M.Sc. Engg. Thesis

Energy-aware Path-Planning for a Mobile Data Collector in Wireless Sensor Network

by

Md. Shaifur Rahman

Submitted to

Department of Computer Science and Engineering

in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology (BUET)

Dhaka 1000

January 2013

Dedicated to my loving parents

Author's Contact

Md. Shaifur Rahman

Lecturer

Department of Computer Science & Engineering

Bangladesh University of Engineering & Technology (BUET).

Email: shaifur.at.buet@gmail.com

The thesis titled "Energy-aware Path-Planning for a Mobile Data Collector in Wireless Sensor Network", submitted by Md. Shaifur Rahman, Roll No. **0409052028P**, Session April 2009, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on January 16, 2013.

Board of Examiners

1. ____________________________________

Dr. Mahmuda Naznin — Chairman (Supervisor)
Associate Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

2. ____________________________________

Dr. A.S.M. Latiful Hoque — Member (Ex-Officio)
Head and Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

3. ____________________________________

Dr. Md. Mostofa Akbar — Member
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

4. ____________________________________

Dr. Masud Hasan — Member
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology, Dhaka.

5. ____________________________________

Dr. Md. Monzur Morshed — Member (External)
Associate Professor
Department of Accounting and Information Systems
Dhaka University, Dhaka.

Candidate's Declaration

This is hereby declared that the work titled "Energy-aware Path-Planning for a Mobile Data Collector in Wireless Sensor Network" is the outcome of research carried out by me under the supervision of Dr. Mahmuda Naznin, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Md. Shaifur Rahman

Candidate

Abstract

Data gathering or the process of collecting and delivering data packet in a resource constrained Wireless Sensor Network (WSN) is a very challenging task as the sensors become out of power anytime during the data gathering period. One of the methods of addressing this problem is to use a dedicated mobile element called *Mobile Data Collector* (MDC) which travels along the network, collects data packets directly from the sensor nodes and carry the data to the sink. The use of MDC has become popular as it elongates the lifetime of the sensor network, reduces the cost of deployment etc. Besides, it can gather data even in a disconnected and sparse sensor network.

Using an MDC in a WSN is challenging in various aspects. It's mobility can be pre-planned or random. In the case of random mobility, one or more sensor nodes may not be visited by the MDC at all. In the case of pre-planned mobility, the most important objective is to cover all the sensor nodes. However, given the infinite number of points in the region, the optimal path-planning of an MDC becomes intractable. In that case, we can use the *Travelling Salesman Problem* tour or *TSP-tour* to find the solution for a good path which ensures the data gathering from all of the sensor nodes. In this thesis, we prove that, a *TSP-tour* ensures the maximum lifetime for the WSN if data is collected by the MDC.

There is another risk involved in using an MDC in the WSN, which is called data delivery latency. The MDC has to halt and collect data from the sensor nodes. The period of a complete tour is comparatively higher than the time required to send packets to the sink by multi-hop forwarding. The packet delivery latency in the case of *TSP-tour* by the MDC may be too high for some real-time applications of the WSN. Therefore, we need to shorten the *TSP-tour* by the MDC.

In our research, we present a shorter tour than the *TSP-tour* by the MDC. Our method iteratively shortens the tour by finding Shortcuts. We find that, to communicate with a sensor node, the MDC

does not have to visit the exact location of the sensor node; instead, visiting any point within the transmission region suffices for the data collection.

We use *OMNET++* simulator to verify the performance of our algorithm. We design a realistic test bed, we compare our tours with the relevant tours and we find that our method has the lower data delivery latency. The lifetime of the WSN in our method is as good as that of the *TSP-tour*. In addition to that, we find the packet-drop rate, the throughput, the tour-time better in using our method.

The running time of our algorithm is polynomial ($O(mn^2)$, where m is the number of iteration and n is the number of sensor nodes). Even though the problem of finding the minimum length *TSP-tour* is *NP-Complete*, there exist many approximate algorithm for this computation which runs in polynomial time. Therefore, Our method runs in polynomial time.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

A *Wireless Sensor Network* (WSN) consists of spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. The sensor nodes then pass data through the network to a location known as *sink* which aggregates and permanently records all the sensed data. There are many applications as well as many challenges of a WSN. In this work, we address the particular problem of collecting data packets from the static sensor nodes. In fact, there are a lot of approaches to address this problem ([3, 4]). We propose a method in which a dedicated mobile element called *Mobile Data Collector* (MDC) is used for collecting data from the sensor nodes and depositing those to the sink. Planning a path for the MDC such that all the sensor nodes can use it to send data to the sink is challenging. It is even more challenging if some other requirements like increasing lifetime of the network, decreasing delay of sending packets to the sink etc. are to be met.

In this chapter, we introduce different methods for collecting data from the sensor nodes, different types of mobile elements which can be present in a WSN, pros and cons of using mobile elements for collecting data and finally few projects involving use of mobile elements.

1.1 Overview of Data Gathering

Usually, the sensor nodes in a Wireless Sensor Network (WSN) monitor their environment, sample data, and forward data packets to a remotely located base station called *sink*. A typical WSN containing sensors and sink is depicted in Figure 1.1. Collecting data packets from the sensor nodes by the sink is known as *Data Gathering* problem [3, 4].

Figure 1.1: A typical WSN

The data packets which cannot be sent from the sensor nodes to the sink, have to be ultimately discarded as tiny sensor nodes suffer from the buffer and power constraint ([5, 6]). Data contained in those packets are then lost. In many applications like monitoring or targets tracking, if packets cannot be sent to the sink within a certain time period, the data contained in those packets become useless. Therefore, data gathering from the sensor nodes is very important and challenging in the power and buffer constrained WSN.

1.1.1 Data Gathering Methods

There are many approaches of data gathering in a WSN. We can classify them as follows.

Direct Contact: If the sink and the sensor nodes are within each others range, then those nodes can communicate directly. This is depicted in Figure 1.2. This method is known as *data gathering by direct contact*. But, this is not very practical as sensor nodes are usually deployed far away from the sink, and the nodes suffer from the limited radio range.

Multihop Forwarding: Sensor nodes can act as relay nodes by forwarding packets received from the other sensor nodes. To forward packets, a path has to exist from the target source node to the sink. The sensor nodes in this path are within the transmission range of each other. Such a path is depicted in Figure 1.3. The problem of finding a suitable forwarding path is known as *routing problem* in a

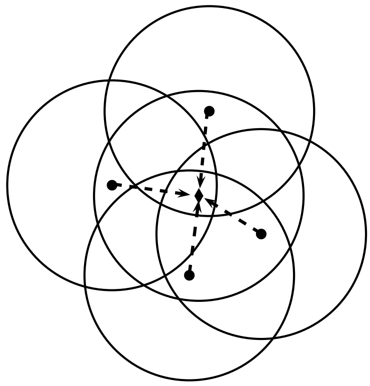


Figure 1.2: Data gathering by direct contact method in a WSN

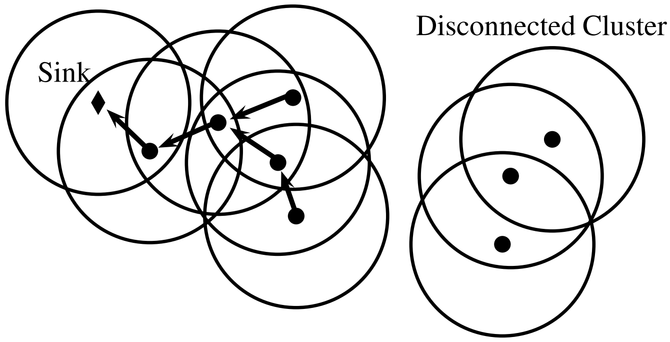


Figure 1.3: Data gathering by multi-hop forwarding in a WSN

WSN. However, multi-hop forwarding or routing is not possible for disconnected clusters or nodes in the WSN. This scenario is very common in a connectivity failure-prone WSN.

Mobile Elements: Packets can be collected from sensor nodes by mobile elements. The sink can itself be mobile, and travel through the network. When the mobile sink comes within the range of a sensor node, the sink collects packets from the sensor node. Some or all of the sensor nodes can be mobile. These mobile sensors can travel to the sink, deposit packets and return to its area of interest in the WSN. Yet, in another approach, there may be one or more dedicated agents for data collection from the sensor nodes. Different types of mobile elements in a WSN is discussed in Section 1.3.

1.1.2 Comparison of Different Data Gathering Methods in a WSN

We present a summary of pros and cons of different data gathering methods in a WSN in Table 1.1. As shown in the comparison, using Mobile Element is the best choice for energy-savings and increasing

Attribute	Direct Contact	Multi-hop Forwarding	Mobile Elements
Data Delivery Latency	Very low, sink is a direct neighbor, packets are delivered almost instantly	Low, depends on the hop-count of the path from the sensor to the sink	High, depends on the speed and path-length of the mobile element
Energy Requirement	Very High, the radio range has to be large to match the width of the WSN	Moderate, the number of sensor nodes has to be sufficient to cover any holes in the network	Low, can be minimal if there is no multi-hop forwarding to the mobile elements
Overhead	No overhead	Overhead for finding path to the sink	Overhead for path-planning of the mobile element

Table 1.1: Comparison among different data gathering methods in the WSN

the lifetime of the sensor nodes. But, high latency is a big challenge in this method. Besides, there is overhead involved in path-planning. Therefore, planning an energy-efficient path with low-latency for the mobile elements for data collection in the WSN is a challenging task which attracts the network researchers in the recent years.

1.2 WSN with Mobile Elements (ME)

A WSN with Mobile Elements (ME) usually has three types of nodes:

Regular Sensor Nodes: These nodes are the sources of information. The main task of these nodes is sensing. These nodes may also forward or relay messages in the network with the multi-hop forwarding. Now a days, different types of sensor nodes are commercially available ([6, 5]). Picture of widely used low-powered *Mica* mote is depicted in Figure 1.4.

Sink or Base Station: This node is the destination of information. It collects data packets generated by sensor nodes either *directly* (i.e., by visiting sensor nodes and collecting data from those nodes) or *indirectly* (i.e., through intermediate nodes or mobile elements). The sink can use data collected from the sensors autonomously or make the data available to the interested users through an Internet connection. A sink node with provision for internet gateway is depicted in Figure 1.5.

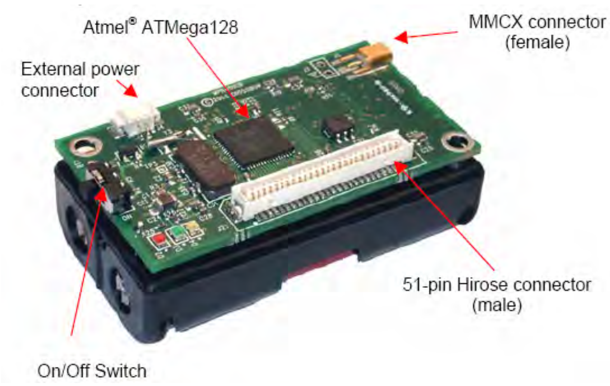


Figure 1.4: Ordinary Sensor Node (Mica2 Mote)

Figure 1.5: Sink of a WSN

Figure 1.6: Robotic car for data collection in a WSN

Support Nodes: These nodes perform a specific task, such as acting as intermediate data collectors or mobile gateways. These nodes are not the sources nor the destinations of messages, but exploit mobility to support network operations such as data collection. A robotic car with navigation capability and a sensor node mounted on top of it, can be used as a mobile data collector. Such an assembly is depicted in Figure 1.6.

We note that mobility might be involved at the different network components. For instance, sensor nodes may be mobile and sinks might be static, or vice-versa. Depending on the specific scenario, the support nodes might be present or not. We term such network as *WSN-ME*. When there are only regular nodes, the resulting WSN-ME architecture is *homogeneous* or *flat*. On the other hand, when support nodes are present, the resulting WSN-ME architecture is *non-homogeneous* or *tiered*. Furthermore, WSN-MEs can be very sparse as there is no concern for coverage. Instead, the mobile elements can cover any network holes.

1.2.1 Advantage of Using Mobile Elements in a WSN

Any element in a WSN which is mobile and can communicate with other nodes is called a *Mobile Element* or ME. There are many advantages of using ME's in a WSN. A few of those are outlined as follows.

Better Connectivity Irrespective of the Number of Nodes: If an ME is used in a WSN, the requirement of dense WSN can be relaxed. The ME can travel to the farthest disconnected nodes or clusters of the network and fetch the data packets to the sink. Thus, this is a very feasible solution for data collection in a sparse sensor network or clustered sensor network.

Cost Reduction: If ME's are used in a WSN, the network can be very sparse and only the required number of sensor nodes need to be deployed. We only have to deploy sensor nodes in the region of interest instead of covering the whole region for full-connectivity. The sensor nodes and the sinks do not have to be mobile which reduces deployment-cost [6, 5].

Handling the Funnelling Effect: In a WSN, traffic streams are created from all sensor nodes towards the sinks. Without an ME in the network, sensor nodes have to forward other nodes' packets. Routing paths are created from each sensor node to the sink for this purpose. The forwarding nodes that are closed to the sink has to transmit more packets than the peripheral ones. As a result, energy of these nodes deplete fast. The neighbors of the sink cause the bottleneck for traffic and this problem is known as the *Funnelling Effect* [7]. If Mobile Elements are used in the WSN, the nodes closer to the sink are not overloaded with traffic. Besides, in a desired scenario, the ME can collect data packets directly from each node. Therefore, there is no forwarding in the network and the sensor node can save power.

Increase in Reliability: A WSN without the ME's is usually dense for full coverage and connectivity ([1]). In that type of WSNs, reliability is penalized by interference and collisions. Communication paradigm in such a WSN is multi-hop routing where the packet loss increases with the increase of the number of hops ([8]). If the WSN uses Mobile Elements, the MEs can visit nodes in the network and collect data directly through the single-hop transmission. This reduces not only the contention and collisions, but also the message loss.

1.2.2 Challenges of Using Mobile Elements in a WSN

There are some challenges involved in using ME's in a WSN. We discuss these challenges as follows.

Path-planning of a Mobile Element: The ME can be in many forms ([9, 5, 6]). For example, it can be another sensor node mounted on a robotic car or it can be a smart vehicle with advanced navigation capability. Planning path for the ME for collecting data packets from the sensor nodes is challenging as it is difficult to obtain an optimized path with the increasing network life-time and with the decreasing latency of data packets. A whole gamut of literature addressing path-planing has been presented in Chapter 2.

Contact Detection: Establishing a contact between a static sensor node and a mobile element requires special architecture. As shown in Figure 1.7, the ME comes within contact of the sensor node only for a finite interval of time. If the interval is not known to the sensor node, it has to poll for the presence of the ME. If the sensor node continuously polls for the ME, its power depletes fast. Another approach may be to use the ME to wake up a sleeping sensor node. However, if the sleep-cycle of the sensor node is not synchronized with that of the ME, the contact fails. This implies that many of the popular MAC's such as *S*-MAC, *T*-MAC etc. are not very useful in this scenario, since those protocols exploit synchronized schedules for energy-efficient communications [8].

Coordination Among Multiple Mobile Elements: If there are more than one ME present in the

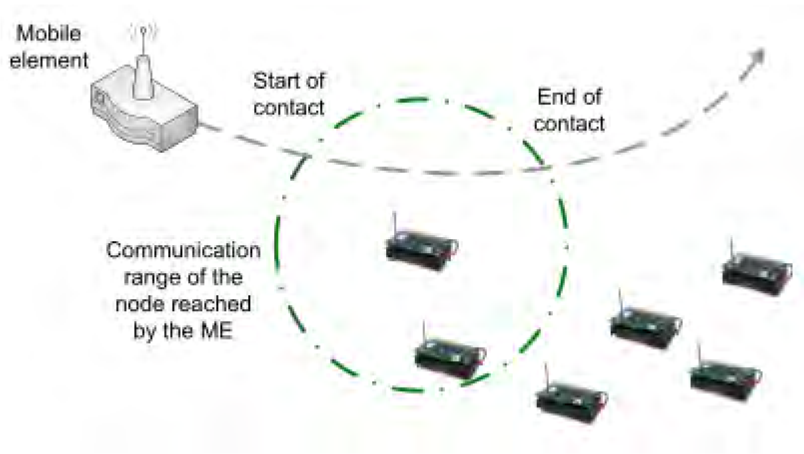


Figure 1.7: Detection of Mobile Element (ME) within the range of the transmission

network, a robust coordination is required among them so that no sensor node is left out. Scheduling these ME's for data gathering is a very challenging research problem.

1.3 Taxonomy of Mobile Elements in a WSN

In this section, we introduce different types of Mobile Elements which collect data in a WSN. We present the classification proposed in [1].

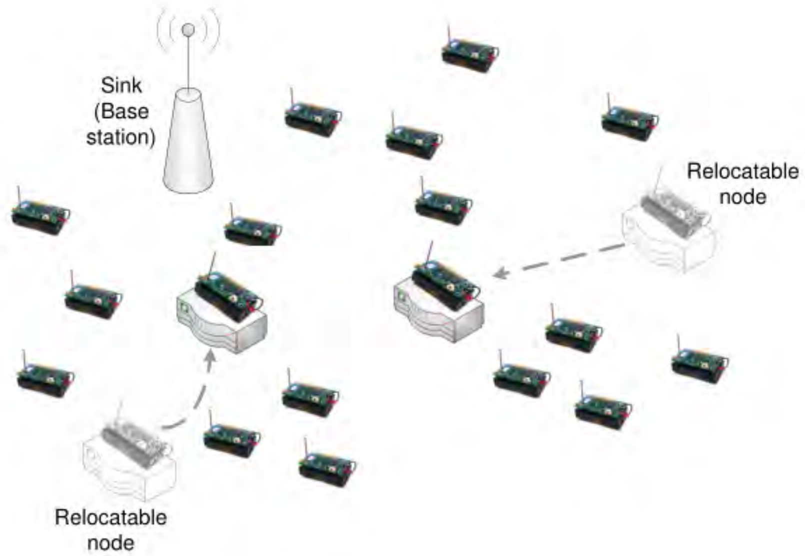


Figure 1.8: Relocatable Nodes in a WSN ([1])

1. **Relocatable Nodes:** These are the mobile nodes which change their locations to characterize the sensing area in a better way, or to forward data from the source nodes to the sink. Relocatable nodes do not carry data as they can move in the network. Rather, they only change the topology of the network. A WSN architecture based on relocatable nodes is depicted in Figure 1.8. Although the ordinary nodes might be relocatable, in most of the cases, special mobile elements (e.g., Support Nodes) are used as relocatable nodes.

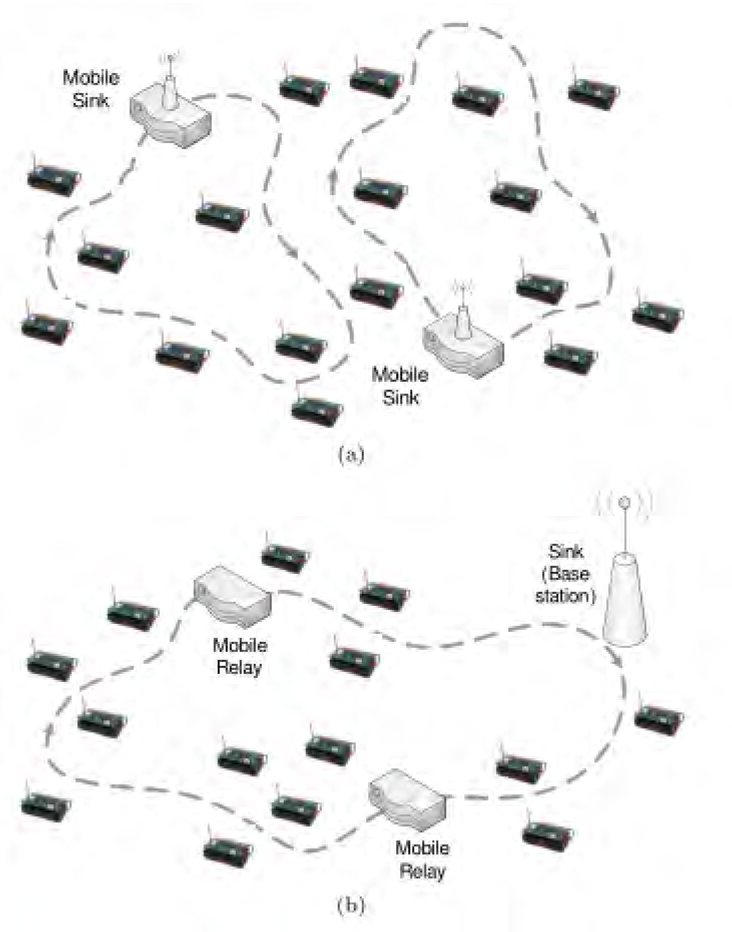


Figure 1.9: (a) A WSN with mobile sinks and (b) A WSN with mobile relays [1]

2. **Mobile Data Collector (MDC):** These are the Mobile Elements which visit the network to collect data from sensor nodes. Depending on whether the MDC's are endpoint or target nodes for communication, these are classified as either *Mobile Sinks* or *Mobile Relays*.

   Mobile Sinks: These are the mobile nodes which are the ultimate destinations of all messages originated by sensors, i.e., they represent the endpoints of data collection in a WSN with Mobile Elements. They can autonomously consume collected data for their own purposes, or they can make them available to the remote users by using a long range wireless internet connection. The architecture is depicted in Figure 1.9(a).

Mobile Relays: These are the support nodes which gather and store packets, and carry those to the sinks. These nodes are not the endpoints of communication and only act as *Mobile Forwarders*. In this case, the collected data packets move along with them, until the *Mobile Relays* get in contact with the sink. The architecture is depicted in Figure 1.9(b).

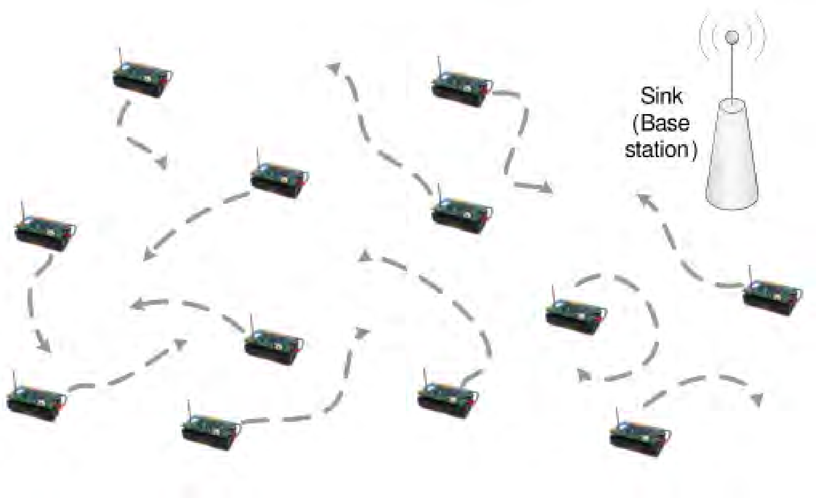


Figure 1.10: Mobile Peers in WSN [1]

3. **Mobile Peers:** $MobilePeers$ are ordinary mobile sensor nodes in a WSN. These nodes can both generate and relay messages in the network. When there is a peer in the communication range of the base station, it transfers its own data as well as those gathered from other peers while moving in the sensing area. A WSN architecture based on $MobilePeers$ is depicted in Figure 1.10.

In this thesis, we address the issues of data gathering technique using a Mobile Element with the goal of the energy saving.

1.4 Applications of Mobile Elements in the WSN

Mobile Elements have been successfully employed in the context of wildlife monitoring applications, such as tracking of zebras in the ZebraNet project [10] or whales in the SWIM system [11]. Sensor nodes are attached to the animals and act as peers, so that not only do they generate their own data, but also carry and forward all data coming from other nodes which they have been previously in contact with. When the ME's get closed to the base station, they transfer all the gathered data. Data which have already been transferred to the base station are flushed by those in order to save storage. ME's can also be used for opportunistic data collection in urban sensing scenarios [12]. Sample applications include personal monitoring (e.g., physical exercise tracking), civil defense (e.g., hazards and hot-spot reporting to police officers) and collaborative applications (e.g., information sharing for tourism purposes). In this context, sensors are not used mainly for monitoring the environment, but are rather exploited to characterize the people in terms of both interactions and context (or state) information. An example is represented by handheld *mobiscopes* [13] where handheld devices such as cell phones or PDAs gather data from the surrounding environment and report them to servers, which provide services to remote users.

1.5 Motivation

The advantages of using Mobile Elements (ME) in the WSN in presented in Section 1.2.1. The main challenge of using the ME is planning an optimal path for it. However, a major disadvantage of using the ME is the delay in delivering data packets to the sink . In this thesis, we present an energy-efficient path-planning for the ME so that the time required to deliver data packets to the sink is minimized. In many applications of the WSN, it is required that the data packets from the sensor nodes must be deposited to the sink node within a particular interval. This type of WSN is called *Real-time* WSN (*RT-WSN*). *RT-WSNs* are used in monitoring and tracking applications [14]. Because of the reduced delay in data delivery in our method, the ME's can be used in the *RT-WSNs* and, consequently, the lifetime of the network will increase.

In this thesis, we propose a method called *Linear Shortcut* to reduce the path-length of the MDC and thus, the delay in data delivery. We analyze the performance of our method by simulation. The experimental results show that the resulting tour for the MDC ensures the maximum lifetime of the network, and at the same time, reduces delay in data delivery, packet drop rate etc.

1.6 Thesis Outline

The remaining of this thesis is organized as follows.

Chapter 2 reviews the recent related research work in this area to analyze the strengths and drawbacks.

Chapter 3 elaborately describes the problem domain. The analysis presented in this chapter justifies using a *TSP-tour* as a starting point for generating a Shortcut tour.

Chapters 4 and 5 describe our method in details. We provide the logical support for our claim.

In Chapter 6, we provide the detail results of the experiments.

Finally, in Chapter 7, we give the conclusions with some future directions of our research.

Chapter 2

Related Work

2.1 Overview

This chapter provides a thorough review of the work related to the path-planning of the Mobile Elements which collect data in a WSN. Historically, path-planning problem was first addressed in the case of Ad hoc Network and it was termed as *Message Ferry* [15]. Therefore, at first, we present the relevant work in the domain of Ad hoc network. Then, we review the work on path-planning of the mobile elements in a WSN according to the taxonomy presented in Chapter 1. We also shed light on the problems for the mobility property of the sink. The findings in this section justify our approach of using a dedicated Mobile Element as a data collector in a WSN. Finally, critical appraisals of some contemporary work on *Mobile Data Collector* or *MDC* are also made in this chapter. Limitations of these work are discussed in details. Findings in this Chapter form the foundation of our research.

2.2 Approaches Using Message Ferry

Like Mobile Ad hoc Network (MANET) and Delay Tolerant Network (DTN), mobility has been introduced to the field of Wireless Sensor Network ([15]). In MANET, the concept of *Message-Ferrying* is introduced by [16]. In this work, the Mobile Element that is used to transfer information among different nodes is called *Message Ferry*. In this work, the authors assume that all other nodes are static, and the Message Ferry is the only medium of message passing between nodes. Here, a TSP-

tour is computed for a single Message Ferry which visits every node in the network to collect data. In this work, the average delay is formulated and it is shown that the problem of finding an optimal tour for the Message Ferry, which minimizes the average delay is a NP-hard problem. The authors provide a sub-optimal tour solution for the Message Ferry. Their algorithm takes an approximate TSP-tour as input, applies two kinds of heuristics. One of the heuristics is involving swapping of edges and the other heuristic is involving the swapping of nodes to minimize the average delay. Their derived tour is modified further to meet the bandwidth requirement of each node. Experiment results show that as network size and network load increase, so does the average delay.

However, *Message Ferrying* approach is not very suitable for addressing the data gathering problem in a WSN. Because, the objective of the data gathering in a WSN is to deposit collected packets to the sink ([3, 4]). But, a *Message Ferry* carries packets between nodes. The problem formulation and the optimization function presented in this work incorporate the delay for the message passing between every pairs of nodes instead of between the sink and a sensor node. This approach does not capture the real scenario of a WSN. Besides, all the heuristics presented in this work are redundant if an exact *TSP-tour* is given as input.

In [17], the authors use the concept of a Mobile Element in a WSN. The Mobile Element is called *Mobile Ubiquitous LAN Extensions* or MULE. In this work, a three-tier architecture for data collection is proposed- *the top tier* consists of WAN connected access points or sinks, *the middle tier* consists of MULEs and the bottom tier consists of static wireless sensor nodes. In this network model, data MULEs move in a random fashion on a $2D$ grid. It is assumed that from a grid position, the MULE has an equal probability for moving to any of the adjacent grid positions. Based on a simple mobility model, closed forms for different quantity of interests are derived. For example, the average inter-arrival time of data transfer from the MULEs to sensor nodes, the average visiting time of sinks by data MULEs can be derived etc. The authors define the fraction of data packets which are successfully delivered to the sinks, the *success rate*. In their method, as the number of grids increases, so does the requirement for the number of data MULEs and the number of access points to sustain the similar success rate. The authors also show that the buffer requirement for the sensor nodes is inversely proportional to the buffer capacity of the data MULEs and when the number of sensor nodes is large, sensor buffer capacity can be traded off with the number of data mules to sustain the similar success rate.

But, the very important issue which is not addressed in this paper is the data delivery latency. Therefore, this approach is not applicable for time-critical operation of the WSN ([14]). The authors also do not explain how the sensor nodes and the data MULEs can communicate to each other. Besides, in their method, they assume that the sensor nodes would be always 'ON'. Thus, the authors have ignored the major challenging issue of the sensor network- the power limitation, which causes the network connectivity failure. Moreover, the *random walk* model of mobility of the MULE is not helpful for optimizing the MULE's path.

In [18], authors give two approaches for message passing in a disconnected sparse network via Mobile Element. The first approach is called *Node-Initiated Message Ferrying Approach* (*NIMF*). In this approach, the node that needs to send a message to another distant node transmits a request to the Message Ferry by a long-range radio transmitter. The Ferry travels in its own path periodically. When the Ferry comes closed to the node that has sent the request for message passing, the node proactively comes closed to the moving Ferry. The node uses the short-range low-powered radio to transmit data to the Ferry. The Ferry periodically advertise its tour path using a long-range radio signal. Therefore, when it comes closer to any receiving node, that node also proactively comes closer to the ferry and retrieves the data from it. In the second approach named *Node-Initiated Message Ferrying Approach* or *NIMF*, the ferry moves proactively closer to the sending and receiving node by detouring from its original tour-path. However, the sending node still has to send out a request to the ferry using a long-range radio. In both of the approaches, the authors calculate the message loss in a particular interval due to the buffer overflows and the timeout of the packets sent by the source nodes. Experimental results show that, the both of the approaches result in a higher number of messages delivered per unit time and per unit energy compared to [19] and its variants.

