forward scanning (line 19-23) is used to help other nodes in the main chain to make

Algorithm 7: FIND SCHEDULE FOR RHIZOME TREE.

Input: $V = \{v(i) \mid 1 \leq i \leq N\} \cup \{v(i,j) \mid 1 \leqslant i \leqslant N, 1 \leqslant j \leqslant degree_{(i)}\}$
Output: $S(t)$ for $t = 1, \ldots, T$; where T is the length of the schedule

```
1  Main()
2  p(i) = 1, capacity(i) = 0 , re(i) = 0, st(i) = 1, for i = 1, 2, ..., N,
   t = 0, re(N) = d_N, st(N) = 0, S(t) = ∅, for t = 1, ..., T;
3  derive the capacity and remainder for each node v_i ∈ V;
4  call Read-all-nodes-with degree-and-schedule()
5  call Schedule-First-three-nodes()
6  Procedure Read-all-nodes-with degree-and-schedule()
7  for i ← N to 1 do
8      step 1 //check remainder of the current node first
9      if remainder(i) > 0 then
10         for j ← 1 to remainder(i) do
11             t ← t + 1
12             S(t) ← S(t) ∪ {v(i, degree(i))};// schedule this leaf
               p(i) ← p(i) + 1, and p(i, degree(i)) ← p(i, degree(i)) − 1 ;
13             Forward-scanning(i-1);// starts at v(i-1)
14             Backward-scanning(i+2);// start at v(i+2)
15     step 2 // schedule the current node in the main chain
16     Update-re-st(i)// schedule this main node
17     Forward-scanning(i-2);// start at v(i-2)
18      Backward-scanning(i+3);// start at v(i+3)
19 Procedure Forward-scanning(i)
20 for j ← i to 1 do
21     if p(j, degree(j)) > 0 then
22         S(t) ← S(t) ∪ {v(j, degree(j))};
23         p(j) ← p(j) + 1 and p(j, degree(j)) ← p(j, degree(j)) − 1 ;
24 Procedure Backward-scanning(i)
25 while (i ⩽ N) do
26     if re(i) == 0 and st(i) == 0 then
27         break;
28     else if ( re(i) > 0 and st(i) == 0) then
29         Update-re-st(i);
30         break;
31     else if ( re(i) ⩾ 1 and st(i) == 1) then
32         Update-re-st(i);
33         i+3; continue;
34     i++;
```

parallel transmissions with the current node; this procedure has been described in Section 4.5.1.

3. In addition to this, backward scanning (line 24-34) in Algorithm 7 is used to help

Algorithm 8: COMPLETION OF THE FIRST ALGORITHM .

```
1  Procedure Update-re-st(index)
2  t ← t + 1;
3  S(t) ← S(t) ∪ {v(index)};
4  p(index) ← p(index) − 1, re(index) ← re(index) − 1;
5  p(index − 1) ← p(index − 1) + 1,re(index − 1) ← re(index − 1) + 1;
6  if ( p(index) == 0) then
7  |  st(index − 1) ← 0;// previous node has finished transmitting
8  |  re(index − 1) ← degree(index − 1) + p(index − 1);
9  Procedure Schedule-first-three-nodes()
10 while p(1) ≠ 0 do
11 |  for i ← 3 to 1 do
12 |  |  if re(i) ≠ 0 then
13 |  |  |  Procedure Update-re-st(i);
14 |  |  Procedure Backward-scanning(i=i+3) ;
```

other nodes in the main chain to make parallel transmissions with the current node; this procedure has also been described in Section 4.5.1.

4. The procedure update-re-st in Algorithm 8 (line 1-8) is used to update the current pair of nodes (transmitter and receiver) regarding their packets and status (re and st).

5. After scheduling all the nodes in the backbone chain, the algorithm has to schedule the first three nodes until they finish transmitting their packets (line 9-14 of Algorithm 8).

6. Finally, the first and last procedure must be utilised inside the main procedure in algorithm 7 (line 1-5) to run the program.

4.6 Summary and Discussion

In summary, the extra-bit technique has been extended towards different topologies of the tree such as the balanced multi-chain, the unbalanced multi-chain, three and four level k-ary trees and the Rhizome tree topology.

Algorithms have been proposed that are optimal for the balanced multi-chain and Rhizome tree, whereas the algorithm is just a few steps away from the optimal solution for the unbalanced multi-chain.

Two frequencies have been used to save energy and to further reduce the latency. The lower bounds have been derived for the single chain and the balanced multi-chain

correspondingly; and finally, the optimal schedule has been proposed.

Although we managed to derive a lower bound for some special cases of the tree, finding the formula of a general tree is still an open question.

Moreover, as we pointed out earlier in chapter three, this technique is not helpful for fast data collection due to the length of the schedule compared to the arbitrary schedule (which has shorter latency).

Furthermore, the achieved results are only theoretical. It would be better to simulate the techniques through any existing simulation software, then finally to assess it in the testbed.

Chapter 5

Mobility in Wireless Sensor Networks

Mobile sinks have been used recently, mainly to minimize energy consumption and to resolve some other issues including data collection from disconnected networks, energy depletion from sensor nodes which are close to the sink, etc. In this chapter, we address the problem of finding an optimal path for the mobile sink to traverse through the sensing field, to collect a single packet from each sensor and return back to its initial point (starting point) such that the total energy use is minimized and subject to the length constraint requiring that the length of the tour is at most L. We refer to this as the minimum energy cost mobile sink restricted tour problem (MMRTP), and show that this problem is NP-hard. Second, we propose two algorithms. The first algorithm is a heuristic one based on a maximum ratio criterion which is based on the maximum reduction in distance and minimum energy consumption (hence termed the 'max-ratio'), while the second algorithm is based on the dynamic programming technique. We consider two scenarios for each algorithm. In the first scenario, there is no restriction on the transmission range of the nodes, whereas in the second scenario, the maximum transmission range is R_{max}. Finally, we evaluate the performance of our proposed algorithms based on MATLAB simulations for three different network sizes and show their effectiveness in terms of energy consumption. Moreover, the simulation results show that our second proposed algorithm has significant impact on the energy consumption in comparison with the algorithm of [70] for the same parameters (i.e., lengths and transmission ranges).

5.1 Introduction

The phenomenon of multi-hop forwarding leads to non-uniform energy depletion among the nodes in the network, and especially those nodes which are close to the sink deplete their energy faster than others, which leads to the cessation of operation across the entire network while some of the nodes still have power to operate [67, 68]; this phenomenon is called an energy hole or bottleneck.

Research, therefore, on mobile sinks has received considerable attention in the last decade to solve some issues such as the bottleneck, disconnection, dispersion and reliability. As a result, several frameworks and algorithms have been designed, proposed and developed to utilize mobile sinks under different scenarios with different constraints for various applications and purposes [6, 54]. In addition to the problem of energy holes, reliability is another concern in WSNs; the probability of message loss increases in a multi-hop fashion, and therefore, the mobile sink can reduce message loss and increase reliability as well [40].

Similarly, in some applications, sensors are sparse and cannot communicate with each other. Hence, mobile sinks can either provide the connectivity between them [83] or be used to visit them to buffer their data before depositing the collected data at the base station; in particular, there are sometimes different disconnected networks in which sensors from one network cannot communicate with another network in order to construct a complete network including the base station. Therefore mobile sinks can be a good solution. Transmission range is another cause of energy consumption, especially when the sensors operate at a high transmission range. To reduce the usage of high transmission ranges, sensors can be either densely deployed, which sometimes leads to high costs, or mobile sinks can be used to approach sensors in order to enable them to use lower transmission ranges, which has a great impact on energy conservation [64].

On the other hand, mobile sinks can cause delays for data collection and they are not desirable in some applications. Hence, mobility and multi-hop forwarding are jointly used to trade off between latency and energy consumption to some extent [80, 30].

In this chapter, we consider a network where sensors are deployed in a region for the purpose of data collection. We are interested in finding an efficient path for the mobile sink to follow and collect the data from sensors via single hop communication and deposit it at the destination for further processing, subject to the length constraint and with minimum total energy consumption. We consider one complete round of data collection as the cycle of the mobile sink traversing from the starting point through all the sensors,

thereby collecting a single packet from each, and returning back to the starting point. For the relevant notations used throughout this chapter refer to Section 5.2.

Our objective is mainly two-fold. Firstly, we want to restrict the length of the tour $\ell(T^S) \leqslant L$ for the mobile sink. It is quite reasonable to restrict the length of the tour due to there being a limited amount of fuel or time for the mobile sink, especially in military operations. Another example is agriculture or habitat monitoring where the mobile sink can collect data within limited time or length. Secondly, we want the mobile sink to select a set of shortcuts (construct a tour) that cover all the nodes and has minimum total energy consumption (E_{total}). E_{total} is the summation of the energy cost that is spent by sensors for sending one packet each to the mobile sink during data collection.

We optimize total energy consumption under the length constraint for two reasons. First to minimise the energy consumption of the sensors in the network, as they use smaller transmission ranges to send their packets to the mobile sink via a single hop. Second to minimize maintenance and costs if we consider battery replacements. In addition, it is eco-friendly. To this end, we propose two different algorithms. A detailed explanation of the algorithms is given in Section 5.3. To the best of our knowledge, we are the first to address this issue.

5.2 System Model and Problem Definition

We consider a network that consists of a number of nodes (sensors), which are deployed in a specific area for the purpose of monitoring. This network can be modelled as a complete undirected graph $G = (V, E)$, where each $v_i \in V$, $i = 1, \ldots, N$, represents a node in the network and is a point in the Euclidean plane ($v_i \in \mathbb{R}^2$). Similarly, each $e \in E$ represents the direct path (distance) between two nodes. Each edge has a set of values (labels), which represent the set of nodes whose communications touch this edge e when it is considered by the mobile sink as its path.

Suppose the locations of sensors are known in advance via GPS or any other method. We assume that each node has one packet to be collected by the mobile sink. We consider two scenarios regarding the transmission range of the sensors. Firstly, we suppose that there is no restriction on the transmission of the sensors, namely, they can all reach each other. Secondly, all the sensors have the maximum transmission range R_{max}. Notably, multi hop forwarding is not considered in this chapter. We assume that the transmission range can be any value between zero and R_{max} for the second scenario, and that the

energy cost over distance d is proportional to d^{α}, where α is the path loss exponent and $2 \leqslant \alpha \leqslant 6$, depending on the environment [60]. In free space, there is no obstacle between transmitter and receiver and the path loss exponent is set to 2. We use the same value in our system while for simplicity, we only consider energy consumption for transmitting a packet from a sensor to the mobile sink, and assume v_1 is the depot for the mobile sink.

Figure 5.1, similar to Figure 2(b) in [70], illustrates the complete graph for 5 nodes wherein each edge is associated with a set of values that refers to the nodes covered by the edge when the mobile sink moves along this edge. In addition, each edge length is the direct distance between its endpoints.

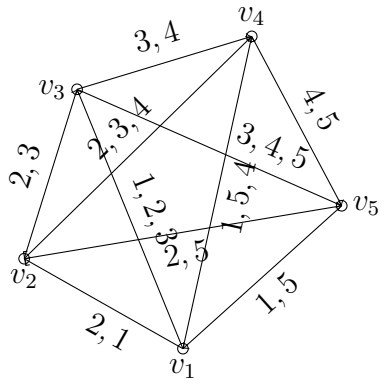


Fig. 5.1 Complete graph for 5 nodes, the numbers next to each edge represent the nodes covered by the edge.

Our main objective is to construct a tour (find an optimal cycle) of length at most L for the mobile sink to follow and collect a single packet from each sensor via single hop communication and return back to its initial position such that the total amount of energy, which is spent by all the sensors in this tour, is minimal.

This problem can be defined as the minimum energy cost mobile sink restricted tour problem (MMRTP).

Definition 3: (MMRTP): Given a set of sensor nodes V with their locations, sink location p_s, transmission range R_{max} and a length constraint L. Find a tour of length at most L for the mobile sink (starting and ending at p_s) that minimizes the total energy cost.

Theorem 15. MMRTP is NP-hard.

Proof. Checking whether a tour of length L and energy cost zero exists is the same as checking whether a TSP tour of length L exists. □

In this chapter, we follow two steps to find a solution for the MMRTP. In the first step, we construct a TSP tour. Then, in the second step, we optimize the TSP tour by

selecting a subset of nodes from the order of the TSP, and construct the trajectory for the mobile sink to follow, by which to collect data from all the nodes (cover all the nodes), and return back to its starting point. In other words, let T be a TSP tour of the nodes in V. Our goal is to find a shortcut tour T^S of length at most L, (i.e., $\ell(T^S) \leqslant L$), by choosing a subset of nodes from the tour (T) of the TSP, skipping others, while ensuring that the sink passes within the transmission range of the nodes that are skipped. The sink then returns back to the starting point, and the process seeks to minimize the total energy consumption E_{total}, which is used by sensors to send their packets to the mobile sink.

Notation	
1- T	denotes the tour of the TSP
2- $\ell(T)$	the length of the tour of the TSP
3- T^S	denotes the shortcut tour obtained from the order of the TSP
4- $\ell(T^S)$	denotes the length of the tour according to our algorithm
5- R_i	the transmission range of node v_i , $0 \leqslant R_i \leqslant R_{max}$
6- $d(v_i, v_j)$	represents the direct distance between node v_i and v_j
7- $dT^S(v_i, v_j)$	total T^S distance of the nodes starting from node v_i and ending at v_j
8- $d(v_i, T^S)$	the distance of the node v_i from the path of the mobile sink
9- E_{v_i}	the energy consumption for node v_i
10- E'_{v_i}	is the new total energy consumption for node v_i in the next round

We summarize the two steps mathematically as follows:

Step 1: We want to find the TSP tour T, therefore the order of the visit in the TSP tour can be represented by a sequence I_i of indices of visited nodes:

$\forall i \quad 1 \leqslant I_i \leqslant N \; and \; \forall i, j \quad i \neq j \Longrightarrow I_i \neq I_j, I_1 = 1.$

Then the length of the tour of the TSP is

$$\ell(T) = \sum_{i=1}^{N-1} d(v_{I_i}, v_{I_{i+1}}) + d(v_{I_N}, v_{I_1}) \tag{5.1}$$

Step 2: We construct the shortcut tour T^S. Therefore, the subset of nodes that are obtained from the order of the TSP can be represented by a sequence of indices of visited nodes.

Let $I'_1, ..., I'_M$ be the list of indices of a subset of nodes that are visited by the mobile sink.

$M \leqslant N$. $I'_1 = 1$.

$$Minimize \;\; E_{total}(T^S) = \sum_{i=1}^{N} d(v_i, T^S)^2 \tag{5.2}$$

$$s.\,t.\; \ell(T^S) = \sum_{i=1}^{M-1} d(v_{I'_i}, v_{I'_{i+1}}) + d(v_{I'_M}, v_{I'_1}) \leq L \tag{5.3}$$

Minimizing $E_{total}(T^S)$ represents the least energy consumption of the nodes when the mobile sink follows the shortcut tour T^S. Inequality (5.3) restricts the length of the shortcut tour to L.

To apply these two steps, we propose two algorithms (the reason for proposing two algorithms instead of one is mentioned at the end of Section 5.3.2. These two algorithms start from the initial point of the TSP tour, and visit a subset of the nodes, skipping others, while ensuring that the sink passes within the transmission range of the nodes that are skipped. The sink then returns back to the starting point, and the process seeks to minimise total energy consumption.

Note: We should make a clear distinction between TSP, travelling salesman with neighbourhoods (TSPN) and our problem. In TSP, the goal is to find the minimum length of the tour while visiting each city exactly once (i.e., each city is a single point). On the other hand, TSPN is a variation of the TSP in which cities are substituted with regions, and the objective is to find a minimal tour through the set of regions that visits at least one point in each region [1]. Clearly both TSP and TSPN are well known NP-hard problems. Our model is a combination of both the TSP and TSPN models in which the mobile sink visits the nodes if it does not exceed the length of the restricted tour; in that case, it is identical to TSP. However, it should skip the central visit to some nodes by choosing alternative paths (shortcuts) during which it can cover the skipped nodes at some points, which is similar to TSPN.

