There is but One God, His name is Truth.

– Guru Nanak, 1468-1539 AD

University of Alberta

Efficient and Reliable In-Network Query Processing in Wireless Sensor Networks

by

Baljeet Singh Malhotra

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Fall 2010
Edmonton, Alberta

Examining Committee

Dr. Ioanis Nikolaidis, Dept. of Computing Science, University of Alberta

Dr. Mario A. Nascimento, Dept. of Computing Science, University of Alberta

Dr. Janelle Harms, Dept. of Computing Science, University of Alberta

Dr. Davood Rafiei, Dept. of Computing Science, University of Alberta

Dr. Petr Musilek, Dept. of Electrical and Computer Engineering, University of Alberta

Dr. Alexandros Labrinidis, Dept. of Computer Science, University of Pittsburgh

To
Preeti and Ishaan,
You are my everything.

Abstract

The Wireless Sensor Networks (WSNs) have emerged as a new paradigm for collecting and processing data from physical environments, such as wild life sanctuaries, large warehouses, and battlefields. Users can access sensor data by issuing queries over the network, e.g., to find what are the 10 highest temperature values in the network. Typically, a WSN operates by constructing a logical topology, such as a spanning tree, built on top of the physical topology of the network. The constructed logical topology is then used to disseminate queries in the network, and also to process and return the results of such queries back to the user. A major challenge in this context is prolonging the network's lifetime that mainly depends on the energy cost of data communication via wireless radios, which is known to be very expensive as compared to the cost of data processing within the network.

In this research, we investigate some of the core problems that deal with the different aspects of in-network query processing in WSNs. In that context, we propose an efficient filtering based algorithm for the top-k query processing in WSNs. Through a systematic study of the top-k query processing in WSNs we propose several solutions in this thesis, which are applicable not only to the top-k queries, but also to in-network query processing problems in general. Specifically, we consider broadcasting and convergecasting, which are two basic operations that are required by many in-network query processing solutions. Scheduling broadcasting and convergecasting is another problem that is important for energy efficiency in WSNs. Failure of communication links, which are common in WSNs, is yet another important issue that needs to be addressed.

In this research, we take a holistic approach to deal with the above problems while processing the top-k queries in WSNs. To this end, the thesis makes several contributions. In particular, our proposed solutions include new logical topologies, scheduling algorithms, and an overall sophisticated communication framework, which allows to process the top-k queries efficiently and with increased reliability. Extensive simulation studies reveal that our solutions are not only energy efficient, saving up to 50% of the energy cost as compared to the current state-of-the-art solutions, but they are also robust to link failures.

Acknowledgements

I am very much indebted to my supervisors, Drs. Ioanis Nikolaidis and Mario A. Nascimento, for their erudite supervision throughout my PhD research work that produced this thesis. I am also very thankful to the examining committee members, Drs. Alexandros Labrinidis, Janelle Harms, Davood Rafiei, and Petr Musilek, for providing me with their valuable suggestions that further improved this thesis.

I extend a special thanks to Dr. Alex Aravind for his assistance and interest in my research work. I am very thankful to Drs. Guohui Lin and Jörg Sander with whom I took courses that helped me in many ways to complete this research. I would like to thank Edith Drummond for her continuing support throughout my research journey at the University of Alberta. Thanks are also due to the teaching and academic staff in the Department of Computing Science that helped me in many ways.

I want to thank Drs. Nilanjan Ray, Reza Sherkat, Baochun Bai, Yuxi Li and Shoudong Zou for their general guidance. I must thank my colleagues at the University of Alberta, Chen Liu, Benyamin Shimony, Nicholas Boers, Sunil Ravinder, Israat Haque, Abhishek Srivastava, Majid Ghanbarinejad, and Ryan Vogt for engaging me in useful research discussions. A warm thanks to Pirooz, Sharmin, Baidya and Amit for their help.

I am thankful to my brother Gurjit and my sisters Happy, Honey and Daisy Didi for their love and support. I will be forever thankful to my most loving brother Navjot and most loving friends Manav and Neeraj, who have influenced my life in so many ways. Lastly but most importantly I cannot thank enough my parents, wife, and son without whom my existence is meaningless. Their love and support will always be a driving force behind me.

The research work carried out in this thesis is financially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada, Graduate Scholarship from iCORE Alberta, Walter H Johns Graduate Fellowship, Queen Elizabeth II Graduate Scholarship, Entrance Scholarship from the Faculty of Graduate Studies at the University of Alberta, Professional Development Grant from the Graduate Student's Association at the University of Alberta, and a Travel Fellowship by VLDB Endowment.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ACK	Acknowledgment
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
BFS	Breadth First Search
BIPST	Biased Shortest Path Tree
BSPT	Balanced Shortest Path Tree
CDS	Connected Dominating Set
DSS	Decision Support System
DST	Dominating Set Tree
F-Node	Filtering Node
MAC	Medium Access Control
MCDS	Minimum Connected Dominating Set
MDST	Minimum Dominating Set Tree
MIMO	Multiple Input and Multiple Output
MIS	Maximal Independent Set
MST	Minimum Steiner Tree
PER	Packet Error Rate
RIBS	Redundancy Injection in Broadcast Scheduling
RICS	Redundancy Injection in Convergecast Scheduling
RFID	Radio Frequency Identification
SPT	Shortest Path Tree
TDMA	Time Division Multiple Access
TM-Node	Temporal Monitoring Node
UDG	Unit Disk Graph
WIRES	Weighted Incremental Scheduling for Convergecast
WISH	Weighted Incremental Scheduling for Broadcast
WSNs	Wireless Sensor Networks