However, the requirement of this approach is expensive as sensor nodes with mobility and/or multi-range radios are very costly ([6],[5]) Besides,the data delivery latency is not considered in the performance measurement of this work. Therefore, this method can not be applied to a delay-sensitive applications ([14]). This work also does not discuss how the coordination between the Ferry and the mobile nodes is done.

2.3 Path-planning of Different Types of Mobile Elements in a WSN

We present some work on data gathering using Mobile Elements in a WSN classified according to the taxonomy presented in Chapter 1.

2.3.1 Work on Mobile Relocatable Nodes

In [20], a system with relocatable nodes targeted for topology management has been proposed. Particularly, special *Predefined, Intelligent, Lightweight Topology Management* (*PILOT*) nodes are used to re-establish network connectivity for faulty links. In details, PILOT nodes move to regions where the connection between nodes is unstable or failed, and they act as *bridges*. As a consequence, they actively change the WSN topology in order to improve both communication reliability and energy efficiency. Algorithms for placement of relocatable nodes in the context of improving network connectivity have been investigated in [21], [22], [23] and [24].

In [22], the authors address the problem of a sensor deployment with load-balancing. The movement-assisted sensor deployment deals with the moving sensors from an initial unbalanced state to a balanced state. In this paper, a *Scan-based Movement-Assisted Sensor Deployment* (*SMART*) has been proposed. SMART addresses the problem of communication holes in sensor networks. Although the proposed method deals with the optimal placement of relocatable nodes, it addresses only the network coverage problem and does not discuss the data-gathering issue at all.

In [23], the authors address the problem of disconnected partitions in a WSN which are caused by the failure of one or more nodes. According to this method, existing mobile sensor nodes reposition themselves to repair the partitions. In this work, the solution involves proper placement of the relocatable nodes to address the issue of connectivity, and it also does not address the issue of data delivery to the sink.

Another method proposed in [24] involves repositioning of the relocatable nodes. Given a network containing one or more source nodes which store data, a number of mobile relay nodes and a static sink, the method presented in this work finds the optimal positions to move the mobile relays in order to minimize the total energy consumed by transmitting a data chunk from the sources to the sink and the energy consumed by the mobile relays to reach their new locations. Assuming a single data-flow from the source to the sink, the authors propose an iterative algorithm that repositions the intermediate nodes one at a time to minimize the cost. Later, the authors extend the problem to the multiple flows of data.

However, this method cannot be applied to a WSN if any node is not mobile. The issue of latency has not been addressed also. Therefore, this method cannot be applied to delay-sensitive WSN too. In this method, the formulation for the optimization function is based on the size of the data packet. But, in most of the applications of WSNs, the sensor nodes have to send data packets intermittently, therefore, the size of the data packet does not contribute to energy efficiency much ([25]).

Mobile relay based approaches for opportunistic networks have been surveyed in [26]. However, due to the difference with WSN, many of the assumptions are costly and are not suitable for WSN.

2.3.2 Work on Mobile Sinks (MDC)

Both mobile sinks and mobile relays have been discussed in existing literature to address the issue of data gathering in WSN. Mobile sinks have been considered extensively in the existing literature [27], [28] etc. In these cases, ordinary sensor nodes are static and densely deployed in the sensing area. One or multiple mobile sinks move throughout the WSN to gather data coming from all nodes. We note that the paths between the source nodes and the mobile sinks are multi-hopped, although the actual paths change with time, since the positions of the sinks are not fixed.

In [27], the authors explore the idea of exploiting the mobile sinks for the purpose of increasing the lifetime of a wireless sensor network with energy-constrained nodes. They give a linear programming formulation for the joint problems of determining the movement of the sink and the sojourn time at different points in the network that result in the maximum overall network lifetime (here defined as the time till the first node dies because of energy depletion) rather than minimizing the energy consumption at the nodes.

In [28] and [29], the authors present a generalized formulation for analyzing stability and performance trade-offs inherent to multi-hop routing in mobile sink based sensor data collection systems. The paper parameterizes the extent of multi-hop routing as a hop-bound factor which is used for representing a wide spectrum of design options including single-hop with mobile sink, multi-hop with static sink, and different levels of mobile sink based multi-hop routing in between. A performance model is developed for studying the impacts of multi-hop routing on energy and collection delay performance. Also, a number of thresholds are derived from the model for determining the amount of multi-hop routing that can be used for stable and efficient data collection in the context of constantly moving sinks. The second part of the paper develops a distributed network-assisted framework for mobile sink trajectory planning and navigation without relying on geographical sensor and sink localizations.

A different approach targeted for data collection in urban scenarios has been considered in [30]. In this case, people act as mobile sink by collecting environmental data (such as pollutants concentration and weather conditions) for their own purposes. The reference WSN scenario is represented by a sparse WSN where multiple mobile sinks can be in contact with a single sensor node at the same time.

Mobile relay based approaches have been used in [17] and [31] as Data-MULE. Methods proposed in [17] has been discussed in earlier section. In [31], authors use the same three-tier architecture i.e. sink, data-mule and sensors but present an analytical model to understand the key performance metrics such as data transfer, latency to the destination, and power.

In [16] and [32], message-ferrying approach for data collection has been outlined. In [32], an extensive power-management scheme for the message-ferry has been developed and performance measure has been compared with dynamic source routing (DSR) [33].

2.3.3 Implication of Mobile Sink

In [34], the authors investigate the real-world applicability of theoretical findings concerning sink mobility. They analytically demonstrate that from the small to the mid size square-shaped WSNs implementing virtual grid topology, the (outer) periphery is not necessarily the best performing mobile-sink trajectory. In such networks, the diagonal-cross appears to be at least as effective as the outer peripheral trajectory. Their OPNET-based study of IEEE 802.15.4 / ZigBee WSNs suggests that in these networks, once all of the protocol overhead is accounted for, no actual benefits can be achieved by deploying a mobile sink. According to this paper, it is proposed that the minimization of protocol overhead must be considered first when mobile sink is deployed in ZigBee-based sensor networks. Therefore, using the mobile sink has the implication of redeeming the MAC-protocol overhead for energy-efficiency. However, adapting the MAC-protocol to suit sink mobility is not always viable. This is why, we have used path-planning algorithm for mobile data collectors or relays instead of mobile sinks.

2.4 Controlled vs. Uncontrolled Mobility

An important characterization of Mobile Elements that collect data in the WSN is the ability of controlling mobility. We can classify those into two categories- *Controlled Mobility* and *Uncontrolled Mobility*.

There are two main patterns for Uncontrolled Mobility- *Deterministic* and *Random Mobility*. The *Deterministic Mobility* pattern is characterized by the regularity in the contacts of the mobile element, which enters the communication range of sensor nodes at a specific time periodically. This can happen when the ME is placed on a shuttle for public transportation, as in [35]. On the other hand, the *Random Mobility* pattern is characterized by contacts which take place not regularly, but with a distribution probability. For instance, Poisson arrivals of a Mobile Element have been investigated in [36], while Random Mobility has been considered in [37].

Different from the former Case, Controlled Mobility exploits nodes which can actively change their location, because they can control their trajectory and speed. As a consequence, motion becomes an additional factor which can be effectively exploited for designing data collection protocols specific to Mobile Elements of WSN.

2.5 Work on Mobile Data Collector (MDC)

In [38], an energy-efficient data gathering mechanism for large-scale multi-hop network has been proposed. In this work, the mobile data collector is called as *SenCar*. This paper deals with the path-planning of the SenCar, balancing the traffic load from the sensors to the SenCar to prolong the network lifetime, and clustering the network along the path of the SenCar. The method is applicable to a network of homogeneous sensor network where data generation-rate and the locations are predefined.

In this research, to fix a path, for any two points A and B are taken in such a way that the x-coordinates of the all sensor nodes' locations are bound by the x-coordinates of the points A and B. Two paths of the SenCar are shown in Figure 2.1. Path shown in Figure 2.1(a) is a straight line between points A and B. Sensor nodes those are reachable from the points of this straight line transmit to the SenCar directly, but the other nodes use relay nodes closer to the points of the straight line.

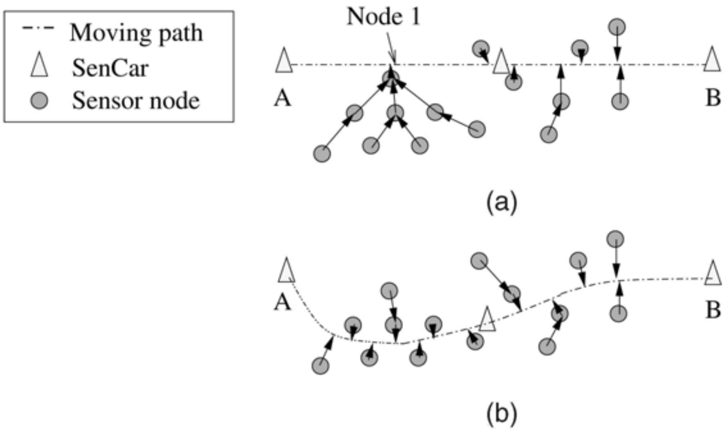


Figure 2.1: (a) The straight line path of SenCar, (b) The curved path of SenCar

As shown in Figure 2.1(a), the depth of one such tree rooted as Node 1 is 2 and this node has to relay many packets of its children. On the other hand, Figure 2.1(b) shows a smooth path of the SenCar that minimizes the tree-depth and also the number of children. Thus, the possible load-balancing is better in this path. In other words, this path minimizes the energy depletion of the sensor nodes due to the packet forwarding, which also maximizes the network-lifetime. The authors call this path as *optimal traffic-relaying path* in [38]. Since, there are infinite number of candidate points for path construction of the SenCar, smoothening the straight line path into an optimal one is intractable. Therefore, the finite number of line segments connected in series between points A and B are used to approximate the optimal path. To select this finite set of line segments, the authors propose a bisector heuristic described as follows.

Initially, a single straight line is chosen. A series of two line segments between Points A and B are derived from it. From a finite set of points which are Δy distance away on the bisector of the current line segment, one point is chosen as the common endpoints of the two line segments. The point is called *Turning Point*. To choose a Turning Point, *MF-trees* rooted at the nodes directly reachable from the SenCar in its current path are generated. The graphs generated in the previous step is transformed into a *Capacitated Flow Network* using quantities like packet generation rate and energy-level of sensor nodes etc. Solving this flow network, the maximum life time of the network for the current connectivity scenario is derived. For each candidate Turning Point, the maximum lifetime of the network (the time after which the first node dies in the network due to energy depletion) is computed. The Turning Point for which this lifetime is of the highest value is selected. For network with disconnected clusters, the authors give a formulation of the Inter-cluster Travelling tour and show that finding the shortest tour is NP-complete. The experimental results show that, as the number of Turning Points in the path of the SenCar increases, the network lifetime decreases. But the gain in lifetime is not that much after the number of Turning Points is eight or more.

However, the paper has some serious limitations described as follows. Although the authors have proved that the problem of finding a inter-cluster tour is NP-hard, no approximation algorithm is given for this computation. Therefore, the optimal solution by this method for a network with disconnected clusters is also NP-hard. The SenCar travels on the planned path periodically. If the path of the SenCar is long, tour time will be high. In sensor nodes, packets will be lost due to buffer-overflow, or collected data will become useless due to the high latency. Moreover, the authors do not provide any solution to handle data latency which leaves the method unsuitable for delay-sensitive wireless network. The authors consider the starting and ending points of the tour by the SenCar to be distinct for each cluster in their formulation of the optimization problem. These endpoints are chosen to be the leftmost and rightmost nodes of a cluster. Here, selection of these points do not contribute to the energy-efficiency and to the path-length minimization. Nothing is mentioned about choosing the value of Δy. If it is too small, the number of candidate Turning Points will be high and so will be the complexity of computation. On the other hand, a large value of it may cause the algorithm to overstep suitable Turning Points. The bi-sector heuristic as proposed in this paper may generate line segments from which not a single sensor node may be reachable. Then, the computation of this line segment will be futile.

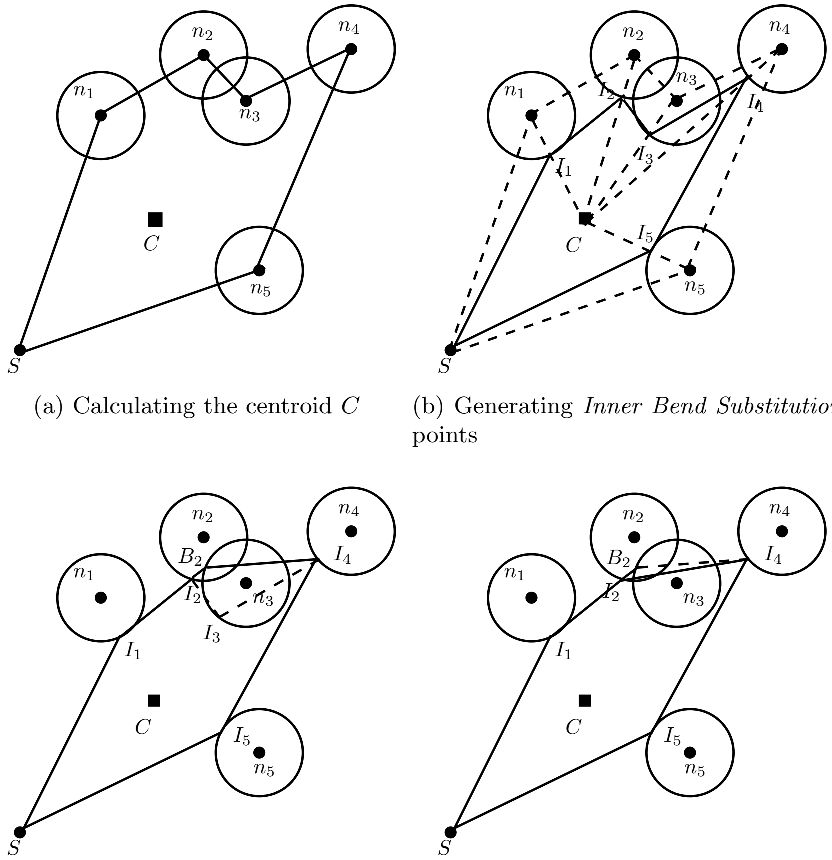
In [39] and [40], the issue of latency was considered while planning path for the Mobile Data Collector. The authors term the Mobile Data Collector as *Data Mule*. In these works,the path selection problem of Data Mule is formalized in to a framework termed as *Data Mule Scheduling Problem* or DMS-problem. The authors observe that Data Mules can be used as an alternative to multihop forwarding in sensor networks and that the use of Data Mules introduces the trade-off between energy consumption and Data Delivery Latency.

In this approach, for the given connectivity graph of the WSN, a near optimal TSP tour is generated using an approximate TSP-solver[41]. Using dynamic programming which runs in $O(n^3)$ time, a tour called *Label Covering* tour or *LC-tour* is generated from the TSP tour. The authors consider three cost metrics for optimization: the number of edges, the total length of the path, and the total uncovered distance i.e. the total length of interval in an edge that is not within the range of any sensor nodes. It is shown that finding the minimum-cost LC tour is NP-hard by showing that the TSP-tour is a special case of LC-tour. Experimental results of this work show that the tour length of the Data Mule and the latency decrease with the increase in the transmission range of the sensor nodes.

However, the methods in [39] and [40] suffer from some serious limitations described as follows. Instead of visiting the exact position of the sensor node, the Data Mule can communicate with the sensor node from any position within its transmission range. The value of this range is typically from 5 to 50 meters [6]. In the LC tour, these values of transmission ranges of the visited nodes add up to the tour-length. The higher the length, the greater the delay of packet delivery. The authors claim their method as energy-efficient. But, they do not mention any thing about using any energy-efficient measures in the mode of communication between the Data Mule and senor nodes.

In [2], authors propose an approximation algorithm which is based on the TSP route constructed from the locations of the deployed sensor nodes. By using some set of heuristics and a shortcut finding step, the authors try to optimize the obtained TSP route within $O(n)$ computation time. This algorithm is applicable only to a sensor network with the static sensor nodes.

The method proposed in [2] is illustrated in Figure 2.2. As shown in Figure 2.2a, there are five sensor nodes n_1 to n_5 and a sink S. Transmission regions of the nodes are shown by circles centered at those nodes. The given *TSP-tour* is also shown in this figure. First, the centroid C of the polygon generated by the *TSP-tour* is calculated. In the next step, a straight line is drawn from C to each node position. This line intersects the circle centered at node n_i at point I_i. This point is called *Inner-lane Substitution Point* or ISP. In Figure 2.2b, all the ISP's are shown. Then, a shorter tour is derived by connecting the ISP's in the order of visiting the nodes in the given *TSP-tour.* In the third step, concave bends are identified. As shown in Figure 2.2c, two edges $< I_2, I_3 >$ and $< I_3, I_4 >$



(a) Calculating the centroid C

(b) Generating *Inner Bend Substitution* points

(c) Substituting Concave Bends

(d) Making shortcut of tour-edges

Figure 2.2: The method of making a shortcut of the TSP-tour [2]

form a concave bend with respect to point C. Using a heuristic, the point I_3 is substituted by point B_3 which is called *Bend Substitution Point* or BSP. Finally, using a dynamic program, shortcuts of the tour edges are made. As shown in Figure 2.2d, the tour edge $< I_2, I_4 >$ is within the range of the circle centered at n_3. Therefore, this edge is a shortcut of the successive edges $< I_2, B_3 >$ and $< B_3, I_4 >$. The final tour is derived by successively connecting the points S, I_1, I_2, I_4, I_5 and S. If the running time of the TSP-approximation algorithm is $O(n^2)$, the running time of the whole path-finding algorithm is $O(n^2)$.

However, we find some limitations of this method: The simple polygon bounded by the edges of the TSP-tour is termed as *TSP-polygon*. In this work, in the proof of a theorem regarding the heuristic for generating the *Inner-lane Substitution Point* (step of Figure 2.2b), it is assumed that the centroid C always lie within the TSP-polygon. This makes method inapplicable for the cases where

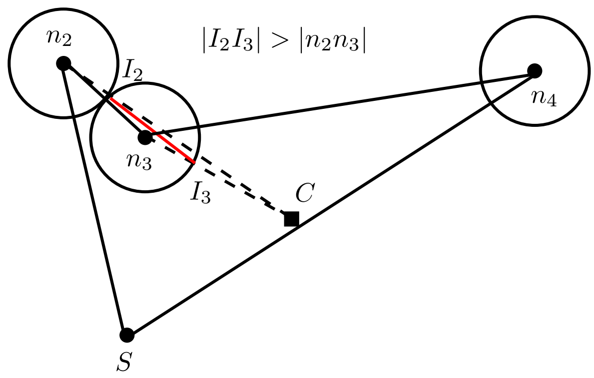


Figure 2.3: The scenario where inner bend (I_2I_3) is greater than the outer tour-edge (n_2n_3)

centroid C lie outside the *TSP-polygon*. Therefore, the method of this work cannot be generalized. Even in case where the centroid C lies within the *TSP-polygon*, the steps of the theorem which proves that- the *inner lanes* are always shorter than the outer edges, is flawed. A counter example is shown in Figure 2.3. In this figure, all the objects are drawn to the scale. Straight lines drawn from C to the point n_2 and n_3 intersect the corresponding circles at point I_2 and I_3. The inner-lane of the tour-edge $< n_2, n_3 >$ is $< I_2, I_3 >$. Using the coordinates of those points, we calculate the lengths of both the line segments and find that inner-lane is longer than the tour-edge. Therefore, it cannot be guaranteed that the tour derived by this method is always shorter than the given tour. The step of finding the *Bend Substitution Points* (step of Figure 2.2c) becomes NP-hard when more than one such bends exist successively. The authors give an approximate solution which depends only on the two endpoints of the surrounding convex bends and which ignores the endpoints of the successive concave bends. An optimal path covering those concave bends would be affected by the concave property of each of those successive bends. However, the authors do not address the effect of this local decision on the global outcome. Making shortcuts at the last step (step of Figure 2.2d) eliminates the gains achieved by the application of heuristics in the previous steps. The Bends or edges with steep concavity may be introduced by the newly added shortcut edges.

The above analysis shows that, the basis of the heuristics applied in this work is poor and, most importantly, the derived tour cannot be guaranteed to be shorter than the input tour.

In [42], the authors address the problem of planning paths of multiple robots to collect data from all sensors in the least amount of time. The solution is applicable to a wireless sensor network with static nodes, static sink and one or more robotic data collectors.

The authors first give an optimal solution for scheduling k data collector robots for a network of n sensors in 1-Dimension (*1-D*). An equation for dynamic programming is formulated to distribute non-overlapping 1-D paths among the k-robots. However, the solution given for the case of 1-D is trivial and cannot be generalized for the cases with higher dimensions of mobility.

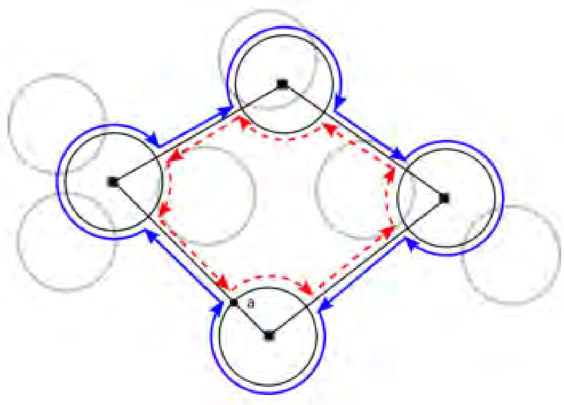


Figure 2.4: Approximate TSPN consisting of clockwise (solid line) and anti-clockwise(dotted line) traversals

In the case of 2-D plane, an approximation solution is given combining earlier works of [43] and [44]. The research work of [43] is used to approximately construct a solution for the TSP with neighborhood (TSPN) problem. The circles/disks representing the radio range of the sensor nodes represent the objects of the TSPN problem. First, a *maximal independent set* I of non-intersecting disks from the given set of disks is computed such that each of the given disks has an overlap with at least one disk included in set I. This is illustrated in Figure 2.4. In this case set I includes four disks shown in bold boundaries. Then, a TSP-tour on the centers of all the disks included in set I is computed. Using this tour, a solution instance for the $TSPN$ problem is generated. This is illustrated in Figure 2.4. The robot or data collector first travels along the *outer* boundaries of the disks of the set I and the edges of the TSP-tour in the clockwise direction (shown in Figure 2.4 by directed solid outer paths) and then along the *inner* boundaries of the disks of the set I and the edges of the TSP-tour in counter-clockwise direction (shown in Figure 2.4 by directed dashed inner paths). Other than the edges of the *TSP-tour*, there is no overlap between the clockwise and counter-clockwise paths. Travels in two types of orientations ensure that all the disks that are neighbors of at least one disk of set I are covered by the robot.

In the next step, the data-collection points for the robot are determined for sensor nodes both within and not within set I. These points are basically intersection of boundaries and edges. In the final step, the whole $TSPN$-tour is split into k-subtours for simultaneous data collection by k robots. This division is done according to the works of [44] which originally involved splitting the TSP-tour into k-subtours.

However, the method also has the following limitations: When there is only one data mule or robot, the method is not anything different than that proposed in [43]. Considering an approximate ratio of 11.5, the resulting tour may be as worse as 11.5 times the length of the TSP-tour on the centers of the disks. The method fails to utilize the available location information of the sensor nodes to the fullest since it allows traversals of the boundaries. There may be many disks in a sparse network where the boundary of a disk does not overlap with any other disk and as a result, the traversal of the boundary is futile and only adds up to the tour-length. Considering the typical radio range from 5 to 50 meters for motes [6, 5], the redundancy of paths is significant for a node (the circumference of the disk or about 34 to 340 meters). The selection of anchor points for the robot is not done carefully to minimize the tour-length. For a dense network with a good coverage, a single anchor point may cover more than one disks. Therefore, visiting the point of intersection of each neighboring disk is redundant. Since, this method does not keep track of nodes already visited or make any shortcut of the $TSPN$-tour, the redundant paths persist in the final solution. No experimental results are provided other than coverage time for this method. The authors do not provide how much path-gain and energy efficiency can be obtained from their method.

2.6 Summary

All these research work conclude with the challenge to find a *good* path for a mobile data collector in a power constraint static sensor network where nodes may be connected or may not be connected. Different research work set objectives for different types of *goodness* of the path. Some work address the issue of meeting bandwidth requirement, some work address maximizing network lifetime, yet some address increasing through-put. However, the only few of these works address the issue of network lifetime and data delivery latency at the same time. Work presented in [25] conclude that most of the energy-saving measures increase the latency. Because of this trade-off between the latency and the network lifetime, finding a data collection method using Mobile Elements which increases network lifetime and, at the same time, decreases data delivery latency is indeed challenging.

Chapter 3

System Model

3.1 Overview

In this chapter, we present some definitions to describe the problem domain. We present the required parameters of the system architecture and data collecting procedures in a WSN. We show that shortening the tour of the MDC reduces the data delivery latency to the sink. Finally, we formulate the problem statement and set our research objectives.

3.2 Preliminaries

3.2.1 Modeling of the Sensor Network and the Tour

Definition 3.1: A *walk* in a given graph $G = (V, E)$ is a sequence $v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n$ where $v_i \in V$ are vertices, $e_i \in E$ are edges and for all i, e_i connects the vertices v_{i-1} and v_i. A *tour* T in a given graph $G = (V, E)$ is a walk with no repeated edges. A *closed tour* is a tour where the starting and ending vertices are the same i.e. $v_0 = v_n$. A *cycle* is a walk with no repeated vertices except for the starting and ending vertices i.e. $v_0 = v_n$.

We use the terms *cycle* and *tour* interchangeably for the travels related to the MDC.

Definition 3.2: A sensor network of n nodes is a complete weighted undirected graph K_n where the weight of the edge e_{ij} connecting two nodes n_i and n_j in graph K_n is the Euclidian distance between the i-th and j-th sensor nodes.

Definition 3.3: A *Hamiltonian Cycle* in a given graph $G = (V, E)$ is a cycle that includes all the vertices of the set V exactly once.

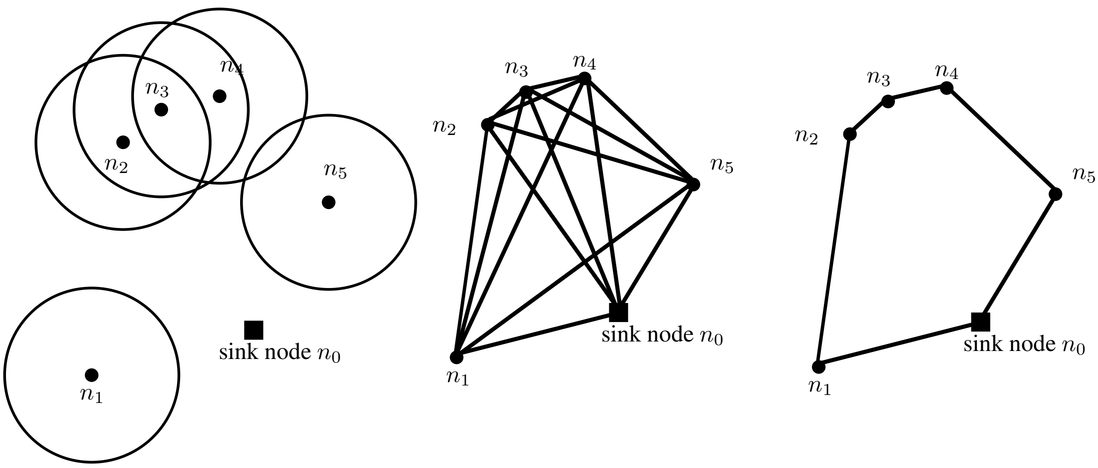
Definition 3.4: The *weight* of the cycle C in a weighted graph G is the sum of the weights of all the edges forming C.

Definition 3.5: A TSP*-tour* or TSP*-cycle* in a given weighted graph G is a Hamiltonian cycle with the minimum weight. If the given graph is undirected, the TSP-tour is called *symmetric* TSP*-tour* or $STSP$*-tour*.

In a sensor network, there are two kinds of nodes, *sensor nodes* and *sink nodes. Sensor nodes* perform sensing, buffering the sensed data and forwarding data packets to the sinks. The *sink node* accumulates the data packets from the sensor nodes. We assume that there is only one *sink node* in our sensor network and each of the sensor sensor nodes sends data packets the sink.

Definition 3.6: Given K_n, a TSP*-tour by the MDC* is a TSP-tour where the tour starts and ends to the sink node.

In Figure 3.1(a), the circles denote the transmission range r of the sensor nodes. The corresponding



(a) A sample of sensor network (b) K_6 (c) The *TSP-tour* by the MDC

Figure 3.1: *TSP-tour* by MDC in sensor network

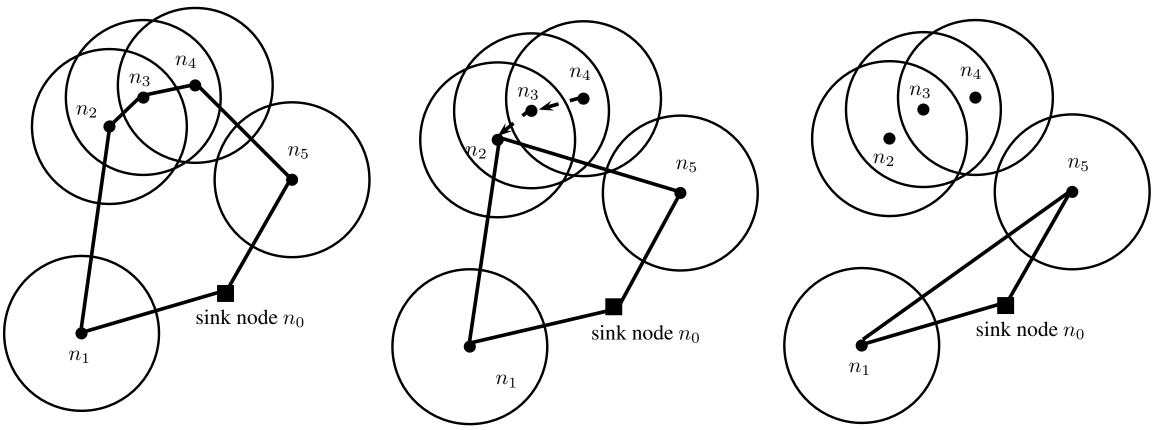
graph representation K_6 is shown in Figure 3.1(b). A possible TSP tour by the MDC is shown in Figure 3.1(c). Here, the MDC starts out from sink node n_0 and after completing the TSP-tour, returns to the same node.

In any arbitrary type of tour, the visiting MDC may never be within the transmission range of all the nodes. Therefore, some nodes must have to forward packets of other sensor nodes. These nodes are known as *Forwarding* or *Relay nodes*. Packets must be forwarded by more than one forwarding nodes i.e via multiple hops. We call this *Multi-hop Forwarding* (MF). If a node is not directly reachable from the MDC, it has to choose one of its neighbors as a forwarding node. This Multi-hop Forwarding path must all the way end up at the node n_{MDC} representing the visiting MDC . In this process, a tree rooted at node n_{MDC} is formed where all the non-leaf nodes are forwarding nodes. This tree is called *Multi-hop Forwarding Tree* or *MF-tree*.