5.3 Proposed Approaches

5.3.1 First Approach: Heuristic Algorithm (max-ratio)

The algorithm mainly consists of three steps. The first step finds the order of the tour according to a TSP algorithm (constructed TSP tour). In the second step, calculations are performed to find the shortcuts and the corresponding energy consumption of the nodes. Each shortcut has a saved distance and energy cost, and we consider dividing the saved

distance by its energy cost as our main criterion (metric) in the algorithm and record the result in a table. Finally, in the third step the shortcut is established, based on the criterion of choosing a maximum value from the table that we obtained in the second part. In other words, the key factor for choosing which shortcut to use, is based on the maximum reduction in distance and minimum energy consumption (termed the 'max-ratio'). Then, only steps 2 and 3 are repeated with the remaining nodes in the tour until the condition of having length at most L is achieved. Our algorithm is a heuristic and there is no guarantee that this metric produces the optimal solution.

The details of this algorithm are as follows:

1. After deployment of the sensors, the order of the visit to the nodes is specified via one of the existing approximation algorithms for the TSP.

2. Assume v_1,v_2,...,v_N is the tour, and v_N is a duplicate of v_1. Once the visiting order is determined, our proposed algorithm selects a subset of nodes accordingly. Choosing this subset of nodes means that some of the nodes are skipped and not visited by the mobile sink. This skipping process is called shortcutting. We consider a shortcut from each node v_i to node v_j to find both the distance saved by the shortcut and the total energy consumption increase for the node(s) v_z in between, where $i = 1, 2, 3, ..., N-2$, $j = i+2, i+3, ..., N$ and $z = i+1, i+2, ...j-1$. The shortcut indicates by how much the length of the tour can be reduced if one or more nodes are skipped, e.g., if the length of the tour $d(v_i, v_{i+1}) + d(v_{i+1}, v_j)$, where $j = i+2$, is 150 meters in TSP, what is the distance saved by the shortcut if the mobile sink goes directly from v_i to v_j. Suppose the distance $d(v_i, v_j)$ is 100 meters. Then the shortcut reduces the tour length by 50 meters ($150 - 100 = 50m$).

   This process of shortcutting is performed as follows:

   (a) The segment distance $d(v_i, v_j)$ is determined via the formula of the Euclidean distance between two points, then the minimum distance of each node lying between these two nodes on the TSP tour from the line segment $\overline{v_i, v_j}$ is determined. This can be calculated via the equation that is used to find the distance of the point from the line segment $d(v_z, \overline{v_i v_j})$, where $\overline{v_i v_j}$ means the line segment from v_i to v_j.

   (b) After finding the distance of the node(s) v_z from the line $\overline{v_i v_j}$, this distance should be squared as energy consumption over distance d is proportional to distance squared. Then, the total squared distance from the line, of the node(s)

bypassed as a result of the shortcut between v_i and v_j should be found. Namely,

$$E_{total}(i,j) = E_{v_{i+1}} + E_{v_{i+2}} + ... + E_{v_{j-1}} \tag{5.4}$$

This indicates the total energy usage by the skipped nodes when the mobile sink goes directly from node v_i to v_j.

Note: In some cases, $d(v_M, T^S)$ could be smaller than $d(v_M, \overline{v_i v_j})$, but we only consider the distance from v_M to line segment $\overline{v_i v_j}$.

Note: Sometimes one shortcut is not enough to reduce the length of the tour. The algorithm, therefore, should be performed for several rounds to find several shortcuts. Hence, the distance of the skipped nodes from the line must be retained on record, as these nodes may be involved in the shortcutting process again in the next round. The reason for this is that when the distance is established between these skipped node(s) and the line in the shortcut of the next round, the marginal difference in power output needs to be calculated and for this, the marginal change in distance between each node and the line must be used (some of this distance was already included in the previous calculations for the shortcut). Namely, if the energy of the skipped node (v_{i+1}) is $E_{v_{i+1}}$ in the first round, and if the node is involved again in the next round then its new energy cost is $E'_{v_{i+1}}$. Finally, the algorithm should account for the difference between the old and new energy expenditures, $\Delta E_{v_{i+1}} \leftarrow E'_{v_{i+1}} - E_{v_{i+1}}$.

(c) Then the algorithm calculates the distance saved by the shortcut when the order of the visits is scheduled directly from node v_i to v_j.

$$S(i,j) = (\sum_{z=i}^{j-1} d(v_z, v_{z+1})) - d(v_i, v_j) \tag{5.5}$$

Then, the algorithm calculates $SE_{total}(i,j) = \frac{S(i,j)}{E_{total}(i,j)}$ based on the two previous calculations, which is used in the next step of the current round for choosing the maximum value in SE_{total}. This maximum value is the main criterion in this algorithm that determines the maximum shortcut and minimum energy consumption possible.

(d) The above two steps are performed for all the nodes and the results are kept in SE_{total}.

3. Finally, the algorithm attempts to choose the optimal values in terms of distance saved and energy expended; that is, the maximum value in SE_{total} determines the shortcutting.

4. When the shortcutting has been executed, SE_{total} is reset to zero to be ready for the next round. Furthermore, in the next round the number of nodes decreases as some nodes are skipped. Then, step 2 and step 3 are repeated for the remaining nodes of the TSP after shortcutting until the condition of the length of the tour $\ell(T^S) \leqslant L$ is satisfied.

Algorithm 9 illustrates the implementation of the proposed technique for the case of unrestricted transmission range. In essence, there are two main functions.

1. The first one is named **Evaluate-all-shortcuts()**, line 11. Inside this procedure three for-loops are used. The first two for-loops are used to find the possible shortcuts (line segments) between a pair of nodes. The third for-loop is used to find the distance of the skipped nodes to the line segment; according to the TSP order, these skipped nodes are ordered sequentially between the pair of nodes joined by the line segment, line 20. Then, this value should be squared as explained before. Inside the third loop, one condition is used to check whether this node is already involved in the process of shortening the tour, line 22; namely, whether it has been skipped. Consequently, the energy conserved relative to the previous calculation should be deducted from the current amount. When the third loop has finished its calculation, the ratio of the distance saved relative to energy consumption should be kept in the table SE_{total}. Then this process is continued until all shortcuts are found for all the nodes.

2. The second procedure is named **Find-max-value()**, line 27. The responsibility of this function is to choose the maximum value in SE_{total} and reduce the length of the tour based on the shortcut with this maximum value. Then the squared distances for the skipped nodes (involved nodes) in this iteration are kept in the vector *record* (line 33), which are necessary for the next iteration, namely, if these skipped nodes will be involved in the shortcutting, only ΔE should be used.

Finally, these two procedures are repeated until the tour length $\leqslant L$ is achieved or all nodes are checked (line 4) or all the values of SE_{total} are zero (line 7).

Algorithm 10 shows the implementation of the proposed technique for the case of

Algorithm 9: MOBILE SINK, SCENARIO 1.

Input: $G = (V, E)$, depot=v_1, L
Output: $sub-tour\ (T^S)\ with\ \ell(T^S) \leqslant L$

Main()
$SE_{total}(i,j) \leftarrow 0, record(i) \leftarrow 0,\ indication(i) = false\ ,\forall i,j = 1,2,3,\ldots,N$;
make a TSP tour T using any approximation algorithm for TSP. $c \leftarrow 0$, $T^S \leftarrow T$;
while $(\ell(T^S)) > L$ *and* $c \leq N$ **do**
    call **Evaluate-all-shortcuts()**
    if $SE_{total}(i,j) == 0\ ,\forall\ i,j = 1,2,3,\ldots,N$; **then**
        break;
    call **Find-max-value(**SE_{total}**)**
    $SE_{total}(i,j) \leftarrow 0, \forall i,j = 1,2,3,\ldots,N$;
    $c \leftarrow c + 1$;
Procedure Evaluate-all-shortcuts()
for $i \leftarrow 1$ ***to*** $N-2$ **do**
    if $indiction(i) == true$ **then**
        continue//skip involved nodes
    for $j \leftarrow i+2$ ***to*** N **do**
        if $indication(j) == true$ **then**
            continue//skip involved nodes
        $S \leftarrow 0$, $d \leftarrow 0$, $sum \leftarrow 0$;
        for $z \leftarrow i+1$ ***to*** $j-1$ **do**
            $d \leftarrow d(v(z), \overline{v(i)v(j)})$;
            $d \leftarrow d^2$;
            if $indication(z) == true$ **then**
                $d \leftarrow d - record(z)$;//$\Delta E'_{v_z}$
            $sum \leftarrow sum + d$;// $E_{total}(i,j)$;
        $S \leftarrow dT^S(v(i), v(j)) - d(v(i), v(j))$;
        $SE_{total}(i,j) = S/sum$;
Procedure Find-max-value(SE_{total}**)**
$maxvalue \leftarrow max(SE_{total})$;
$find\ i,j\ locations\ of\ maxvalue$;
$\ell(T^S) \leftarrow \ell(T^S) - dT^S(v(i), v(j)) + d(v(i), v(j))$;
for $z \leftarrow i+1$ ***to*** $j-1$ **do**
    $d \leftarrow d(v(z), \overline{v(i)v(j)})$;
    $record(z) \leftarrow d^2$;
    $indication(z) = true$;

restricted transmission range (For simplicity we set the maximum transmission range to 100m). The general idea of this algorithm is almost identical to the first algorithm, the only difference is in the **Evaluate-all-shortcuts()** function, where the distance of the nodes that lie between a pair of nodes in the order of TSP should be checked to know whether they can be covered by the line segment of the pair. This condition is checked in

Algorithm 10: MOBILE SINK, SCENARIO 2.

```
1  Evaluate-all-shortcuts()
2  R_max ← 100;
3  for i ← 1 to N − 2 do
4      if indiction(i) == true then
5          continue
6      for j ← i + 2 to N do
7          if indication(j) == true then
8              continue
9          S ← 0, d ← 0, sum ← 0, count1 ← 0, count2 ← 0;
10         for z ← i + 1 to j − 1 do
11             count1 ← count1 + 1;
12             d ← d(v(z), v(i)v(j));
13             if d ⩽ R_max then
14                 count2 ← count2 + 1;
15                 d ← d^2;
16                 if indication(z) == true then
17                     d ← d − record(z);//ΔE'_{v_z}=E'_{v_z} − E_{v_z} ;
18                 sum ← sum + d;
19         S ← dT^S(v(i), v(j)) − d(v(i), v(j));
20         if count1 == count2 and S > 0 then
21             SE_total(i, j) = S/sum;
```

line 13. It should be pointed out that only some of the nodes may be covered by R_{max} instead of all. Hence we consider this line segment as a shortcut only when all the nodes are covered by the shortcut (the line segment). This is the reason that two counters are used (count1 and count2). When these two values are matched, line 20, then we consider this line segment and fill in SE_{total} with the ratio value, line 21. Then the second function is called to perform the shortcutting based on the value of the maximum ratio.

5.3.1.1 Time Complexity of the Heuristic Algorithm

The complexity of the first proposed algorithm is $TSP + O(N^4)$ where TSP is the complexity of the TSP based on the used approximation algorithm. N is the number of nodes in the system. The TSP should be found for our algorithm as the first step, thus, the complexity for the TSP is needed, then our algorithm needs at most $O(N^3)$ steps to find shortcut saving, first function (**Evaluate-all-shortcuts()**), next there are at most $O(N^2)$ steps to choose the best shortcut in the second function (**Find-max-value()**). Finally, these two functions should be repeated at most N times until one of the conditions of stopping is satisfied (main function). As a result, the complexity be-

comes $O(N * (N^2 + N^3)) = O(N^4 + N^3)$, and the total complexity of the algorithm is $TSP + O(N^4)$.

5.3.2 Second Approach: Dynamic Programming (DP) Algorithm

We observed that our problem is similar to the 0-1 knapsack problem and can be solved using a similar dynamic programming approach. Therefore we would like to describe the Knapsack problem briefly.

The Knapsack problem is a problem in combinatorial optimization and belongs to the family of NP-hard problems. This means that it is unlikely to admit an algorithm that finds an optimal solution in polynomial time. Suppose there is a knapsack with limited capacity C and there are N different items; moreover, suppose each item has two attributes, namely weight (volume) and value (profit). As all the items cannot be loaded into the knapsack due to its limited capacity, a subset of items should be determined, chosen and loaded into the knapsack such that they have maximum value (profit) and do not exceed the capacity of the knapsack.

There are several types of the knapsack problem but we are interested in the 0-1 knapsack problem where the item can either be completely taken or not. I.e, we cannot take a fraction of the item. For more detail about the knapsack problem and its solution refer to [57].

The simple approach to solve this problem is to go through all possible subsets of the N items and output the best of them. It should be noted that the complexity of this simple technique is exponential, which is not desirable for large input N. The second technique is a dynamic programming approach, which is used to find a solution to this problem and is usually better than the brute force technique.

When the dynamic programming technique is used to find a solution to the knapsack problem, two tables (arrays) are required, the first one is used to record the solution and only return back the optimal solution value (maximum profit) at the end without determining the chosen items. Therefore, the second one is used to keep track of the subset of items that makes the optimal solution.

We observed that our problem is similar to the 0-1 knapsack problem and can be solved using a similar dynamic programming approach. To further understand how to apply the dynamic programming approach to our problem, refer to Algorithm 11 and 12.

Note: It should be pointed out that in the knapsack, we want to maximize the profit

whereas in our solution we want to minimize energy consumption.

In our problem, when the DP technique is used to find a solution, two tables (arrays) are required to record the solution, we call these two tables Table E and Table P respectively. The first table is used to record the calculation for the problem and finally return the optimal solution (least energy cost). Moreover the table entry $E(v, l)$ stores the optimal solution value (least energy cost) for the subtour node v from node v_1 up to node v with length constraint l. $E(N, L)$ stores the overall optimal solution for the problem with length constraint L. The detailed calculation is shown in line 30-40 in Algorithm 11.

Since the first table only returns the optimal solution value we do not know how many nodes have been exactly chosen to be visited by the mobile sink to construct the tour. Therefore the second table is required to record the involved subset of nodes (selected nodes to be visited by the mobile sink) during the calculation of the first table and traced-back at the end.

Algorithm 11 based on the dynamic programming technique solves the problem as follows:
After constructing the TSP tour via any existing algorithm, two essential functions are used in order to find a solution for our problem. The first one is again named **Evaluate-all-shortcuts()**: this procedure finds all the possible distances (shortcuts) and corresponding energy wastage respectively between pairs of nodes. For simplicity, all the results are kept in two separate tables; we call them distance table **(d)** and energy table **(energy)** respectively. Then, these two tables are used in the second function to apply the idea of the dynamic programming to find a solution to the problem.

The second function **dynammic-programming()**, which starts in line 22, is the main idea of the dynamic programming technique for our problem. First of all, the two tables E and P, described earlier, should be initialized as shown in line 23-29. Then three main loops are required to perform the calculation to find the optimal solution for the problem based on the dynamic programming approach, line 30-40. Since the first node is the depot, the first loop goes from node two through N. Line 34 is used to check whether there is a shortcut between a pair of nodes w, v; if there is (line 35), we should specify the optimal solution for node v with length l, this can be done either by inheriting the optimal solution of node w with length s plus the energy cost of the shortcut (line 38) and replacing the value of $E(v, l)$ with this optimal solution (line 40) or keeping its own value. Then this iteration is repeated for all the nodes until the optimal solution is achieved. In other words, these iterations try to choose the best optimal solution for node v from all its shortcuts. Fi-

Algorithm 11: MOBILE SINK, DYNAMIC PROGRAMMING TECHNIQUE.