List of Symbols

N	Number of nodes
R	Number of rounds
i	A node
j	A round
r	The sink/root node
P_i	The logical parent of node i
ξ_i	A set containing logical children of node i
N_i	A set containing neighbors of node i
$\mathcal{N}$	A set containing neighbors of N nodes
$\mathcal{P}$	A set containing logical parents of N nodes
$\mathcal{S}$	A schedule for N nodes
S	A subset of nodes
$V(S)$	A set containing values of nodes in S
t_i	Time slot allocated to node i
$v_{i,j}$	A value observed by node i during round j
τ	A threshold value
k	Number of values
Q	A set containing pairs $(i, v_{i,j})$
G	A communication graph representing a network of N node
V	A set containing the N nodes of G
E	A set containing the edges of G
Ψ	Network density
ω	Transmission range
γ	Probability of change in node's value
δ	Percentage of change in node's value
L	Length of a side of a square area
R_{rx}	Reception cost
R_{tx}	Transmission cost
R_c	Ratio of reception to transmission cost

Chapter 1

Introduction

1.1 Wireless Sensor Networks

A wireless sensor network (WSN) is a collection of sensor nodes that are also equipped with computing and communication capabilities. Nodes have on-board radio transceivers through which they can communicate with other nodes, thus creating a wireless network. Through sensors these nodes capture data from the physical environment, which are then stored and possibly processed for extracting useful information. Though these nodes can be equipped with different kinds of sensors to perform various sensing activities, they are generally constrained by limited memory, processing power, and communication channel capacity. Typically the nodes operate on batteries; hence they have a limited energy reserve and therefore energy conservation is one of the design goals in WSNs.

In the past few years WSNs have grown rapidly in their capabilities, e.g., nodes with a low-power 32 bit PXA271 XScale processor with 32MB of RAM and 32 MB of Flash memory, an integrated 802.15.4 radio with a built-in 2.4GHz antenna are now available commercially [18]. The way these networks are beginning to be deployed in research and the commercial sphere [71], it is not unreasonable to expect that in the next 10-15 years a vast amount of information gathered by widely deployed WSNs will be accessible over the Internet [63]. This trend favors the integration of the existing Internet with our physical world to create new interesting applications. Before listing some of these applications we review a generic architecture of a WSN and its most important characteristics.

1.1.1 Architecture

Due to specific objectives of various applications, WSNs do not have a fixed "one-size-fits-all" architecture. As surveyed in [36], the architecture of WSNs drastically varies at a node level as well as at the network level. At the node level, physical dimension, storage

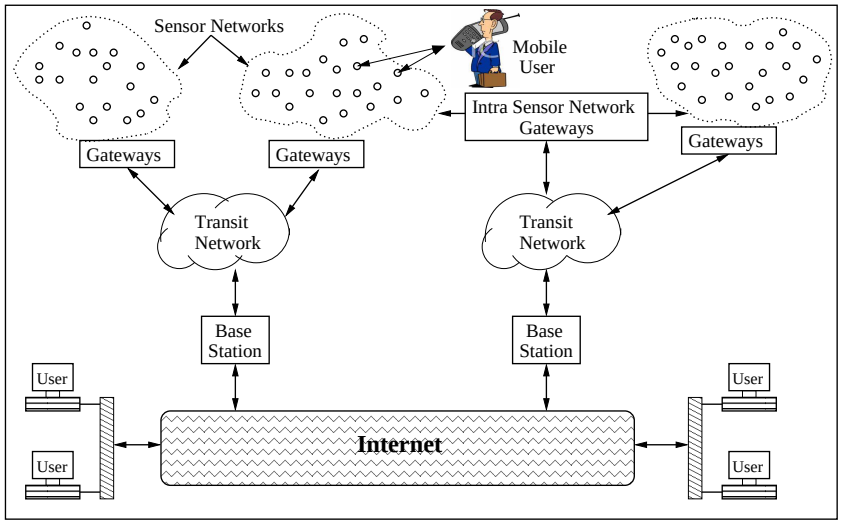


Figure 1.1: A WSN architecture.

capacity, and computational and communication power, are some of the important design considerations. At the network level, the nodes' organization and their communication strategies on a collective basis influence the architecture. However, a common desirable characteristic across all WSNs is low power consumption.

In general, we would like WSNs to be integrated with the existing wireless or wired networks, e.g., the Internet. Figure 1.1 represents a typical design of a WSN architecture. Multiple sensor networks, possibly at different geographical locations, can be setup to monitor areas of interest. Heterogeneous sensor networks may communicate with each other using *intra* sensor network *gateways*. A user may move in a sensor network area to *inquire* individual nodes directly, or sensor networks may have a base station where data from individual nodes can be gathered and processed. Base stations may be connected to the Internet and users at various locations may access the information gathered by the nodes.