Observation: *The maximum hop count of the MF-tree in TSP-tour is* 1.

Definition 3.7: A tour T by the MDC is *complete* if each of the sensor nodes can send data packets to either the sink node or the visiting MDC directly or via the MF-tree. Otherwise, the tour is *incomplete*.

Three tours are shown in Figure 3.2. The TSP-tour shown in Figure 3.2(a) is a complete tour. The



(a) TSP-tour (b) A tour with a 2-hop MF-tree (c) An incomplete tour

Figure 3.2: Examples of complete and incomplete tour by MDC

tour shown in Figure 3.2(b) is also complete because Nodes n_3 and n_4 can send packets to the visiting MDC via Node n_2. But, the tour shown in Figure 3.2(c) is incomplete as none of the nodes n_1, n_2 and n_3 can send data packets to either the MDC or the sink.

Observation: *A TSP-tour by the MDC is complete.* (By Definition 3.5)

3.2.2 Energy Modeling of the Sensor Network

We adopt the energy model presented in [25]. The energy to send one packet from Node n_i to Node n_j is:

$$E_{i,j} = k_0 + [(h(n_i, n_j)]^w \tag{3.1}$$

where w is the path-loss exponent, the function $h(n_i, n_j)$ returns the hop-count of the path between Nodes n_i and n_j. We call k_0 the energy constant, which includes all energy consumption, such as receiving energy, idle-state energy, processing circuitry energy etc. which are unrelated to the distance or path of transmission. A node n_j has to transmit its own packets in addition to the packets of all the descendant nodes of the subtree T_{n_j} rooted at n_j in the MF-tree.

Let us assume that the data generation rate of all the sensor nodes are the same and the sensor nodes are homogeneous. Therefore, the leaf-nodes in the MF-tree expend the least amount of energy as they don't forward other nodes' packets. The nodes nearest to the MDC i.e. at Level 1 consume the highest amount of energy. According to this model, the total energy consumption by a node n_j for a period of MDC's travel is:

$$\begin{aligned} E_{n_j} &= \sum_{\forall \text{ node } m \ \in T_{n_j}} (k_0 + 1^w) \\ &= |T_{n_j}|(1 + k_0) \end{aligned} \tag{3.2}$$

Here, $|T_{n_j}|$ is the total number of nodes in the subtree T_{n_j} rooted at the node n_j. From Equation 3.2 it is clear that the total energy consumption by a sensor node is directly proportional to the number of packets it relays in addition to its own packets. Deducing similar equations, we can show that, in case of heterogeneous network and in scenarios where traffic generation rates are different, the energy consumption is still proportional to the frequency of packet forwarding actions. In other words, the life-time of the sensor node is proportional to the number of forwarded packets in addition to is own packets.

Definition 3.8: *m-lifetime* is defined as the period after which exactly m nodes of a sensor network die due to energy depletion.

Lemma 3.1: *TSP-tour* has the maximum m-lifetime of all the complete tours by the MDC in a sensor network.

Proof: Let us compare 1-life-time of the TSP-tour T_{TSP} with that of an arbitrary complete tour T_i. There may be two cases described as follows:

Case (a): The maximum hop-count of all the MF-trees of tour $T_i > 1$.

In this case, let n_j be the nearest node to the root i.e. a node at depth 1 in the MF-tree with the maximum hop count in tour T_i. Under the similar traffic scenario and the similar network topology, this node dies faster than the first node n_k to die in TSP-tour; because node n_k does not forward other nodes' packets whereas node n_j forwards all the packets of the nodes of the MF-tree sub-rooted at node n_j. Thus, 1-lifetime of tour T_i which is the lifetime of node n_j is shorter than that of TSP-tour T_{TSP}.

Case(b): The Hop-count of all MF-trees of tour T_i is 1.

In this case, the TSP-tour can still beat the 1-lifetime of tour T_i by decreasing the transmission radius (TXR) r by a small amount ϵ. TSP-tour is invariant to the value of TXR. When all other things are equal, the energy consumption by a node is directly proportional to the value of TXR according to the our adopted energy model. Therefore, the 1-lifetime of TSP-tour with $TXR = (r-\epsilon)$ is higher than that of tour T_i with $TXR = r$. Using the similar approach, we can show that $2, 3, \ldots, m$-lifetime of TSP-tour are higher than $2, 3, \ldots, m$-lifetime respectively of any arbitrary complete tour T_i. ■

Observation on *TSP-tour* by *MDC*: Now, we find the following observations in a *TSP-tour* of the MDC:

A. The tour is complete,

B. The tour ensures that there is no forwarding or relay action in the network,

C. The tour is invariant to the transmission range TXR

Due to the Observations A, B and C, the tour ensures the maximum m-lifetime of the network.

3.3 Problem Statement

Observation: *The problem of finding a complete tour for the MDC for a sensor network is intractable.*

The above observation follows from the innumerable possible *Anchor* or *Halting Points* for the

MDC in the whole network. Finding the order of node visit is an *NP-hard* problem. However, *TSP-Tour* is complete and has the maximum m-lifetime. These facts motivate us to use the *TSP-Tour* for offline or static path-planning of the MDC. Though finding a *TSP-tour* is *NP-complete*, there exist good heuristics approximation and software tools to find *TSP-tour* for thousands of nodes in a reasonable amount of time. ■

We use the solution of the *TSP-Tour* as the basis of our tour because it ensures the maximum lifetime of the WSN. There is a penalty to pay for the maximum m-lifetime of the *TSP-tour*. In the *TSP-tour*, the delay of delivering packets to the sink is at most the time the MDC takes to complete the current tour (tour-time). The speed of the MDC is lower than the speed at which packets are forwarded to the neighbors in the wireless medium. Therefore, the tour-time of the MDC is higher than the time it takes to send packets to the sink via multi-hop forwarding. But we know that *TSP-Tour* does not allow any forwarding of packets. Therefore, *Data Delivery Latency* is comparatively higher in *TSP-Tour* than any other tours which allow multi-hop forwarding.

Definition 3.9: *Data Delivery Latency* (DDL) of a data-packet is the time-difference between packet generation and delivery.

Let a packet i be generated at t_g time after the MDC sets out from the sink node position. The MDC completes the current tour in t_T time according to some tour plan T. The *Packet Delivery Latency* t_l for this particular packet i is given by:

$$t_l(i) = t_T - t_g(i) \tag{3.3}$$

If n packets in total are collected in this tour T, the average *Packet Delivery Latency* denoted by t_{avg} is computed as follows:

$$\begin{aligned} t_{avg} &= \frac{\sum_{i=1}^{n}[t_T - t_g(i)]}{n} \\ &= t_T - \frac{\sum_{i=1}^{n} t_g(i)}{n} \end{aligned} \tag{3.4}$$

The quantity $\frac{\sum_{i=1}^{n} t_g(i)}{n}$ in Equation 3.4 known as the *average packet generation time* is not controllable as it depends on the sampling rate of sensor nodes and event frequencies. However, we can improve both the per packet delivery latency and the average packet delivery latency by decreasing the tour-time t_T as evident from both Equation 3.3 and 3.4.

The tour-time of the TSP-tour i.e. t_{TSP} has two components: the fraction of tour-time t_h that the MDC halts and collects data from nearby nodes and the fraction of tour-time t_m that the MDC travels between the node positions. Therefore, we can calculate the *TSP-tour time* t_{TSP} as follows:

$$t_{TSP} = t_h + t_m \tag{3.5}$$

When the number of nodes is very high and/or the network is sparse, $t_h << t_m$, and thus, t_m dominates tour-time t_{TSP}. This assumption is logical for practical scenario where the speed of a commercially available robotic car used as MDC is usually at most 5 ms^{-1} where as packet transfer from a sensor node to the MDC happens in the order of miliseconds [42, 8]. Thus, decreasing motion time t_m contributes to decreasing the latency. If the speed of the MDC is v_{MDC}, and if we assume that it accelerates to this speed instantly and also stops instantly, then,

$$t_m = \frac{|t_{TSP}|}{v_{MDC}} \tag{3.6}$$

where $|t_{TSP}|$ is the path-length of the TSP-tour. We can always come up with a speed v_{MDC} that can approximate the case where acceleration and halting both take finite time. Usually, given a particular MDC, v_{MDC} is fixed [1]. Therefore, the only way to decrease the tour-time is decreasing the length of the tour i.e. $|t_{TSP}|$ (see Equation 3.6). However, by decreasing the tour length, we have the risk of making the resulting tour incomplete. Therefore, we address the issue carefully so that, the resulting tour is complete and shorter than the TSP-tour.

Now, we formulate the problem of balancing the lifetime of the network and the data delivery latency described as follows:

Problem Statement: *Given a TSP-tour by the MDC, we find a tour T_d that is complete and shorter than the TSP-tour.*

3.4 Research Objective

The given TSP-tour can be modified in many ways to derive a tour which is *shorter* and *complete*. Now, we provide the reasons discussed as follows:

Reason 1: The number of nodes visited in the *TSP-tour* can be decreased by making *shortcut* of the *TSP-tour*.

Reason 2: The length of the edges in the resulting tour can be decreased by taking into consideration the value of the transmission radius TXR. ■

In our thesis, we present two algorithms for the two steps stated above. For both of the purposes, we present the notion of finding *Linear Shortcuts* on any given tour to derive a complete tour. For Reason 1, we present the notion of *Label Covering* tour [39, 40]. For Reason 2, we present the notion of *Tight Label-covering* tour. Finally, we show that both of the tours are equally energy-efficient in terms of m-lifetime for a constant TXR. But, *Tight Label Covering* tour has the least data delivery latency among the three tours. Though, the *TSP-tour* computation is *NP-Complete* [45], our algorithms can be computed in polynomial time.

Chapter 4

An Efficient Path Planning

4.1 Overview

In this chapter, we present our method in details. At first, we present a simple strategy of finding Shortcuts in an arbitrary tour. We call this *Linear Shortcut* method. Then, using Linear Shortcut, we derive a tour shorter than the *TSP-tour*. We give a framework which can shorten any tour iteratively. We illustrate the steps of this iterative improvement. Finally, we analyze the time complexity of our method.

4.2 Improving Latency by Finding a Shortcut

4.2.1 Linear Shortcut of a Tour

Definition 4.1: Given a cycle or tour T in an undirected graph, a *Linear Shortcut* T_s is a tour that is derived by selecting some finite number of points on the path of the tour T, and joining them by straight lines successively in the order of the edges of T.

In Figure 4.1, an example of a *Linear Shortcut Tour* is shown. Here, the tour constitutes five edges which successively connect five nodes each denoted by n_i where $i = 1, \ldots, 5$. At first, to form a linear shortcut tour, we choose zero or more points from each edge. We call each of these points *Anchor Points*. Here, five Anchor Points are chosen, each from one of the five edges. These Anchor Points are denoted by p_i where $i = 1, \ldots, 5$. We then join these p_i's by straight lines to derive a Linear Shortcut

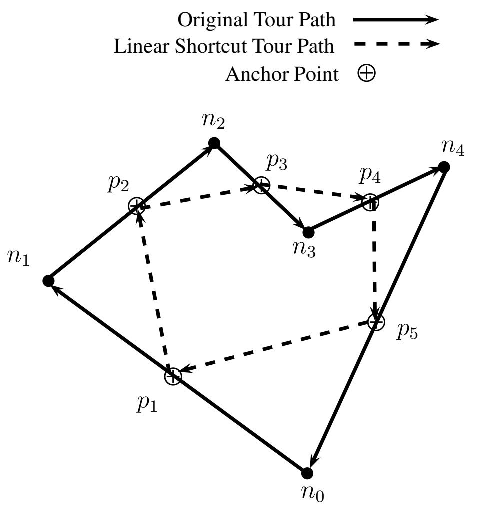


Figure 4.1: An example of a *Linear Shortcut tour*

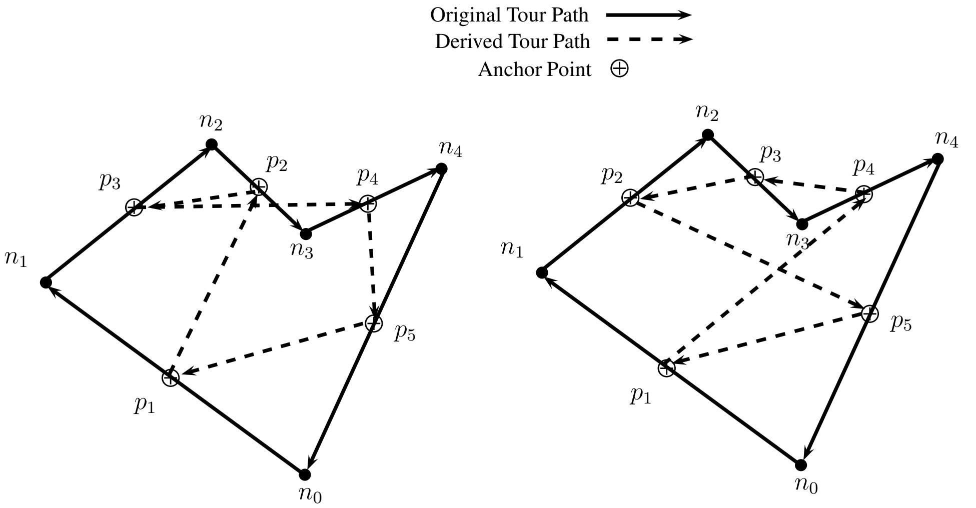
tour. However, the label p_i reflects their order of choosing as follows:

1. If $p_i < p_j$ and both the points are on the same edge connecting nodes n_k and n_l such that n_k is visited before n_l, then p_i is closer to n_k than p_j is to n_k, or in other words, p_i is farther to n_l than p_j is to n_l.
2. If $p_i < p_j$ and, p_i and p_j lie on edges e_k and e_l respectively, then the edge e_k is visited before edge e_l in the tour.

Thus, the order of choosing any number of Anchor Points from any edges of the given tour must be according to the above rule, and the Anchor Points must also be connected successively by the edges in the same order to form a Linear Shortcut Tour.

In Figure 4.2(a), the labels of Anchors p_2 and p_3 are swapped and thus, the resulting tour is not a Linear Shortcut Tour. In Figure 4.2(b), the order of connecting the Anchor Points by edges successively is $< p_1, p_4, p_3, p_2, p_5, p_1 >$. Therefore, this is not a Linear Shortcut Tour either. There is no condition attached to the total number of Anchor Points or the number of Anchor Points from each edge.

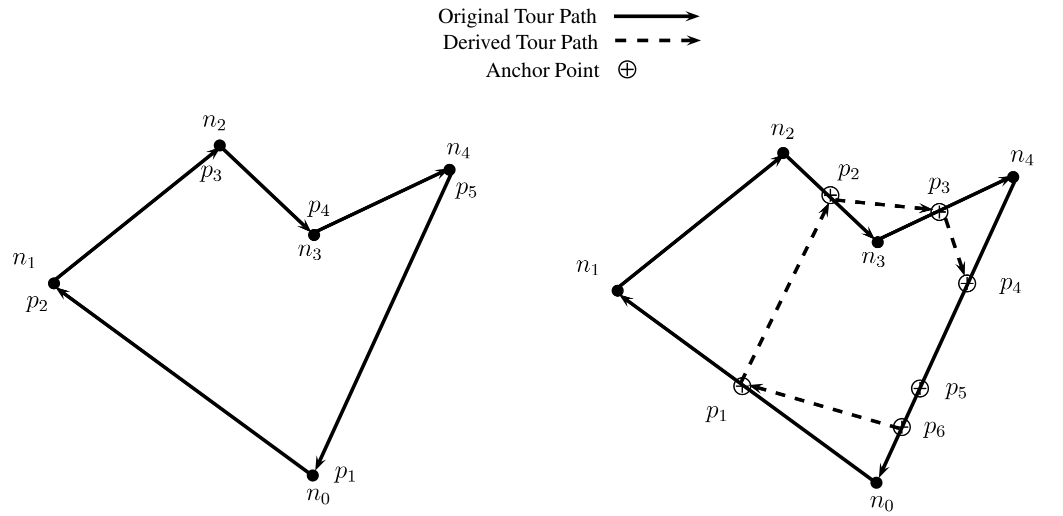
For example in Figure 4.3(a), the Anchor Points selected are the same as the node positions, hence the given and the derived tour are the same. But in Figure 4.3(b), the edge $< n_1, n_2 >$ does not contain any Anchor Point, whereas edge $< n_4, n_0 >$ contains three Anchor Points. The total number



(a) Anchor point p_2 and p_3 are chosen in a wrong way

(b) Anchors not connected by edges according to the order of p_i's

Figure 4.2: Examples of derived tours which are not Linear Shortcut tours



(a) Derived tour coincides with the given tour

(b) Edges contain different number of Anchor Points

Figure 4.3: Examples of different Linear Shortcut tours

of Anchor Points can also be only one, a point that lies on the path of the given tour, therefore, the derived tour is effectively of zero length.

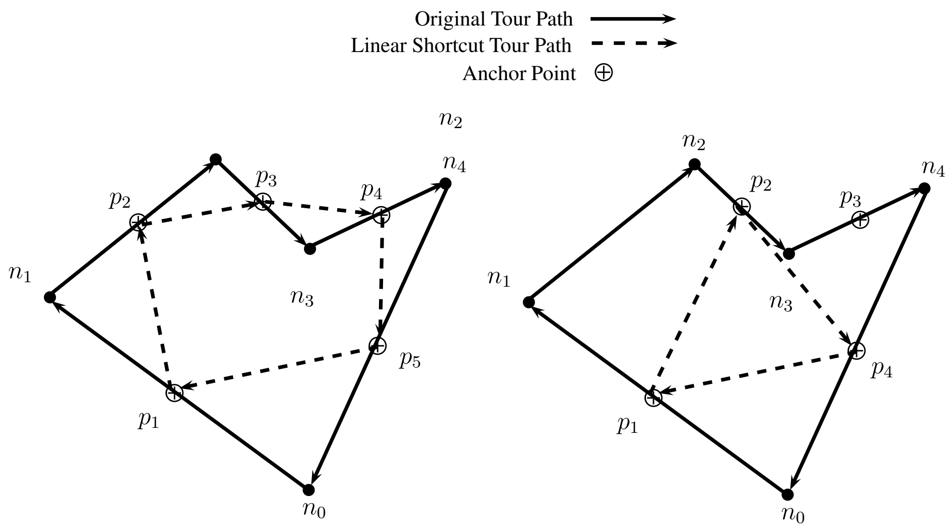
Algorithm 4.1 generates a Linear Shortcut Tour according to some *Anchor Point Selection Strategy* S. S controls the points and their number on a tour edge. For example, if the strategy is to choose the middle-point of each edge then the derived Linear Shortcut tour is as shown in Figure 4.4(a). If the strategy is to choose the midpoint of each odd numbered edge of the given tour, then the derived

Algorithm 4.1 Generating a *Linear Shortcut Tour*

Input: A tour T with K edges in undirected graph $G = (V, E)$, a strategy S for choosing Anchor Points

for all i-th edge in tour T where $i = 1, \ldots, K$ **do**
    choose a_i Anchor Points according to strategy S and label them accordingly
end for
for all $i = 1, \ldots, \left(\sum_{i=1}^{K} a_i\right) - 1$ **do**
    connect Anchor Points p_i and p_{i+1} by an edge and add it to tour T_s
end for
connect the first and last Anchor Points of tour T_s to make it a cycle

Output: T_s is a linear shortcut tour of tour T



(a) Midpoint of each edge chosen as anchor point (b) Midpoint of each odd edge chosen as anchor point

Figure 4.4: Example of different strategies for finding Linear Shortcut tour

Linear Shortcut tour is as shown in Figure 4.4(b).

Lemma 4.1 The length of a Linear Shortcut Tour is at most that of the given tour.

Proof: This can be proved by *Triangle Inequality* [46]. As shown in Figure 4.5, the Anchor Points p_m and p_{m+1} lie on Edges $< n_i, n_{i+1} >$ and $< n_{i+1}, n_{i+2} >$ respectively. These are also the last and the first Anchor Point of their respective edges. Using *Triangle Inequality* we get,

$$|n_i p_m| + |p_m p_{m+1}| + |p_{m+1} n_{i+2}| \leq |n_i n_{i+1}| + |n_{i+1} n_{i+2}|$$

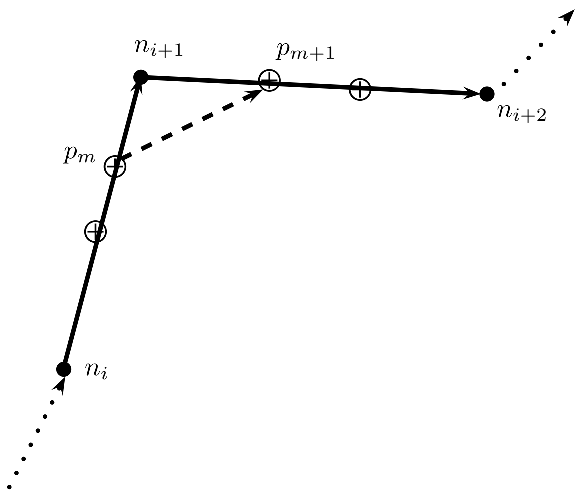


Figure 4.5: Corner-cutting in Linear Shortcut Tour

Thus, the edge connecting Anchor Points p_m and p_{m+1} effectively corner-cuts node n_{i+1}. This can be proved for all the Corner-cutting edges in the derived tour. If there is no Corner-cutting edge then the Anchor Points coincide with the nodes and the resulting tour is of the equal length of the given tour. Thus, if T_s is the Linear Shortcut tour of T then $|T_s| \leq |T|$, where the length of a tour T is given by $|T|$. ■

4.2.2 Linear Shortcut Tour in the Context of MDC

At the Anchor Points, the MDC stops and collects data from adjacent nodes . The *first* and the *last* Anchor Points of an edge of a given tour are the two points where the MDC changes direction. For example, MDC stops at all Anchor Points but changes direction or rotates itself for alignment with a new path segment at all Anchor Points except p_5 shown in Figure 4.3(b).

Anchor Point Choosing Strategy: The strategy S is such that the derived Shortcut tour is complete i.e. MDC can collect data from all sensor nodes. For example, if TXR of node n_1 is so small that the MDC fails to communicate while traveling on the Shortcut tour shown in Figure 4.3(b), then, the derived tour becomes incomplete. If TXR is so large that each node is reachable from point p_1, the MDC can rather stop at p_1 and collect each node's data. The resulting Linear Shortcut tour has only one Anchor Point and is thus of length zero.

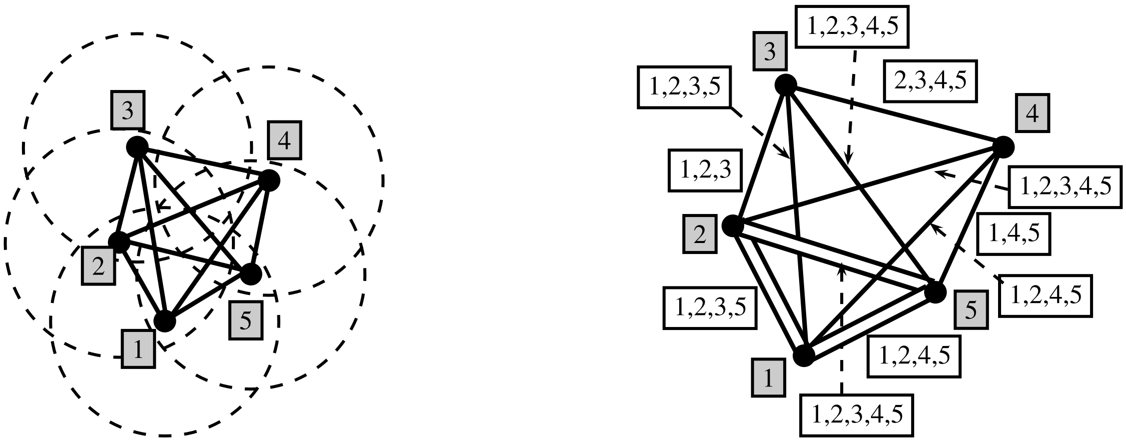
In the following sections, we present our strategy for making a Linear Shortcut. We term the resulting tour as *Tight Label Covering Tour* or *TLC-tour*.

4.3 Label Covering Tour [Sugihara et. al. 2008]

Let us consider a complete graph $G_l = (V_l, E_l)$ which have the set of vertices $V_l = V$ of a graph $G = (V, E)$. There is an edge between any two nodes of the graph i.e $E_l = \{e_{n_i,n_j} | i \neq j \text{ and } n_i, n_j \in V_l\}$. The cost function associated with each edge is the Euclidian distance between the nodes connected by that edge; that is $f(e_{n_i}, n_j) = distance(n_i, n_j)\ \forall n_i, n_j \in V_l$. Each node is given a unique label from 1 to $|V_l|$. The set of all labels is $L = \{1, 2, 3, \ldots, |V_l|\}$. Associated with each edge e_{n_i}, n_j is a set of labels $L(e_{n_i}, n_j) \subseteq L$ which represents the set of nodes whose communication ranges intersect with this edge. We determine the set of labels as follows: $k \in L(e_{n_i}, n_j)$ if Node n_k's communication range intersects the edge e_{n_i}, n_j. If $distance(n_k, e_{n_i}, n_j) \leq r$, where r is the transmission range of the communication. For any edge e_{n_i}, n_j, we have $i, j \in L(e_{n_i}, n_j)$. In Figure 4.6, a complete graph is generated for the network. Each edge in this graph is marked with the associated labels. For example, edge between Nodes 1 and 2 passes through the transmission ranges of Nodes $1, 2, 3$ and 5. Similarly, the edge between Nodes 2 and 5 passes through the transmission ranges of all the nodes of this cluster. Hence, its label is $\{1, 2, 3, 4, 5\}$. Now, instead of the intractable task of finding the shortest tour of the MDC using arbitrary number of Anchor Points from a domain of infinite points, we have to find the shortest tour on this complete Labeled graph so that the union of the labels of the edges in this tour forms the set of all labels L. Let us formally define this tour as follows.

Definition 4.2: A tour t_{LC} defined on a graph $G = (V, E)$ is *Label Covering Tour* when it satisfies at least one of the following conditions for $k = 1, 2, \ldots, |V|$:

1. $\exists e_{n_i}, n_j \in t(E)$, $k \in L(e_{n_i}, n_j)$, where $t(E)$ is the set of edges of tour t, or
2. Euclidian distance between Nodes n_s and n_k i.e. $distance(n_s, n_k) \leq r$ where n_s is the starting node of the tour t



(a) Complete Graph derived from the Connectivity Graph (b) Label Covering Tour (tour-edges makred in double-line)

Figure 4.6: Label Covering tour in a cluster with five nodes

Label-Covering Tour Problem: Given graph $G = (V, E)$ with all its edges labeled, we find a label-covering tour t_{LC} in this graph so that the cost of the tour is the minimum i.e

$$min \sum_{\forall e \in t_{LC}(E)} f(e) \text{ where } t_{LC}(E) \text{ is the set of edges in this tour,}$$

where $f(e)$ is a cost function $f : E \to \mathbb{R}$ defined on the edges and $t_{LC}(E)$ is the set of edges of the Label Covering Tour t_{LC}.

In Figure 4.6, the MDC starts from the node 1 and travels the minimum cost tour $[1, 2, 5, 1]$. The union of the labels of these edges is the set of all labels i.e. $L(1) \cup L(2) \cup L(5) = \{1, 2, 3, 5\} \cup \{1, 2, 3, 4, 5\} \cup \{1, 2, 4, 5\} = \{1, 2, 3, 4, 5\}$. Therefore, this tour is the minimum-cost Label Covering tour.

NP-hardness of Label Covering Tour Problem: In [39], the authors show that the *Label Covering Tour* problem is *NP-hard*. If we choose a small TXR, we find a *Label Covering Tour* to include all the nodes. Thus it becomes a *Travelling Salesman Tour*. For this new value of TXR, the optimal *Label Covering Tour* is also an optimal *TSP-tour*. Since finding a *Travelling Salesman Tour* is NP-hard [45], so is finding a *Label Covering Tour*.

Algorithm 4.2 Generating the Minimum Length Label-Covering Tour

Input: A TSP tour T_{TSP}

```
d[0] ← 0                                              ▷ Array d[i] stores the path-cost from Node 0 to i
d[1...n] ← +∞
tour[0] ← {T_TSP[0]}
tour[1...n] ← ∅
for all i = 0, ..., n − 1 do
    for all j = i + 1, ..., n do
        shortCuttable ← true
        anchorSet ← ∅
        for all k = i + 1, ..., j do
            if line segment T_TSP(i)T_TSP(j) is NOT within range r of node T_TSP(k) then
                shortCuttable ← false
                break-loop
            else
                a_k ← Anchor Point for node T_TSP(k)         ▷ Anchor Point Computation
                anchorSet ← anchorSet ∪ {a_k}                ▷ Anchor Point Computation
            end if
        end for
        if shortCuttable = true & d[i] + |T_TSP(i)T_TSP(j)| < d[j] then
            d[j] ← d[i] + |T_TSP(i)T_TSP(j)|
            tour[j] ← {tour[i], T_TSP(j)}
            a[i][j] ← anchorSet ∪ {T_TSP(i), T_TSP(j)}       ▷ Anchor Point Computation
        end if
    end for
end for
T_LC ← tour[n]
```

Output: T_{LC} is the minimum length LC-tour

4.3.1 Label Covering Tour as a Result of Linear Shortcut

In [39], Sugihara and Gupta introduce *Label Covering* (LC) tour as a measure to reduce the data delivery latency. Given a TSP-tour, the authors give a polynomial-time algorithm for finding a shorter tour of the *TSP-tour* which they call *Label Covering Tour* (*LC-tour*). However, we are approaching the problem from the perspective of finding a linear shortcuts of a given *TSP-tour*, the challenge is how to derive *LC-tour* from *TSP-tour* by means of finding Linear Shortcuts. Given a TSP-tour T_{TSP}, Algorithm 4.2 generates *LC-tour* in polynomial time $O(n^3)$. There is a strategy S that describes finding shortcuts of a given *TSP-tour* to derive *LC-tour*. We summarize the strategy as follows:

1. Given a *TSP-tour* T_{TSP}, we select an Edge set E_{LC} according to Algorithm 4.2. Each edge $e \in E_{TSP}$ is may be included to E_{LC} or not.