Input: $G = (V, E)$, depot=v_1, L
Output: $\ell(T^S) \leqslant L$

```
1  Main()
2  { energy(i, j) ← 0, d(i, j) ← ∞ ,∀ i, j = 1, 2, 3, ..., N ;
3  make a TSP tour T using any approximation algorithm for TSP;
4  Evaluate-all-shortcuts()
5  Dynamic-programming()
6  }
7  Evaluate-all-shortcuts()
8  R_max ← 100;
9  for i ← 1 to N − 2 do
10     for j ← i + 2 to N do
11         sum ← 0, count1 ← 0, count2 ← 0;
12         for z ← i + 1 to j − 1 do
13             count1 ← count1 + 1;
14             d ← d(v(z), v(i)v(j));
15             if d ⩽ R_max then
16                 count2 ← count2 + 1;
17                 d ← d^2, sum ← sum + d;
18         if count1 == count2 then
19             d(i, j) ← d(v(i), v(j));
20             d(j, i) ← d(v(i), v(j));
21             energy(i, j) = sum;
22 dynamic-programming()
23 for v ← 1 to N do
24     for l ← 0 to L do
25         if v ≠ 1 then
26             E(v, l) = ∞;// this table for dynamic programming;
27         else
28             E(v, l) = 0;
29         p(v, l) = 0;// this table for trace-back;
30 for v ← 2 to N do
31     for l ← 0 to L do
32         for w ← 1 to v − 1 do
33             if d(w, v) == ∞ then
34                 continue; // skip non-existing shortcuts
35             s = l − d(w, v);
36             if s < 0 then
37                 continue;// skip negative path lengths
38             e = E(w, s) + energy(w, v);
39             if E(v, l) > e then
40                 E(v, l) = e, P(v, l) = w;
```

nally, Algorithm 12 is used to specify the involved nodes (visited node by the mobile sink).

Algorithm 12: MOBILE SINK, DYNAMIC PROGRAMMING,TRACE-BACK.

```
1  Trace-back()
2  l = 1;
3  for j ← 1 to L do
4      if E(N, j) < E(N, l) then
5          l = j;
6  output(E(N, l));
7  v = N;
8  while v > 1 do
9      w = P(v, l);
10     if w ⩽ 0 then
11         break;
12     l = l − d(w, v);
13     output w;
14     v = w;
```

5.3.2.1 Time Complexity of the Second Algorithm

It is worth noting that the complexity of the second approach is $O(N^3 + N^2L)$, where N is the number of nodes and L is the length constraint of the tour. For the first function at most $O(N^3)$ steps are required to find shortcut savings, then at most $O(N^2L)$ steps are required for the DP approach to find the best possible solution. It can be observed that the length constraint can change the complexity of the algorithm, if we use meter as a unit then the accuracy is precise but the computation cost is high, on the other hand we can reduce the complexity of the algorithm by changing the unit length but the accuracy is decreased.

Note: There are two main differences between our problem and the 0-1 knapsack problem. Firstly, in the knapsack problem, each item has two fixed values (p=profit and w=weight), which cannot be changed (updated) during the calculation, and these two fixed values are the inputs. In other words, the DP technique selects the best combination of items and returns the optimal solution at the end without modifying the two fixed values for any item. Secondly, the optimal solution for item i, $\forall i = 1, 2 \ldots, N$, depends on the optimal solution for item $i - 1$. Consequently, the optimal solution for item i will be inherited from item $i - 1$ either with or without including item i. Refer to [57] for further details.

Two crucial points should be mentioned for the proposed algorithms. Firstly, despite the fact that the second proposed algorithm can achieve the best possible solution for the problem if one exists, its running time increases substantially with length L. That is, for large input L, the complexity is very high. Secondly, all the distance values between nodes must be integers. Therefore, in our calculation all the values have been rounded to integer by using the round function.

On the other hand, the complexity of the first proposed algorithm (heuristic max-ratio) is polynomial, as we have explained before. Therefore, we can choose between these two algorithms based on the size of the problem in terms of input and length. Moreover, the values for the first algorithm do not need to be only integers.

5.4 Simulations and Performance Evaluations of the Proposed Algorithms

5.4.1 Simulation Results for Different Network Sizes

In this section, we present and evaluate the results of simulations for both proposed algorithms with three different network sizes; that is, 20 nodes, 40 nodes and 100 nodes, respectively.

The algorithm is applied to 10 iterations of the experiment and the average length constraint and energy usage are calculated for each network size.

The first step of the experiment is the deployment of sensors at random locations in a specific area; 300m*300m for the 20-node experiment and 1000m*1000m for the 40 and 100-node experiments, respectively. Secondly, we obtain the TSP tour by using NEOS Server for Concorde [1], which is available online, to find the TSP tour and its length ($\ell(T)$=1300m for 20 nodes, $\ell(T)$=5750m for 40 nodes, and $\ell(T)$=7762m for 100 nodes). Then, we finally apply our proposed algorithms.

In all the plots, the X-axis represents length constraints and the Y-axis, energy consumption per length constraint. For all of the following simulations we use no restriction on the transmission range for the first scenario. This freedom of range allows both algorithms to find a solution at an early step of the constraint. For simplicity, in the second scenario we set the maximum transmission range to 100m thereby allowing both algorithms to find a solution at a late stage of the length constraint.

[1] http://neos.mcs.anl.gov/neos/solvers/co:concorde/TSP.html.

In general, when the length constraint reaches the length of the TSP, energy consumption approaches 0 due to the fact that the mobile sink traverses through each sensor, reducing their energy consumption for sending packets since the transmission range is 0 whereas if the length constraint is lower, the transmission range of the sensors would be higher to reach the mobile sink, causing them to use more power for packet transfer.

Figure 5.2 illustrates the results of simulations for the heuristic algorithm on 20 nodes. The dotted line represents scenario one while the dark solid line represents scenario two. As expected, the algorithm can find a solution at low length constraints according to the first scenario, meaning there is high energy consumption due to some sensors transmitting over a long range to reach the mobile sink. Moreover, the energy consumption is the same for the first and second length constraints as the algorithm achieves the same solution for both of these data points (although a valid solution was found for only 1 out of the 10 deployments, there is no guarantee to find a solution for all of the deployments, because the algorithm is heuristic). At higher length constraints, the energy consumption reduces sharply for 300m and 400m, with valid solutions found for 4 and 8 out of 10 of the deployments, respectively. For length constraints from 500m up to 1100m, there is a further decline in energy consumption, and at each length constraint, valid solutions are found for 10 out of 10 of the deployments. At higher length constraints of 1200m and 1300m only 6 out of 10 and 1 out of 10, respectively, of the deployments yield a valid solution. This is due to the dispersal pattern of the nodes which do not allow for such a large length constraints in all deployments. Furthermore, at the length constraint of 600m, there is a valid solution for the second scenario and the energy consumption is lower than the energy consumption in the first scenario because for the second scenario, only 2 out of 10 take part in the solution, while in the first scenario 10 out of 10 take part in the solution, so that the lower average value may reflect stochastically less dispersal in the data averaged. There is a valid solution in general in the second scenario for constraints of 700m and onwards. This is because the transmission range and length constraint are very restricted and not all of the nodes are able to send their packets to the sink. Then, with an increase in length constraint for both scenarios, energy cost decreases gradually until it eventually approaches zero. Notably, for the second scenario, from 600m up to 1100m, 10 out of 10 of the deployments are involved in the solution. and for 1200m and 1300m only 6 and 1 out of 10 have this length.

Similarly, Figure 5.3 shows the results of simulations for the same 20 nodes according to the dynamic programming algorithm for both scenarios. Evidently, the solution can be

achieved at an early step of the length constraint with high energy consumption according to the first scenario for 10 out of 10 of the deployments, then it gradually decreases with an increase in length constraint until it approaches zero. On the other hand, the solution can be found at a very late stage (i.e., 500m) for 2 out of 10 the deployments according to the second scenario with slightly lower energy consumption compared to the first scenario in which 10 out of 10 involve in the solution. For the length constraint of 600m, again only 5 out of 10 involve in the solution. From 700m to 1100m, 10 out of 10 involve in the solution and the energy consumption steadily decreases until it finally becomes zero.

It can be observed that dynamic programming technique can find a solution if there exits one, therefore, at the early stage of the length constraint 10 out of 10 involve in the solution, whereas for the heuristic algorithm less than 10 deployments involve in the solution.

Figure 5.4 demonstrates and compares both heuristic and dynamic programming algorithms for the first scenario. As we have explained, both algorithms can find a solution at a very early step with high energy consumption. Moreover, the energy cost is high and constant at some length constraints (i.e., 100m and 200m) then decreases substantially towards zero in the heuristic algorithm. However, for the dynamic programming algorithm the energy cost is not constant and decreases gradually with each length constraint until it becomes zero. Unexpectedly, the energy consumption under length constraint 100m is higher for dynamic programming compared to the heuristic algorithm. Again, this is due to the involvement of 1 out of 10 the deployments in the solution for heuristic algorithm, while 10 out of 10 are involved in the solution at that stage.

Likewise, Figure 5.5 compares both heuristic and dynamic programming algorithms for the second scenario. Expectedly, a solution is found at a very late step of the length constraint(i.e., 500m for dynamic programming and 600m for heuristic algorithm), with different points varying in their energy consumptions. The energy consumption then gradually decreases towards zero.

Similarly, Figure 5.6 shows the results of simulations for the heuristic algorithm (first and second scenarios) on 40 nodes. For the first scenario the solution starts at the length constraint of 1000m with the involvement of 2 out of 10 the deployments in the solution and very high energy consumption. Notably, for three consecutive length constraints (1250m, 1500m and 1750m) the energy consumption show only slight differences due to the fact that their achieved lengths are very close to each other and only 7 out of the 10 deployments are involved in the solution. From the length constraint of 1750 to 2000m only 8 out of 10

the deployments involve in the solution with a steep decline in energy consumption. From 2250m up to 5000m, 10 out of 10 the deployments involve in the solution and the energy consumption steadily decreases until it finally becomes zero. Again for 5250m, 5500m and 5750m only 7, 5 and 1 out of 10 the deployments, respectively, have this length.

Figure 5.7, shows the results of simulations for the dynamic programming algorithm (first and second scenarios) on 40 nodes. Evidently, the solution can be achieved at an early step of the length constraint with high energy consumption according to the first scenario for 10 out of 10 of the deployments, then it sharply decreases in the first three consecutive lenght constraints and then gradually decreases with an increase in length constraint until it approaches zero. On the other hand, the solution can be found at a very late step (i.e., 4000m) for 2 out of 10 the deployments according to the second scenario with slightly lower energy consumption compared to the first scenario in which 10 out of 10 involve in the solution. Finally, energy consumption gradually decreases with an increase in length constraint until it approaches zero. Note that for length constraint of 5250m, 5500m and 5750m, again only 7, 5 and 1 out of 10 involve in the solution, respectively.

Figure 5.8 compares both heuristic and dynamic programming algorithms for the first scenario on 40 nodes with different points varying sharply and gradually in their energy consumptions with the involvement of different number of simulations. Eventually, the energy consumption gradually decreases towards zero.

Similarly, figure 5.9 compares both heuristic and dynamic programming algorithms for the second scenario on 40 nodes. Expectedly, finding a solution is achieved at a very late step of the length constraint(i.e., 4000m for dynamic programming and 4500m for heuristic algorithm). It can be observed that in dynamic programming, energy consumption can remain at the same level for both lengths 4000m and 4250m respectively, then sharply decreases up to 5000m. Similarly, for the heuristic algorithm, there is a steep decrease for the two length constraints. Then energy consumption in both algorithms gradually decreases until it finally becomes zero.

Figure 5.10, 5.11, 5.12 and 5.13 are the results of simulations involving 100 nodes.

Figure 5.10 shows the results of simulations for the heuristic algorithm (first and second scenarios) on 100 nodes. Unexpectedly, the energy consumption in the first length constraint compared to the second one is low due to the fact that only 2 out of the 10 and 4 out of 10 deployments involve in the solution, respectivley. Then there is a combination of steep and gradual decline in the energy consumption up to 2750m. Finallly, the energy

consumption in both algorithms gradually decreases until it finally becomes zero.

Similarly, Figure 5.11 shows the results of simulations for the same 100 nodes according to the dynamic programming algorithm for both scenarios. Clearly, a solution can be achieved at an early and late step of the length constraint with high energy consumption according to the first and second scenarios respectively. Then, it gradually deceases with an increase in length constraint until it approaches zero with the involvement of different deployments in the solutions.

Similarly, Figures 5.12 and 5.13 compare both heuristic and dynamic programming algorithms for the first scenario on 100 nodes, for energy consumptions with different length constraints and with the involvement of different deployments in the solutions.

In general, as we expected, the algorithm can find a solution at an early stage according to the first scenario for any network size in comparison with the second scenario. Furthermore, the second algorithm can always find a better solution with lower energy consumption compared to the first algorithm due to the power of the dynamic programming.

Note: Some plots for three network sizes and specific length constraints for one Random deployyment foe each network size are shown in Appendix A in order to visually see the tour after simulations.

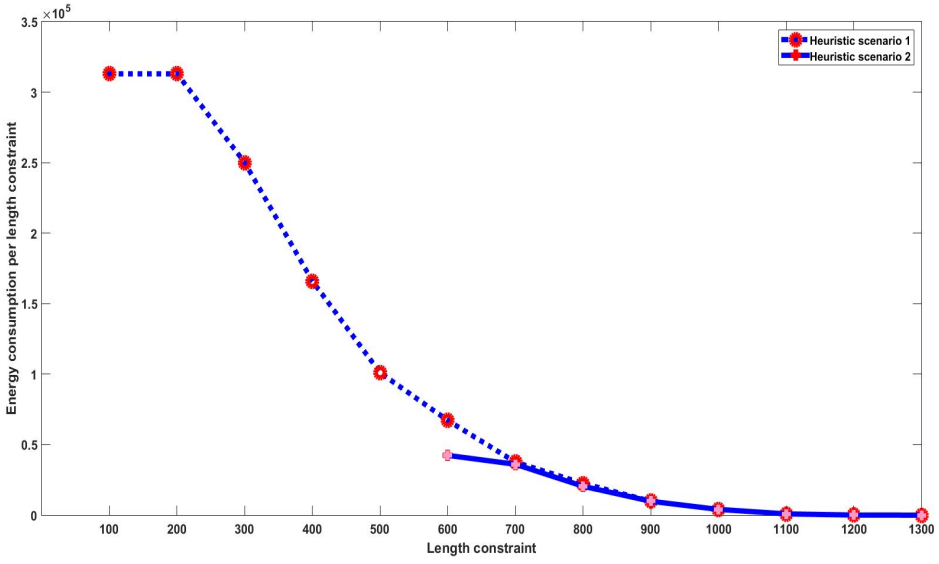


Fig. 5.2 Heuristic algorithm, scenario 1 & 2, R_{max}=100, 20 nodes.