1.1.2 Characteristics

WSNs have been characterized according to several parameters like node deployment, node capabilities, applications, energy and communication constraints, etc. [2, 3, 36]. Some of those general characteristics include:

- *Ad Hoc Deployment*: Nodes are generally designed to be deployed in an arbitrary fashion. An extreme example of such deployment is where the nodes are dropped from an airplane onto a geographical region of interest.

- *Dynamic Topology*: Because of the unreliable wireless communication at certain times and the failure of individual nodes due to depletion of their energy, the network topology may change unpredictably and randomly at any point in time.

- *Application Specific*: It is very likely that sensor nodes are designed for specific applications. That means the functionality of nodes will highly depend upon the type of applications for which the nodes were designed.

- *Energy Constrained*: In most cases, nodes in a network rely upon a limited supply of energy from their batteries. Energy consumption of individual nodes may account for the overall lifetime of the network as failure of some nodes may leave the network disconnected and less useful.

- *Bandwidth Constrained*: Sensor nodes will primarily be dependent on their wireless radios for communicating data and/or control messages. In a typical situation nodes share the limited bandwidth of the available communication channels.

- *Self-Reconfiguration*: Due to limited external (user) involvement, WSNs are expected to have self-reconfiguration capabilities. The tasks of self-reconfiguration and networking may mostly depend on nodes' knowledge about their relative positioning with respect to their neighbors.

- *Multi-Hop Communication*: Due to the power constraint of wireless radios and also to restrict the communication interference, nodes have limited transmission range. In this situation, two nodes that are not in each other's transmission range, may use multi-hop communication, i.e., they may use *intermediate* nodes to communicate with each other.

1.1.3 Applications

WSNs may play a crucial role around us. Surveillance, tracking and smart spaces are some of the important applications of WSNs. We list some of the popular applications next.

- *Military Applications*: Military applications are one of the promising areas in which WSNs are being explored on a large scale. Such applications include tracking of moving objects [12, 25, 67, 74], classification of ground vehicles based on their acoustic signals [11, 21, 47], monitoring of hostile environments in a battlefield [42], and so on. Deployment of sensor nodes in a hostile environment may reduce human injuries

and other costs. Nodes can be dropped from a plane over a vast geographical region to detect harmful and dangerous materials. Tracking of tanks and other vehicles in a war zone may provide the observer with better strategic decisions.

- *Urban Applications*: Sensor nodes are typically being deployed to solve urban problems like traffic congestion, vehicular parking, security and health care [10, 65]. Nodes can be deployed along the busy highways for route information and traffic diversions in case of accidents. Sensor nodes can also be deployed within the vehicles to collect and exchange useful information as they cross each other while moving along the roads/highways. Parking management is another application where sensor nodes can be used for effective parking services in busy urban places [8].

- *Industrial Applications* : Tracking inventory through sensor nodes in a warehouse is another application which has generated interest in retail and other related industries [62]. Based on the information available with the nodes deployed in a large warehouse inventory items can be managed efficiently. Intelligent forklifts have also been designed using low-cost sensors [53]. RFID systems [35] that are widely used in industrial applications can be thought as forms of low capability WSN systems.

- *Environmental Studies*: Nodes capable of measuring variations in temperature, humidity, pressure, etc., can be very useful in environmental health monitoring systems [27]. For example, nodes can be deployed for an early warning system to check the spread of forest fires. Habitat monitoring is one such application in which WSNs have been experimented with success [45]. Environmental studies that involve visits to regions of harsh weather can benefit from WSNs. Nodes in such regions can be deployed to collect data over a period of time without involving human experts. Remotely accessing the data may help in reducing the operational cost of environmental studies. Also, WSNs are preferred to avoid human interference with the natural habitat, especially in cases where continuous study is required and hence frequent visits to the area. This way WSNs may reduce the negative side effects of such studies, and at the same time, are capable of providing useful information about the inhabitants in the deployed region. Towards that end there are several approaches proposed in the literature for data collection and query processing in WSNs [17, 33, 34, 43, 44, 59].

  Next, we present a more detailed discussion on query processing in WSNs, which is the main research topic of this thesis.

In a typical situation in WSNs, nodes perform the tasks of sensing, processing, data storage and reporting of useful data according to the objectives of an application. Depending on the application's objectives nodes can pro-actively process the captured data within the network for taking appropriate actions, such as informing other nodes in the network of an event occurrence or sending an alert message to an observer. Another option is for users to issue queries over the network to collect data at a particular node, the so-called *sink*, for further processing to extract useful information. Towards that end nodes must collaborate with each other in a distributed way, which comes at an increased cost of communication required to exchange data and/or control messages. One of the main research challenges here is to reduce the amount of data and the number of messages being transmitted by the nodes in order to reduce their energy consumption, which is crucial for prolonging the network lifetime. Primarily due to this reason energy efficient data processing techniques have received a considerable attention from the research community [17, 43, 46, 48, 58, 59, 60].

A major source of energy consumption in WSNs is the radio component of the nodes, which makes data communication very expensive as compared to the cost of data sensing and processing [54]. Because in most cases nodes are battery operated, with a limited supply of energy, lowering energy consumption by reducing the volume of transmitted data is the most important goal of this thesis. It is worth noting that the radios consume significantly less energy during *sleep* mode as compared to their *transmission* or *reception* mode [28]. Due to this reason it may be beneficial for the nodes to switch off their radios whenever they are neither transmitting nor receiving. In this thesis, apart from our efforts on reducing the volume of transmitted data, we also pay attention to the scheduling solutions, which can allow the nodes to sleep as long as they can to further reduce their energy consumption.