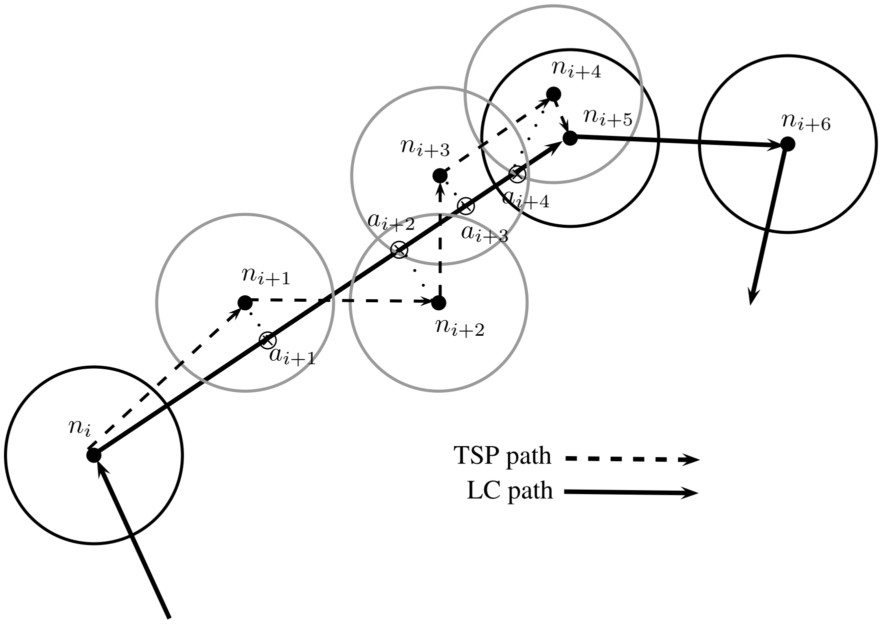


Figure 4.7: Anchor Points in an LC-tour as a result of finding a Linear Shortcut

2. If an edge of E_{TSP} is included, it will have exactly two Anchor Points on it. As shown in Figure 4.7, the edge $< n_{i+5}, n_{i+6} >$ cannot not be a Shortcut, Therefore, it is included in the set E_{LC}. Hence, it has exactly two Anchor Points i.e. its two end-points n_{i+5} and n_{i+6}.
3. If an edge $e \in E_{LC}$ is derived by finding Shortcut among two or more edges of E_{TSP}, then it has two or more Anchor Points. Two anchor-points are its end-points. Let n_j be a node that is not visited at its position due to the Shortcut on Edge e. The other Anchor Points on e are derived by calculating the intersection between the normal from node n_j and the edge e or, in case the intersection lies outside the line segment of e, the intersection of the circle of radius r centered at node n_j and edge e. As shown in Figure 4.7, the edge $< n_i, n_{i+5} >$ is derived by finding shortcuts on five consecutive edges of *TSP-tour.* Two Anchor Points of this edge are its endpoints. At these two points, the MDC collects data from node n_i and n_{i+5}. There are four more Anchor Points on this edge i.e. $a_{i+1}, \ldots, a_{i+4}$. For example, the normal drawn from node n_{i+1} to this edge intersects it at point a_{i+1} and hence this Anchor Point. But, the normal drawn from node n_{i+4} intersects this Shortcut outside its line segment. Therefore, the corresponding Anchor Point a_{i+4} is derived by the intersection of the circle of radius r centered at node n_{i+4} and the line segment of the edge itself.

We give Algorithm 4.2 to generate the minimum cost *Label Covering Tour*. Here, Line 14, 15 and 21 track the computation of Anchor Points for each prospective edge selected but does not affect the running time of the algorithm. If an edge in resulting LC-tour connects node n_i and n_j, the Anchor Points for the MDC on this particular edge can be found in the array element $a[i][j]$.

Lemma 4.2: If TXR of only the visited nodes in the LC-tour is zero, any tour derived by making linear shortcut of the LC-tour will not be complete.

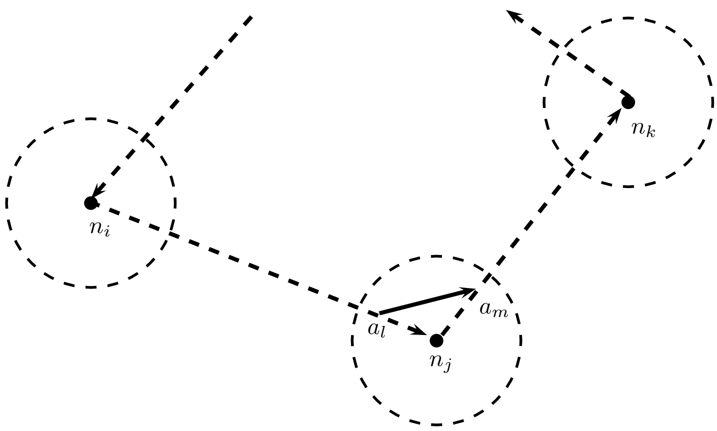


Figure 4.8: Making a Linear Shortcut of a Label Covering tour

Proof: As shown in Figure 4.8, we choose two Anchor Points a_l and a_m on two different successive edges of our tour T_{LC}. Connecting the two Anchor Points, we derive a tour T_d which is shorter than the min-cost LC-tour. The set of edges of the derived tour is as follows:

$$E_d = E_{LC} - \{< n_i, n_j >, < n_j, n_k >\} \cup \{< n_i, a_l >, < a_l, a_m >, < a_m, n_k >\}$$

using *Triangle Inequality*, $|T_d| < |T_{LC}|$. As shown in Figure 4.8, if $TXR \neq 0$, it is always possible to choose two such Anchor Points a_l and a_m different from the node n_j such that the resulting tour is shorter and complete. However, when $TXR = 0$, the derived tour T_d misses Node n_j unless $a_l = a_m = n_j$. Thus, when $TXR = 0$ for the visited nodes, any Linear Shortcut results in an *Incomplete Tour*. ■

Lemma 4.2 shows that, there is further scope of Linear Shortcut if $TXR \neq 0$. Let us explore this opportunity in the following sections.

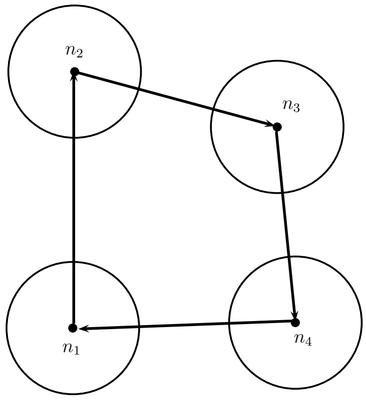
4.4 Tight Label Covering Tour (*TLC-tour*)

Goal: Given an *LC-tour* for the MDC in a sensor network with the non-zero TXR, we derive a tour by making linear shortcut such that the resulting tour is *complete.*

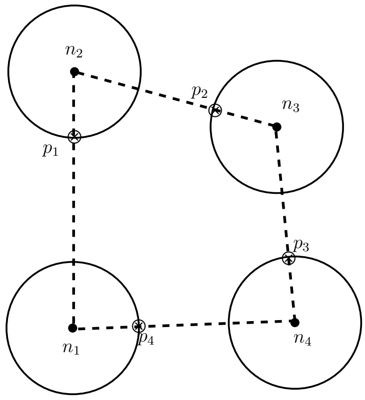
We present two possible cases of making linear shortcuts in the following sections.

Case I: *There are no overlapping intermediate nodes*

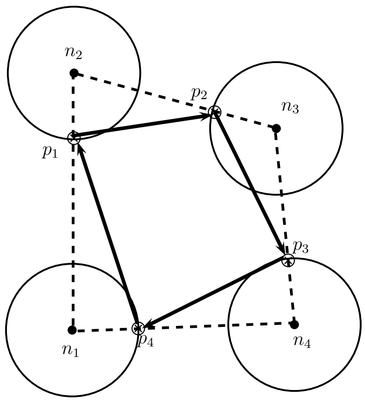
If there is no overlapping intermediate nodes in any edge of the resulting LC-tour derived from a TSP-tour, then the number of nodes visited in it is the same as in the TSP-tour, so is the length of the tour. But in our approach, it is also possible to shorten the length even further.



(a) LC-tour



(b) Anchor Points in LC-tour



(c) TLC-tour generated from the LC-tour

Figure 4.9: Deriving TLC-tour from LC-tour

As shown in Figure 4.9a, the initial TSP-tour and LC-tour are the same. The MDC visits all the nodes from n_1 to n_4 in the order of the minimum cost TSP-tour. To derive a *Tight Label Covering*

Tour, at first, we find the intersection of the incoming edge with the circles of radius TXR centered at the nodes. For example, the edge connecting Nodes n_1 and n_2 is incoming to the circle centered at n_2. This edge intersects the circle at p_1. However, this edge is not incoming to the circle centered at n_1, rather the edge connecting n_4 and n_1 is incoming to that circle. Thus, we derive the Anchor Points p_1, p_2, p_3 and p_4. Connecting these points, we derive a tour that is shorter than the LC-tour.

To generalize the rule, if there is no overlapping circle in any edge connecting nodes n_i and n_j,

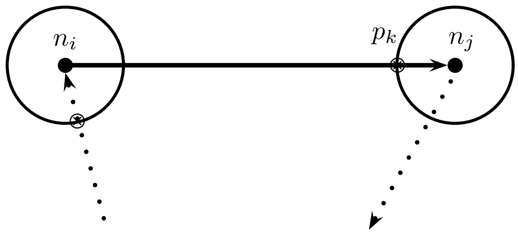


Figure 4.10: Selecting an Anchor Point from an edge with no overlapping circle

we pick only one point for finding Shortcut on that edge- this point is the intersection of the circle centered at node n_j. This is illustrated in Figure 4.10. Here, p_k is the intersection of circle centered at n_j and the edge connecting n_i and n_j. Therefore, p_k is the only point chosen from this edge. In the next section, we discuss the other case and the technique of iterative improvement.

Case II: *There are overlapping intermediate nodes.*

When there is one or more overlapping intermediate circles in an edge of the given LC-tour, the number of points p_i for finding Shortcuts can be more than one. To illustrate the technique, at first, let us define some terms related to the technique.

Definition 4.3: A line segment of a tour on which any particular node is reachable from the MDC is called the *Contact Interval* or CI for that node.

For example, in Figure 4.11, the node n_k is only reachable on the line segment $< ln_k, rn_k >$ on the

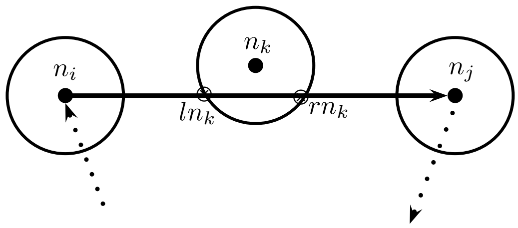


Figure 4.11: Contact Interval $< ln_k, rn_k >$ of Node n_k on a tour-edge

tour edge connecting n_i and n_j. Hence, it is the Contact Interval for this node n_k. We represent any Contact Interval for any intermediate node n_k by two points on that edge as follows:

Definition 4.4: The point which is encountered first by the MDC on the CI of a Node n is called the l-Point of the CI and it is denoted by ln. Similarly, the point encountered last by the MDC on the CI is called the r-Point and it is denoted by rn. These two points i.e. l and r Points mark the boundary of a CI.

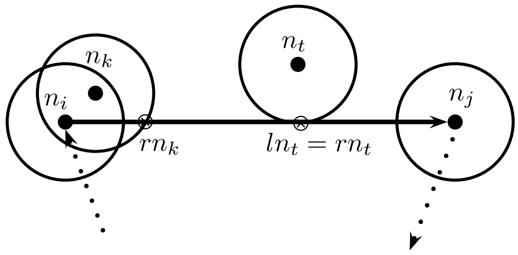


Figure 4.12: Contact Intervals of different nodes on a tour-edge

If the edge is tangent to the intermediate circle, we have $ln_k = rn_k$ as shown in Figure 4.12. Here l and r Points for the Contact Interval are the same for the node n_t as the edge is a tangent to its transmission circle. If the intersection of the edge and the circle lies outside the line segment of the edge, the Contact Interval contains at least one end-point of the edge. For example, the Contact Interval of Node n_k, as shown in the same figure, is $< n_i, rn_k >$.

4.4.1 Representation of the Contact Interval

From the given *LC-tour*, we can identify the intermediate nodes with circles having overlaps with the particular tour-edge. Then, we determine l and r Points of the CI for each such node and sort those CI's according to the non-decreasing distance of l Points from the first visited node on that edge.

As shown in Figure 4.13, there are five intermediate nodes whose transmission radii intersect with the tour edge connecting nodes n_i and n_j. For node n_{i+1}, the CI is given by $ln_{i+1} = (316, 122)$ and $rn_{i+1} = (398, 120)$. For node n_{i+5}, the right intersection point lies beyond the line segment of the edge. Therefore, r Point of its CI is the position of node n_j. In Table 4.1, the sorted CI's of the intermediate nodes are shown. For example, the coordinate of the first visited node of this edge n_i is $(234, 120)$. The l Point of any CI closer to Node n_i is that of the Node n_{i+1}. Therefore, its entry

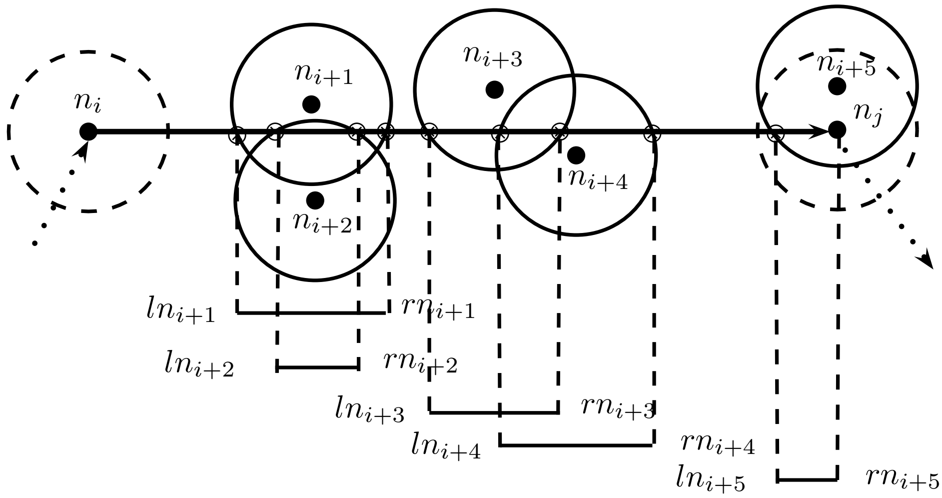


Figure 4.13: Contact Intervals of intermediate nodes of the edge connecting Nodes n_i and n_j

Node	***ln(x,y)***	***rn(x,y)***
n_i	N/A	N/A
n_{i+1}	$(316, 120)$	$(398, 120)$
n_{i+2}	$(337, 120)$	$(382, 120)$
n_{i+3}	$(422, 120)$	$(494, 120)$
n_{i+4}	$(461, 120)$	$(554, 120)$
n_{i+5}	$(613, 120)$	$(647, 120)$
n_j	N/A	N/A

Table 4.1: Sorted Contact Intervals for Figure 4.13

comes first in the sorted list. The l Point of Node n_{i+5} is the farthest from node n_i. Therefore, it is the last entry in the sorted list.

The Algorithm 4.3 generates the sorted CI's for any particular edge. If the sorting function in Line 12 runs in $O(n \log n)$ (there are many sorting algorithms available like heapsort) then its running time is $O(n + n \log n) = O(n \log n)$. The number of edges in the given LC-tour is $O(n)$. Therefore, determining the sorted Contact Intervals for the tour takes $O(n^2 \log n)$ time.

4.4.2 Critical Contact Interval (CCI)

Once we have determined the sorted list of Contact Intervals, the next step is to find the Critical Contact Interval.

Definition 4.5 Given a list of Contact Intervals CI_e of an edge e in a tour, *Critical Contact Interval*

Algorithm 4.3 Generating sorted *Contact Interval* for any tour-edge

Input: An edge $e \in E$ that connects Node n_i and Node n_j in an *LC-tour* and the list of intermediate nodes I_e

$CI_e \leftarrow \{\}$
for all node $n_k \in I_e$ **do**
  Find intersections (ln_k, rn_k) of edge e and circle of radius TXR centered at n_k
  if ln_k is outside of line segment of edge e **then**
    $ln_k \leftarrow n_i$
  end if
  if rn_k is outside of line segment of edge e **then**
    $rn_k \leftarrow n_j$
  end if
  $CI_e \leftarrow CI_e \cup \{(n_k, ln_k, rn_k)\}$
end for
sort CI_e using ln_k as key

Output: CI_e is the sorted *Contact Intervals*

or CCI is the interval of the minimum length that has at least one point from each the Contact Interval.

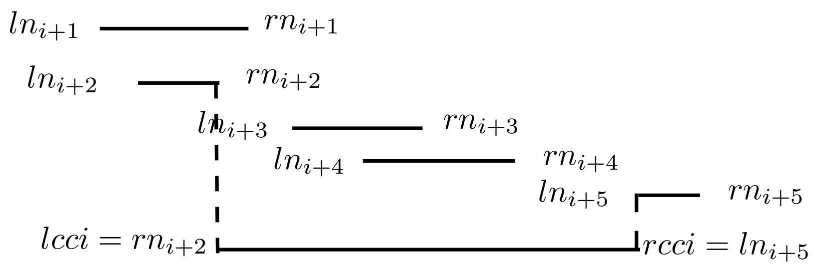


Figure 4.14: *Critical Contact Interval* for a given list of intervals starting from Node n_{i+1}

In Figure 4.14, the critical Contact Interval for the edge described in Figure 4.13 is shown. Here, the CCI has the left endpoint $lcci = rn_{i+2}$ and the right end point $rcci = ln_{i+5}$. If we assume that, the MDC travels along this edge, any interval having left endpoint any farther than the $lcci$ from n_i or right endpoint closer than the $rcci$ to n_i will not cover one or more intermediate nodes. For example if $lcci = (388, 120)$ instead of $rn_{i+2} = (382, 120)$, the MDC misses the intermediate Node n_{i+2}'s CI along this edge.

Generating CCI may be quite simple- for example, taking the rn of the leftmost interval and ln

Algorithm 4.4 Generating *Critical Contact Interval* (CCI)

Input: List of sorted intervals CI_e of an edge $e \in E$ of a tour

$(n_t, ln_t, rn_t) \leftarrow firstElementOf(CI_e)$
$lcci_e \leftarrow rn_t$
$(n_k, ln_k, rn_k) \leftarrow nextElementOf(CI_e)$
while ln_k closer to n_i than rn_t **do** ▷ scan all the intervals contained within the leftmost interval
  if rn_k closer to n_i than $lcci_e$ **then**
    $lcci_e \leftarrow rn_k$
  end if
  $(n_k, ln_k, rn_k) \leftarrow nextElementOf(CI_e)$
end while
$(n_s, ln_s, rn_s) \leftarrow lastElementOf(CI_e)$
$rcci_e \leftarrow ln_s$
if $rcci_e$ closer to n_i than $lcci_e$ **then**
  $rcci_e \leftarrow lcci_e$
end if

Output: $CCI_e = (lcci_e, rcci_e)$ is the Critical Contact Interval of edge e

of the rightmost interval. Since, the intervals are already sorted, this takes $O(1)$ time.

$$lcci \leftarrow rn \text{ of the leftmost interval}$$

$$rcci \leftarrow ln \text{ of the rightmost interval}$$

However, we have sorted the intervals according to ln values. Therefore, assigning ln of the rightmost interval to $rcci$ is correct, but assigning the rn of the leftmost interval may result in missing one or more intervals totally contained within the leftmost interval. For example, in Figure 4.14, the leftmost CI is of intermediate node n_{i+1}. However, Contact Interval of node n_{i+2} is fully contained within this interval. If we assign $lcci = rn_{i+1}$, then the resulting interval misses CI of Node n_{i+2}. Therefore, unlike $rcci$, the value of $lcci$ cannot be determined in $O(1)$ time. We have to scan the sorted list starting from the beginning till all intervals contained within the leftmost intervals are checked for the leftmost rn value. This will be the correct value for $lcci$. Algorithm 4.4 does the job in $O(n)$ time for any edge.

If there is no intermediate node having overlapping circle with any edge, the CCI of that edge is Null. It is to be noted that, we have deliberately left out the nodes visited by the LC-tour. The next section deals with the incorporation of non-intermediate nodes.

4.4.3 Finding Shortcut by Bypassing the Visited Nodes

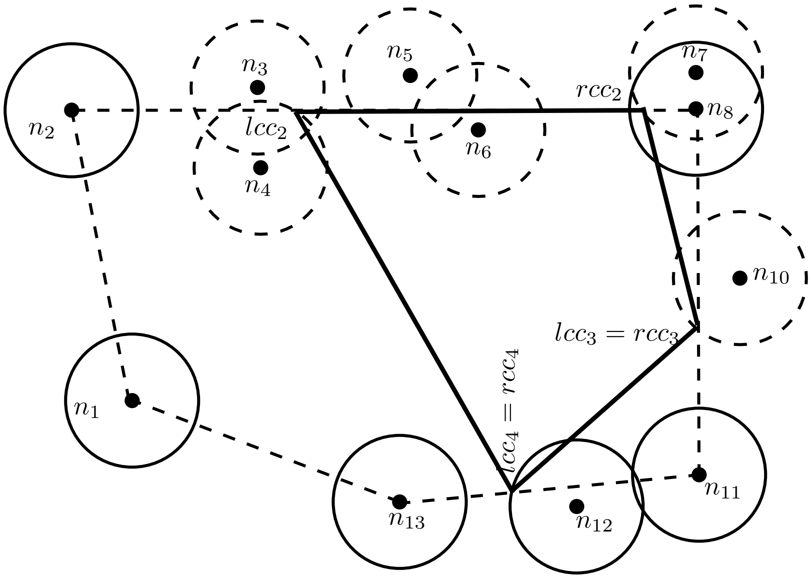


Figure 4.15: Connecting the *Critical Contact Intervals* of successive edges to form a Shortcut tour

Let us ignore the visited nodes which are the endpoints of the edges of the LC-tour. Then, we can connect the CCI's of successive edges of the LC-tour to derive a shortcut tour. This has been illustrated in Figure 4.15. We have already derived the CCI of the edge connecting Nodes n_2 and n_8 in the previous section. The CCI on edge $< n_8, n_{11} >$ is a single point according to Algorithm 4.4, which is the r point of the circle centered at n_{10}. So is the case for edge $< n_{11}, n_{13} >$. However, edges $< n_1, n_2 >$ and $< n_{13}, n_1 >$ do not have any intermediate nodes with overlapping circles. Therefore, they have no CCI's. If we connect the endpoints of the CCI's of successive edges, we derive a Shortcut tour that covers only the intermediate nodes in the LC-tour. This tour is shown by solid lines in Figure 4.15. We denote the $lcci$ of i-th edge (as ordered in a given tour) as the point lcc_i and $rcci$ of i-th edge as rcc_i.

4.4.4 A Complete Shortcut Tour

The tour derived in the previous section is shorter than the LC-tour but is not complete. To make the tour complete, we have to extend its path to cover nodes visited in LC-tour; for example, Nodes n_1, n_2, n_8, n_{11} and n_{13} must also be covered in the example given by Figure 4.15. To do this, we

Algorithm 4.5 Generating Tight Label Covering Tour

Input: A tour t with CCI's associated with each edge

for all node n_i visited in the tour t **do**
  if both the edges e_s (incoming) and e_t(outgoing) incident with n_i have CCI **then**
    if line l_{st} connecting r point and l point of the CCI's of edges e_s and e_t respectively does not intersect circle centered at n_i **then**
      l_{n_i} is the line parallel to l_{st} and tangent to the circle centered at n_i
      update r point of edge e_s as the intersection of l_{n_i} and e_s
      update l point of edge e_t as the intersection of l_{n_i} and e_t
    end if
  else
    p_{n_i} is the intersection of the incoming edge e_s and circle centered at n_i
    if p_{n_i} is closer to n_i than r point of the CCI of incoming edge e_s **OR** CCI for incoming edge e_s does not exist **then**
      update r point of edge e_s as p_{n_i}
      if CCI for incoming edge e_s does not exist **then**
        update l point of edge e_s as its r point
      end if
    end if
  end if
end for
$t_{TLC} \leftarrow \{\}$
Join r point of an edge to the l point of the next edge successively and add it to tour t_{TLC}

Output: t_{TLC} is a TLC tour

choose one or more Anchor Points p_i's both from the edges which have CCI's and edges which do not. Algorithm 4.5 is used to generate a shorter tour from the given LC-tour provided that the CCI's of the edges are already calculated according to Algorithm 4.4.

To cover the visited nodes in the resulting Shortcut tour, we take different actions depending on the status of the CCI's of the two edges adjacent with the visited node n_i:

1. If both of the edges have non-null CCI's, i.e. there are intermediate nodes on both the edges, then we just add the r point of the incoming edge with the l point of the outgoing edge. We call this line segment *r-l line segment.*

   (a) If r-l line segment is intersecting with the node n_i under inspection, then we do nothing. For example, in Figure 4.16, Node n_8 has both the edges with non-null CCI's. We connect r-point of the incoming edge rcc_2 with the l-point of the outgoing edge lcc_3. The resulting r-l line segment intersects the circle centered at n_8. Hence, Node n_8 is covered by this

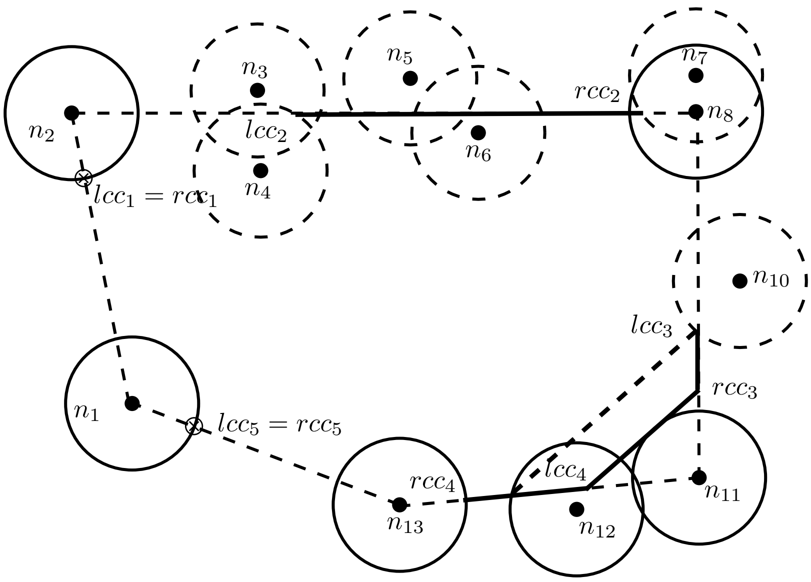


Figure 4.16: Updated l and r points to cover visited nodes

newly added edge.

(b) If r-l line segment is non-intersecting with the Node n_i, then we draw a straight line that is parallel to the r-l line segment and tangent to the circle centered at n_i. Let this line intersects the incoming and outgoing edges at Points p_i and p_o respectively. We, then, add a new edge connecting p_i and p_o, which is a tangent to the circle centered at n_i and therefore, covers Node n_i. We also update the r point of the incoming edge as p_i and l point of the outgoing edge as p_o. For example, in Figure 4.16, for Node n_{11}, the r point of incoming edge and the l point of the outgoing edge are rcc_3 and lcc_4 respectively. The straight line connecting these two points does not intersect the circle centered at n_{11}. Therefore, we draw a line segment parallel to this straight line and tangent to the circle stated before. The resulting edge covers the node n_{11}. We also update the r point of the incoming edge as rcc_3 which is the intersection of this newly added edge and the incoming edge. Before, the l and r Points were similar for this incoming edge, but now, the two points become different. The l point of the outgoing edge is also updated as $lcci_4$ which is the intersection of the newly added edge and the outgoing edge.

2. If the incoming edge does not have any intermediate Node with overlapping circle or (if it has then) its r Point is farther from Point n_i by at least TXR, then we compute p_i as the intersection

between the incoming edge and the circle centered at n_i. If the incoming edge has a non-null CCI, then we update its r Point as p_i. Otherwise, we set the incoming edge's r and l Point as p_i. For example, in Figure 4.16, n_{13} has only one adjacent edge i.e. the incoming edge with non-null CCI. Previously, the r Point of this edge was the intersection of n_{12} with this edge that lies closer to n_{13}. We update this r Point as rcc_4, which is the intersection of this edge with Node n_{13}. In the same figure, Node n_1 has both the adjacent edges with null CCI. Therefore, we update both the l and r Point of the incoming edge as the intersection of this edge with the circle centered at n_1 i.e. the Point $lcc_5 = rcc_5$. For Node n_2, the outgoing edge has a non-null CCI but the incoming edge does not. Therefore, we determine the r and l Point of its incoming edge as $lcc_1 = rcc_1$.

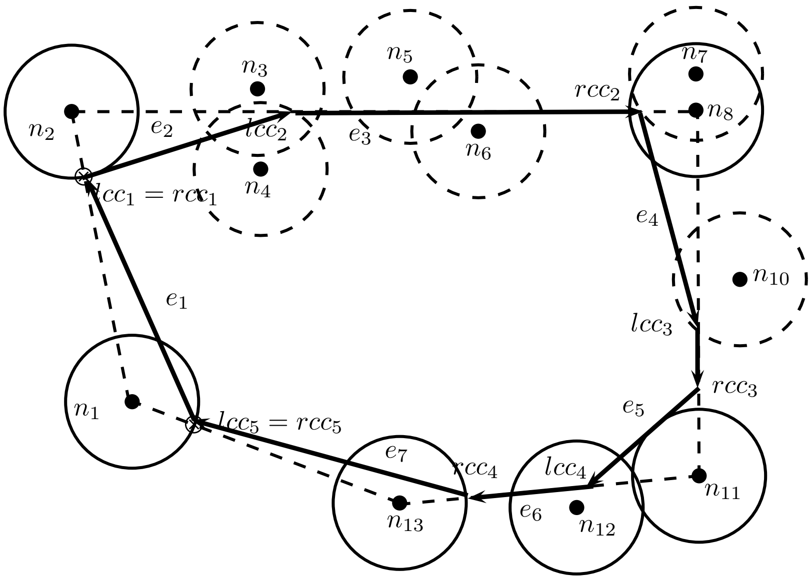


Figure 4.17: TLC-tour derived in Iteration 1

Now, we have all the edges with non-null CCI i.e. with both l and r points and we can join the r Point of the previous edge with the l Point of the next edge. The final edges are shown by solid lines in Figure 4.17. The resulting path will be a cycle and has been derived according to the strategy of the Algorithm 4.1, since we have chosen at most two Anchor Points from each edge and connected them in succession. Therefore, the resulting tour will be shorter than the given tour according to Lemma 4.1. In other word, we have derived a tour that is shorter in length than the given tour. We call this process *tightening* of the given tour by Linear Shortcut. We call the shorter tour derived from the Label Covering Tour *Tight Label Covering* tour or *TLC-tour*.