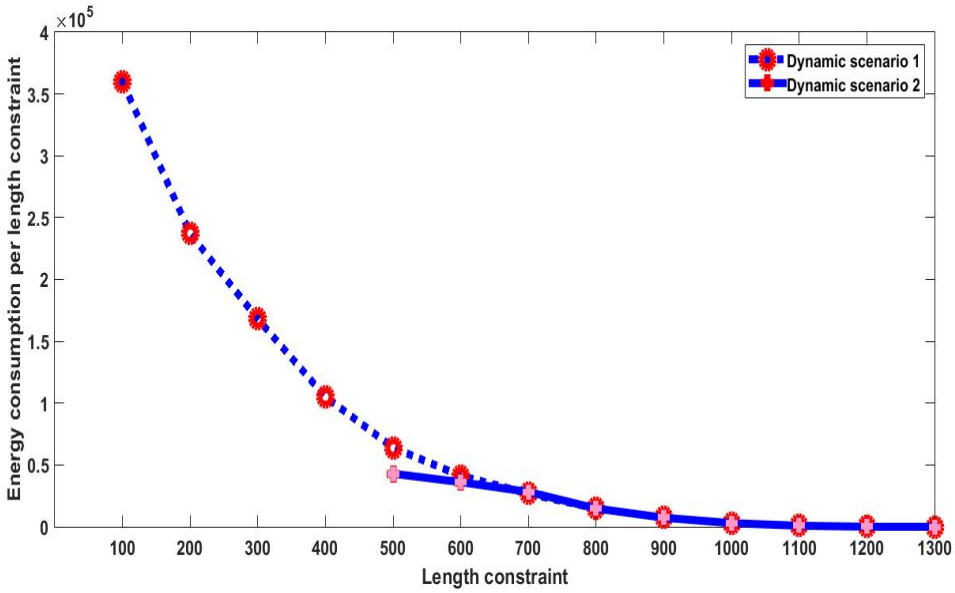


Fig. 5.3 Dynamic programming algorithm, scenario 1 & 2, R_{max}=100, 20 nodes.

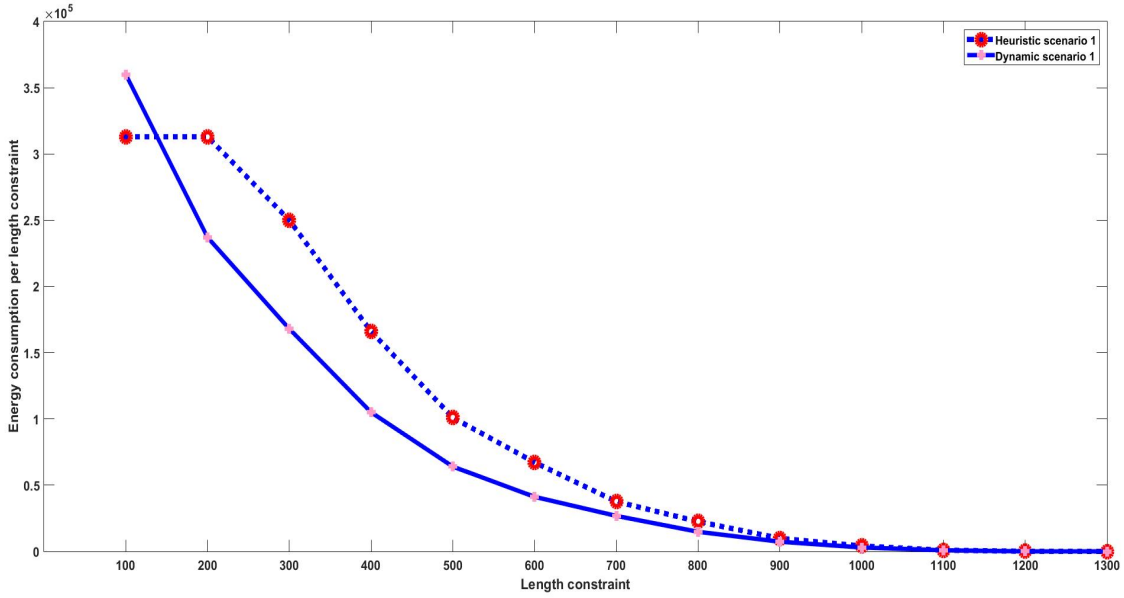


Fig. 5.4 Heuristic & dynamic programming algorithms, scenario 1, 20 nodes.

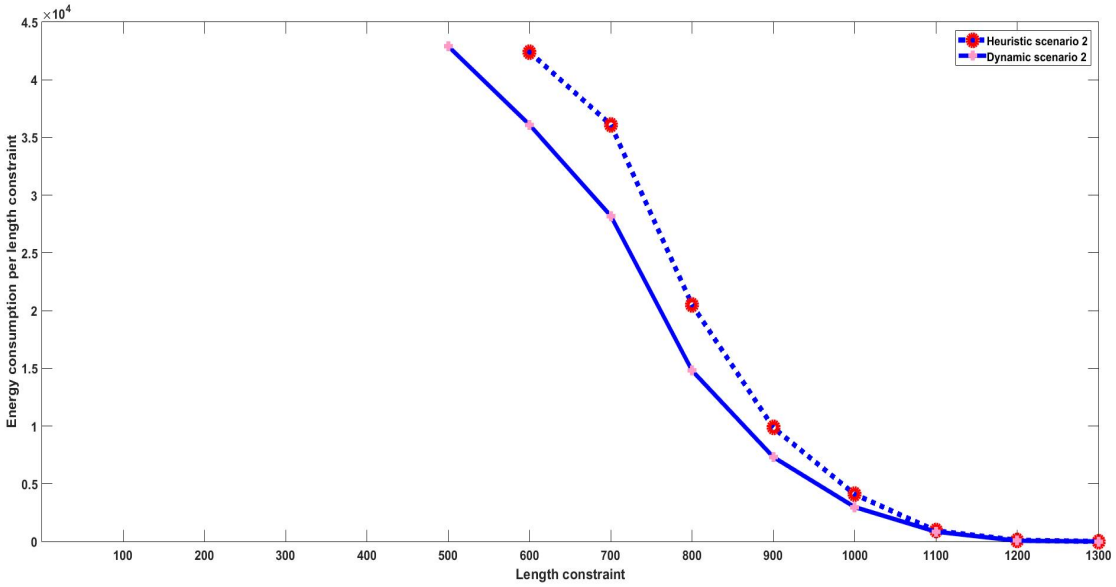


Fig. 5.5 Heuristic & dynamic programming algorithms, scenario 2, R_{max}=100, 20 nodes.

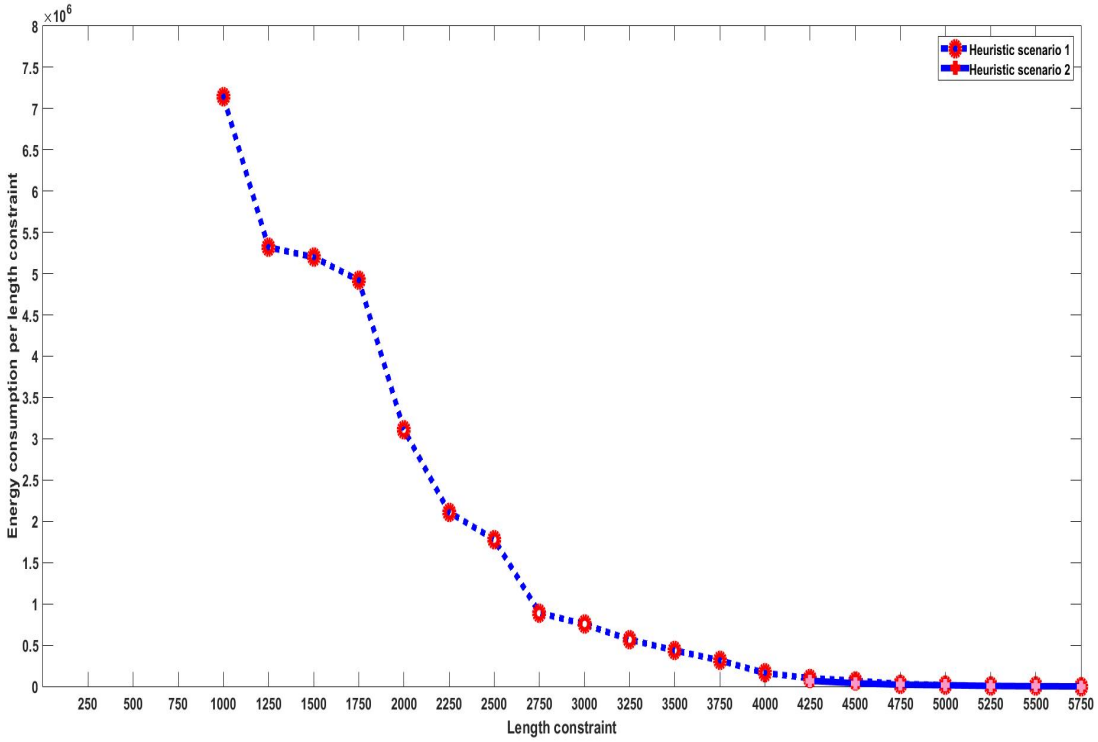


Fig. 5.6 Heuristic algorithm, scenario 1 & 2, R_{max}=100, 40 nodes.

5.4.2 Comparing Our Results with the Algorithm for the Label Covering Problem

In this section, we present the numerical results achieved from the experiments for three different network sizes, namely 20, 40 and 100 nodes. In particular we compare our second proposed algorithm with the algorithm for the label covering problem [70].

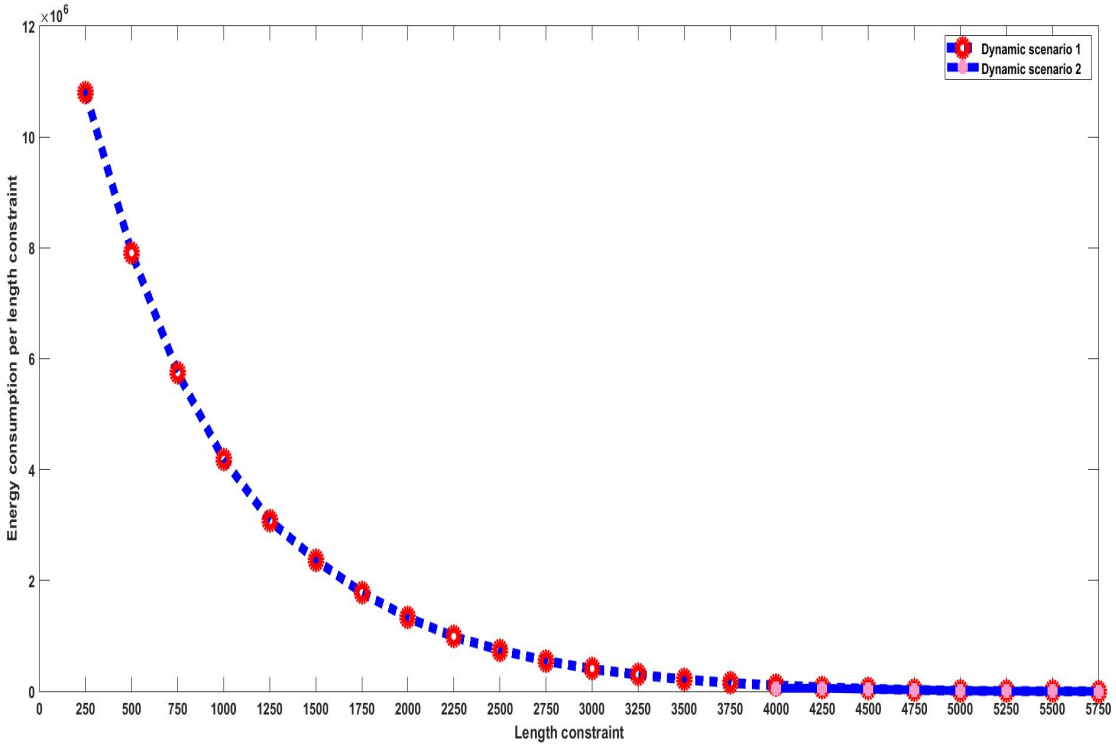


Fig. 5.7 Dynamic programming algorithm, scenario 1 & 2, R_{max}=100, 40 nodes.

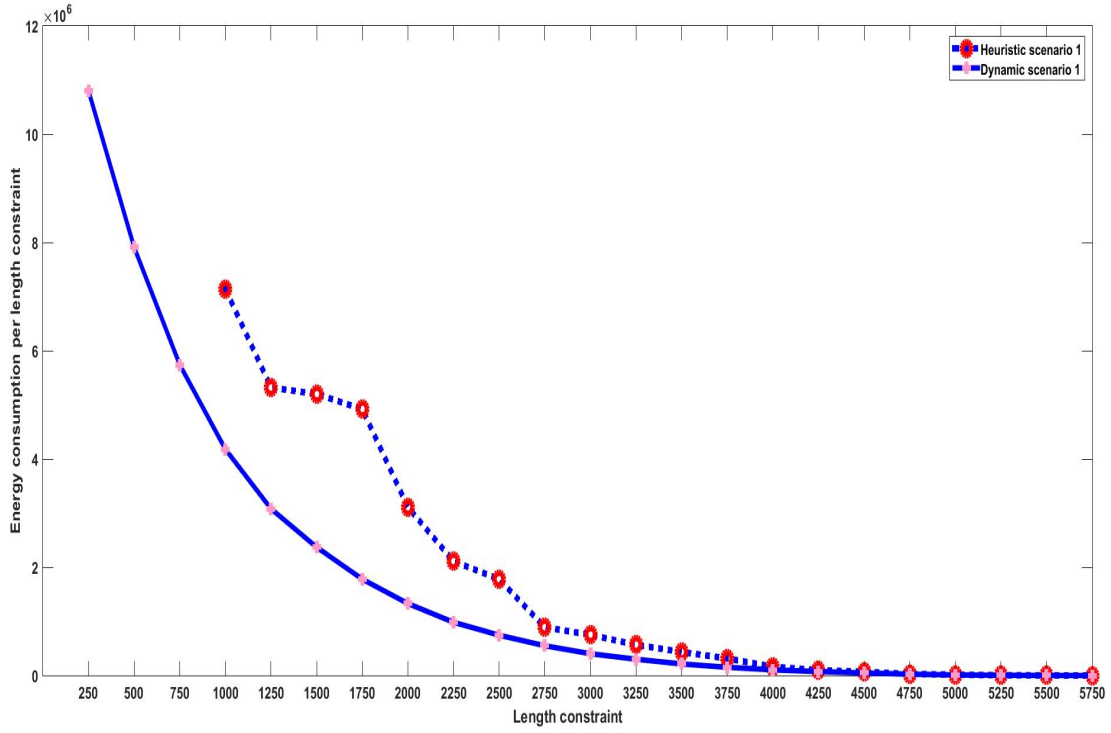


Fig. 5.8 Heuristic and dynamic programming algorithms, scenario 1, 40 nodes.

In the experiments, for simplicity, we consider four different transmission ranges (75m, 100m, 125m, 150m) for each network size. The results obtained for 20 nodes are shown in Table 5.1. The first column refers to the achieved minimum length tours based on

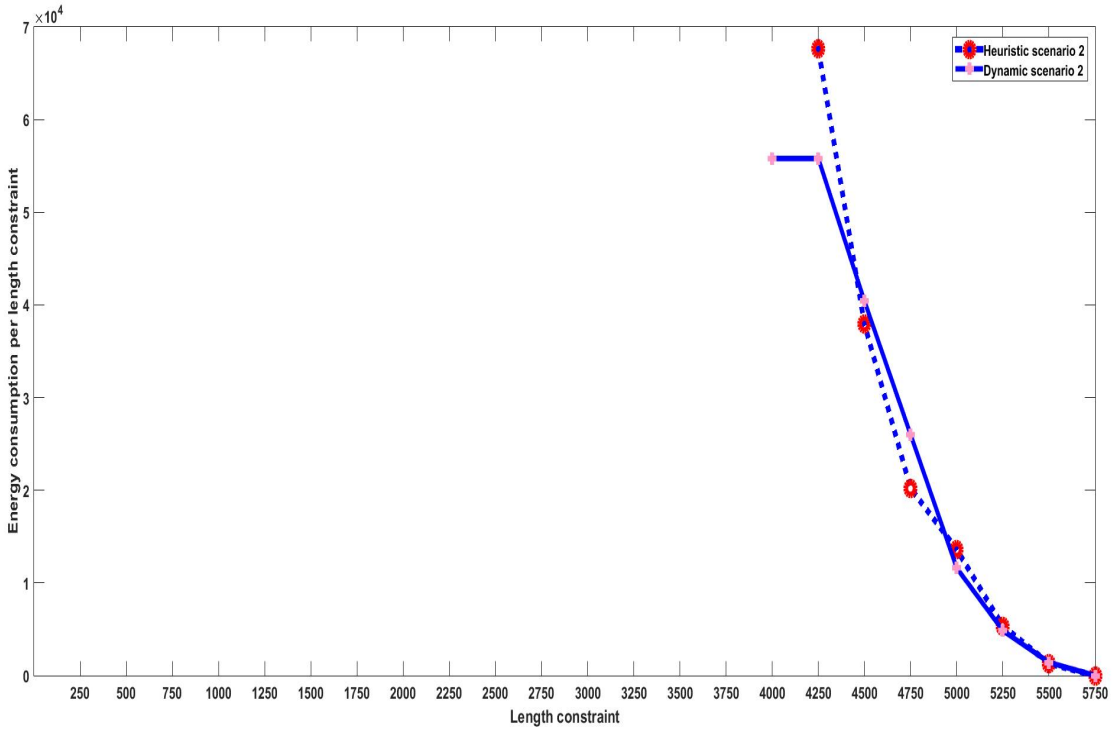


Fig. 5.9 Heuristic and dynamic programming algorithms, scenario 2, R_{max}=100, 40 nodes.