1.2 In-network Query Processing

In-network processing of sensor data is a basic technique used by many applications in which computations are performed by nodes "locally" with the hope that less information will need to be transmitted in order to reduce the communication overhead. That is, in-network processing is about performing *local computations* on nodes within the network in order for the volume of transmitted data to be as small as possible. The main problem of in-network data processing that we consider in this research is how to disseminate queries in the network and collect data of interest at a particular node, i.e., the sink node. The data

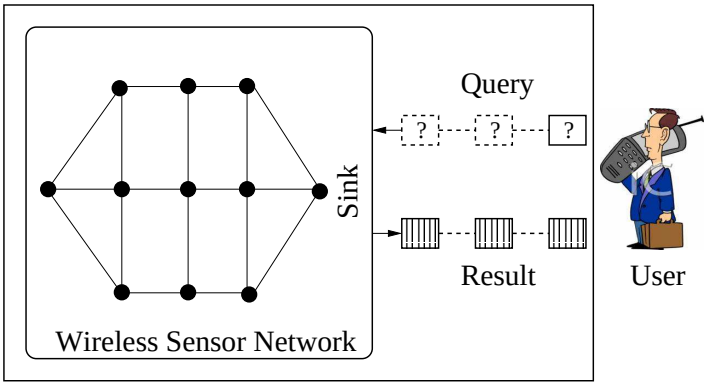


Figure 1.2: A WSN based Decision Support System.

of interest is defined by the users through the semantics of the queries.

In a typical situation a user initiates in-network query processing by issuing queries to the sink, e.g., to find what are the 10 highest temperature values in the network, or which nodes have detected a particular type of tank in a battlefield. In this thesis we consider continuous queries that seek data (query answer) periodically, i.e., a query is issued only once, but it is *executed* multiple times for collecting data periodically as defined by the user. Basically a WSN operates as an element of a Decision Support System that guides the decision-making activities of individuals or organizations.

Figure 1.2 shows the interaction between a user and a WSN. The sink propagates the query issued by the user into the network and the nodes respond appropriately. The nodes reply back to the sink, if necessary, while making local decisions, i.e., the query is processed in-network. Finally, the sink responds to the user by returning the result of the query back to the user. Essentially, we have two phases of in-network query processing that are involved here: *dissemination* and *data collection* phases, which both might require multi-hop communication. In this context of in-network query processing scenario as depicted in Figure 1.2, this thesis considers the following core research problems:

- How to disseminate the query from the sink to every other node in the network. The objective here is to reduce the number of messages used for query dissemination and hence to reduce the energy consumption of the nodes. It is trivial to understand that any solution for query dissemination can also be used for any message that needs to be disseminated from the sink to the nodes.

- How to exploit the semantics of a given query to make local decisions during the data collection phase. The local results computed by the nodes then must be propagated to

the sink to answer the query. The objective here is to reduce the number of messages as well as the volume of data transmitted by the nodes.

- How to schedule transmissions in the network for exchanging control/data messages during dissemination and data collection phases. It is trivial to understand that a schedule will facilitate the nodes to be *awake* at specific times during query processing, i.e., only when they are supposed to either transmit or receive messages. That in turn will allow the nodes to sleep as long as they can in order to further reduce their energy consumption. Therefore, a "tight" schedule will be considered a "good" schedule. More importantly, since multiple nodes may participate during query processing, it is likely that their wireless transmissions may interfere with each other. Therefore, the objective is to construct "tight" and "interference-free" schedules.

- How to deal with communication failures that are inherent in a wireless network. Because of failures, at certain times during query processing, some nodes that were able to communicate previously may suddenly be devoid of any communication to exchange control/data messages. That may cause the sink to return incorrect or incomplete results. Therefore, in-network query processing must also offer solutions to recover from the failures either partially or fully. Obviously, there will be some communication overhead for the nodes to recover from the failures. Nonetheless, the objective here is to opportunistically recover from the failures with the minimum possible overhead.

It is beyond the scope of this thesis to consider every type of query that can be processed in a WSN. Instead, we consider a class of aggregation queries, prominently represented by the top-k queries, to pursue an investigation of in-network query processing problem in WSNs. Top-k queries are simple, yet an important class of queries that are widely used in various applications [6, 30, 73, 80]. In the context of WSNs, remotely monitoring a physical environment is a typical application. There exist many situations where one is interested in monitoring extreme and atypical behavior. For example, finding the highest temperature values in a building or a patch of forest, the most frequently visited locations by animals, etc. Oftentimes, there are limited resources, e.g., dispatching vehicles to deal with such observed extremes and as a consequence, one may be interested only in the top-k observations. In this context, top-k queries represent an important class of aggregation queries, e.g., min-k, MAX, and MIN queries.