The process of deriving the *TLC-tour* is formally presented in Algorithm 4.5. For an LC-tour, we compute CCI for each edge according to Algorithm 4.4. Then, using Algorithm 4.5, we compute the TLC-tour. Algorithm 4.5 updates the l and r Points, and adds new edge if necessary, for each visited node in the given tour. This computation takes $O(n)$ time. The successive joining in Line 19 takes $O(n)$ time. Therefore, Algorithm 4.5 takes $O(n)$ time.

4.4.5 Iterative Improvement of *TLC-tour*

The path found in Figure 4.17 can be further shortened using method of finding Linear Shortcut outlined before. To apply Linear Shortcut, we select $0, 1$ or 2 points from each tour edge and connect them successively. We divide each iteration of improvement into two steps:

1. We connect the r Point rcc_i of i-th edge with l Point lcc_j of the next edge (j-th edge such that $j > i$) with non-Null CCI and include the edge connecting lcc_j and rcc_j in the edge set.
2. We *re-associate* the intermediate circles with the resulting edges and recompute the CCI's for each edge.

We outline the steps of generating sorted CI of each associated circle in Algorithm 4.3 and the steps to compute CCI in Algorithm 4.4. However, we need a policy to *re-associate* the circles when existing tour-edges *break* into shorter ones and new edges are *added*. Let us illustrate the method by applying on the tour derived in Figure 4.17.

As shown in Figure 4.18, there are eight edges. We label the edge that connects points within the range of Nodes n_1 and n_2 as the first edge, the edge next to it as the second edge and so on. Node n_1 overlaps both of the first edge (outgoing) and the last edge (incoming). The outgoing first edge has a non-zero overlap with the circle centered at Node n_1, but the incoming last edge does not. Therefore, we associate Node n_1 with the first edge rather than the last one in the tour. Node n_2 has zero overlap with both of the first (incoming) and the second edge (outgoing). As a tie-breaker, we associate it with the incoming first edge. Therefore the first edge is associated with two circles- one centered at n_1 and the other at n_2. Their CI's are computed according to Algorithm 4.3. The CCI is also computed for the first edge according to Algorithm 4.4. The l and r Points of the first edge is

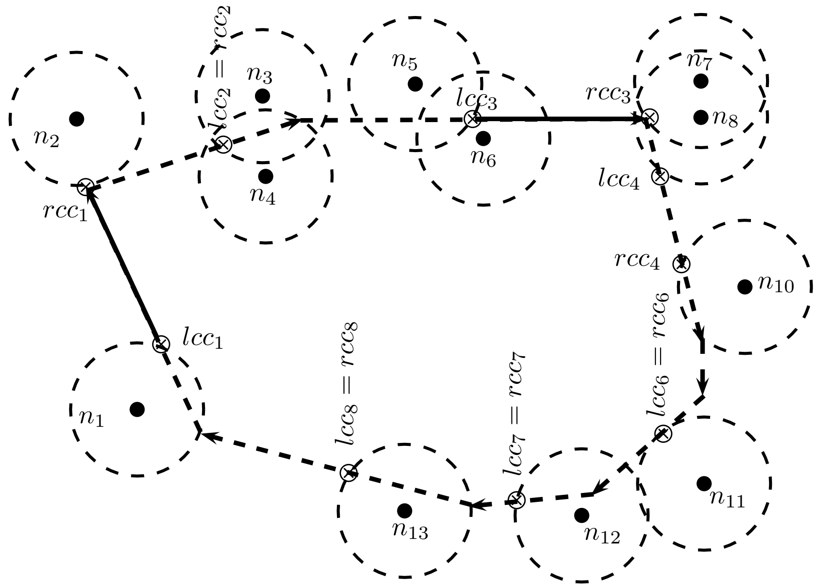


Figure 4.18: Updating the l and r point in the path derived in Figure 4.17

lcc_1 and rcc_1 respectively.

The second edge of the given tour *broke* at the boundary of the circle centered at n_4 to give out the second edge and the third edge. We need to decide which of these two edges to associate to circles centered at n_3 and n_4. Both of the circles have bigger CI's with the second edge than with the third one. Therefore, we associate both the circles with the second edge. Now, the CI's of both of these circles border on the right end point of the second edge. There are no other CI's on this edge. Therefore, we select both of the l and r Point of CCI as the rightmost l Point of the CI's of these two circles, this point is $lcc_2 = rcc_2$, as shown in Figure 4.18.

The third tour edge has CI's of circles centered at n_5, n_6, n_7 and n_8. Since, n_8 has larger CI with the next edge than this edge, we don't associate it with this edge. The circle centered at n_7 has CI with both of this third edge and the next edge. In both of the cases, the CI is a single point. Since, the third edge is incoming, we associate this circle with the third edge. In the similar fashion as done in previous tour edges, the l and r Points of this tour edge is determined as lcc_3 and rcc_3 respectively.

In the same way, we determine the l and r points of the CCI's of the remaining edges. It is to be noted that, the edge exiting circle centered at node n_{10} has no CCI and hence, it has no l and r Points.

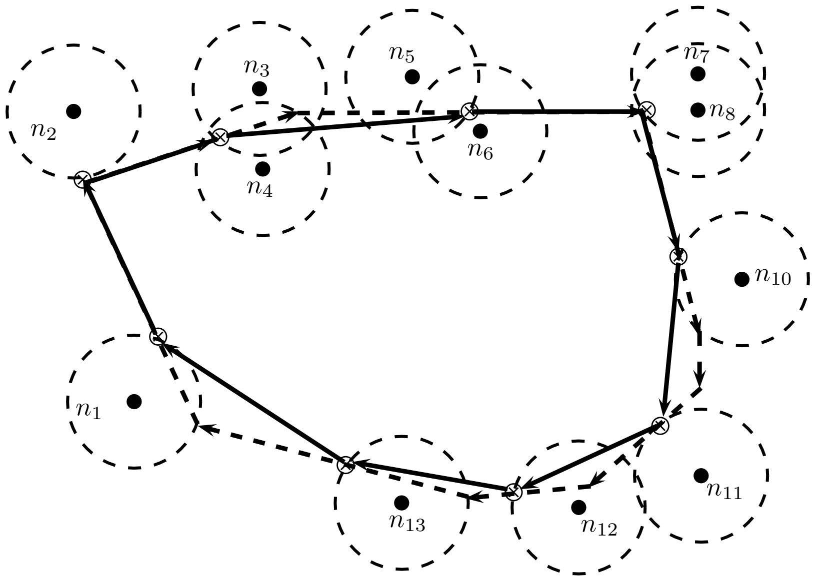


Figure 4.19: *TLC-tour* derived in Iteration 2

Therefore, this edge will be skipped out of the resulting tour. After this round of re-associating of circles and computation of l and r Points of CCI's of respective edges, we join the r Point of an edge with the l Point of the next edge with CCI. Thus, the resulting tour is shown in Figure 4.19. Now, there are one more edge than the given tour but since it is derived by finding Linear Shortcut, it is shorter than the given tour.

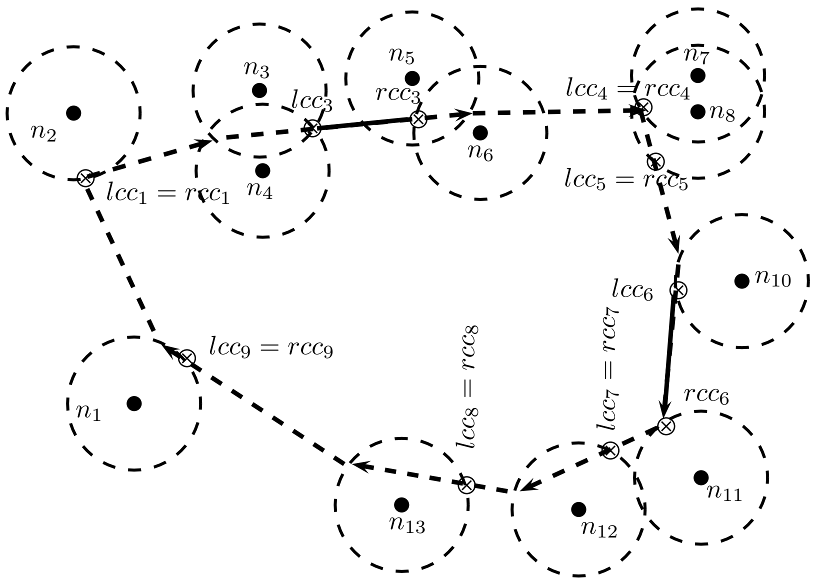


Figure 4.20: Updating the l and r Points after Iteration 2

We can continue in this way to successively *tighten* the given tour. For example, in Figure 4.20, the l and r Points of the tour derived in Iteration 2 are computed after re-associating the circles. We note that, of the nine tour edges, only two have distinct l and r Points for the CCI's, one edge has none of those and the rest of the edges have $lcc_i = rcc_i$.

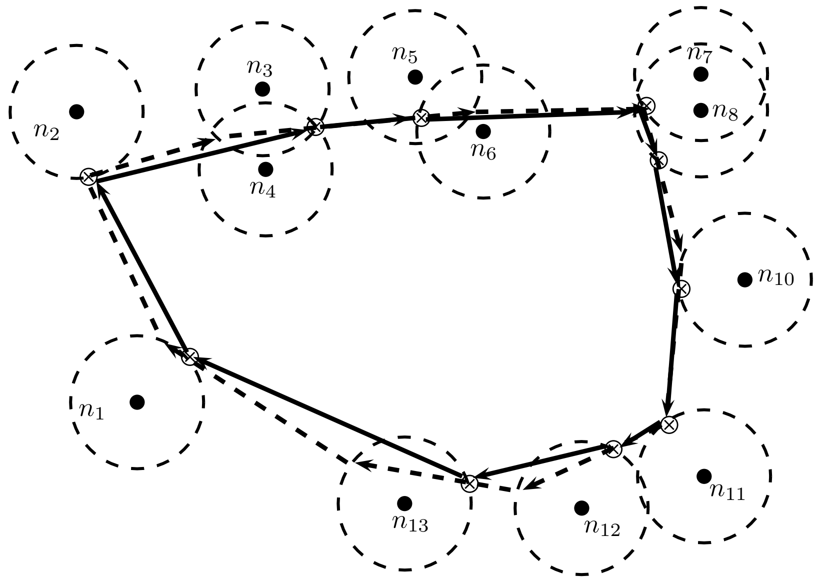


Figure 4.21: *TLC-tour* derived in Iteration 3

Using the same method as in the previous iteration, we connect the r points with then next edge's l Point to derive a tightened tour as shown in Figure 4.21. We note that, the amount of path saving has decreased in successive iterations, the highest saving being attained in the very first iteration.

We, again re-associate the circles and compute the CCI's on the tour derived after Iteration 3, as shown in Figure 4.22. After Iteration 4, we derive the tightened tour shown in Figure 4.23. We note that, the path savings have become even more smaller. Therefore, we stop our iterative improvement here and choose Anchor Points for the MDC.

From each node position n_i, we draw perpendicular to the edge with which the corresponding circle is associated with. The intersection of this perpendicular with the edge is the Anchor Point a_i for that node n_i. When the MDC reaches the point a_i, it polls the target node n_i for data packets. If the point a_i is out of the line segment of the edge, we choose the intersection of the corresponding circle and the edge as the Anchor Point. For example, in Figure 4.23, node n_7 has no such perpendicular on its associated incoming edge. Therefore, we choose this edge's endpoint that intersects the circle

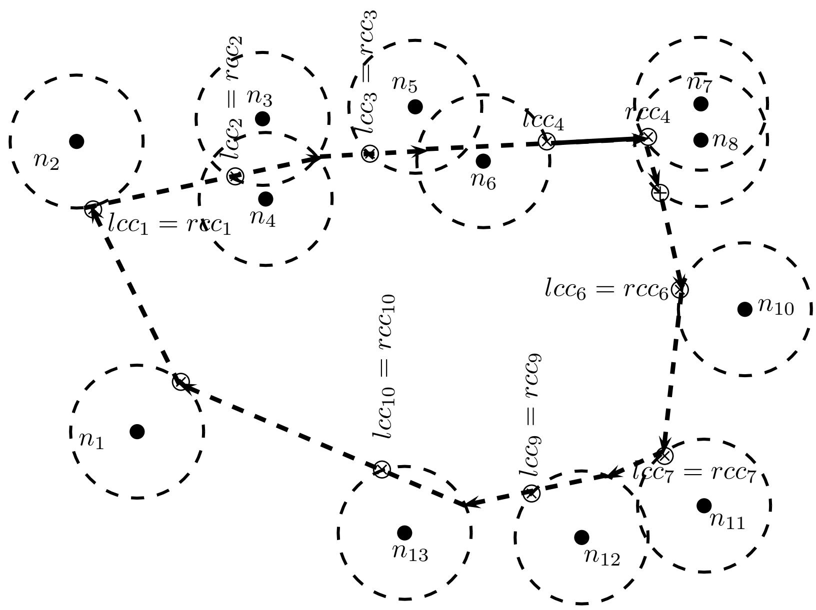


Figure 4.22: Updating the l and r Points after Iteration 3

centered at node n_7.

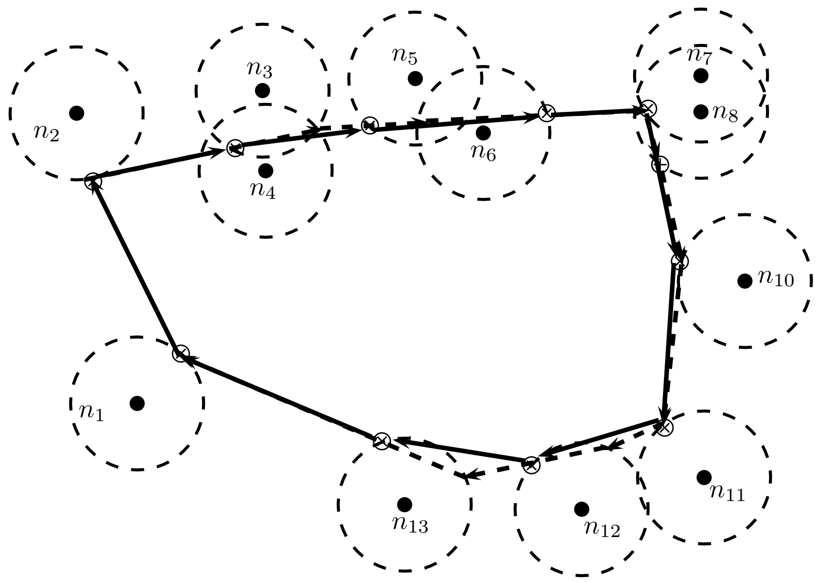


Figure 4.23: TLC-tour derived in Iteration 4

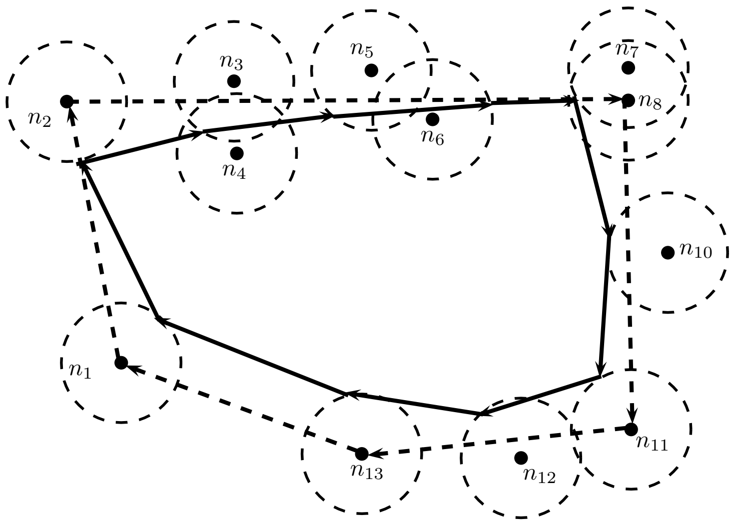


Figure 4.24: Comparison between input LC-tour(doted path) and TLC-tour derived in Iteration 4

We have already noticed that, in successive iterations, the path gain deceases. We can define the path gain $g_i(t_{TLC})$ for a derived TLC-tour in iteration i as follows:

$$g_i(t_{TLC}) = \frac{|t_{TLC}|_{i-1} - |t_{TLC}|_i}{|t_{TLC}|_i} \tag{4.1}$$

Here, $|t_{TLC}|_i$ is the length of the TLC-tour derived in Iteration i. Nevertheless, the resulting gain is a significant improvement over the given *LC-tour.* The given *LC-tour* and the *TLC-tour* derived after Iteration 4 are over-imposed on each other for comparison (see Figure 4.24).

We have described our method for successive iterative tightening. Now, we give the formal algorithm. In Iteration 1, we use Algorithm 4.3 to generate sorted CI's for each circle associated with each edge and use Algorithm 4.4 to derive CCI's from the list of sorted CI's and finally, use Algorithm 4.5 to generate the first TLC-tour. For successive iterations, we also need to do the similar things except we don't have to sort all the CI's; rather we have checked only few marginal circles for re-association and when those are associated with a different edge, their CI's are re-computed and inserted into the list of CI's of the respective edge in non-decreasing order of the l value (the distance between the first endpoint and the l Point of a CI).

To speed up the task of re-association, specially the process of updating contact interval of each node, we maintain a node-list where current associated edge of each node is also kept. For example, after the first iteration as shown in Figure 4.17, the list of node associated with the edge after re-association step is shown in Table 4.2. This table allows the retrieval of the associated edge with a node in $O(1)$ time and helps us avoid searching the edge list repeatedly. After the first iteration, the number of edges change, Therefore, we need to scan this list and update the edge number. This scan takes $O(n)$ time. Then, we can scan the list again and re-associate a node, which is currently associated with edge e_i, with either the previous edge e_{i-1} or the next edge e_{i+1} or just keep its current association with edge e_i. Again, this step takes $O(n)$ time. The decision regarding the re-association has been illustrated in Figure 4.25.

Node	**Associated Edge**
n_1	e_5
n_2	e_1
n_3	e_2
n_4	e_2
n_5	e_2
n_6	e_2
n_7	e_2
n_8	e_2
n_{10}	e_3
n_{11}	e_3
n_{12}	e_4
n_{13}	e_4

(a) The list for the given *LC-tour*

Node	**Associated Edge**
n_1	e_8
n_2	e_1
n_3	e_3
n_4	e_3
n_5	e_3
n_6	e_3
n_7	e_3
n_8	e_3
n_{10}	e_5
n_{11}	e_6
n_{12}	e_7
n_{13}	e_7

(b) The list after updating Labels in Iteration 1

Node	**Associated Edge**
n_1	e_1
n_2	e_1
n_3	e_2
n_4	e_2
n_5	e_3
n_6	e_3
n_7	e_3
n_8	e_4
n_{10}	e_4
n_{11}	e_6
n_{12}	e_7
n_{13}	e_8

(c) The list after re-associating with edges in Iteration 1

Table 4.2: The node-list with associated edges for the *LC* and *TLC-tour* for Figure 4.17

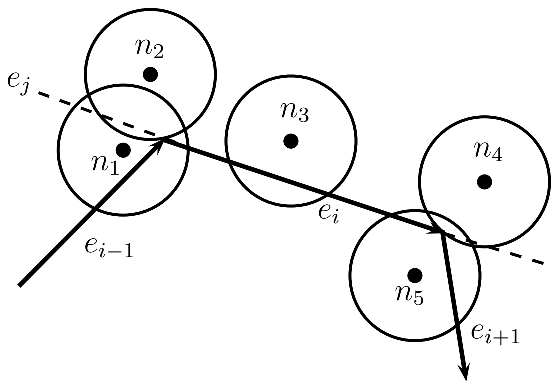


Figure 4.25: Example of re-association process for generating *TLC-tour*

As shown in Figure 4.25, in the previous iteration all the nodes from n_1 through n_5 were associated with edge e_j. After addition of new edges and deletion of existing edges, this edge e_j is labeled as e_i in the current iteration. We run the re-association test for all the five nodes of edge e_i as follows:

1. We compute the intersections between circle centered at n_1 and straight line representing edge e_i. We check that these two points are not the same as the l and r Points of the circle with respect to this edge. This tells us that, the circle is not fully contained with edge e_i. Since the l Point is different from the intersection, we infer that the circle has an overlap with the previous Edge e_{i-1}. Therefore, we determine the l and r Points for this circle with respect to Edge e_{i-1}. Finally, we determine the length of CI's for both of the edges and find that it is longer for edge e_{i-1}. Therefore, we delete this node's CI from the list of Edge e_i and add it the corresponding list of Edge e_{i-1}. We also update the node-association list.
2. For Node n_2, we find that the l and r points are same and is aligned with the first endpoint of Edge e_i. Therefore, its a candidate for checking with previous Edge e_{i-1} and we find the same case for this edge too i.e. l and r Points are same. This means that the CI's are of same length for these two edges. In this case, we associate this node with the incoming edge e_{i-1}. This is our tie-breaking measure.
3. Using the test as outlined in above cases, we find that Node n_3 is fully contained by endpoints of edge e_i. Therefore, it is kept associated with it.
4. Just like node n_2, node n_4 is associated with the incoming edge e_i.
5. Like the case for Node n_1, Node n_5 is associated with Edge e_{i+1} because its CI is longer for this edge.

Theses steps have been outlined in Algorithm 4.6. It does the above computation in $O(1)$ time for each node, however, we have sorted the list of CI's for each edge by the non-decreasing *l-values*. Therefore, when we insert the new interval in an existing or new list during re-association, we still keep the list sorted by maintaining data structure like heap keyed on the *l-values* and it is done in $O(\log n)$ time. This computation is done for each of the n nodes. Therefore, the total running time of Algorithm 4.6 is $O(n \log n)$.

Algorithm 4.6 Re-associating nodes with edges

Input: Set of all nodes V with each node $n_i \in V$ indexed with associated edge e_j and set of tour edge E of TLC-tour, with each edge $e \in E$ having Contact Interval CI_e sorted on l points

for all node $n_i \in V$ **do**
  $e_j \leftarrow$ current associated edge of node n_i
  $l_j \leftarrow$ updated l point of node n_i with respect to edge e_j
  $r_j \leftarrow$ updated r point of node n_i with respect to edge e_j
  if $l_j =$ left-end point of edge e_j **then** ▷ node's CI aligned with left end-point of the edge
    $l_{j-1} \leftarrow$ updated l point of node n_i with respect to edge e_{j-1}
    $r_{j-1} \leftarrow$ updated r point of node n_i with respect to edge e_{j-1}
    if $distance(l_{j-1}, r_{j-1}) > distance(l_j, r_j)$ **then**
      remove node n_i from I_{e_j}
      add node n_i to list $I_{e_{j-1}}$ in non-decreasing order of l_{j-1}
      update n_i's associated edge as e_{j-1}
    end if
  end if
  if $r_j =$ left-end point of edge e_j **then** ▷ node's CI aligned with right end-point of the edge
    $l_{j+1} \leftarrow$ updated l point of node n_i with respect to edge e_{j+1}
    $r_{j+1} \leftarrow$ updated r point of node n_i with respect to edge e_{j+1}
    if $distance(l_{j+1}, r_{j+1}) > distance(l_j, r_j)$ **then**
      remove node n_i from I_{e_j}
      add node n_i to list $I_{e_{j+1}}$ in non-decreasing order of l_{j+1}
      update n_i's associated edge as e_{j+1}
    end if
  end if
end for

Output: Set of nodes V with each node $n_i \in V$ with updated associated edge, Set of tour edges E with each edge $e \in E$ with updated Contact Interval CI_e

After re-association and updating of CI's, we update the CCI for each edge. We may use the same algorithm used in Iteration 1. However, in Iteration 1, nodes visited in the *LC-tour* are not considered; rather those are handled specially (in generating *TLC-tour*) by drawing tangents to those nodes and finding intersections of the tangent with the existing edges. We also speed up the computation as follows.

As shown in Figure 4.26, both of the circles' r Points are the right endpoint of the associated edge and this point is also the l point of the CCI i.e. lcc_i. The computed r Point of the CCI is the l Point of the second circle. Because r Point is encountered before the l Point, according to the Algorithm 4.4 of Iteration 1, we may set $rcci \leftarrow lcc_i$. Instead, we attempt to put the coincident l and r Points of the CCI closer to middle of the edge as possible. For this reason, we choose to set

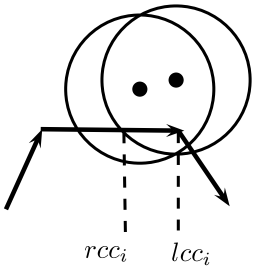


Figure 4.26: Special check for computing CCI for Iteration $i > 1$

$lcc_i \leftarrow rcc_i$. Similar step is followed for the case involving left end point of the edge. Other than this check, the computation is similar to that of Iteration 1 by Algorithm 4.4. Algorithm 4.7 does this job for Iteration $i > 1$ in $O(n)$ time for each edge and $O(n^2)$ time for the complete tour.

Algorithm 4.7 Generating Critical Contact Interval for Iteration $i > 1$

Input: List of sorted intervals CI_e of an edge $e \in E$ of a tour

$(n_t, ln_t, rn_t) \leftarrow firstElementOf(CI_e)$
$lcci_e \leftarrow rn_t$
$(n_k, ln_k, rn_k) \leftarrow nextElementOf(CI_e)$
while ln_k closer to n_i than rn_t **do** ▷ scan all the intervals contained within the leftmost interval
  if rn_k closer to n_i than $lcci_e$ **then**
    $lcci_e \leftarrow rn_k$
  end if
  $(n_k, ln_k, rn_k) \leftarrow nextElementOf(CI_e)$
end while
$(n_s, ln_s, rn_s) \leftarrow lastElementOf(CI_e)$
$rcci_e \leftarrow ln_s$
if $lcci_e$ = right end point of edge e **then**
  $lcci_e \leftarrow rcci_e$
else if $rcci_e$ closer to n_i than $lcci_e$ **OR** $rcci_e$ = left end point of Edge e **then**
  $rcci_e \leftarrow lcci_e$
end if

Output: $CCI_e = (lcci_e, rcci_e)$ is the CCI of Edge e

The next step is connecting the r Points with the l Points successively. The complete steps of generating TLC-tour is shown in Algorithm 4.8. According to this algorithm, we always keep the CCI's of the edges updated for the next iteration. Therefore, in the beginning of any iteration, we join the r Points with the l Points successively. This is done in Lines 1 - 5 in $O(n)$ time. In Line 6, the circles are re-associated in $O(n \log n)$ time. Finally, the CCI's are updated for each edge in Lines 7 - 9 in $O(n^2)$ time. Therefore, the total running time of the Algorithm 4.8 is $O(n^2)$.

Algorithm 4.8 Generating TLC-tour for Iteration $i > 1$

Input: Set of all nodes V and set of all edges E_{i-1} of TLC-tour of iteration $(i-1)$

$E_i \leftarrow \{\}$
for all edge $e \in E_i$ with non-null CCI **do**
  add CCI of edge e to E_i
  connect r point of edge e to the l point of next edge with CCI and add it to E_i
end for
Re-associate nodes for the set of edge E_i according to Algorithm 4.6
for all edge $e \in E_i$ **do**
  Update CCI according to Algorithm 4.7
end for

Output: Set of all nodes V and set of all edges E_i of TLC-tour of iteration i

4.4.6 Selecting the Anchor Points

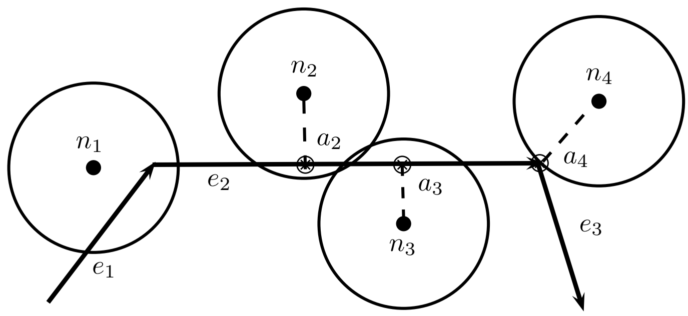


Figure 4.27: Selection of Anchor Points for nodes after TLC-tour is generated

After our algorithm has generated a *TLC-tour*, we fix some points in each edge where the MDC halts and initiates communication with one or more nodes attached to that particular tour edge. Each of these points is called *Anchor Point* and is denoted by a_i. A set of nodes $\{n_i\}$ is attached with Anchor Point a_i.

For example, in Figure 4.27, the Anchor Points for Edge e_2 are shown. Three nodes n_2, n_3 and n_4 are attached to this edge. We draw a perpendicular from the centers of the circles to the edge and the resulting intersections are the Anchor Points for the corresponding nodes. For example a_2 and a_3 are two Anchor Points for Nodes n_2 and n_3 respectively. MDC halts at Point a_2 and communicates with Node n_2 for data packets. But, the intersection of the perpendicular drawn from the circle centered at n_4 with the straight line representing Edge e_2 lies outside the line segment representing this edge. Therefore, instead of drawing a perpendicular, we use the right endpoint of this edge as the Anchor

Point a_4 for Node n_4.

After completion of the iteration generating TLC-tour, we determine the Anchor Point a_i for each node n_i. Computation for each node takes $O(1)$ time. Therefore, the whole tour including finding the Anchor Points for all nodes $O(n)$ time.

4.4.7 Computational Time Complexity

In Iteration 1, we use Algorithm 4.3 which runs in $O(n \log n)$ time and Algorithm 4.4 which runs in $O(n^2)$ time and finally, Algorithm 4.5 to generate the first TLC-tour, which runs in $O(n)$ time. Therefore, the time complexity for generating the *TLC-tour* in Iteration 1 is $O(n \log n) + O(n^2) + O(n) = O(n^2)$. As explained previously, the running time for computation in Iteration $i > 1$ is $O(n^2)$. Therefore, we generalize that, the running time for generating *TLC-tour* by m iterations is $O(mn^2)$.

We can stop the iterative improvement as soon as the path gain as defined by Equation 4.1 is below a certain threshold like 5%, 1% etc.

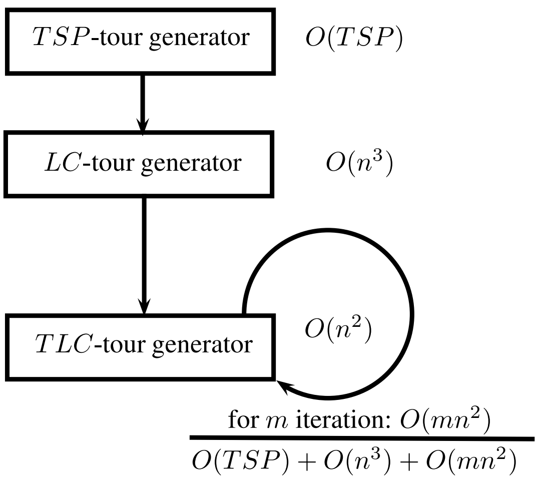


Figure 4.28: Time complexity of the algorithms for generating *TLC-tour*

The stages of computing *TLC-tour* is shown in Figure 4.28. The time complexity of the computation depends on two factors- the time complexity of the algorithm used to find the *TSP-tour* and the number of iterations m in the steps of making a Linear Shortcut. However, the combined stages of computation after finding the *TSP-tour* runs in polynomial time ($O(n^3 + mn^2)$). If the algorithm

Node	**x-Coord.**	**y-Coord.**
n_1	100.4	201.9
n_2	30	10
n_3	95	2
...	...	...
n_{100}	301	202

(a) Input

anchor-x	**anchor-y**	**Node-list**
302	308	starting point(sink)
290	205	n_{60}
202	192	n_{75}, n_{80}
...	...	...
302	308	starting point(sink)

(b) Output

Figure 4.29: Sample input and output of our method of generating *TLC-tour*

to find *TSP-tour* does not run in polynomial time, then its time complexity dominates that of our algorithm.