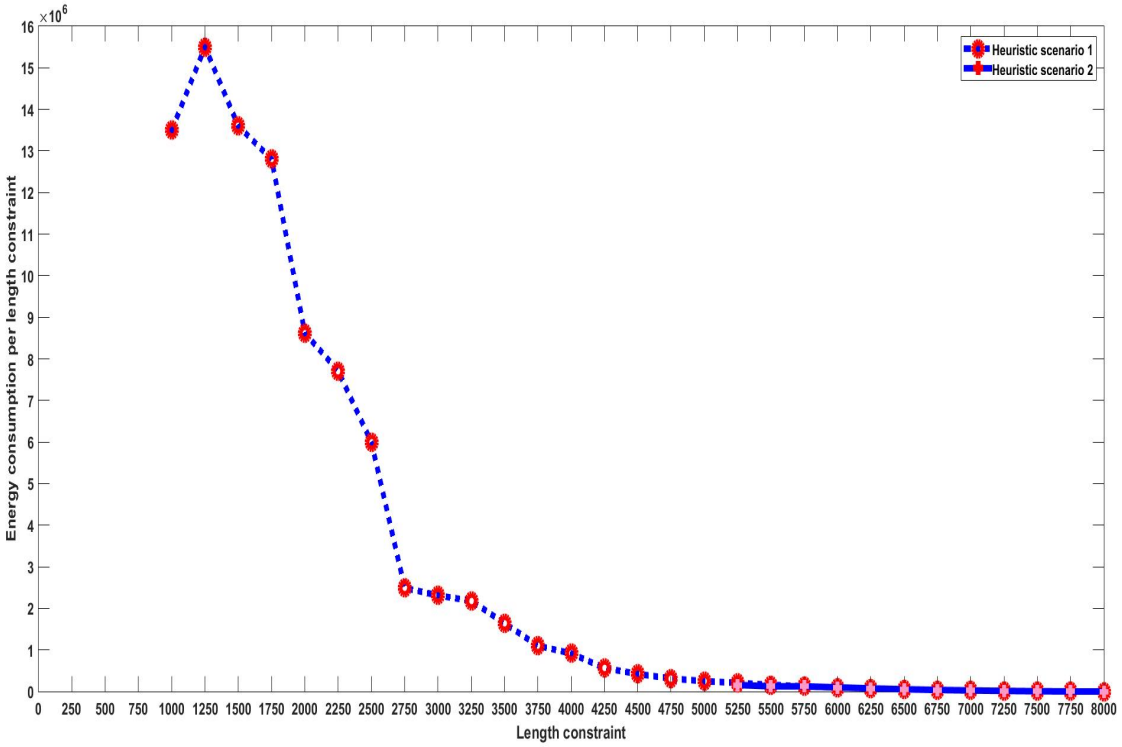


Fig. 5.10 Heuristic algorithm, scenario 1 & 2, R_{max}=100, 100 nodes.

the algorithm for the label covering problem. The second column is the corresponding energy consumption, and the third column shows energy consumptions according to our algorithm when its length constraint is set to the achieved minimum length of the label

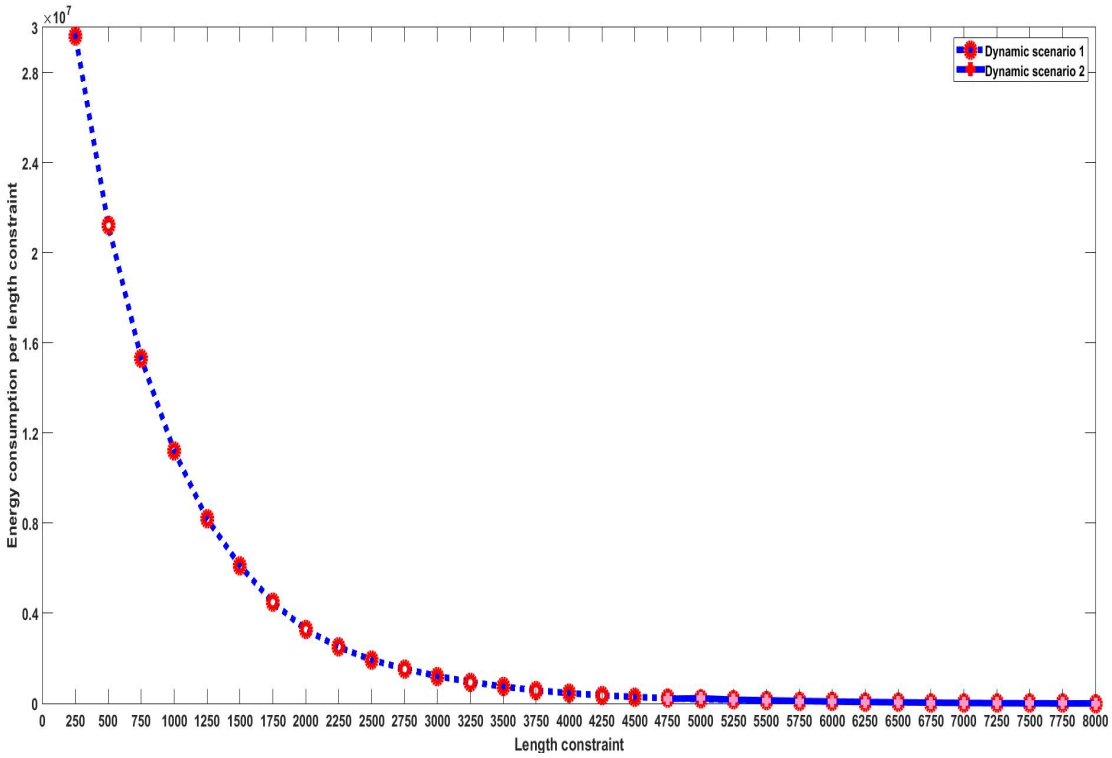


Fig. 5.11 Dynamic programming algorithm, scenario 1 & 2, R_{max}=100, 100 nodes.

covering problem (i.e., we use the same lengths that are obtained from the algorithm for the label covering problem). It is observed that our algorithm has slightly lower energy consumption for the first two transmission ranges (75m and 100m), whereas the last two transmission ranges (125m and 150m) have the same energy consumptions.

The numerical results for the network of size 40 are illustrated in Table 5.2. Interestingly, there is a significant reduction in the energy consumption according to our algorithm. Similar results are shown in Table 5.3 for 100 nodes. In our algorithm, we observe that when the network size increases, there is a great reduction in the energy consumption in comparison with the algorithm for the label covering problem. This reduction can be referred to our method by which we find the shortcuts that satisfy the length constraint and total energy consumption rather than finding only the minimum length to cover all of the nodes in label covering problem which does not focus on energy consumption.

5.5 Summary and Discussion

In this chapter, we have considered the problem of selecting an optimal trajectory for the mobile sink to follow such that the sink can collect data from sensors with minimal total energy consumption while satisfying the imposed length constraint L. To this end, this

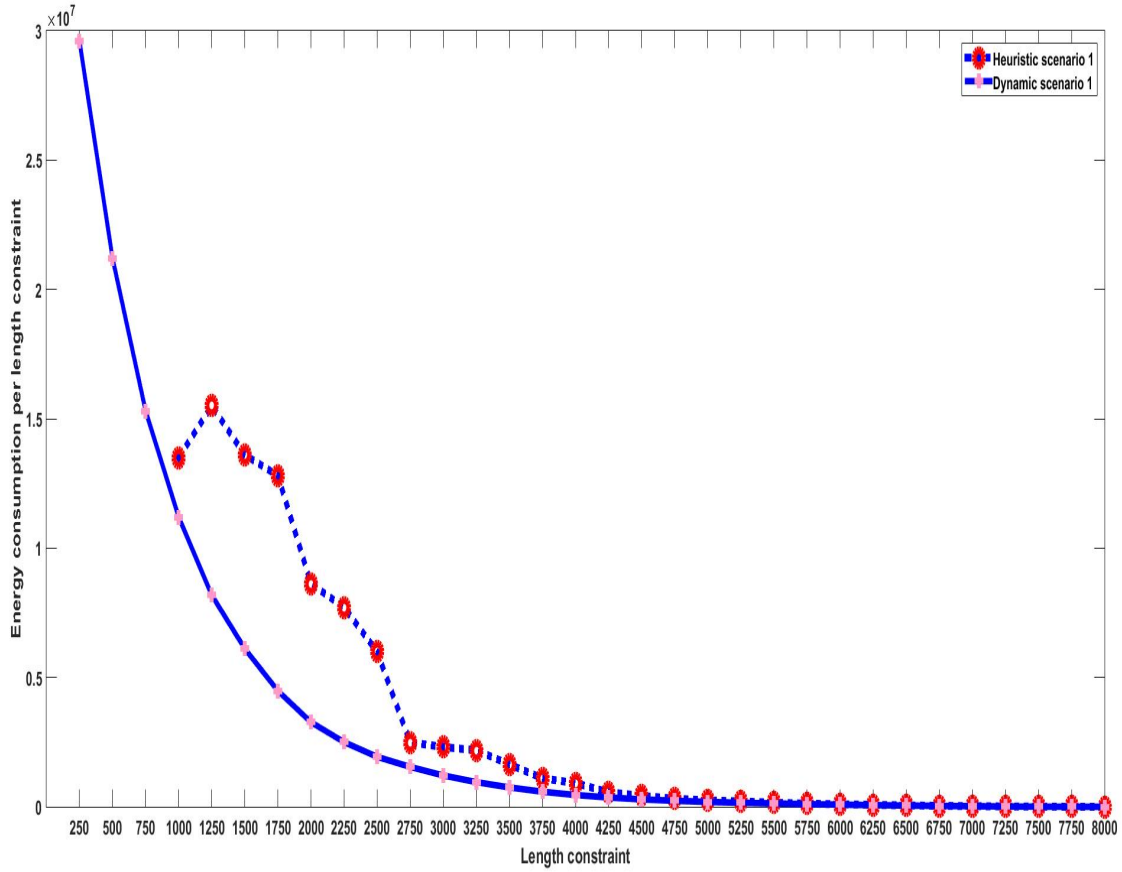


Fig. 5.12 Heuristic and dynamic programming algorithms, scenario 1, 100 nodes.

problem has been defined formally and proven to be NP-complete. We have proposed and presented two algorithms that utilize a mobile sink efficiently to traverse through a sensing field and collect data from sensors in a single-hop pattern, without considering multi-hop forwarding, after which the mobile sink deposits the collected data at the base station. In the first algorithm, the key factor for choosing the optimal shortcut (sub path) , as explained earlier, is based on the maximum reduction in distance and minimum energy consumption (termed the 'max-ratio'), while the second algorithm is based on the dynamic programming technique. Moreover, two scenarios have been considered for each algorithm. In the first scenario the maximum transmission range is not specified, whereas in the second scenario it is specified.

As shown in the experimental MATLAB simulations, the second proposed algorithm (dynamic programming) can achieve better results in comparison to the max-ratio criterion due to the influence of the dynamic programming architecture. Furthermore, in some cases the max-ratio criterion cannot find a solution, even if one existent, due to the fact that there is no guarantee to find a solution for all of the deployments in all cases since the

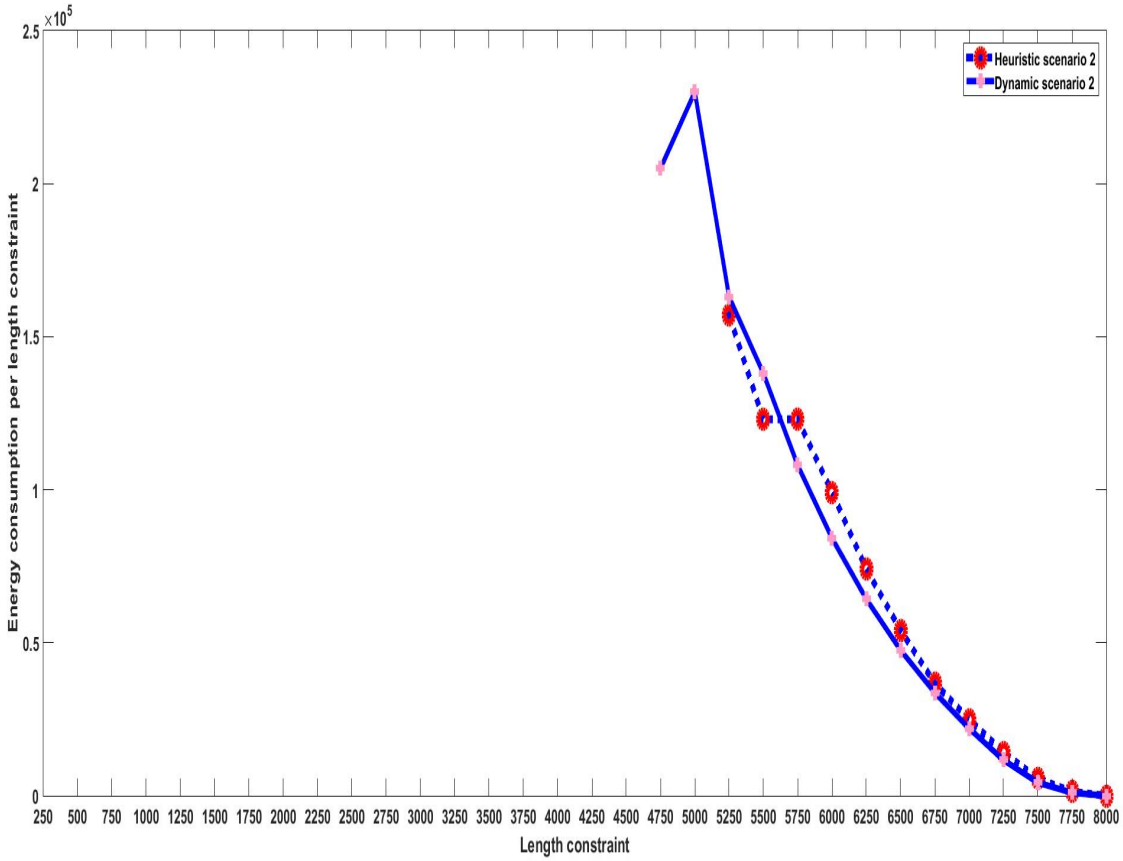


Fig. 5.13 Heuristic and dynamic programming algorithms, scenario 2, R_{max}=100, 100 nodes.

first algorithm is heuristic, whereas, the dynamic programming technique can always find the best solution among the available solutions. The second algorithm has been compared with the proposed algorithm for the label covering problem. Interestingly, our algorithm can achieve significant reduction of energy consumption for the same minimum length. This reduction has a great impact on the lifespan of the network since the label covering problem focusses only on the achieving minimum length.