Through a systematic study of the top-k query processing in WSNs, we strive to gain

more insights about in-network query processing in general. Nonetheless, we will show in this thesis that our proposed solutions for processing top-k queries are not *tied* to top-k queries only, but they are also applicable to a wide range of in-network query processing problems in WSNs. Since performing local computations is much cheaper than transmitting data to a sink, the main idea behind these solutions is to *push* the local computations into the network to reduce the communication cost and thus to increase the network's lifetime. Towards that end we design new *logical* topologies that are effective for efficient query processing in WSNs. Logical topologies basically provide a virtual infrastructure for efficient communication in a wireless network. Our intuition is that by exploiting the spatial proximity of the nodes one can create various logical topologies built on top of the physical topology of a network, which cannot only support the in-network local computations, but they also provide an effective infrastructure that is robust to failures and efficient for exchanging data/messages between the nodes to reduce the communication cost.

Our proposed logical topologies are unique in the sense that they fully exploit the two basic properties of a WSN, i.e., its multi-hop communication setup and wireless nature of transmission/reception, to reduce the communication cost significantly. The multi-hop setup of WSNs enforces the nodes to process sensor data in an incremental fashion while providing an opportunity to make local decisions, e.g., we will show in this research how to exploit the "multi-hop" property effectively for data *aggregation* and *filtering*. Wireless transmission/reception is essentially performed in a "broadcast" environment, i.e., a single transmission by a node (transmitter) can be received by multiple nodes (receivers) present within the radio range of the transmitting node. Though reception also accounts for energy consumption in WSNs, it usually comes at a reduced cost as compared to the transmission cost [28]. In this research we will show novel techniques that exploit the "broadcast" property not only to reduce the transmission cost (at the expense of an increased reception cost that is cheaper than the transmission cost), but also to recover from the failures in WSNs. Nevertheless, reducing the overall communication cost, which includes the cost for transmission as well as reception, remains the primary research objective of this thesis. Next, we present a background discussion on what is going to follow from Chapters 2 to 6.

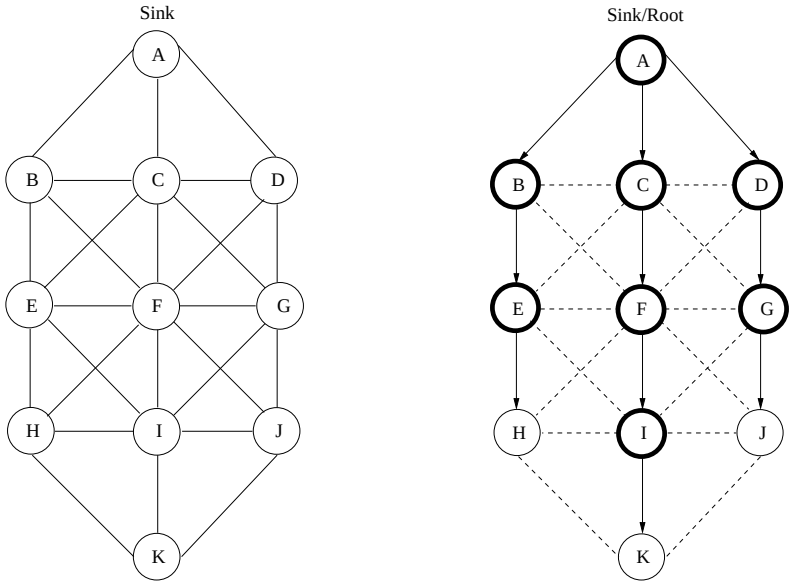
1.2.1 In-Network Processing of Top-k Queries

Top-k queries in distributed systems is a widely studied problem. Olston *et. al.* addressed the problem of caching approximate values with appropriate precision [50]. Their work lead to the idea of implanting *arithmetic filters* in a distributed environment to suppress

communication messages. Babcock and Olston [6] extended that and applied the idea of cached values for the top-k monitoring problem in data streams. The key idea is to use the cached values as range-based arithmetic filters. Filters are adjusted dynamically when they are violated. A coordinator node monitors the filter constraints of the rest of the nodes and also maintains the top-k result. This fundamental idea of installing filters for suppressing unnecessary updates has turned out to be especially useful within WSNs. A number of algorithms proposed in the literature, e.g., [56, 57, 59, 60, 73], rely on this idea for continuous monitoring of sensor values, a problem that is closely related to the problem that we investigate in this thesis. The difference among the previously proposed algorithms lies in the various strategies being used for maintaining the filters at successive periods. There are also solutions that only apply *aggregation* and do not use any filtering mechanism. TAG [43] is a classical example of such non-filtering based approach that can be used for the top-k query problem that is addressed in this thesis. We will discuss more about aggregation and filtering based solutions of the top-k query problem in the next chapter.

Not surprisingly, several other versions of the top-k query problem exist. For instance, a variation of the top-k query is to rank the objects based on the aggregated scores on a set of attributes stored at distributed locations. The Threshold Algorithm [22] is the best known solution for this problem. The main constraint of this algorithm is that it assumes single-hop communication. Zeinalipour-Yazti *et. al.* [80] propose a solution to a similarly defined problem but which is developed in the context of a multi-hop WSN. Similar, yet different problems have been investigated elsewhere. In [58] the authors use a model-based optimization technique for answering the approximate top-k queries; the goal is to minimize the number of true answers missed in approximate answer. More recently the authors of [15] propose exploiting the spatially correlated sensor data to build partial order trees (POT) to answer the top-k queries. Their main idea is to select "hot spots" (sensors with highest readings) in a network to build a logical topology to reduce unnecessary sensor updates.