In Figure 4.29, a sample input and a sample output of our method of generating *TLC-tour* are shown. The input consists of nodes and their coordinates. The *TSP-tour* is generated from this information. The output is a set of coordinates, each of which is associated with some actions. For example, the first point in the list is the starting point or sink. The MDC starts its tour from this point. Since, the tour is a cycle, the first point is also the last point in the tour. The second point is an Anchor Point that is associated with node n_{60}. The third point is an Anchor Point for both the nodes n_{75} and n_{80}. The MDC moves from one point of the list to the next point and does the job associated with the point. This is the final output of our algorithm that minimizes data delivery latency and maximizes node lifetime by avoiding packet-forwarding.

Chapter 5

Energy-efficient Communication

5.1 Overview

Medium Access Control (MAC) Layer plays an important role for energy conservation of the sensor nodes in a WSN [25]. In this chapter, we present a novel design for the MAC Layer for energy-efficient communication between a sensor node and the visiting MDC. Typically, a sensor node's radio is turned "off" for an interval to conserve energy [8]. The interval is known as *sleep-interval*. We present a MAC Layer for the MDC in which the interval is dynamically modified to reduce the delay related to data collection, and at the same time, to save energy of the sensor nodes.

5.2 Communication Layers

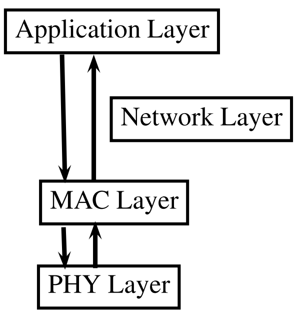


Figure 5.1: Communication between different layers

Typically, there are three layers in the communication module of a Wireless Sensor Network:

Network Layer, Medium Access Control or MAC Layer and Physical Layer. Application Layer runs on top of the communication module and it is responsible for gathering data packets which are generated as a result of sensing activity. In our network scenario, the MDC is supposed to collect data packets directly from each of the sensor node. Therefore, *Network Layer*, which is responsible for discovering adjacent neighbors and maintain various routing information for forwarding, is optional in our scenario. Even, if it is present, we can bypass Network Layer by directly handing over the data packets from the Application Layer to the MAC layer as shown in Figure 5.1 and vice versa. This approach relieves the sensor nodes from all the overhead related to maintaining routing tables and building path to the sink. On the other hand, it does not result in packet loss as all the sensor nodes are covered by the visiting MDC.

5.3 Data Deposition Method

Every sensor node buffers data packets until it comes in contact with the visiting MDC. Then, it can upload or transmit all of the data packets to the MDC that brings those to the sink which is its starting point of the tour. We have designed the mode of communication between the MDC and the target sensor node. Therefore, uploading data is

- quick to minimize the overall PDL, and
- energy-efficient to maximize the life-time of the network

With each Anchor Point programmed into the MDC, there is attached one or more target sensor node. Suppose that, node n_i is tied to Anchor Point a_i. As the MDC reaches the point a_i, at first, it tries to draw the attention of the target node. For this purpose, it sends out a unicast packet of type *RESPOND_NOW* with n_i embedded in the header. As soon as n_i picks up the packet, it responds with the same type of packet with *totalPackets* flag set. For example, if there is the total of 130 data packets in the buffer of the sensor node n_i, its *RESPOND_NOW* packet contains $totalPackets \leftarrow 130$. As soon as the MDC picks up this *RESPOND_NOW* packet from n_i, it sends out unicast packet of type *DATA_REQUEST* with *allowedPackets* field set.

For example, if the MDC has time to collect only 80 data packets from n_i, it sets $allowedPackets \leftarrow 80$ in *DATA_REQUEST* packet. The rest $130 - 80 = 50$ packets is left the buffer of the sensor node for the next visit. In response to this *DATA_REQUEST* packet from MDC, Node n_i starts sending its chosen 80 packets at a stretch. As soon as the MDC has collected a total of *allowedPackets* or a certain time have elapsed (the time required for receiving such packets computed based on the system parameters), it starts for its next Anchor Point. Sometimes, the MDC may bring some instruction or information from the sink for the sensor node, these are also passed over to the sensor node by packets of type *CONTROL_PACKET*. It is sent before the *DATA_REQUEST* packet.

As outlined above, sensor nodes are passive in the communication between those and the MDC in the sense that, they never send out any kind of packets spontaneously but only in response to requests by the MDC. There is no Application-level acknowledgement packet because according to design of communication mode, the shared medium is supposed to be contention-free as only the MDC and the target sensor node take turn in using it. For these reasons, the data uploading to the MDC is quick and energy-efficient for the sensor nodes.

5.4 Adaptive Duty Cycle in MAC Layer

There are many energy-efficient MAC protocols for sensor nodes like S-MAC, T-MAC [8] etc. However, most of these protocols abide by strict time synchronization and data packets are transmitted only at the beginning of each synchronized interval. Thus, the above protocols are not suitable for the wandering MDC; because when it starts its journey from the sink, many sensor nodes are out of its reach. The schedule of those nodes are clearly different from the MDC's. When the MDC comes in their vicinity, those node become out of sync with the MDC. Therefore, no data collection is possible for the MDC.

For the reason stated above, instead of existing MAC protocols, we adopt a duty-cycled CSMA MAC protocol that is simple and energy-efficient.

The periodic interval P (in milisecond) is the same for the sensor nodes and the mobile MDC, so is the percent of this interval λ that the wireless radio will be in listening mode. This is called *duty cycle* of the physical layer. Every sensor node and the MDC maintains these values by MAC layer. The value a is called *stable duty cycle*. During the rest of the time $P(1 - \lambda)$, the radio is in "sleep mode" and does not receive or transmit any signal. When any node or the MDC tries to transmit

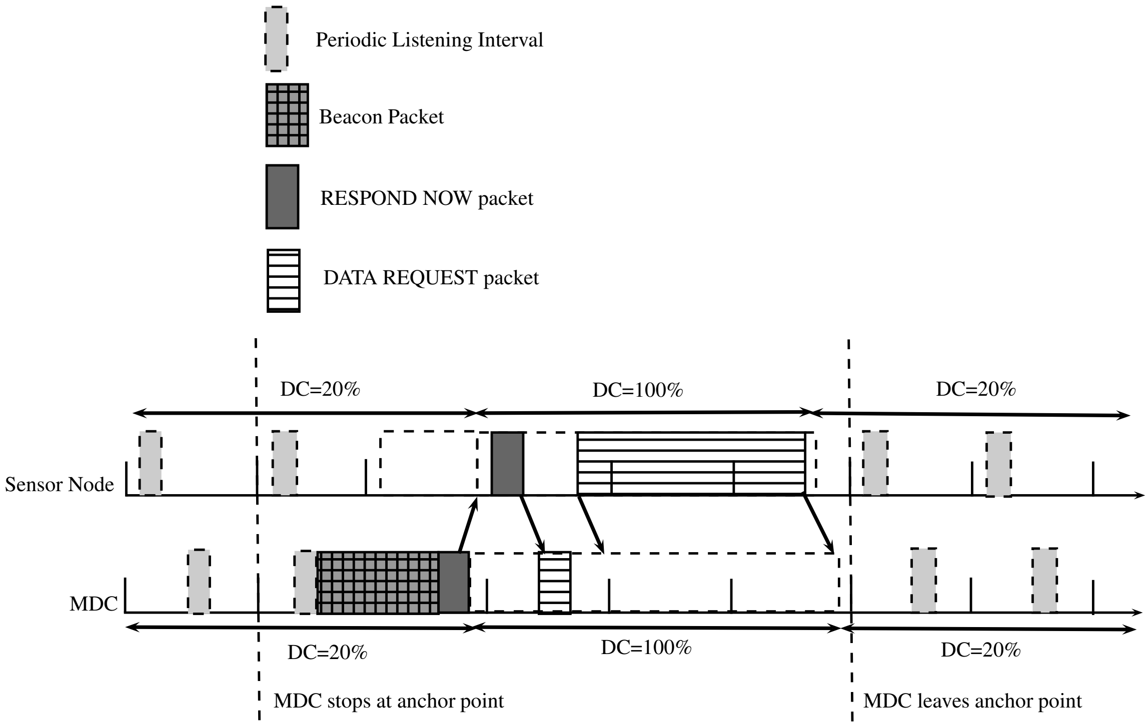


Figure 5.2: Duty cycle modulation in MAC Layer during data packet retrieval by the MDC

any packet, it sends out a packet known as *beacon* packet that takes at least $P(1-\lambda)$ time. When neighboring nodes wake up and picks up this train of beacons, it's radio stays in receiving mode and abandon sleep schedule for the current interval. The node that sent the beacon trains, then starts to send the data packets to the target node. Every other node except the target node then goes to its normal sleep cycle in the next interval. This way, there is no requirement for synchronization of "wake-up" or "sleep" cycles. When the MAC Layer finds the medium busy or detects collision, it starts backs-off timer; otherwise it transmits with 1-persistence.

We integrated changing duty cycle into MAC Layer. During data packet uploading to the MDC, the duty cycle is changed to 100% from stable duty cycle λ and after uploading finishes, duty cycle is restored to λ. This allows quick uploading of the data packets to the MDC and minimizes latency. This is illustrated in Figure 5.2.

Two schedules of MAC are shown in Figure 5.2- one belongs to the MDC and the other to a sensor node. Though *stable duty cycle* is the same for the MDC and the sensor node, the "wake-up" or "sleep" time is not synchronized. MDC initiates communication with the potentially sleeping target sensor node by sending out *beacon trains.* When the sensor node wakes up after a time of $P(1-\lambda)$, it receives the beacon packets and cancel sleep schedule in the current interval. The MDC then sends out the *RESPOND_NOW* packet. As soon as the transmission of this packet is over, MDC's MAC changes duty-cycle to 100%. As soon as the sensor node receives *RESPOND_NOW* packet, its MAC changes the duty cycle to 100%. The sensor node then replies with the number of packets it want to upload to the MDC. MDC sends out the *DATA_REQUEST* packet and after receiving it, the sensor node starts sending out the data packets. After the last data packet is uploaded, it changes its duty cycle to *stable duty cycle* λ. After receiving the last data packet from the sensor node, MDC also changes its duty cycle to λ and starts for its next Anchor Point. If duty cycle is not changed to 100%, packet uploading is delayed by a factor of sleep cycle fraction i.e $(1-\lambda)$. For simplicity and the lack of potential contention, we discard the provision for per-packet acknowledgement. Instead, when collision is detected and/or signal quality is poor, MAC attempts a fixed number of retries.

Chapter 6

Experimental Result

In this chapter, we report the experimental results and provide the analysis. We also present the details of the test bed of experiments.

6.1 Experimental Setup

6.1.1 The Test Bed

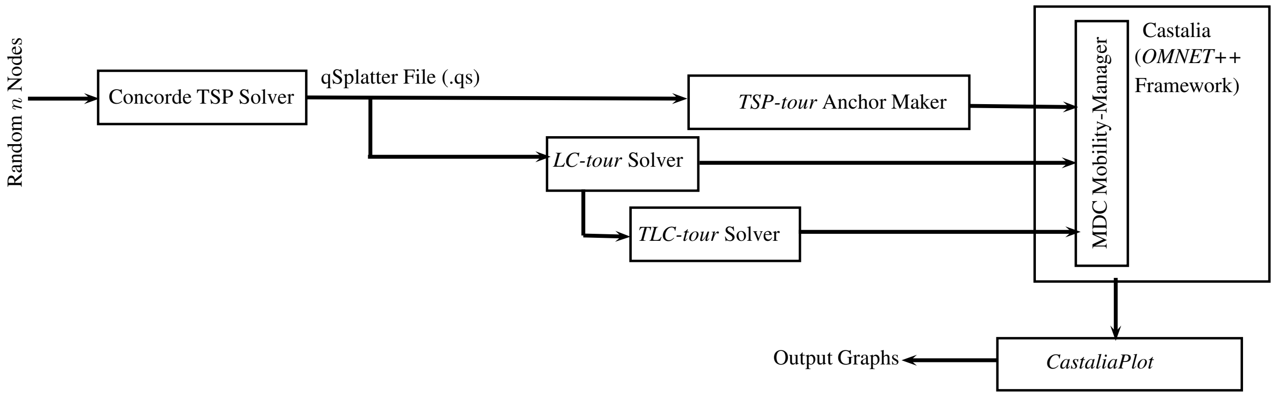


Figure 6.1: Simulation steps

We use *Castalia* 3.2 [47], a very latest and reliable sensor network framework which is run on one of the most widely used network simulator *Omnet++ 4.2.2* [48]. The steps of the experiment are shown in Figure 6.1. We use *Concorde TSP Solver* [41] to find the exact *TSP-tour*. The output *qSplatter* of file containing the *TSP-tour* and node coordinates is fed into the *LC-tour solver* whose output is fed into our *TLC-tour solver*. *LC-tour* and *TLC-tour* produces the *Anchor Points* which are the intermediate points of the tours where the MDC halts to retrieve packets from the nearby nodes.

To compare the performance among *TSP-tour*, *LC-tour* and our *TLC-tour*, we keep the node positions same across scenarios and vary the transmission radius TXR from $2m$ to $32m$. When TXR is only $2m$, the network is sparse and the path lengths for all kinds of tours are of the highest values. When TXR is $32m$, the network becomes dense and the path lengths for all kinds of tours are of the shortest values.

Castalia 3.2 uses realistic radio modeling, and simulates the signal fall due to distance by square-shaped *Path Loss Cell* [47]. The signal reception quality measured by $RSSI$ value is the same in a particular cell. The smaller the path-loss cells are, the more fine-grained is the signal propagation model. However, memory requirement increases drastically with the number of such cells. Therefore, we peg the cell-size (length of a side of the square) with the value of the TXR as follows

$$cellSize = TXR/2 \tag{6.1}$$

Equation 6.1 ensures that there are exactly $2 \times 2 \times 2 = 16$ cells within each circular transmission

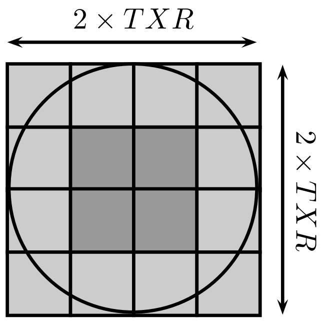


Figure 6.2: A (4×4) grid of path-loss cells which cover the circular transmission range

range, as shown in Figure 6.2. We vary the scenarios just changing the TXR, but keep the radio model of *CC2420* intact. As a result, the energy consumption per packet reception and transmission are the same across scenarios and thus comparisons related to the energy measures across scenarios are also fair. All other network parameters are also similar across the scenarios.
The experimental setup parameters are illustrated in Table 6.1.

6.1.2 Traffic Generation

Parameter Name	**Value/Description**
Simulator Name	*Castalia 3.2* on *OMNET++ 4.2.2*
Operating System	Linux Fedora Core 14
Hardware Type	Processor: Intel Core i5, RAM: 2 GigaByte, Standard Workstation
Simulation Run Time	7200 seconds for each run
Pseudo-Random Number Generator (RNG)	Mersenne Twister (Period length $2^{19937} - 1$)
Total Number of Runs	10

Table 6.1: Experimental setup parameters

We generate the traffic at the sensor nodes randomly. For each value of TXR and for each type of tours (TSP, LC, TLC), we use a common *Pseudo-Random Number Generator* (RNG) for random packet generation in sensor nodes. This RNG's provided by *OMNET++* is *Mersenne Twister* type and has a long period of $2^{19937} - 1$. The event of random packet generation in the simulation is free from the repetition and correlation to other events. The RNG has been used to produce a packet in the interval between 15 seconds and 30 seconds. All output measures are averaged over 10 simulation runs.

We set the total time of each run as 7200 seconds. The MDC set out from the initial point and continuously travels in constant speed (1 meter/second) and complete as many tour as possible in this 2-hour time and gather as many packets as possible from all sensor nodes.

The same set of 10 seeds is used in all tour-types and TXR values for fairness of comparison. We provide the histograms of packet delivery latency for each scenario and for each simulation run in Appendix A. The number of entries in the buckets of the histogram are almost the same across different simulation runs for the similar scenario. We continue each run of the simulation for such a long time (2-hour) that the output measures become independent of the traffic generation pattern.

6.1.3 PHY and MAC Parameters

The parameters used for physical layer and MAC layer are shown in Table 6.2. The widely used *CC2420* radio model is chosen. The data-rate of the radio is set to be 250-kbps. This radio model along with the underlying wireless channel of *Castalia* simulates signal interference, path-loss, cross-

Protocol Layers	Parameter Name	Parameter Value
Physical Layer	Radio Type	CC2420
	Transmitting Power	57.42 miliWatt
	Receiving Power	62 miliWatt
	Data Rate	250 kbps
	Base-band	20 MHz
	Noise-bandwidth	194 MHz
	Sensitivity	-95 dBm
	Idle Power Consumption	1.4 miliWatt
	Modulation Type	Ideal
	PHY-Frame Overhead	6 Byte
MAC Layer	MAC Type	Tunable MAC
	MAC Buffer Size	32 Protocl Data Unit
	Access Type	CSMA
	CS-Persistence	1-persistent
	Delay for Vaid CS	128 mili-second
	Transmission Retries	only 1
	Stable Duty Cycle	0.1
	Listen Interval	10 mili-second
	Back-off Type	Random Interval Drawn From Constant Range
	Back-off Base Value	16 mili-second
	Random offset Time before Retransmission	5 mili-second
	MAC Packet Overhead	9 Byte
	MAC Beacon Frame size	125 Byte
Mobility Controller	Stable Speed	1 meter/second
	Acceleration Type	Instant

Table 6.2: The list of parameters used for Physical and MAC Layers in the experiment

fading and other PHY phenomena present in a shared wireless medium [49].

The parameters used for the MAC layer is also listed in Table 6.2. 1-persistent Carrier Sense Multiple Access (CSMA) is used with only 1 transmission retries as explained in Chapter 5. The sensor nodes modulate their duty cycle from 10% (stable duty-cyle) to 100% when they come in to the contact of the visiting MDC and receive its *RESPOND_NOW* packet. After data transaction with the MDC is over, the sensor nodes reset their duty-cycle to the stable one (10%). To wake up the target sensor node, MDC continuosly sends out beacon frames. The total time length of the beacon trains must be at least equal to the sleep-interval (90%) of the stable duty cycle. Values for *beacon frame size*, *listen interval*, *sleep interval* etc. are chosen carefully to this end.

The MAC Layer has its own timer for *back-off*. When carrier sensing detects the medium as busy, MAC Layer is backed off for some random time drawn from a constant interval (16 milisecond) using a separate RNG. The MAC buffer-size is set to 32 PDU.

Protocol Layers	**Parameter Name**	**Parameter Value**
Network Layer	Address Translation	Node ID (constant)
	Network Packet Overhead	10 Byte
Application Layer (Sensor Node)	Sensor Sampling Type	Random
	Maximum Interval of Sampling	30 second
	Minimum Interval of Sampling	15 second
	Application Buffer Size	120 Application Packets
	Duty Cycle Modulation	Present
	Hibernation After Contact with MDC	10 second
Application Layer (MDC)	Packet Overhead	5 Byte
	Pre-tour Delay	30 second
	RESPOND_NOW Packet Retries	2
	Waiting Time After Sending RESPOND_NOW packet	333 mili-second
	Waiting Time After Sending DATA_REQEUST packet	343 mili-second

Table 6.3: The list of parameters used for Network and Application Layers in the experiment

6.1.4 Network and Application Layer Parameters

The parameters used for Network and Application Layers are shown in Table 6.3. The Application Layer is completely built for our experiment. The Application Layer programs are in the MDC and in the sensor node and these are different. These are known as *MdcApp* and *ResponseApp* respectively. In *MdcApp*, there are provisions for sending *RESPOND_NOW* packet targeting a particular sensor node, collecting packets from the target sensor node based on the *number of packets* field set by the target sensor node in response to the *RESPOND_NOW* packet, calculating the number of packets dropped or not received from the sensor nodes based on the sequence number of the Application Layer packet etc. Another improvisation builds the mobility manager of the MDC so that it can send interrupts to

Application Layer when a particular anchor or halting point is reached and when a tour is complete. Prior to the beginning of a run, this mobility manager known as *MDCMobilityManager* loads the Anchor Points generated by the relevant programs to find *TSP, LC* and *TLC* tours. The MDC sends and receives packets only in halting states. No data packet is sent or received in motion. The acceleration type from halting to motion state is instant. The MDC sets out from the sink, travels along the tour-path, stops at the *anchor points*, collect data from the sensor nodes and return to the sink to deliver packets. A time delay of 10-second is kept for the delivery operation. There are few other timers related to the *waiting time* for a reply packet from the target sensor node in response to *RESPOND_NOW* packet and data packets as explained in Chapter 4. Values for these time windows are also listed in Table 6.3. Since, reply to the *RESPOND_NOW* packet is vital for data transaction in a particular tour, it is sent with two retries. *ResponseApp* generates packets randomly within an interval from 15 seconds to 30 seconds and buffer packets.

6.2 Results and Analysis

6.2.1 Impact on Packet Delivery Latency (PDL)

Packet Delivery Latency (PLD) in brief, is the time difference between the packet generation and the packet delivery to the sink. There is a time-stamp of the packet-creation embedded in the packet header. As MDC completes a single tour, it calculates PDL for each packet using this time-stamp value. This statistics gathered by the MDC per run is represented by PDL histogram shown in the Appendix A. We observe that in those 10 buckets of the histograms, the bucket with the highest packet count is either the 5-th or 6-th one in almost all of the cases. Thus, the skewness of the PDL distribution in those long-running simulations is almost zero, and the average value is very close to the median value. This indicates that, we can reliably compare the central tendency values for the different measures from different kinds of tours of the MDC.

In Figure 6.3, the average PDL is compared for *TSP, LC* and *TLC* tours for different TXR's. The average PDL does not vary much for *TSP-tour* as its path does not change in response to the change in TXR. However, as TXR increases, the network becomes dense and both the LC and *TLC-tour* paths decrease, so does the tour-time of the MDC. As a result, the average PDL also decreases

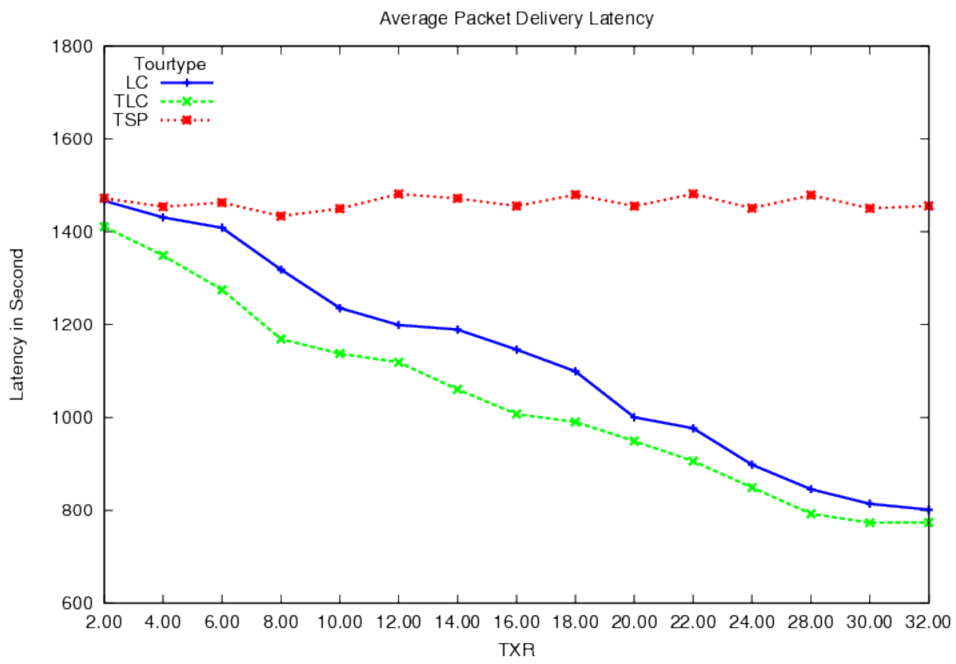


Figure 6.3: The average Packet Delivery Latency vs. TXR

for LC and TLC tours. However, average PDL for *TLC-tour* outperforms that of *LC-tour* by at most 150 seconds and *TSP-tour* by at most 500 seconds.

The comparisons of the maximum PDL for different TXR's for *TSP, LC* and *TLC* tours are shown in Figure 6.4. The maximum PDL decreases for both LC and TLC tours as the network becomes dense. Here also, *TLC-tour* beats *LC-tour* by 200 seconds. This happens because the maximum PDL cannot be larger than the tour time of the MDC which is roughly proportional to the tour-length, and *TLC-tour* has the shortest tour-length among the three tours. *TSP-tour* is not affected by the change of TXR, so maximum PDL does not vary much with the change in TXR.

6.2.2 Impact on Packet Delivery Rate (PDR)

Since we generate traffic randomly, it is important to measure the *throughput* which we define as the number of packets delivered to the sink by the MDC per second. In Figure 6.5 the throughput has been plotted for *TSP, LC* and *TLC-tour* for different TXR's. Because of the random traffic, throughput does not consistently change for varied TXR. However, the throughput in *TLC-tour* is always the highest whereas the throughput in *TSP-tour* is always the lowest among the three kinds

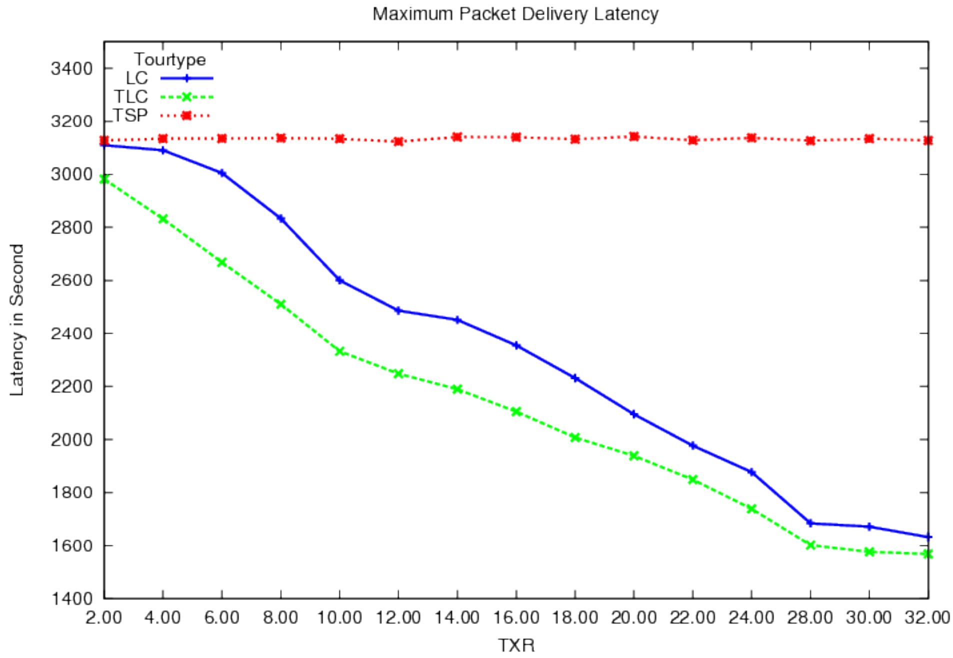


Figure 6.4: The maximum Packet Delivery Latency vs. TXR

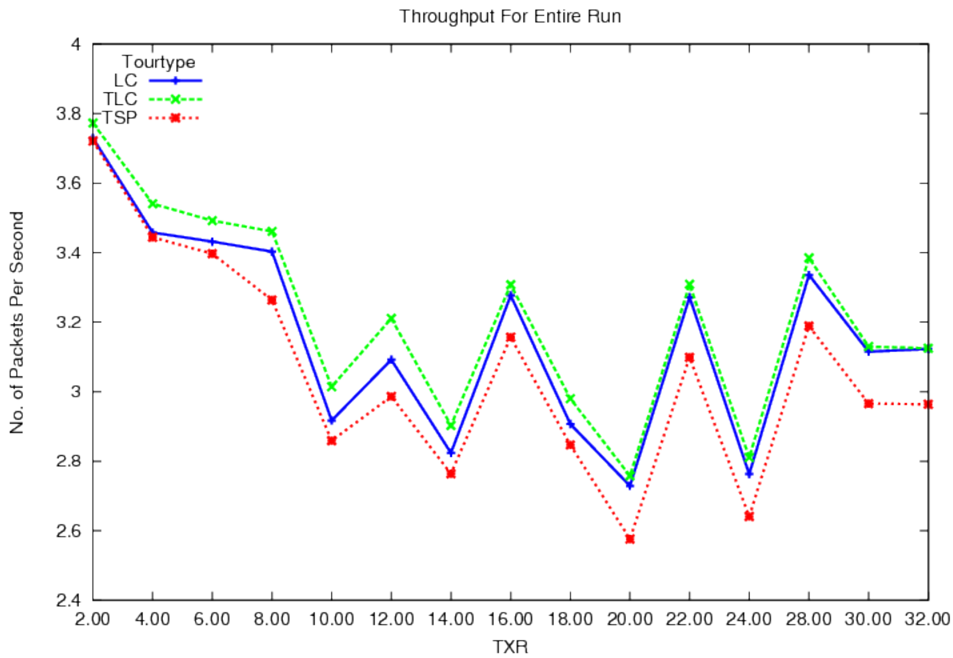


Figure 6.5: Impact on Packet Delivery Rate (PDR)

of tours across all scenarios. The throughput for *TLC-tour* is higher than *LC-tour* by a significant margin (as much as 0.15 packets/second). As the network becomes dense, the path savings by *TLC-tour* is minimal and the length of *LC-tour* gets decreased; therefore throughputs are almost the same but still better than *TSP-tour*.