On the other hand, the proposed algorithms may in some cases be unable to achieve good performance due to a large geographical area or sparse deployment of sensors such that some sensors cannot be covered by the mobile sink under the imposed restrictions.

Also, another limitation of the proposed algorithms is that they are based on directly implementing a shortcut in the TSP tour, which is not energy efficient in all cases. For example, in Figure 5.15 there is a shortcut involving going from node one to node three and returning back to node one, which means that we can skip the second node. The second node then requires a higher transmission range, while the third node uses zero energy. It would therefore be better for the mobile sink to approach both nodes, to some

Table 5.1 Results for label covering problem and our algorithm, 20 nodes.

Minimum-length	energy-cost(label-cover)	Energy-cost(our problem)	R_{max}
639	26857	26742	75
518	48985	48874	100
464	88746	88746	125
437	103020	103020	150

Table 5.2 Results for label covering problem and our algorithm, 40 nodes.

Minimum-length	energy-cost(label-cover)	Energy-cost(our problem)	R_{max}
4945	45468	38115	75
4646	81471	61442	100
4102	150940	116130	125
3655	218980	207600	150

Table 5.3 Results for label covering problem and our algorithm, 100 nodes.

Minimum-length	energy-cost(label-cover)	Energy-cost(our problem)	R_{max}
5790	134450	97246	75
5183	48985	164010	100
4631	402130	260080	125
3686	675810	568060	150

extent, in order to balance energy consumption between the nodes as shown in Figure 5.16.

Therefore, in our future work, we intend to optimise the tour to have minimal energy consumption. Moreover, we intend to find a solution for scenarios with sparse sensor deployment (or a large geographical area), which cannot be covered by a single mobile sink. This will be done by either using multiple sensors and mobile sinks or allowing mobile sinks to utilise multi-hop forwarding. Another potential approach is to find the solution for the problem without considering the TSP tour as the first step.

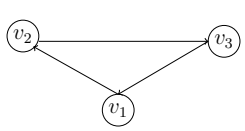


Fig. 5.14 TSP, 3 nodes

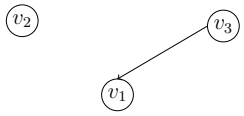


Fig. 5.15 Direct shortcut for Figure 5.14.

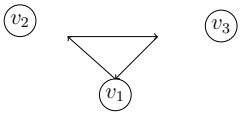


Fig. 5.16 Better shortcut for Figure 5.14.

Moreover, occasionally multiple nodes are closer to each other due to the random deployment and therefore when shortcuting is performed energy consumption is increased. As a result, this situation should be considered to improve the quality of the algorithm.

Furthermore, our heuristic algorithm produces a smaller tour than the imposed length

constraint with higher energy consumption in both scenarios.

Finally, another limitation with regards to the dynamic programming is that the values of the shortcuts are being rounded up to integers which could cause misguidance of the actual length of the tour to the mobile sink.

Chapter 6

Conclusion

This chapter reiterates the study of this thesis, followed by some directions for further investigations by others.

6.1 Thesis Summary

The main requirement of resource constrained sensors is to utilize their resources, in particular their energy more efficiently, to prolong their functionalities as much as possible. Hence, designing energy efficient algorithms remains a key challenging issue in WSNs. Therefore, this study examines energy consumption from two different points of view. In other words, this investigation revolves around two points: idle listening and mobility mechanism.

The primary goal of this study is to propose energy efficient mechanisms in WSNs for data collection. The first part of this study has focused on the idle listening perspective since the idle listening state lets the radio of the nodes be in the active state for a possible incoming message. Thereby a significant amount of energy is wasted.

Zhao and Tang [81, 82] have explored this problem in WSNs with tree topology under the restriction that data collection proceeds from bottom to top and only some nodes have data. Two issues in this setting are a rise. First, the issue of idle listening because each parent p must listen to its child c $|T_c|$ times, where $|T_c|$ indicates the number of nodes in the sub-tree rooted at the child c. Second, the side effect of the idle listening leads to huge latency. As a result Zhao and Tang have proposed an approach called successive-slot schedules to reduce the amount of idle listening in this restricted topology. In their technique, each parent, when it detects the idle listening from any of its children, then it should next time stop listening toward this child. In other words, detecting the idle

listening means that this child has no more packet for forwarding, therefore the parent node can turn off its radio towards this child.

Considering the interesting technique of Zhao and Tang, one can observe that there is more room to reduce idle listening further. My project has been built up upon their earlier work and proposed an optimization technique, called the extra-bit technique, which reduces idle listening further and also minimizes latency. This project has proposed that each child node adds one extra bit (0/1) to the packet as an indication to inform the parent whether or not more packets are coming. 1 means that more packets are coming while 0 means the end of transmission from this child. This extra bit alleviated this extra idle listening of the previous technique. This study has investigated further deriving the lower bound for data collection in a chain using successive-slot or extra-bit schedules. It has been proved that the optimal number of time slots is $4N - 6$, where $N \geq 3$ is the number of nodes in the network excluding the sink. The research has also presented how to calculate the expected amount of idle listening for extra-bit schedules and successive-slot schedules in chains and trees where each node has data with a fixed probability. We have also demonstrated that the expected amount of idle listening is significantly smaller with the extra-bit technique.

As the second contribution, the study has sought to derive the minimum length of the successive-slot/extra-bit schedule (latency) for various special cases of the tree and to provide the corresponding schedule. For example, $N + NM - 1$ is the lower bound for M balanced chains, where N is the length of each chain and M is the number of the chains. Similarly, for a four level K-ary tree $k(d + 2 + 2k + k^2 + k^3)$ indicates the lower bound, where d is the depth of the tree, $d = 4$, and $k \geq 2$ is the degree of each node in the tree. For the results for Rhizome trees, refer to the two scenarios in Chapter 4. Next, the project has proposed an algorithm that can obtain the optimal schedule for two of these special cases (balanced chains and Rhizome trees) individually that matches the lower bound. Nonetheless, and for unbalanced multi-chains, the length of the proposed algorithm is a few steps away from optimal. Subsequently, one can observe that the extra idle state, which is available among the nodes due to the nature of the communication, to avoid interference, can be alleviated by utilizing two frequencies. This leads to further energy saving and the further minimization of latency. Therefore, the study has sought to utilize two frequencies for all the above-mentioned cases, with the exception of the k-ary tree, and to prove the correctness of the theorems.

In order to further explore the issue of energy expenditure from a different perspective,

as another main part of our work, the study has investigated and used sink mobility. The used of a mobile sink is quite significant to reduce energy consumption by eliminating multi hop forwarding among the sensors. It is also beneficial to cover disconnected areas. However, in general finding the optimal trajectory for the mobile sink to traverse through the sensing field and collect data from sensors is NP-complete, thus various algorithms have been presented to find a good possible solution (trajectory). In this thesis, Chapter 5 has focused on an energy efficient restricted tour construction, which is also NP-complete. We have proposed two distinct algorithms to find an energy efficient constrained tour for the mobile sink to follow and collect data from sensors via single hop communication such that the total amount of energy that is spent by all the sensors is minimum. The first algorithm is heuristic, while the second one uses a dynamic programming technique. Furthermore, for each algorithm two scenarios have been considered. In the first scenario, we have assumed that there is no restriction on the transmission range of the sensors, whereas in the second setting we have considered that the maximum transmission range is limited. In the first algorithm, the main key factor to construct the tour is "max-ratio" for both scenarios. Max-ratio means that we choose the path that cover the maximum sensors such that the total energy that is spent by the internal nodes is minimum. Note that for both algorithms when the tour is constructed, then sensors adjust their transmission ranges according to the constructed tour to save energy.

This project has examined the complexity of each proposed algorithm. One can argue that while the complexity of the heuristic algorithm is polynomial (i.e., $TSP + O(N^4)$), nevertheless it cannot always find a solution. On the other hand, the complexity of the dynamic programming technique, which is $O(N^3 + N^2L)$, where N is the number of nodes and L is the length constraint of the tour, may be polynomial or exponential depending on the length of the tour, and furthermore, all the distance values must be integer. At the end, the simulation results in Matlab have illustrated that the proposed techniques, in particular the second algorithm, can significantly reduce the total energy consumption in the network in comparison with an existing algorithm (label covering problem) for the same length. All in all, this interesting project has been productive in terms of energy consumption.

6.2 Future Work Directions

Research is an endless journey and there are various unfolding questions which remain unresolved.

1. One of the biggest remaining challenging issue with regard to the successive-slot/extra-bit technique is either to find the optimal schedule for general trees or to provide sufficient evidence that it is NP-complete, and then propose an algorithm to achieve a good solution.
2. Devising a distributed algorithm can be a potential further direction for the extra-bit technique.
3. In Chapter 5, the project has considered TSP as the first step to find the tour, then based on it we constructed the restricted tour. It would be interesting to find the solution for the problem without considering the TSP tour as the first step.
4. As another potential future direction, it could be highly interesting to examine the combination of the optimal number of sinks and multi-hop transmissions in order to either trade-off between energy consumption and latency, or cover sparse or disconnected areas.
5. Further investigation can be done on multiple frequencies for densely deployed sensors and multiple antennas for mobile sinks to shorten latency. Additionally, replenishing sensors through mobile sinks is another interesting project.
6. It would be interesting to implement the proposed techniques in a real sensor testbed.
7. Another potential direction that could be taken into consideration is that choosing a subset of nodes which can cover the target area, namely, deactivation of redundant nodes that cover the same area.

Bibliography

[1] Sergey Alatartsev, Marcus Augustine, and Frank Ortmeier. Constricting insertion heuristic for traveling salesman problem with neighborhoods. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*, 2013.

[2] Seema Bandyopadhyay and Edward J Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1713–1723. IEEE, 2003.

[3] Jean-Claude Bermond, Jerome Galtier, Ralf Klasing, Nelson Morales, and Stephane Perennes. Hardness and approximation of gathering in static radio networks. *Parallel Processing Letters*, 16(02):165–183, 2006.

[4] Jean-Claude Bermond and Min-Li Yu. Optimal gathering algorithms in multi-hop radio tree-networks with interferences. In *Ad-hoc, Mobile and Wireless Networks*, pages 204–217. Springer, 2008.

[5] Archana Bharathidasan and Vijay Anand Sai Ponduru. Sensor networks: An overview. Technical report, Department of Computer Science, University of California, Davis, 2002.

[6] Yanzhong Bi, Limin Sun, Jian Ma, Na Li, Imran Ali Khan, and Canfeng Chen. Hums: an autonomous moving strategy for mobile sinks in data-gathering sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2007.

[7] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[8] F. Bouabdallah, N. Bouabdallah, and R. Boutaba. Load-balanced routing scheme for energy-efficient wireless sensor networks. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–6, Nov 2008.

[9] Fatma Bouabdallah. *Minimizing the Energy Consumption in Wireless Sensor Networks*. PhD thesis, Composante universitaire : IFSIC, IRISA - campus de Beaulieu - F - 35 042 Rennes Cedex, Novomber 2008.

[10] Gurashish Brar, Douglas M Blough, and Paolo Santi. Computationally efficient scheduling with the physical interference model for throughput improvement in wireless mesh networks. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 2–13. ACM, 2006.

[11] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5):481–494.

[12] Xujin Chen, Xiaodong Hu, and Jianming Zhu. Minimum data aggregation time problem in wireless sensor networks. In *Proceedings of the First International Conference on Mobile Ad-hoc and Sensor Networks (MSN'05)*, LNCS 3794, pages 133–142, Berlin, Heidelberg, 2005. Springer-Verlag.

[13] Hongsik Choi, Ju Wang, and Esther A Hughes. Scheduling for information gathering on sensor network. *Wirel. Netw.*, 15(1):127–140, January 2009.

[14] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1):165 – 177, 1990.

[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[16] David Coudert, Herv Rivano, and Xavier Roche. A combinatorial approximation algorithm for the multicommodity flow problem. In Roberto Solis-Oba and Klaus Jansen, editors, *Approximation and Online Algorithms*, volume 2909 of *Lecture Notes in Computer Science*, pages 256–259. Springer Berlin Heidelberg, 2004.

[17] Xuewu Dai, Peter E Omiyi, Kaan Bür, and Yang Yang. Interference-aware convergecast scheduling in wireless sensor/actuator networks for active airflow control applications. *Wireless Communications and Mobile Computing*, 14(3):396–408, February 2014.

[18] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2008.

[19] Anthony Ephremides, J.E. Wieselthier, and D.J. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE*, 75(1):56–73, Jan 1987.

[20] Sinem Coleri Ergen and Pravin Varaiya. TDMA scheduling algorithms for wireless sensor networks. *Wireless Networks*, 16(4):985–997, 2010.

[21] C. Florens and R. McEliece. Packets distribution algorithms for sensor networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1063–1072 vol.2, March 2003.

[22] Cédric Florens, Massimo Franceschetti, and Robert J McEliece. Lower bounds on data collection time in sensory networks. *Selected Areas in Communications, IEEE Journal on*, 22(6):1110–1120, 2004.

[23] Mounir Frikha. *Ad Hoc Networks: routing,QoS and Optimization*. John Wiley & Sons, Hoboken, NJ, 2011.

[24] Shashidhar Gandham, Ying Zhang, and Qingfeng Huang. Distributed time-optimal scheduling for convergecast in wireless sensor networks. *Comput. Netw.*, 52(3):610–629, February 2008.

[25] Shashidhar Rao Gandham, Milind Dawande, Ravi Prakash, and Subbarayan Venkatesan. Energy efficient schemes for wireless sensor networks with multiple mobile base stations. In *Global telecommunications conference, 2003. GLOBECOM'03. IEEE*, volume 1, pages 377–381. IEEE, 2003.

[26] Quang Gao, Keith J Blow, David J Holding, Ian W Marshall, and XH Peng. Radio range adjustment for energy efficient wireless sensor networks. *Ad hoc networks*, 4(1):75–82, 2006.

[27] Shuai Gao and Hongke Zhang. Energy efficient path-constrained sink navigation in delay-guaranteed wireless sensor networks. *Journal of Networks*, 5(6):658–665, 2010.

[28] Michael R Garey and David S Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*, 1979.

[29] Luisa Gargano and Adele A Rescigno. Optimally fast data gathering in sensor networks. In *Mathematical Foundations of Computer Science 2006*, pages 399–411. Springer, 2006.

[30] Mukhtar Ghaleb, Shamala Subramaniam, Mohamed Othman, and Zuriati Zukarnain. Predetermined path of mobile data gathering in wireless sensor networks based on network layout. *EURASIP Journal on Wireless Communications and Networking*, 2014(1):1–18, 2014.

[31] Thomas Grant. *Algorithms for Wireless Communication and Sensor Networks*. PhD thesis, University of Leicester, 2013.

[32] Jonathan L Gross and Jay Yellen. *Graph theory and its applications*. CRC press, 2005.

[33] Xin Guan, Lin Guan, Xin Gang Wang, and Tomoaki Ohtsuki. A new load balancing and data collection algorithm for energy saving in wireless sensor networks. *Telecommunication Systems*, 45(4):313–322, 2010.

[34] Liang He, Jianping Pan, and Jingdong Xu. Reducing data collection latency in wireless sensor networks with mobile elements. In *Computer communications workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 572–577. IEEE, 2011.

[35] Wendi B Heinzelman, Anantha P Chandrakasan, and Hari Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *Wireless Communications, IEEE Transactions on*, 1(4):660–670, 2002.

[36] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, pages 10 pp. vol.2–, Jan 2000.

[37] Ö. D. Incel, Amitabha Ghosh, Bhaskar Krishnamachari, and Krishnakant Chintalapudi. Fast data collection in tree-based wireless sensor networks. *IEEE Transactions on Mobile Computing*, 11(1):86–99, January 2012.

[38] Aravind Iyer, Sunil S Kulkarni, Vivek Mhatre, and Catherine P Rosenberg. A taxonomy-based approach to design of large-scale sensor networks. In *Wireless Sensor Networks and Applications*, pages 3–33. Springer, 2008.