The precise top-k query problem that we address in this thesis is presented in the next chapter. Before presenting this problem and our proposed solution we briefly review the different aspects of in-network query processing that we investigate in this thesis. We start with a discussion on logical topologies.



(a) Physical topology (Connectivity Graph) (b) Logical topology (Shortest Path Tree)

Figure 1.3: Solid lines represent edges, and arrowed lines represent messages with the distinction that all arrowed lines coming out from a node represent a single transmission from that node. Dashed lines in Figure (b) represent edges that are in the graph but not in the tree.

1.2.2 Logical Topologies : A Virtual Infrastructure for Query Processing

Throughout this thesis we use a Unit Disk Graph (UDG) to represent a WSN. UDGs are frequently used to model the communication of wireless nodes with identical circular ranges, deployed on 2–dimensional space. UDGs are, therefore, considered a "benchmark" class of graphs for the study of wireless algorithm complexity [16]. Nonetheless, the solutions proposed in this thesis are *independent* of UDGs, i.e., they are also applicable to general topology graphs with some modifications.

A WSN often operates by constructing logical topologies, such as a spanning tree, built on top of the physical topology of the network. The constructed logical topologies are then used to disseminate queries in the network, and also to process and return the results of such queries back to the user. To demonstrate the importance of logical topologies for in-network query processing we present an example as illustrated in Figure 1.3. A connectivity graph representing which wireless nodes a node can communicate with is shown in Figure 1.3(a). Consider that node A is the sink, which is responsible for disseminating the query issued by a user into the network, and to return the results of the query back to the user.

To start the query processing, the first step is to disseminate the query in the network

to all nodes. An obvious solution is for A to transmit the query in its neighborhood, and request all of its neighbors to do the same. This is repeated until all nodes in the network have received the query, i.e., dissemination by *flooding*. It is trivial to see that a node may receive the query message more than once, to be precise, up to once from each of its neighbors. However, note that the query message is the "same" for every node, and therefore a node does not need to transmit the query message more than once. In the specific example flooding generates 11 query messages.

Each node can also forward its reply message containing its data. Since every reply message, unlike the query message, is "unique" (containing data from a particular node), every node may forward the reply messages for all of its neighbors. Eventually the sink will receive the data from all the nodes, which can then answer the query. However, note that redundant messages are transmitted by multiple nodes traversing multiple-paths thus creating a high communication traffic in the network. That makes flooding an inefficient solution. In the specific network shown in Figure 1.3(a) flooding generates 40 reply messages.

In the network shown in Figure 1.3(a) we note that several nodes can be reached by one single message due to the wireless nature of transmissions. Therefore, a sensible alternative to flooding is to use a logical tree topology, e.g., a Shortest Path Tree (SPT) shown in Figure 1.3(b) built from the graph shown in Figure 1.3(a) in which the sink, node A, becomes the root of the logical tree. Using a logical tree topology such as an SPT a query can be received by all nodes if the root and every non-leaf node of the tree transmit the query. This specific SPT requires a total of 8 messages (transmitted by nodes A, B, C, D, E, F, G, I) to disseminate the query in the network as compared to 11 messages required by flooding.

The advantage of using a logical tree topology appears more prominently during the reply phase. Data arriving at a parent node from its children can be *aggregated* in-network, and only the aggregated data can be forwarded further up in the tree. Consider, e.g., node I that aggregates the data it receives from its child K with its own data, before forwarding the aggregated data to its parent F. TAG [44] is a well-known method that uses in-network aggregation to reduce the communication cost substantially. This technique will be discussed in detail in the next chapter. For the particular SPT shown in Figure 1.3(a) a total of 10 reply messages are generated as compared to 40 reply messages in the case of flooding.

It is trivial to understand from the above discussion that logical topologies play a crucial role in query processing in WSNs. Basically, logical topologies provide a virtual infrastructure for query processing in WSNs. In this research we pay attention to the logical tree topologies that can efficiently process queries in WSNs.

1.2.3 Broadcasting and Convergecasting

An important observation from the above discussion is that in-network query processing basically involves two types of communication messages, i.e., sink-to-the-nodes, and nodes-to-the-sink. Recall the example in which the sink is required to disseminate the query in the network, and nodes need to send an appropriate response back to the sink. In the literature sink-to-the-nodes and nodes-to-the-sink communication are commonly known as *broadcasting* and *convergecasting*, respectively. In a typical situation, in-network processing of a query, irrespective of its type, will require at least one broadcast phase to disseminate the query and at least one convergecast phase to collect the response from the nodes. In summary, broadcasting and convergecasting are the two basic operations in WSNs that may have a profound impact on in-network query processing solutions, which is the precise topic that we discuss in Chapters 3 through 5 of this thesis. In particular, we discuss the results of our investigation with respect to logical topologies that are better suited for broadcasting and convergecasting as well as energy-efficient scheduling for those.