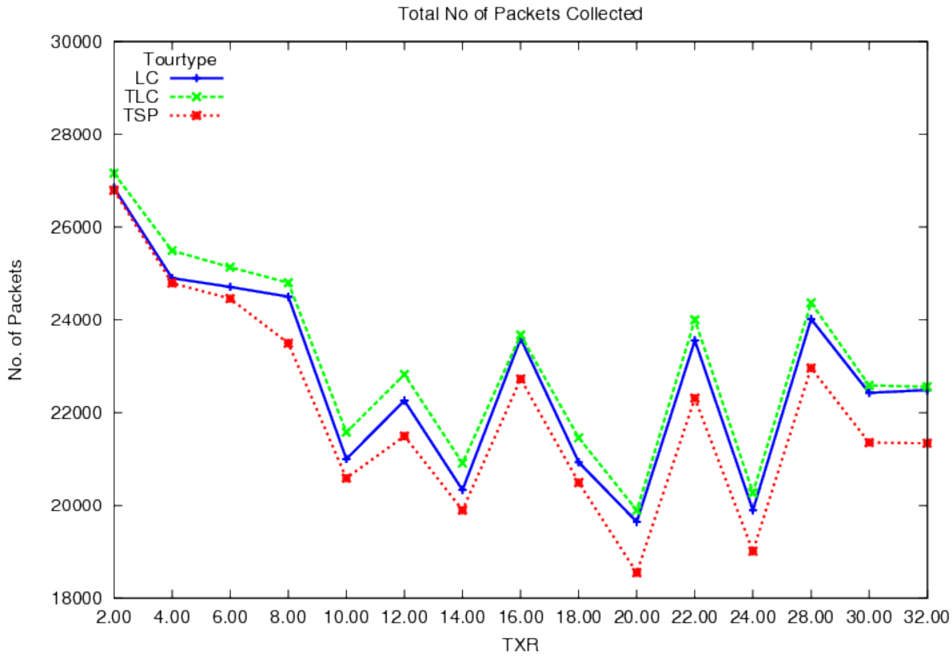


Figure 6.6: The impact on the total number of packets collected by the MDC

In Figure 6.6, the total number of packets collected by the MDC is compared for three types of tours. Here, we find that throughput is directly proportional to the total number of packets collected. Here, the MDC in *TLC- tour* collects the highest number of packets (500 more packets than *LC-tour* and 800 more packets than *TSP-tour*).

In Figure 6.7, the total packets dropped by nodes due to buffer constraint has been plotted. Here, the *TLC-tour* has significant upper-hand than the other two types of tours. When TXR is small and the network is sparse, the path savings by TLC is significant. The resulting tour-time is also smaller. Thus, the MDC can make more number of tours and visit the sensor nodes more frequently than the other two types of tours. This significantly reduces the number of packets dropped at sensor nodes. As the network becomes dense, the spread of values for total number of packets dropped between TLC and *LC-tour* decreases but *TLC-tour* always has the least value.

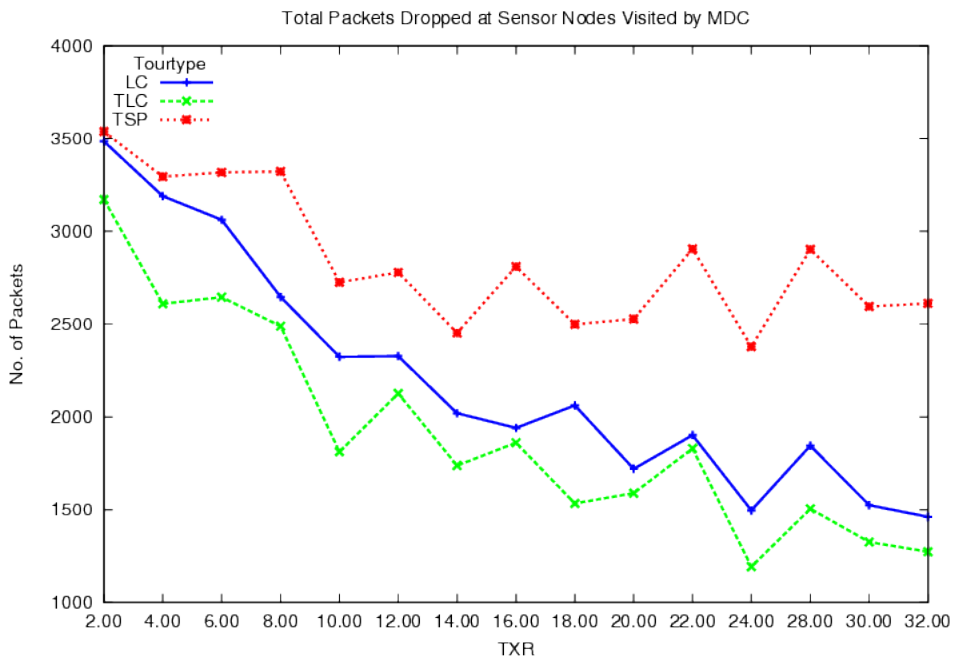


Figure 6.7: The impact on the total number of packets dropped by nodes

6.2.3 Impact on Tour Time

In Figure 6.8, the average tour time of the MDC for *TSP, LC* and *TLC-tour* are compared. Since, the path-length of the MDC is the shortest among these three types of tours, the tour-time of *TLC-tour* is also the smallest. However, as the TXR increases and the network becomes dense, the difference in the path-lengths of *TLC-tour* and *LC-tour* becomes smaller. For *TSP-tour*, the path-length is unaffected as the MDC must visit the exact position of the nodes every time.

For the similar reason stated in the above paragraph, the number of tours covered by the MDC is always the highest in *TLC-tour* and the lowest in *TSP-tour* as shown in Figure 6.9. Here, the tour count is shown as the percentage of the total path length of a single tour. For example, for $TXR = 8.00m$, the MDC covers about 550% of the single tour that is roughly equal to five complete tours plus 1/2 of a single tour. As TXR increases, the path lengths for both of the LC and *TLC-tour* decrease and the tour count increases accordingly. For *TSP-tour*, this value is invariant to the change in TXR.

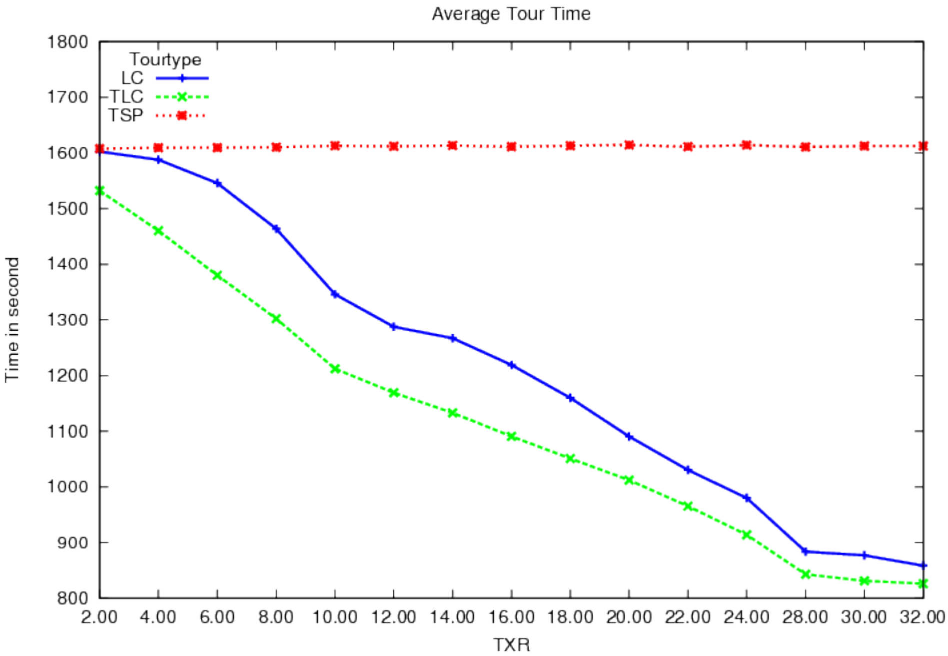


Figure 6.8: The average tour time of the MDC

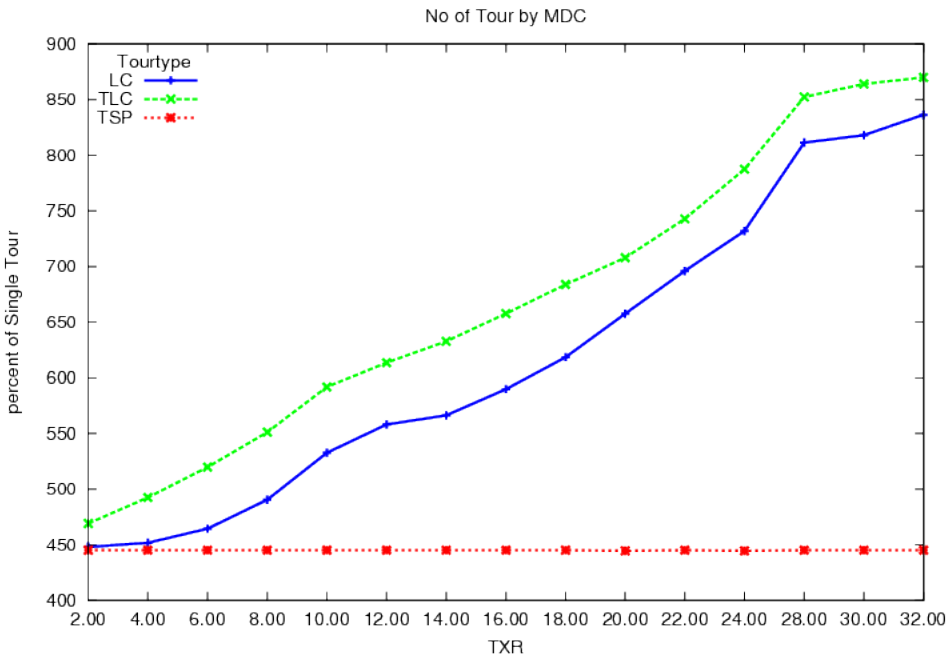


Figure 6.9: The total number of tours by the MDC

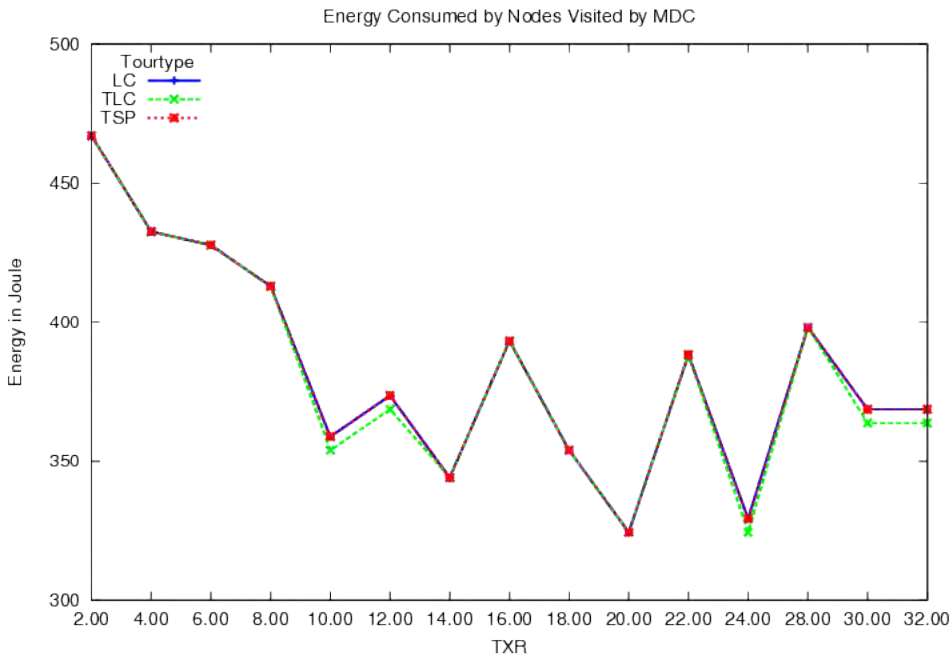


Figure 6.10: Average Energy Consumed by the Sensor Nodes vs. TXR

6.2.4 Impact on Energy Consumption

In Figure 6.10, the average energy consumed by the sensor nodes are compared for *TSP-tour, LC-tour* and *TLC-tour*. Since, in all of the cases, there is no packet forwarding by sensor nodes but only direct sending to the visiting MDC, the variations in the average energy consumed among *TSP-tour, LC-tour* and *TLC-tour* are the minimal. In fact, the spread is so small that the three lines almost overlap with each other. However, due to the randomness of traffic pattern and differences in tour-time of the MDC, there is a considerable variation among different scenarios. It is to be noted that we set the energy consumption for sensing and generating a single packet as 10% of that for the transmission of a single packet. The idle power consumption by Radio is $16mJ$ whereas the power for *radio-TX* and *radio-RX* are $57.42mJ$ and $62mJ$ respectively. Therefore, the energy consumption is dominated by radio transmission and reception activities or the number of packets transmitted or received by the sensor nodes. The average energy consumption pattern shown in Figure 6.10 matches the graph pattern of Figure 6.6 where the total number of packets sent to the MDC by the sensor nodes is shown. Also to be noted that, we changed the TXR by varying the property of the wireless medium but not by varying the radio model or power levels. Therefore, this comparison among scenarios are fair and valid.

6.2.5 Overhead of Computation

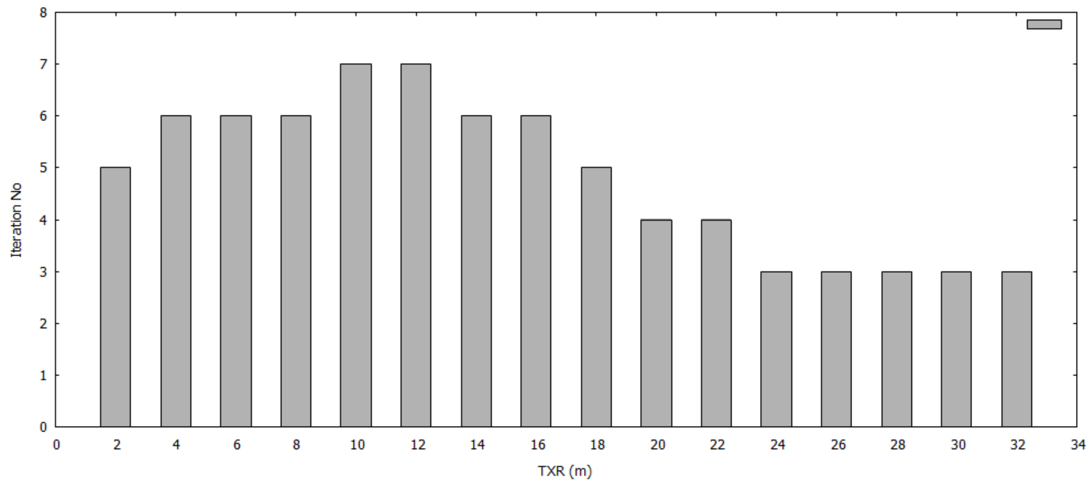


Figure 6.11: No of iterations vs. TXR (Total nodes 100)

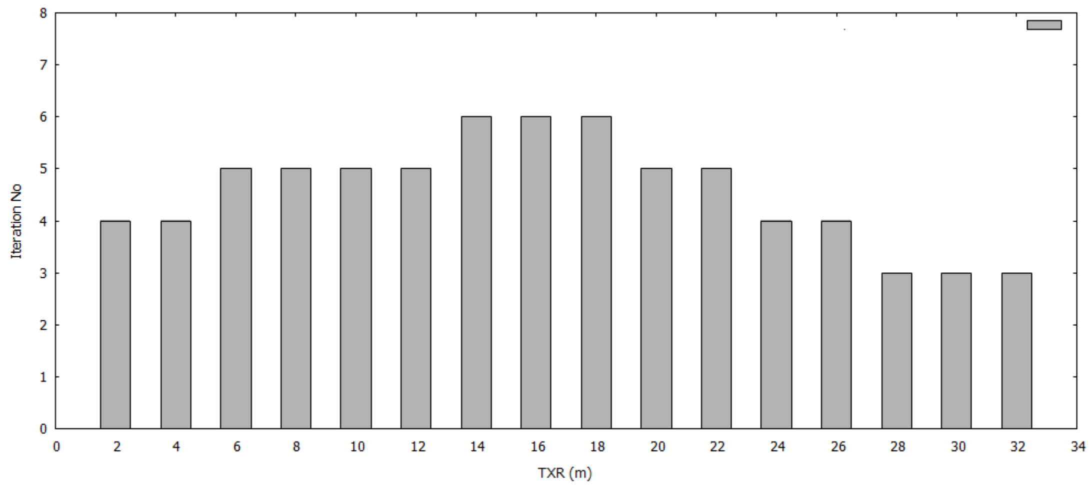


Figure 6.12: No of iterations vs. TXR (Total nodes 75)

In Figure 6.11, the number of iterations after which the *path-gain* falls below 5% is plotted against TXR for the scenario with 100 nodes. For example, when TXR is $10m$, the *path-gain* falls below 5% after the 7th Iteration, which is the maximum for all the scearios. This value is very small compared to the number of nodes in the scenario. Therefore, our algorithm converges quite fast irrespective of the types of the network. Similar plots are shown in Figure 6.12 and 6.13 when the number of nodes

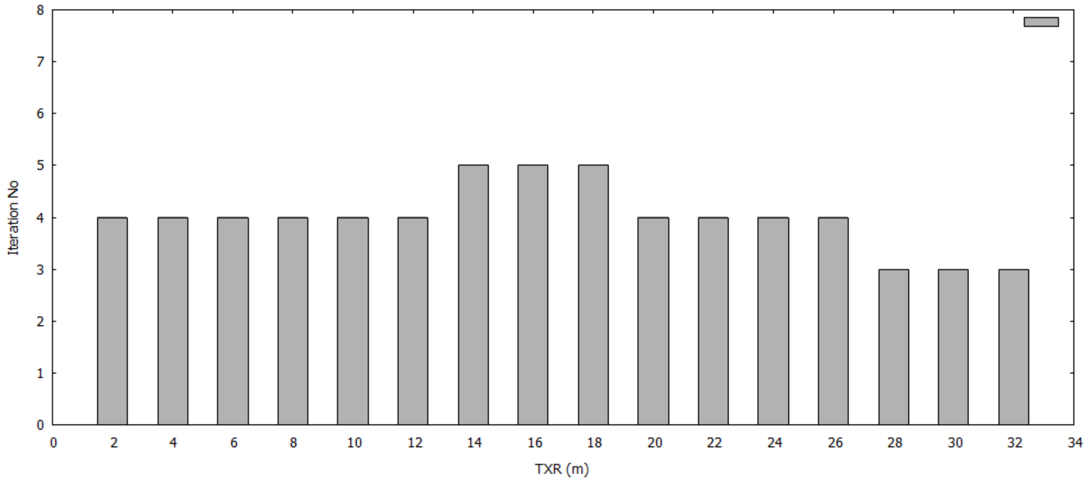


Figure 6.13: No of iterations vs. TXR (Total nodes 50)

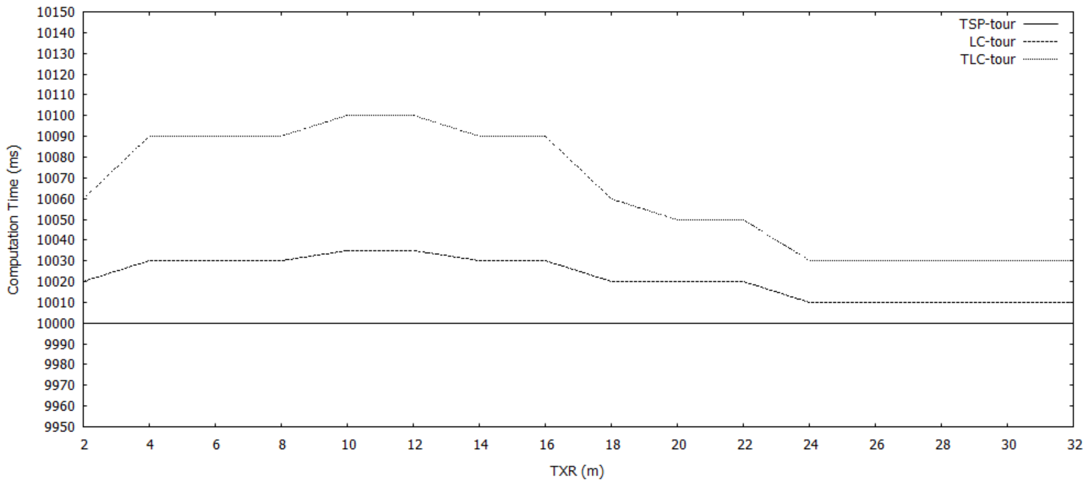


Figure 6.14: Computation Time vs. TXR

are 75 and 50 respectively.

In Figure 6.14, computation time for different tours are plotted. As shown in this figure, the marginal overhead of computation time for both *LC-tour* and *TLC-tour* are small compared to that of the exact *TSP-tour*. Therefore, our method does not impose any significant overhead of computation.

6.3 Discussion

Now, we can summarize our findings as follows:

1. The shortcut *TLC-tour* ensures the lower Packet Delivery Latency (See Figure 6.3).
2. The packet drop-rate by the sensor nodes is, on the average, lower in *TLC-tour* as evident from Figure 6.7
3. The tour time and the maximum Packet Delivery Latency in *TLC-tour* are the minimum compared to *LC-tour* and *TSP-tour* as evident from Figure 6.9 and 6.4 respectively
4. *TLC-tour* ensures the higher throughput as evident from Figure 6.5
5. The energy consumption and thus the network life-time in *TLC-tour* is as good as those in *LC* and *TSP-tour* as evident from Figure 6.10

The summary stated above points out that, *TLC-tour* should be always used instead of *LC* or *TSP-tour* since there exists algorithms which run in polynomial time. We can remember that we derive *LC-tour* from *TSP-tour* and *TLC-tour* from *LC-tour*.

Chapter 7

Conclusion

In this chapter, we provide our research summary. Meeting the latency requirement by using the *Mobile Data Collector* or MDC in the WSN depends on how fast the MDC can complete its tour, which in turn depends on how short the tour-length is. The advantages of using the MDC has been already stated in Chapter 1. To achieve these advantages in our application of the WSN, we minimize the *data delivery latency* which is a major downside of using the MDC. In Chapter 3, we prove that shortening the path of the MDC is the only viable option to minimize the latency. Therefore, research in shortening the tour of the MDC is of utmost significance for energy-efficient data collection in a WSN.

7.1 Summary

We provide a simple data collection method based on *TSP-tour*. In our method, to communicate with a sensor node, the MDC does not have to visit the exact location of the sensor node; instead visiting any point of the transmission region suffices for the communication. We adopt the *disk* model of the transmission range whose radius is denoted by TXR. The value of the TXR typically ranges from 5 to 50 meters. This distance adds up to the length of the *TSP-tour* for each visited node. In our method, we save this distance by making a shortcut of the tour. On one hand, the MDC does not have to visit each node. Though we have used the similar disk model to represent the sensor nodes arbitrary shapes of sensor transmission area can be applicable. The MDC can halt at a sensor node and collect data from its neighborhood. For example, for elliptical shape, the eccentricity and the focci are required to compute the intersections between the edge and the ellipse. Our method can also be extended to the 3-dimension. In that case, the third coordinate or z-coordinate is required for each point. This may be helpful for aerial or underwater MDC.

We also test the performance of our algorithm using realistic test beds. The objective is to measure to what degree latency has been minimized as a result of shortening the tour-path. We compare the performance measures for the *TLC-tour* derived by our method with those of the *TSP-tour* and *LC-tour*. Our *TLC-tour* is the shortest of the three types of tours under comparison.

From the experimental results, among all tours, we find that the average packet delivery latency is the smallest in the case of *TLC-tour*. The *TLC-tour* has the shortest path, therefore, the MDC takes the minimum time to complete a tour on the path compared to *TSP-tour* and *LC-tour*. Since packet delivery latency is directly proportional to the tour-time as explained in Chapter 3, it is logical that *TLC-tour* ensures the minimum data delivery latency among the three types of tours. For the same reason, maximum packet delivery latency is also the minimum for *TLC-tour*.

Because of the minimum tour-time in the case of *TLC-tour*, the MDC visits the nodes most frequently than the other two types of tours. Consequently, the time interval between two successive visits by the MDC is the smallest and the least number of packets are dropped by the sensor nodes in the case of *TLC-tour*. Therefore, the packet drop-rate is the smallest in the case of *TLC-tour*.

Since the packet drop rate and the tour-time are the smallest in the case of *TLC-tour*, the MDC can collect the most number of packets in that tour. Therefore, the throughput is the highest in the case of *TLC-tour* compared to the other two tours.

In our strategy for data collection, no nodes forward packets of the neighboring nodes. The MDC collects packets directly from each node. Therefore, the m-lifetime of the WSN in the case of the *TLC-tour* is the same as those of *TSP-tour* and *LC-tour*. In other word, like *TSP-tour*, *TLC-tour* is the most energy-efficient tour.

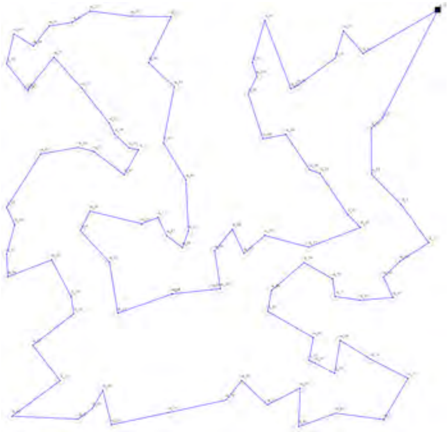
7.2 Future Research Extension

For future work, we plan to test our method in real scenarios using sensor motes [6, 5] and *iRobot* [9] used as a low-cost MDC. We also plan to compute path for multiple MDC's. In our approach, we do not ration the time allocated by the MDC for a particular sensor node for data collection. Rather, the MDC collects all the packets buffered in the sensor node currently in its contact. In future, we plan to develop a framework by which the MDC can learn, from its initial periodic tours, some parameters like- how much time to allocate for a sensor node for data collection and which nodes to visit and which ones to skip in a particular tour.

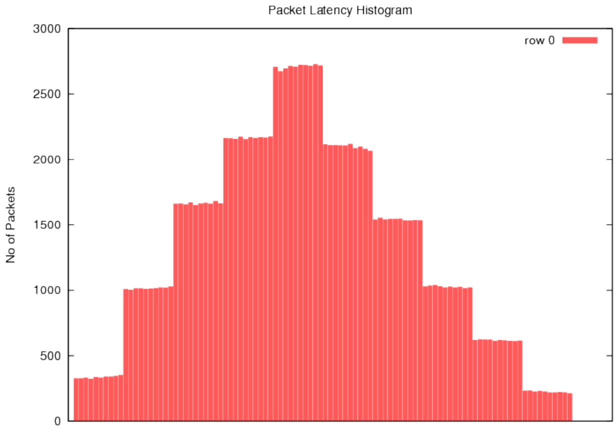
Appendix A

Result Per Simulation Scenario

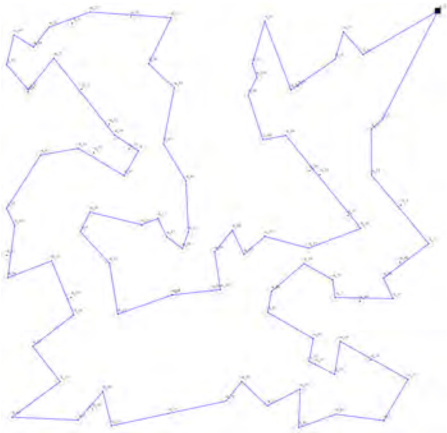
In this appendix, we present the histograms of latencies in different scenarios of our simulation runs for the three types of tours. We use 10 buckets for each histogram. In all the cases, the maximum counts occur in either the 5^{th} or the 6^{th} buckets. The skewness of the histogram is almost zero. Therefore, the average value is very close to the median value and we can reliably compare the values of the *average packet delivery latency* among different types of tours



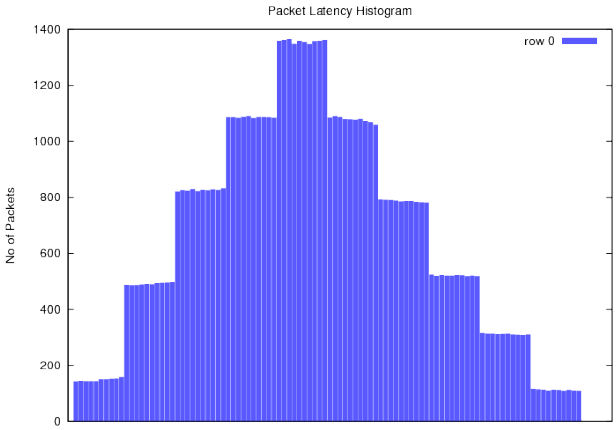
(a) TSP Tour for TXR=2.00



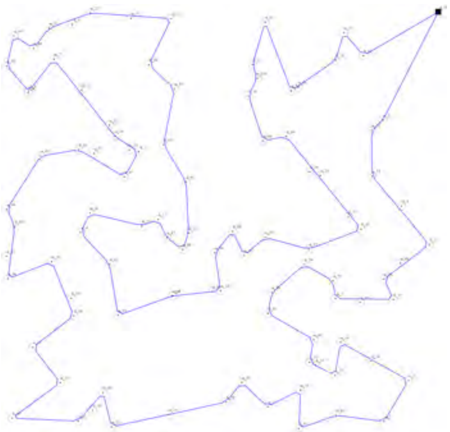
(b) Latency Histogram for TSP Tour of TXR=2.00