[39] Sushant Jain, Rahul C Shah, Waylon Brunette, Gaetano Borriello, and Sumit Roy. Exploiting mobility for energy efficient data collection in wireless sensor networks. *Mobile Networks and Applications*, 11(3):327–339, 2006.

[40] Aman Kansal, Arun A Somasundara, David D Jea, Mani B Srivastava, and Deborah Estrin. Intelligent fluid infrastructure for embedded networks. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 111–124. ACM, 2004.

[41] Arie M.C.A. Koster and Xavier Muoz. *Graphs and Algorithms in Communication Networks: Studies in Broadband, Optical, Wireless and Ad Hoc Networks*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[42] Fabian Kuhn, Rogert Wattenhofer, and Aaron Zollinger. Ad-hoc networks beyond unit disk graphs. In *Proceedings of the 2003 joint workshop on Foundations of mobile computing*, pages 69–78. ACM, 2003.

[43] Sumit Kumar and Siddhartha Chauhan. A survey on scheduling algorithms for wireless sensor networks. *International Journal of Computer Applications*, 20(5):7–13, April 2011.

[44] Tung-Wei Kuo and Ming-Jer Tsai. On the construction of data aggregation tree with minimum energy cost in wireless sensor networks: Np-completeness and approximation algorithms. In *INFOCOM, 2012 Proceedings IEEE*, pages 2591–2595, March 2012.

[45] E. Lebhar and Z. Lotker. Unit disk graph and physical interference model: Putting pieces together. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8, May 2009.

[46] Liron Levin, Michael Segal, and Hanan Shpungin. Cooperative data collection in ad hoc networks. *Wireless networks*, 19(2):145–159, 2013.

[47] Xu Li, Amiya Nayak, and Ivan Stojmenovic. Sink mobility in wireless sensor networks. *Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Communication*, pages 153–184, 2010.

[48] Yingshu Li and My T Thai. *Wireless sensor networks and applications*. Springer Science & Business Media, 2008.

[49] Weifa Liang and Jun Luo. Network lifetime maximization in sensor networks with multiple mobile sinks. In *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*, pages 350–357. IEEE, 2011.

[50] Weifa Liang, Jun Luo, and Xu Xu. Prolonging network lifetime via a controlled mobile sink in wireless sensor networks. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6. IEEE, 2010.

[51] Junchao Ma, Wei Lou, and Xiang-Yang Li. Contiguous link scheduling for data aggregation in wireless sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 25(7):1691–1701, 2014.

[52] Junchao Ma, Wei Lou, Yanwei Wu, Xiang-Yang Li, and Guihai Chen. Energy efficient TDMA sleep scheduling in wireless sensor networks. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*, pages 630–638, 2009.

[53] Ming Ma and Yuanyuan Yang. Sencar: an energy-efficient data gathering mechanism for large-scale multihop sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 18(10):1476–1488, 2007.

[54] Mirela Marta and Mihaela Cardei. Using sink mobility to increase wireless sensor networks lifetime. In *World of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008. 2008 International Symposium on a*, pages 1–10. IEEE, 2008.

[55] Ioannis Papadimitriou and Leonidas Georgiadis. Maximum lifetime routing to mobile sink in wireless sensor networks. In *Proc. of the 13th IEEE SoftCom*, pages 1–5, 2005.

[56] Junyoung Park, Kyoungjin Moon, Sungjoo Yoo, and Sunggu Lee. Optimal stop points for data gathering in sensor networks with mobile sinks. *Wireless Sensor Network*, 4(01):8, 2011.

[57] David Pisinger. *Algorithms for knapsack problems.* PhD thesis, Dept. Computer science, University of Copenhagen, 1995.

[58] Ramesh Rajagopalan and Pramod K. Varshney. Data-aggregation techniques in sensor networks: A survey. *IEEE Communications Surveys & Tutorials*, 8(4):48–63, 2006.

[59] Subramanian Ramanathan and Errol L. Lloyd. Scheduling algorithms for multihop radio networks. *IEEE/ACM Trans. Netw.*, 1(2):166–177, April 1993.

[60] Theodore S. Rappaport. *Wireless Communications: Principles and Practice*. IEEE Press, Piscataway, NJ, USA, 1st edition, 1996.

[61] A. Rasul and T. Erlebach. The extra-bit technique for reducing idle listening in data collection. *International Journal of Sensor Networks (IJSNET)*, accepted, 2016.

[62] Aram Rasul and Thomas Erlebach. Reducing idle listening during data collection in wireless sensor networks. In *10th International Conference on Mobile Ad-hoc and Sensor Networks, MSN 2014, Maui, HI, USA, December 19-21, 2014*, pages 16–23, 2014.

[63] Aram Rasul and Thomas Erlebach. An energy efficient and restricted tour construction for mobile sink in wireless sensor networks. In *12th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2015, Dallas, TX, USA, October 19-22, 2015*, pages 55–63, 2015.

[64] Rahul C. Shah, Sumit Roy, Sushant Jain, and Waylon Brunette. Data mules: modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks*, 1(2-3):215–233, 2003.

[65] Yi Shi, Y Thomas Hou, Jia Liu, and Sastry Kompella. How to correctly use the protocol interference model for multi-hop wireless networks. In *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing*, pages 239–248. ACM, 2009.

[66] Kazem Sohraby, Daniel Minoli, and Taieb Znati. *Wireless sensor networks: technology, protocols, and applications*. John Wiley & Sons, Hoboken, NJ, 2007.

[67] A.A. Somasundara, A. Ramamoorthy, and M.B. Srivastava. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 296–305, Dec 2004.

[68] Arun A. Somasundara, Aditya Ramamoorthy, and Mani B. Srivastava. Mobile element scheduling with dynamic deadlines. *IEEE Transactions on Mobile Computing*, 6(4):395–410, April 2007.

[69] Wen-Zhan Song, Fenghua Yuan, and R. LaHusen. Time-optimum packet scheduling for many-to-one routing in wireless sensor networks. In *Mobile Adhoc and Sensor Systems (MASS), 2006 IEEE International Conference on*, pages 81–90, Oct 2006.

[70] Ryo Sugihara and Rajesh K. Gupta. Improving the data delivery latency in sensor networks with controlled mobility. In *Distributed Computing in Sensor Systems, 4th IEEE International Conference, DCOSS 2008, Santorini Island, Greece, June 11-14, 2008, Proceedings*, pages 386–399, 2008.

[71] My T Thai and Ding-Zhu Du. Connected dominating sets in disk graphs with bidirectional links. *Communications Letters, IEEE*, 10(3):138–140, 2006.

[72] Sameer Tilak, Nael B. Abu-Ghazaleh, and Wendi Heinzelman. A taxonomy of wireless micro-sensor network models. *ACM Mobile Computing and Comunications Review*, 6(2):28–36, 2002.

[73] Ashraf Wadaa, Stephan Olariu, Larry Wilson, Mohamed Eltoweissy, and Kennie Jones. Training a wireless sensor network. *Mobile Netw. Appl.*, 10(1-2):151–168, February 2005.

[74] Jin Wang, Xiaoqin Yang, Jian Shen, Ping Guo, and Feng Xia. Mobile-sink routing algorithm based on energy and distance for wireless sensor networks. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 951–956. IEEE, 2013.

[75] Z Maria Wang, Stefano Basagni, Emanuel Melachrinoudis, and Chiara Petrioli. Exploiting sink mobility for maximizing sensor networks lifetime. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 287a–287a. IEEE, 2005.

[76] Ya Xu, John Heidemann, and Deborah Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, pages 70–84, New York, NY, USA, 2001. ACM.

[77] Haibo Zhang, Pablo Soldati, and Mikael Johansson. Optimal link scheduling and channel assignment for convergecast in linear wirelesshart networks. In *Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, 2009. WiOPT 2009. 7th International Symposium on*, pages 1–8. IEEE, 2009.

[78] Xinyu Zhang and Kang G Shin. E-mili: energy-minimizing idle listening in wireless networks. *Mobile Computing, IEEE Transactions on*, 11(9):1441–1454, 2012.

[79] Ying Zhang, Shashidhar Gandham, and Qingfeng Huang. Distributed minimal time convergecast scheduling for small or sparse data sources. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 301–310. IEEE, 2007.

[80] Miao Zhao and Yuanyuan Yang. Bounded relay hop mobile data gathering in wireless sensor networks. *Computers, IEEE Transactions on*, 61(2):265–277, 2012.

[81] Wenbo Zhao and Xueyan Tang. Scheduling data collection with dynamic traffic patterns in wireless sensor networks. In *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China*, pages 286–290, 2011.

[82] Wenbo Zhao and Xueyan Tang. Scheduling sensor data collection with dynamic traffic patterns. *IEEE Transactions on Parallel and Distributed Systems*, 24(4):789–802, April 2013.

[83] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 187–198. ACM, 2004.

[84] Jin Zheng, Xinlin Xu, and Guojun Wang. Energy efficient data aggregation scheduling in wireless sensor networks. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 1662–1667. IEEE, 2011.

[85] M. Zorzi and R.R. Rao. Geographic random forwarding (geraf) for ad hoc and sensor networks: multihop performance. *Mobile Computing, IEEE Transactions on*, 2(4):337–348, Oct 2003.

Appendix A

A.1 Tour for Three Network Sizes with Specific Length Constraint

In this section, we show some plots for specific length constraints in order to visually see the tour after simulations. Note that these results are extra simulations for the algorithms proposed in Chapter 5.

Figure A.1 represents the TSP tour for 20 nodes whereas Figure A.2 and A.3 are the results of simulations based on the heuristic algorithm for both scenario 1 and 2 respectively when the length constraint is set to 700 (L=700m). The achieved length is 666m and energy consumption is 27343 units after the simulation for the first scenario, whereas the achieved length is 677m and energy consumption is 31028 units after the simulation for the second scenario when the transmission range is set to 75m (R_{max}=75m).

On the other hand, Figure A.4 and A.5 are the result of simulation based on the dynamic programming algorithm for both scenario 1 and 2 with the same conditions. Interestingly, despite the fact that the achieved length and energy consumption are the same, that is, L=700m and energy consumption is 21671 units, they have slightly different tours.

Similarly, Figure A.6 shows the TSP tour for 40 nodes whereas Figure A.7 and A.8 are the result of simulation based on the heuristic algorithm for both scenario 1 and 2 when the length constraint is set to 4200 (L=4200m). The achieved length is 4007 and energy consumption is 159180 units after the simulation for the first scenario; whereas the achieved length is 4167m and energy consumption is 138060 units after the simulation for the second scenario when the transmission range is 150m (R_{max}=150m). However, the achieved results for the same specifications based on the dynamic programming algorithm for both scenarios is coincidentally the same and shown in figure A.9; namely, achieved

length is exactly 4200m and energy consumption is 104785 for scenario 1 and 2.

Likewise, Figure A.10 illustrates the TSP tour for 100 nodes whereas Figure A.11 shows the tour after simulation for both scenarios based on the heuristic algorithm, i.e. they have the same result. Figure A.12 shows the tour after simulation for both scenarios based on the dynamic programming algorithm.

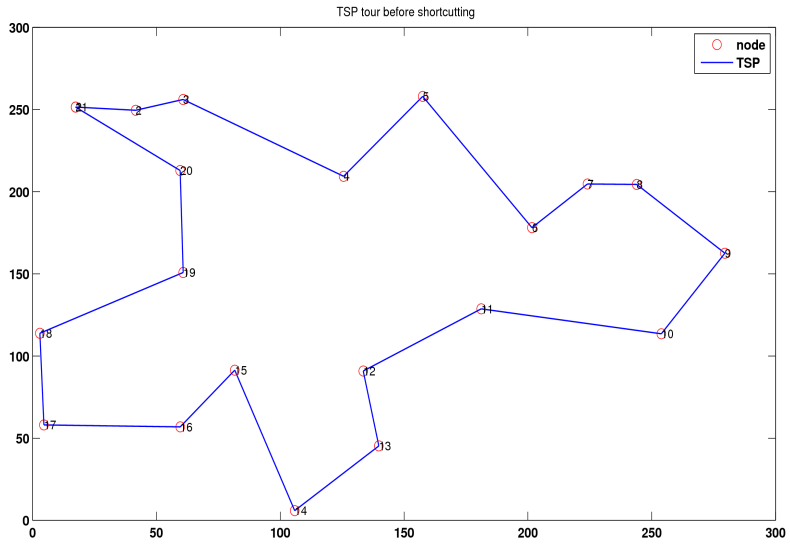


Fig. A.1 TSP tour, 20 nodes.

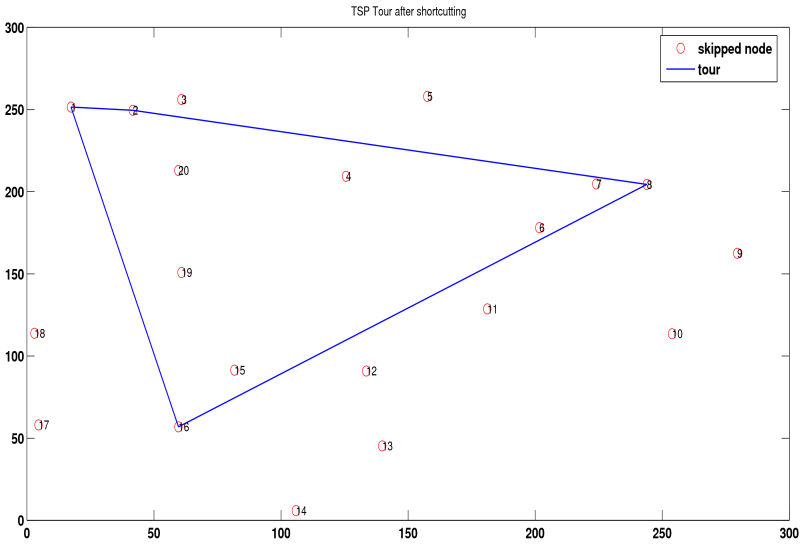


Fig. A.2 Heuristic algorithm, scenario 1, L=700, 20 nodes.

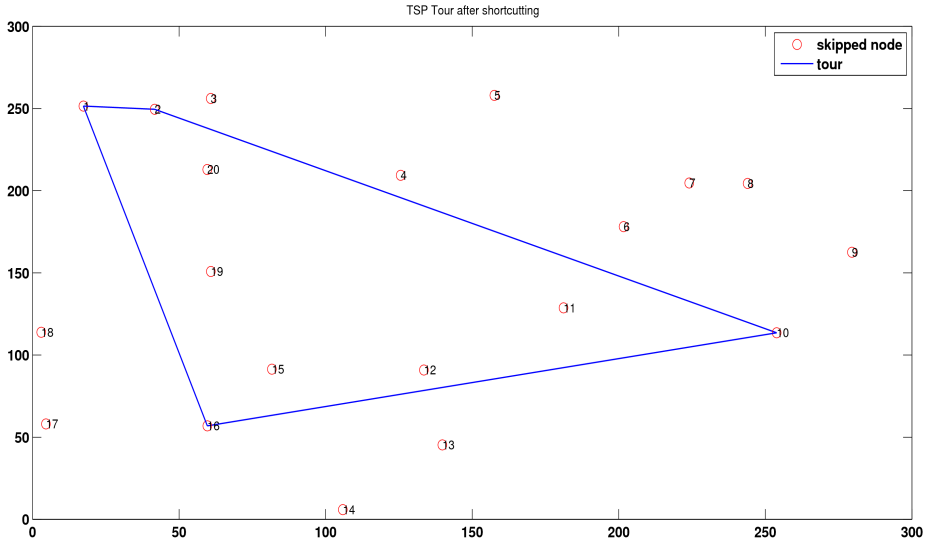


Fig. A.3 Heuristic algorithm, scenario 2, L=700 & R_{max}=75, 20 nodes.