1.2.4 Failure Recovery

Failures are part of wireless networks that may disrupt the logical topologies, which are used for in-network query processing. Failures are equally likely to occur during the broadcast as well as convergecast. Consider the scenario of query broadcasting using SPT as shown in Figure 1.3(b). The failure of the communication link between nodes A and C will result in many nodes not receiving the query message. One obvious solution here is that node A can try to re-transmit the message if it does not receive an acknowledgment (ACK) from its children. In this particular case since node C did not receive the message and hence node A did not receive an ACK from C, node A can re-transmit the message. This process can be repeated for every parent node and as long as they did not receive an ACK from all of their children. It is trivial to see that an excessive amount of messages (including re-transmissions and ACKs) will be used to recover from the failures. In this research we will present novel solutions for opportunistic failure recovery during broadcast as well as convergecast, which *do not require re-transmissions* and ACKs. As we will show in Chapter 6 that is indeed possible by exploiting the broadcast and convergecast schedules.

1.3 A Summary of Contributions

- We propose an efficient distributed algorithm for processing the top-k queries in WSNs. Our carefully designed algorithm is significantly better than the current state-of-the-art solution in terms of energy consumption (Chapter 2).

- We propose new algorithms for constructing logical tree topologies that are better than the existing solutions in reducing the communication cost required for broadcasting and convergecasting in WSNs (Chapter 3).

- We establish new theoretical bounds for broadcast and convergecast scheduling problems in WSNs and prove their correctness. We also propose an efficient framework for broadcast and convergecast scheduling in WSNs. Our scheduling solutions have shown better performance than the existing solutions (Chapters 4 and 5).

- We present a novel solution for failure recovery in WSNs. The proposed solution is an efficient communication framework that opportunistically exploits the convergecast and broadcast schedules for robust query processing in WSNs (Chapter 6).

- We demonstrate the importance of our proposed in-network processing solutions by putting them together for efficiently processing top-k queries. In particular, we evaluate the impact of various logical topologies, scheduling schemes and failure recovery mechanisms on the top-k query processing in WSNs (Chapter 7).

- Another important aspect of this thesis is that, apart from synthetic datasets, it uses real world WSN setup and sensor data to evaluate the performance of various solutions proposed in Chapters 2 through 7.

Chapter 2

Tok-k Query Processing

2.1 Introduction

As mentioned in the previous chapter, there are several versions of the top-k queries that have been considered in the literature [6, 58, 59, 60, 73]. The precise top-k query as considered in this thesis satisfies the following requirements: (a) k is not restricted (but naturally cannot be larger than the number of nodes in the network), (b) the query determines the exact k highest observed values, (c) the query determines the full set of nodes that reported the k highest values, and (d) the query is executed periodically starting at some point in time and reporting values for a number of subsequent rounds.

We want to avoid trivial and energy-inefficient solutions for processing top-k queries. One such solution is a centralized approach whereby, in every round, all nodes send their measurements directly to the sink (using one hop communication) which then locally calculates the top-k values. This solution is of little practical interest because it introduces a large communication overhead, and hence energy consumption. An alternative approach here is to construct a logical topology built on top of the physical topology, e.g., a spanning tree rooted at the sink, in which nodes are connected to the root using multiple hops. Recall the example SPT, a logical tree topology that was discussed in the previous chapter. A logical topology not only provides a multi-hop *infrastructure* for nodes to communicate with each other and the root, but it also allows to impose some *form* of data aggregation to reduce the volume of data being transmitted to save on the communication cost. Before presenting our energy-efficient solution we formally describe the top-k query problem in WSNs.

2.2 Problem Statement

We consider a WSN consisting of a set of nodes, $S = \{i : i = 1, 2, ...N\}$. Time is discrete and counted in rounds. Each node produces one value per round. Let $S_{p,j}$ be the set of nodes that produced the p^{th} highest value, $V(S_{p,j})$, during the j^{th} round. The exact top-k query problem then is to find the set of k highest values, $D_j = \{V(S_{p,j}) : p = 1, 2, ..., k\}$, for each round, j. This implies also obtaining the set of nodes, $\cup_{p=1}^{k} S_{p,j}$, that observed the k highest values. An example of 8 nodes and their corresponding values during a given round is presented in Table 2.1. A top-2 query would return $D_j = \{23, 20\}$. Note that because of ties, the number of nodes reporting the top-k values may be larger than k, e.g., nodes $\{s_3, s_4, s_8\}$ in this case. Solutions not concerned with dealing with ties would return only the top-k *nodes*, i.e., either $\{s_3, s_8\}$ or $\{s_4, s_8\}$. In some applications, failing to report a node that has a top-k value may be problematic. For example, if one needs to plan and schedule resources according to the number of points of interest (reporting sensors), not being able to know the exact number of (correct) points is likely to have undesirable consequences.

Sensors	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
Values	10	15	20	20	16	15	18	23

Table 2.1: An example of 8 sensor values.

To start the top-k query processing using a logical topology, the first step is to disseminate the query in the network to all nodes, i.e., a broadcasting phase is needed to disseminate the query. In response to the query message propagated by the root, nodes take appropriate action by forwarding their messages (containing data) to the root using the multi-hop structure of the logical tree, i.e., a convergecasting phase is needed to collect the responses. We will assume in this chapter the existence of a logical tree topology, e.g., an SPT during both phases of query processing, i.e., broadcasting and convergecasting. In Chapters 3 through 5 of this thesis we will present a detailed discussion on which logical tree topologies are best suited for efficiently processing the top-k queries. For the purpose of this chapter it is sufficient to assume the existence of an underlying logical tree topology that is built on top of the physical topology for broadcasting as well as convergecasting. We have already seen how a logical tree topology is useful for broadcasting the query in the network. Shortly we will look into ways in which the top-k query processing solutions can benefit from the structure of a logical tree topology during convergecasting.