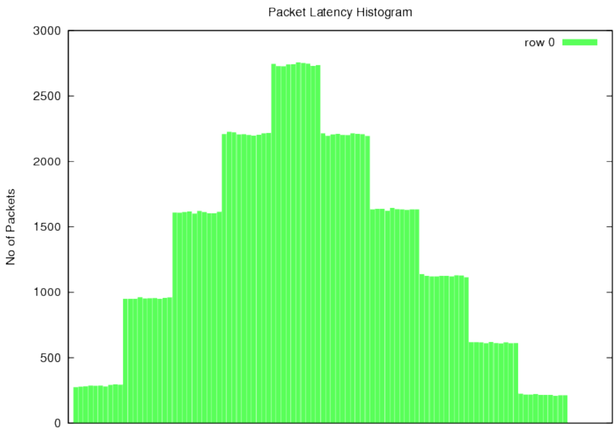
(c) LC Tour for TXR=2.00



(d) Latency Histogram for LC Tour of TXR=2.00

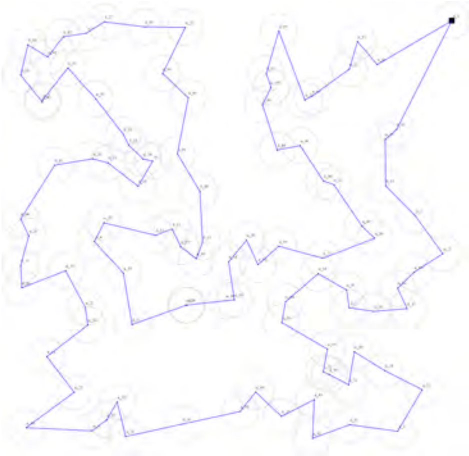


(e) TLC Tour for TXR=2.00

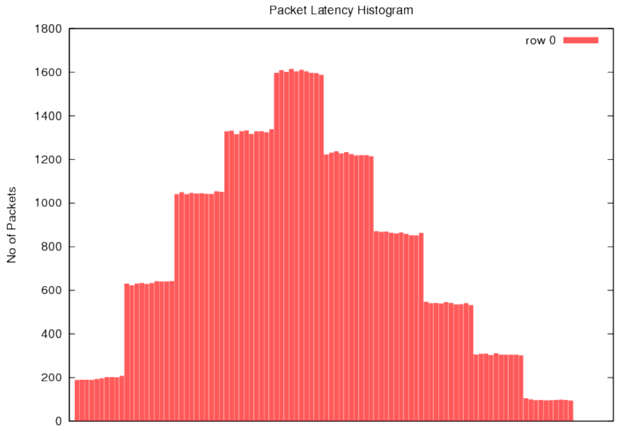


(f) Latency Histogram for LC Tour of TXR=2.00

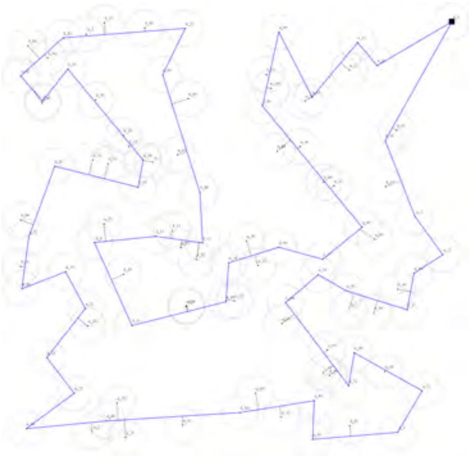
Figure A.1: Comparison of Latency Histogram for TXR=2.00m



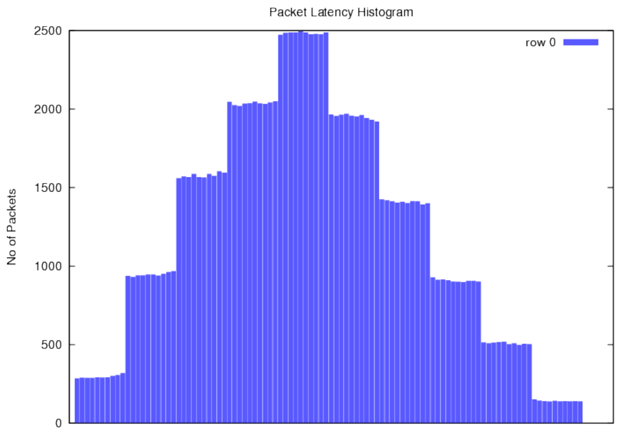
(a) TSP Tour for TXR=8.00



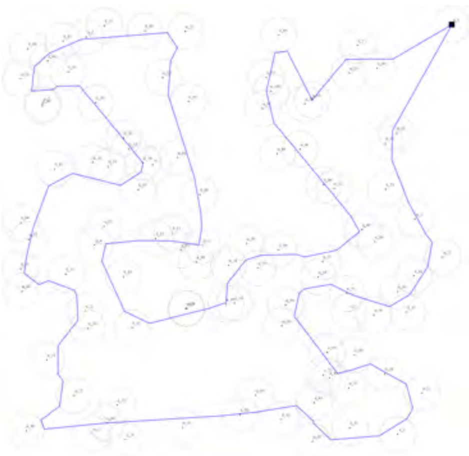
(b) Latency Histogram for TSP Tour of TXR=8.00



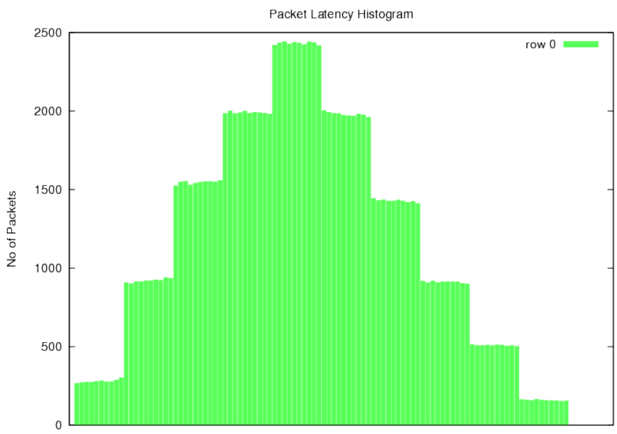
(c) LC Tour for TXR=8.00



(d) Latency Histogram for LC Tour of TXR=8.00



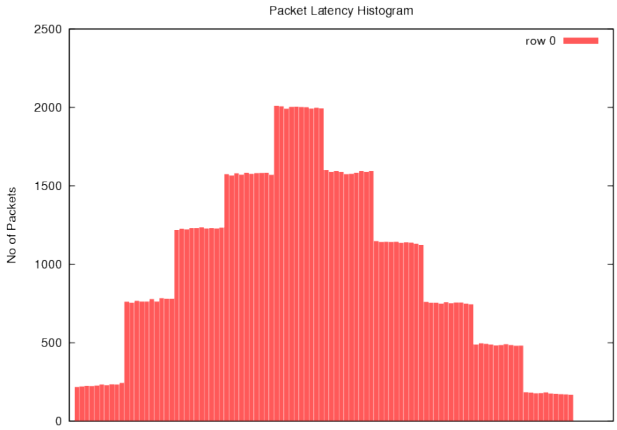
(e) TLC Tour for TXR=8.00



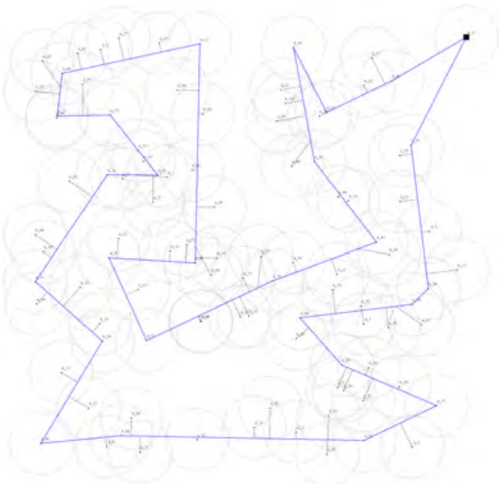
(f) Latency Histogram for LC Tour of TXR=8.00

Figure A.2: Comparison of Latency Histogram for TXR=8.00m

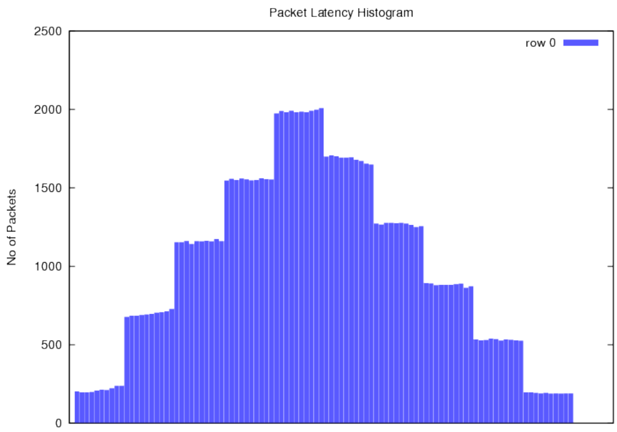
(a) TSP Tour for TXR=14.00



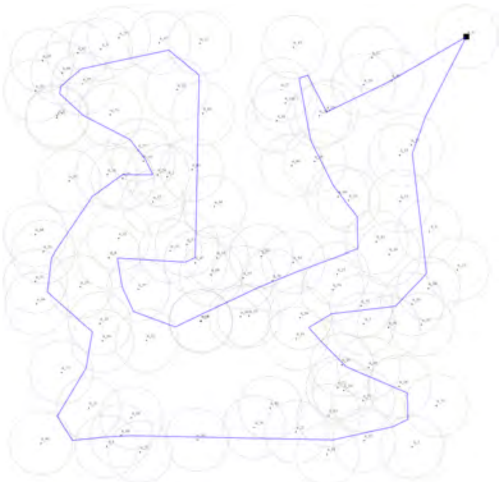
(b) Latency Histogram for TSP Tour of TXR=14.00



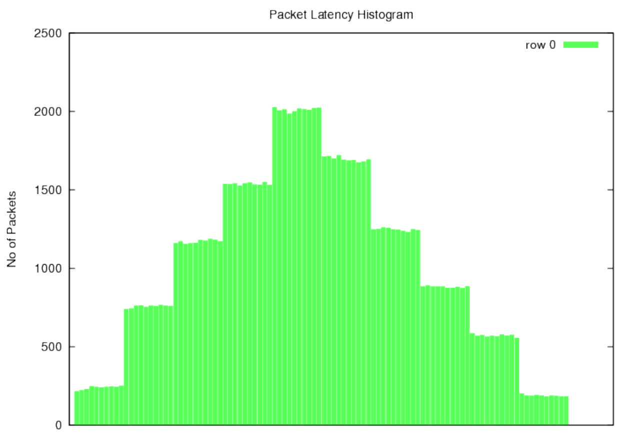
(c) LC Tour for TXR=14.00



(d) Latency Histogram for LC Tour of TXR=14.00



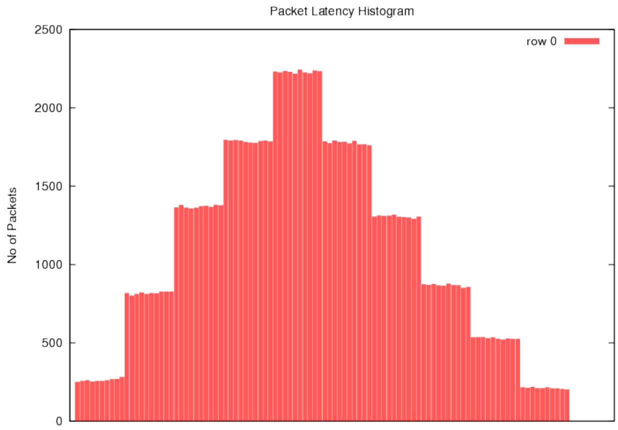
(e) TLC Tour for TXR=14.00



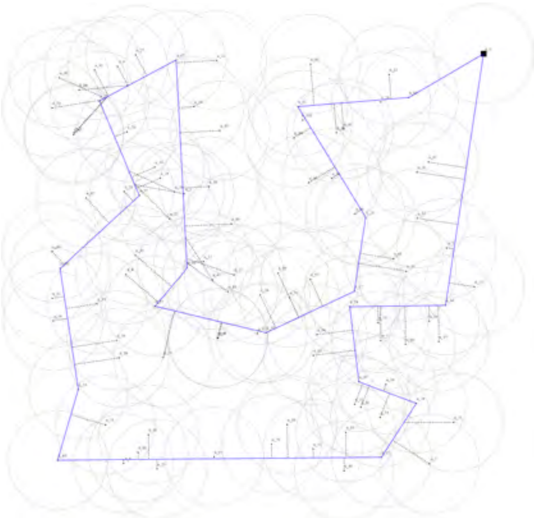
(f) Latency Histogram for LC Tour of TXR=14.00

Figure A.3: Comparison of Latency Histogram for TXR=14.00m

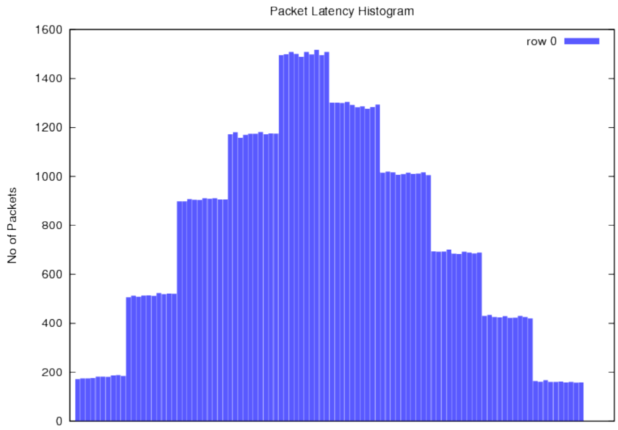
(a) TSP Tour for TXR=22.00



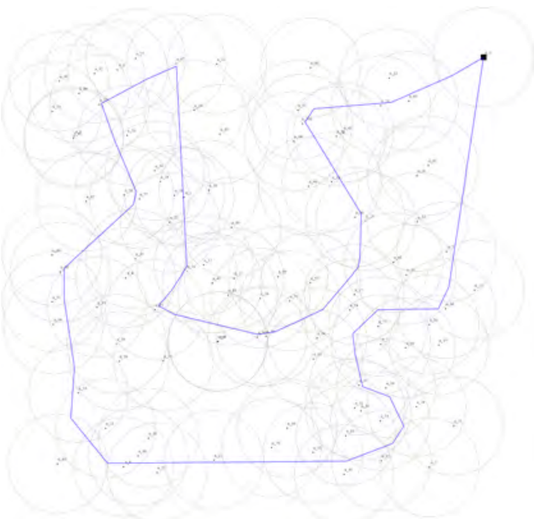
(b) Latency Histogram for TSP Tour of TXR=22.00



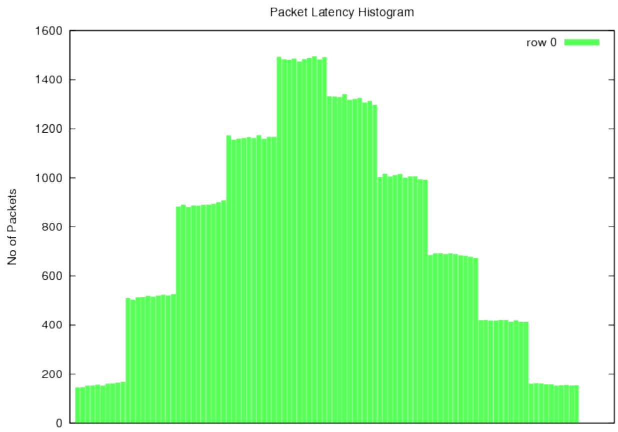
(c) LC Tour for TXR=22.00



(d) Latency Histogram for LC Tour of TXR=22.00

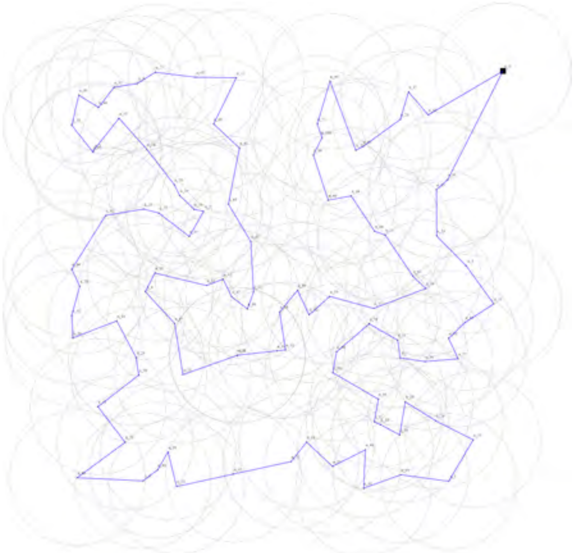


(e) TLC Tour for TXR=22.00

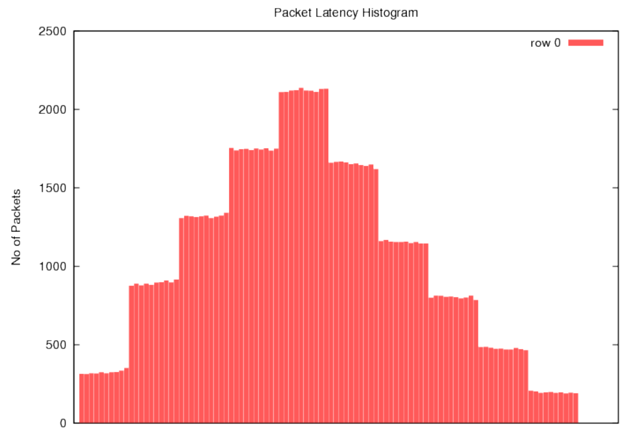


(f) Latency Histogram for LC Tour of TXR=22.00

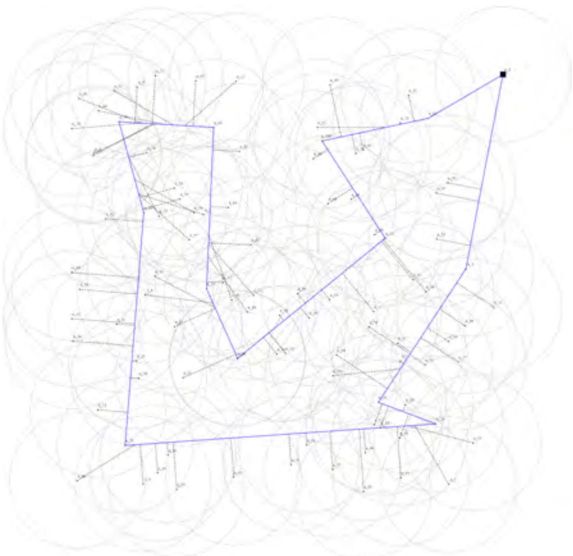
Figure A.4: Comparison of Latency Histogram for TXR=22.00m



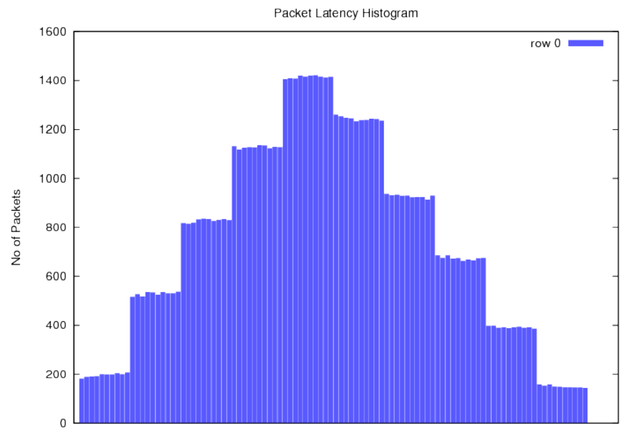
(a) TSP Tour for TXR=30.00



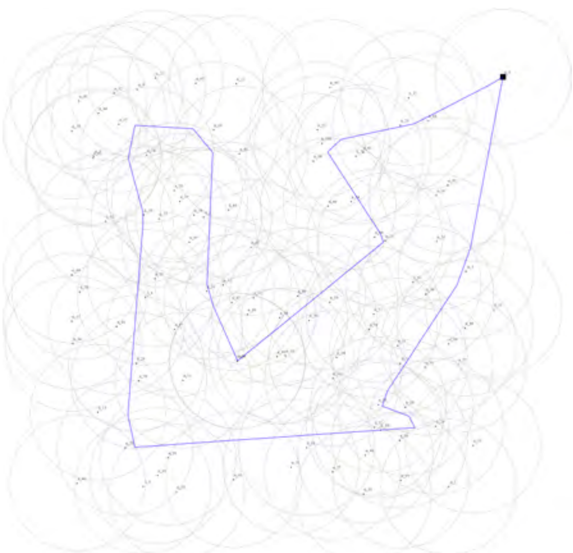
(b) Latency Histogram for TSP Tour of TXR=30.00



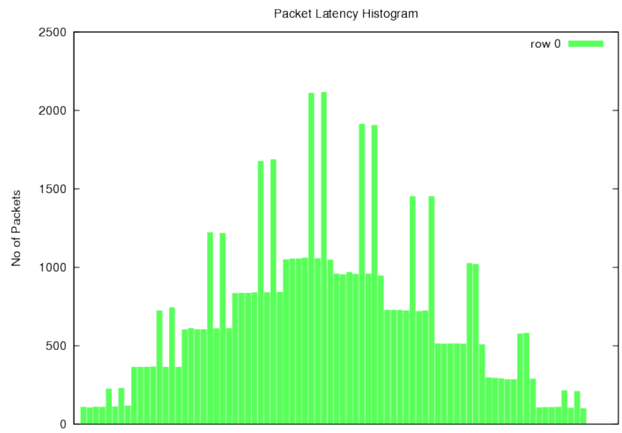
(c) LC Tour for TXR=30.00



(d) Latency Histogram for LC Tour of TXR=30.00



(e) TLC Tour for TXR=30.00



(f) Latency Histogram for LC Tour of TXR=30.00

Figure A.5: Comparison of Latency Histogram for TXR=30.00m

Bibliography

[1] M. Di Francesco, S. K. Das, and G. Anastasi, "Data collection in wireless sensor networks with mobile elements: A survey," *ACM Transaction on Sensor Networks*, vol. 8, pp. 7:1–7:31, August 2011. [Online]. Available: http://doi.acm.org/10.1145/1993042.1993049

[2] Y. Yuan and Y. Peng, "Racetrack: an approximation algorithm for the mobile sink routing problem," in *Proceedings of the 9th international conference on Ad-hoc, mobile and wireless networks*, ser. ADHOC-NOW'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 135–148. [Online]. Available: http://dl.acm.org/citation.cfm?id=1881991.1882002

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Comput. Netw.*, vol. 38, no. 4, pp. 393–422, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1016/S1389-1286(01)00302-4

[4] R. Rajagopalan and P. Varshney, "Data-aggregation techniques in sensor networks: a survey," *Communications Surveys Tutorials, IEEE*, vol. 8, no. 4, pp. 48 –63, quarter 2006.

[5] "Xbow sensor network solutions," http://www.moog-crossbow.com.

[6] "Wasp-mote," http://www.libelium.com/products/waspmote.

[7] J. Li and P. Mohapatra, "Analytical modeling and mitigation techniques for the energy hole problem in sensor networks," *Pervasive Mob. Comput.*, vol. 3, no. 3, pp. 233–254, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.pmcj.2006.11.001

[8] P. Huang, L. Xiao, S. Soltani, M. Mutka, and N. Xi, "The evolution of mac protocols in wireless sensor networks: A survey," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1 –20, 2012.

[9] "irobots: Robots that make a difference," http://www.irobot.com.

[10] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet," in *10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2002)*, 2002, pp. 96–107.

[11] Z. J. Haas and T. Small, "A new networking model for biological applications of ad hoc sensor networks." *IEEE/ACM Transactions on Networking (TON)*, vol. 14, pp. 27–40, February 2006.

[12] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, and X. Zheng, "The rise of people-centric sensing," *IEEE Internet Computing*, vol. 12, pp. 12–21, 2008.

[13] T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibdas, A. Kansal, S. Madden, and J. Reich, "Mobiscopes for human spaces," *IEEE Pervasive Computing*, vol. 6, pp. 20–29, April–June 2007.

[14] Y. Li, C. S. Chen, Y.-Q. Song, and Z. Wang, "Real-time QoS support in wireless sensor networks: a survey," in *7th IFAC International Conference on Fieldbuses & Networks in Industrial & Embedded Systems - FeT'2007*, Toulouse, France, 2007. [Online]. Available: http://hal.inria.fr/inria-00188265

[15] M. Khabbaz, C. Assi, and W. Fawaz, "Disruption-tolerant networking: A comprehensive survey on recent developments and persisting challenges," *Communications Surveys Tutorials, IEEE*, vol. 14, no. 2, pp. 607 –640, quarter 2012.

[16] W. Zhao and M. H. Ammar, "Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks," in *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, ser. FTDCS '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 308–. [Online]. Available: http://dl.acm.org/citation.cfm?id=795675.797105

[17] R. C. Shah, S. Roy, S. Jain, and W. Brunette, "Data mules: Modeling a three-tier architecture for sparse sensor networks," in *IN IEEE SNPA WORKSHOP*, 2003, pp. 30–41.

[18] W. Zhao, M. Ammar, and E. 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*, ser. MobiHoc '04. New York, NY, USA: ACM, 2004, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/989459.989483

[19] X. Zhang, G. Neglia, J. Kurose, and D. Towsley, "Performance modeling of epidemic routing," *Comput. Netw.*, vol. 51, no. 10, pp. 2867–2891, Jul. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2006.11.028

[20] T. Srinidhi, G. Sridhar, and V. Sridhar, "Topology management in ad hoc mobile wireless networks," in *Proceedings of the 24thIEEE International Real-Time Systems Symposium (RTSS 2003)*, 2003.

[21] C. Tang and P. K. McKinley, "Energy optimization under informed mobility," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, pp. 947–962, September 2006. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2006.122

[22] S. Yang, M. Li, and J. Wu, "Smart: A scan-based movement-assisted sensor deployment method in wireless sensor networks," in *In Proc. of IEEE INFOCOM*, 2005, pp. 2313–2324.

[23] G. Dini, M. Pelagatti, and I. M. Savino, "An algorithm for reconnecting wireless sensor network partitions," in *5th European conference on Wireless sensor networks (EWSN)*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 253–267. [Online]. Available: http://dl.acm.org/citation.cfm?id=1786014.1786036

[24] F. El-Moukaddem, E. Torng, G. Xing, and S. Kulkarni, "Mobile relay configuration in data-intensive wireless sensor networks," in *IEEE 6th International Conference on Mobile Adhoc and Sensor Systems (MASS)*. Macao, China: IEEE, 2009, pp. 80–89.

[25] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Netw.*, vol. 7, pp. 537–568, May 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1465737.1465996

[26] M. Conti, L. Pelusi, A. Passarella, and G. Anastasi, *Adaptation and Cross Layer Design in Wireless Networks, chapter: Mobile-relay Forwarding in Opportunistic Network.* CRC Press, 2008.

[27] Z. M. Wang, S. Basagni, E. Melachrinoudis, and C. Petrioli, "Exploiting sink mobility for maximizing sensor networks lifetime," in *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Volume 09*, ser. HICSS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 287.1–. [Online]. Available: http://dx.doi.org/10.1109/HICSS.2005.259

[28] J. Rao, T. Wu, and S. Biswas, "Network-assisted sink navigation protocols for data harvesting in sensor networks," in *WCNC 2008, IEEE Wireless Communications & Networking Conference, March 31 2008 - April 3 2008, Las Vegas, Nevada, USA, Conference Proceedings.* IEEE, 2008, pp. 2887–2892.

[29] J. Rao and S. Biswas, "Network-assisted sink navigation for distributed data gathering: Stability and delay-energy trade-offs," *Computer Communications*, vol. 33, pp. 160–175, February 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2009.08.009

[30] G. Anastasi, E. Borgia, M. Conti, and E. Gregori, "A hybrid adaptive protocol for reliable data delivery in wsns with multiple mobile sinks," *Comput. J.*, vol. 54, no. 2, pp. 213–229, 2011.

[31] S. Jain, R. C. Shah, W. Brunette, G. Borriello, and S. Roy, "Exploiting mobility for energy efficient data collection in wireless sensor networks," *Mob. Netw. Appl.*, vol. 11, pp. 327–339, June 2006. [Online]. Available: http://dx.doi.org/10.1007/s11036-006-5186-9

[32] H. Jun, W. Zhao, M. H. Ammar, E. W. Zegura, and C. Lee, "Trading latency for energy in wireless ad hoc networks using message ferrying," in *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, ser. PERCOMW '05.

Washington, DC, USA: IEEE Computer Society, 2005, pp. 220–225. [Online]. Available: http://dx.doi.org/10.1109/PERCOMW.2005.88

[33] D. B. Johnson, D. A. Maltz, and J. Broch, "Ad hoc networking," in *Ad hoc networking.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001, ch. DSR: the dynamic source routing protocol for multihop wireless ad hoc networks, pp. 139–172. [Online]. Available: http://dl.acm.org/citation.cfm?id=374547.374552

[34] N. Vlajic and D. Stevanovic, "Sink mobility in wireless sensor networks: a (mis)match between theory and practice," in *International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly.* Leipzig, Germany: ACM, 2009, pp. 386–393. [Online]. Available: http://doi.acm.org/10.1145/1582379.1582464

[35] A. Chakrabarti, A. Sabharwal, and B. Aazhang, "Using predictable observer mobility for power efficient design of sensor networks," in *Proceedings of the 2nd international conference on Information processing in sensor networks*, ser. IPSN'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 129–145. [Online]. Available: http://dl.acm.org/citation.cfm?id=1765991.1766001

[36] A. A. Somasundara, A. Kansal, D. D. Jea, D. Estrin, and M. B. Srivastava, "Controllably mobile infrastructure for low energy embedded networks," *IEEE Transactions on Mobile Computing*, vol. 5, pp. 958–973, August 2006. [Online]. Available: http://dx.doi.org/10.1109/TMC.2006.109

[37] S. Poduri and G. S. Sukhatme, "Achieving connectivity through coalescence in mobile robot networks," in *ROBOCOMM*, 2007, p. 4.

[38] M. Ma and Y. Yang, "Sencar: An energy-efficient data gathering mechanism for large-scale multihop sensor networks," *IEEE Transaction on Parallel and Distributed System*, vol. 18, pp. 1476–1488, October 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1313042.1313066

[39] R. Sugihara and R. K. Gupta, "Improving the data delivery latency in sensor networks with controlled mobility," in *4th IEEE international conference on Distributed Computing in Sensor Systems.* Berlin, Heidelberg: Springer-Verlag, 2008, pp. 386–399. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69170-9_26

[40] ——, "Path planning of data mules in sensor networks," *ACM Transaction on Senson Networks*, vol. 8, no. 1, pp. 1:1–1:27, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/1993042.1993043

[41] "Concorde tsp solver," http://www.tsp.gatech.edu/concorde.html.

[42] D. Bhadauria, O. Tekdas, and V. Isler, "Robotic data mules for collecting data over sparse sensor fields," *J. Field Robot.*, vol. 28, no. 3, pp. 388–404, May 2011. [Online]. Available: http://dx.doi.org/10.1002/rob.20384

[43] A. Dumitrescu and J. S. B. Mitchell, "Approximation algorithms for tsp with neighborhoods in the plane," *J. Algorithms*, vol. 48, no. 1, pp. 135–159, 2003.

[44] G. N. Frederickson, M. S. Hecht, and C. E. Kim, "Approximation algorithms for some routing problems," *SIAM J. Comput.*, vol. 7, no. 2, pp. 178–193, 1978.

[45] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007.

[46] "Wikipedia: Triangle inequality," http://en.wikipedia.org/wiki/Triangle_inequality.

[47] "Castalia wsn simulator framwork for omnet++," http://www.castelia.org.

[48] "Omnet++ simulator," http://www.omnetpp.org/.

[49] D. Pediaditakis, Y. Tselishchev, and A. Boulis, "Performance and scalability evaluation of the castalia wireless sensor network simulator," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTools '10. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010, pp. 53:1–53:6. [Online]. Available: http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8727