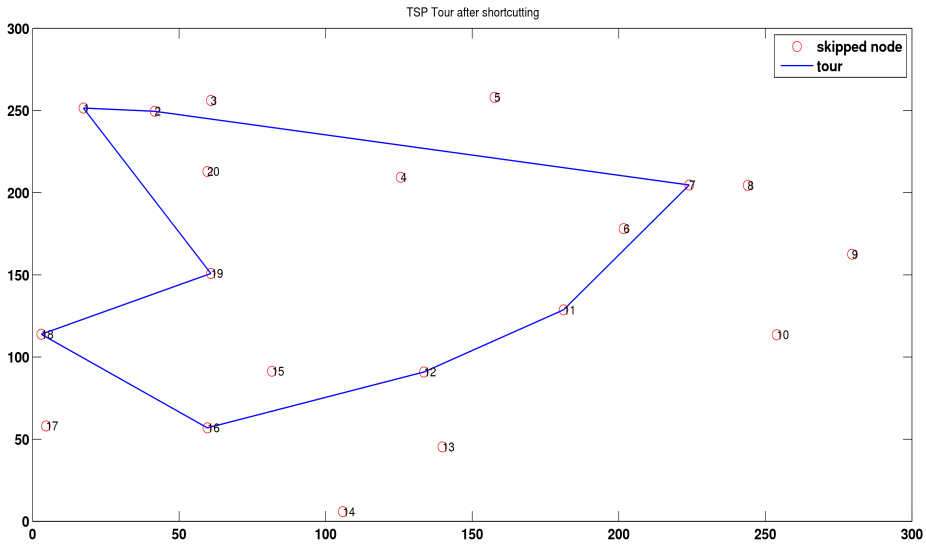


Fig. A.4 Dynamic programming, scenario 1, L=700, 20 nodes.

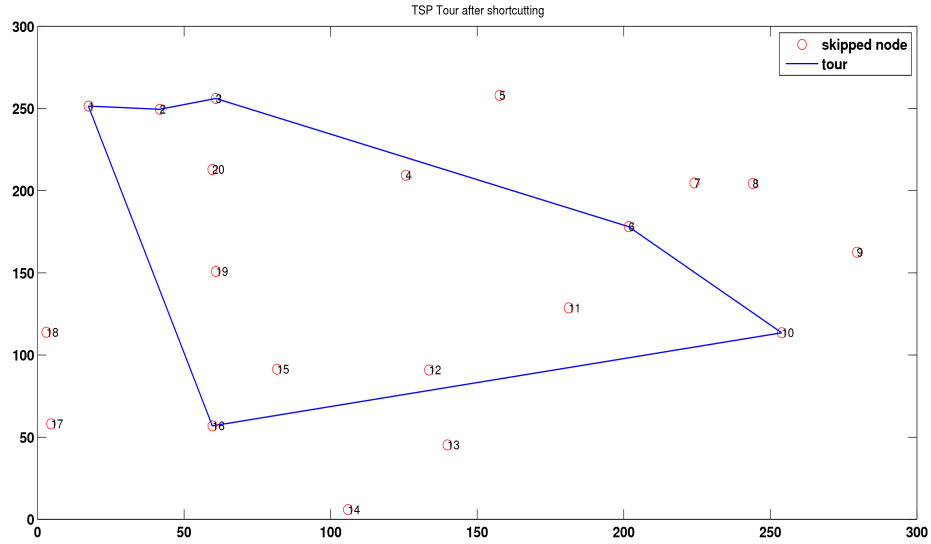


Fig. A.5 Dynamic programming, scenario 2,L=700 & R_{max}=75 , 20 nodes.

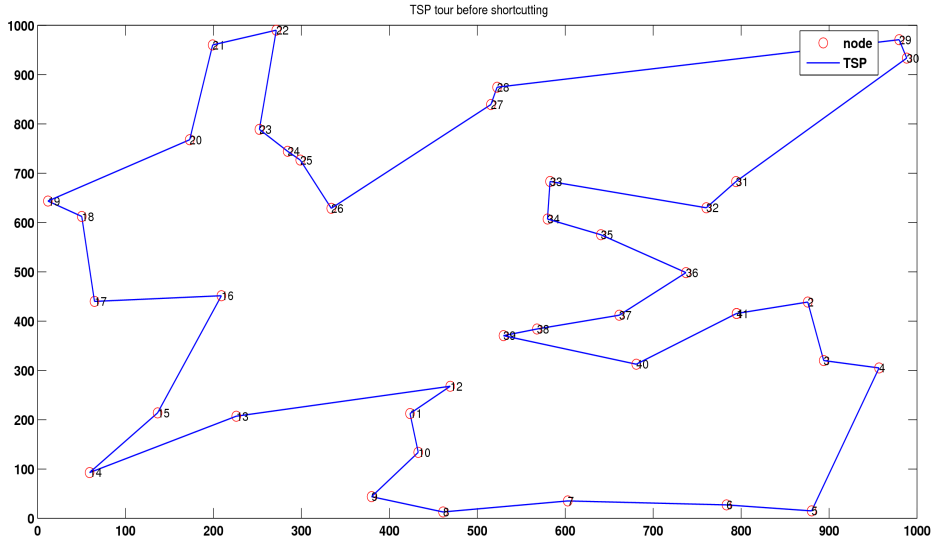


Fig. A.6 TSP tour, 40 nodes.

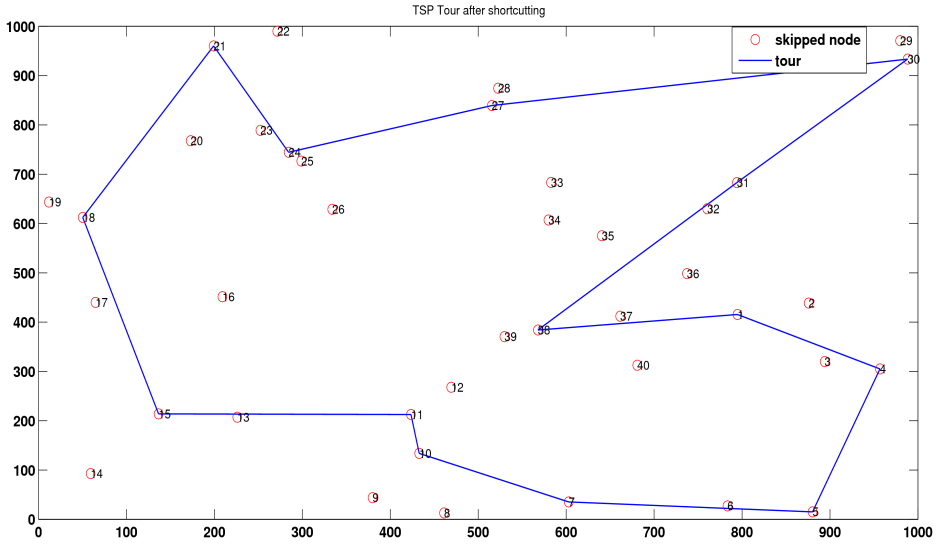


Fig. A.7 Heuristic algorithm, scenario 1, L=4200, 40 nodes.

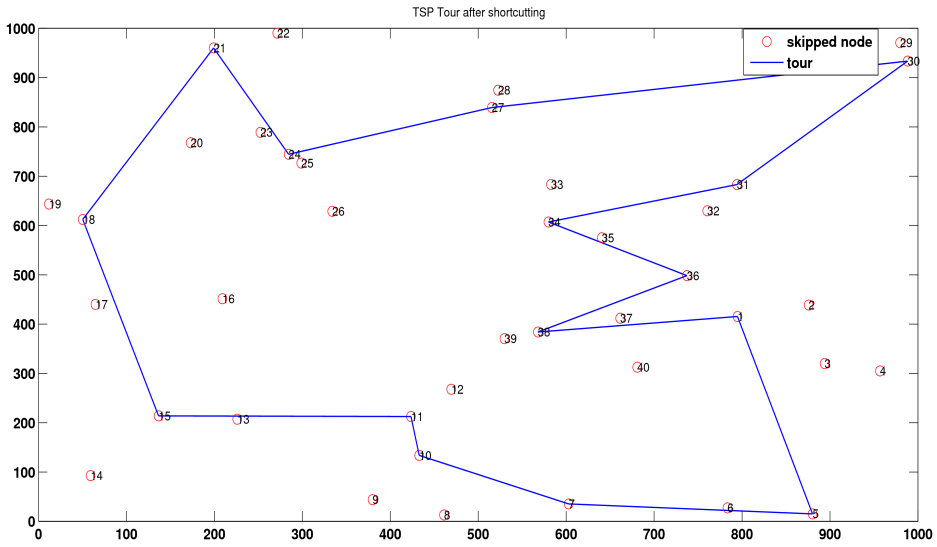


Fig. A.8 Heuristic algorithm, scenario 2, L=4200 & R_{max}=150, 40 nodes.

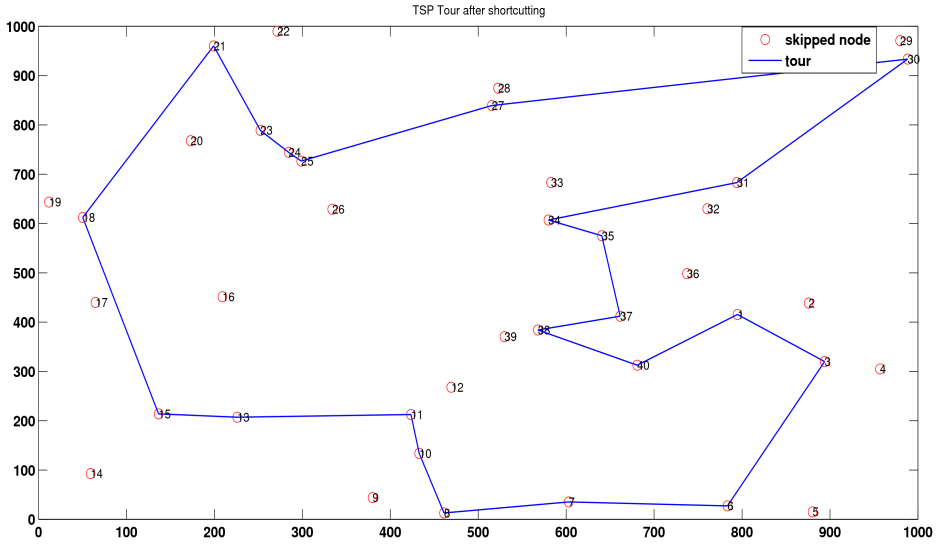


Fig. A.9 Dynamic programming, scenario 1 & 2, L=4200 & R_{max}=150, 40 nodes.

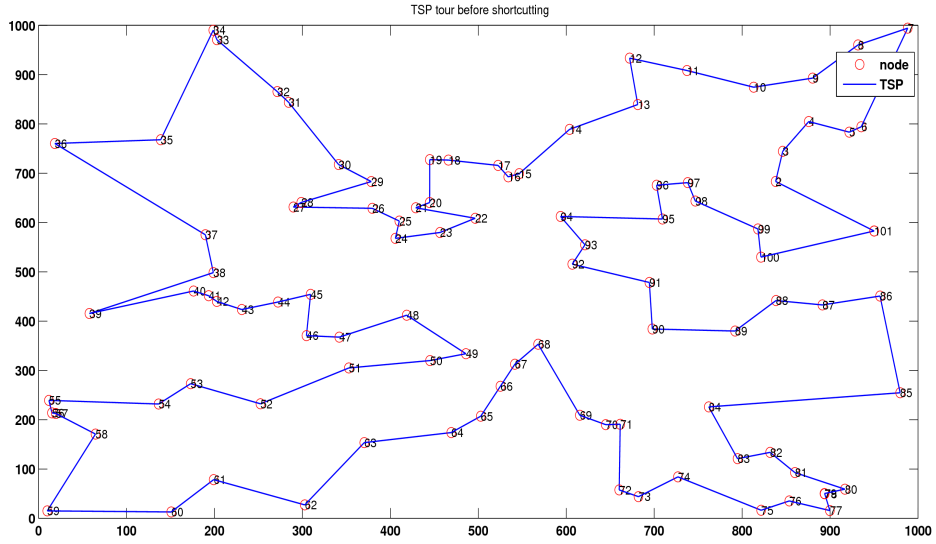


Fig. A.10 TSP tour, 100 nodes.

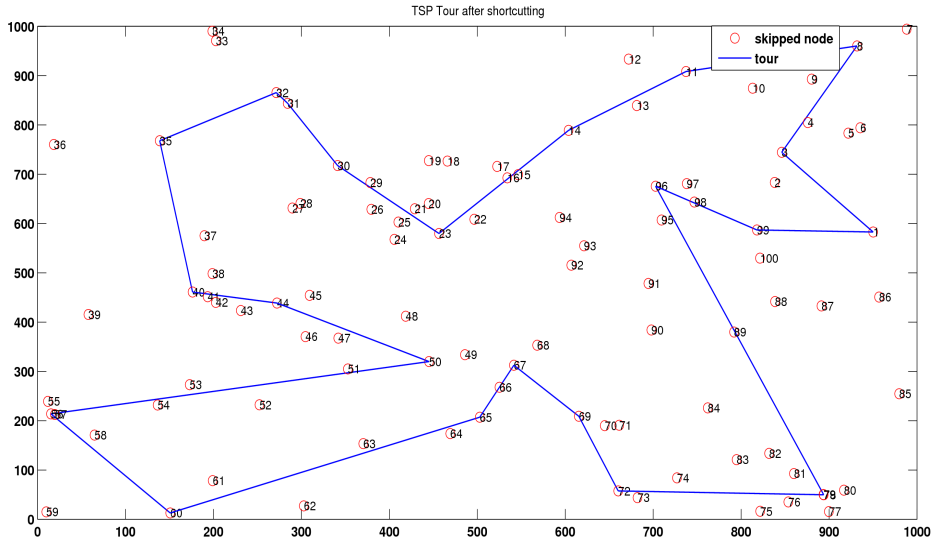


Fig. A.11 Heuristic algorithm, scenario 1 & 2, L=500 & R_{max}=150, 100 nodes.

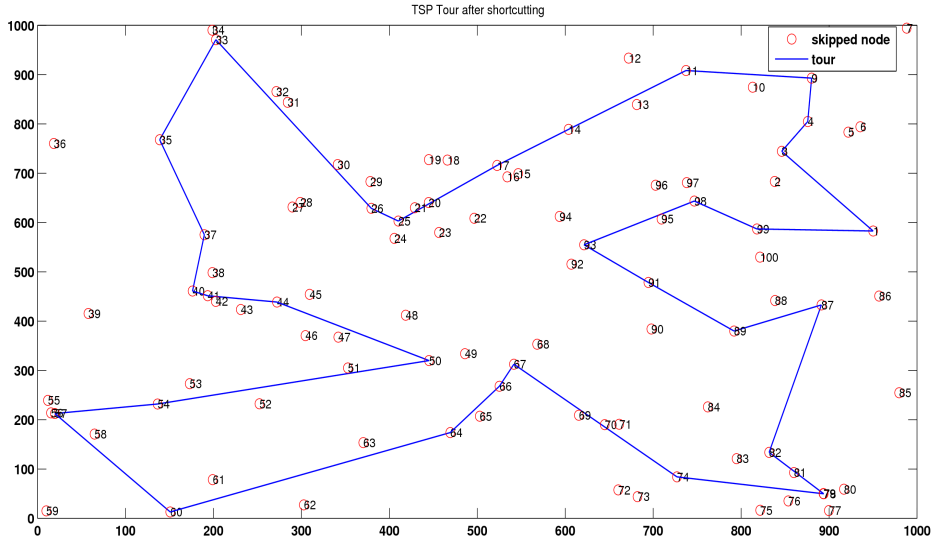


Fig. A.12 Dynamic algorithm, scenario 1 & 2, L=500 & R_{max}=150, 100 nodes.