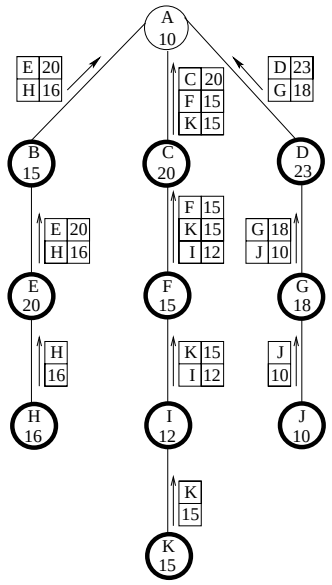


Figure 2.1: An example of a top-2 query processing using TAG. Darker circles represent nodes that triggered an update. Rectangles represent the packets transmitted by the corresponding nodes.

2.3 Related Work

Earlier solutions of processing the top-k queries in WSNs include TAG [43]. In TAG an SPT is used as an underlying logical topology for converegecast. In a given round processing starts from the leaf nodes and every non-leaf node first receives data from *all* of its children before sending the *combined* data to its own parent. A scenario of top-2 query processing using TAG on top of SPT is presented in Figure 2.1 in which nodes are annotated with the values observed during a specific round. One prominent feature of TAG is that non-leaf nodes exploit the semantics of a top-k query to perform *aggregation* in order to reduce the volume of transmitted data during the convergecast. For instance as shown in Figure 2.1, node C, after receiving values from its child F, discards the result $\{$I:12$\}$ to forward its local top-2 result only, i.e., $\{$C:20, F:15, K:15$\}$. Since unique top-k values are considered, the local top-2 result forwarded by node C contains 2 unique values observed at 3 different nodes. Finally, the root finds the top-2 result, i.e., $\{$D:23, C:20, E:20$\}$.

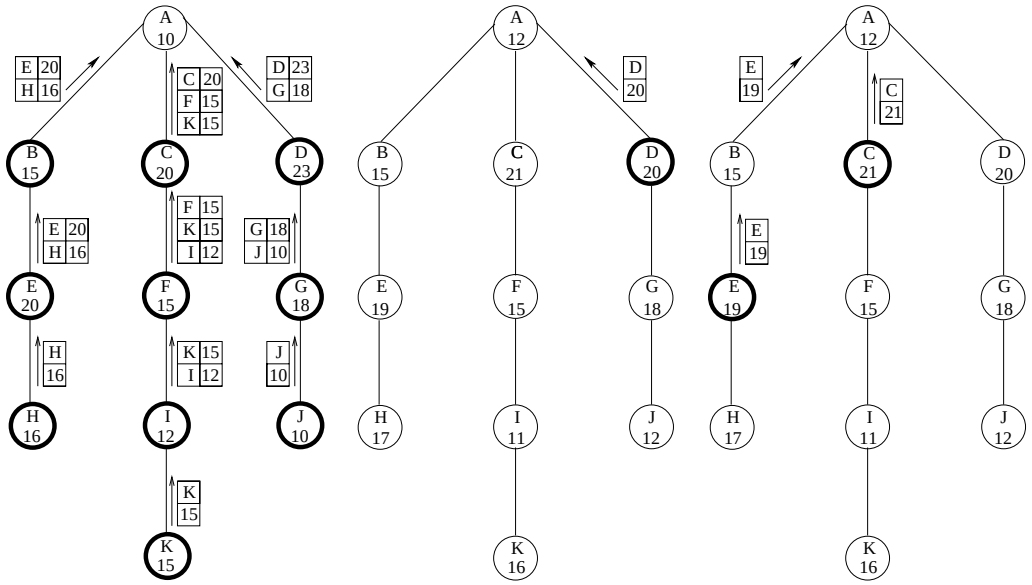
Unfortunately, TAG requires every node to send an update (containing its own value or aggregated values) during every round irrespective of the fact that only k such values will eventually become part of the actual answer. In an ideal solution only the nodes that have values among the top-k ones should send their values to the sink. However, these nodes do not know of their own "special" status *a-priori*. In order to address these issues, recently

FILA [73] has been proposed to process the top-k queries. The basic idea of FILA is to use arithmetic-*filters* for suppressing updates from nodes that are unlikely to become part of the solution to the top-k query. The intuition behind using filters is that nodes that reported the top-k values during a round are more likely to produce the top-k values again in the next round. It also means that the updates from the nodes, which had not produced the top-k values are potentially not required to compute the result in the next round.

There are other similar works presented elsewhere [4, 6, 58, 59, 60, 79]. Our work differs from the ones presented in [6, 58, 59, 60, 73] in a number of ways. Range caching [6] as well as FILA [73] use "range-based" filters, due to which they return approximate answers. In contrast to that, and as shown in this chapter, our algorithm is guaranteed to produce exact answers. Furthermore, FILA assumes a particular topology in which the root can directly communicate with the nodes (i.e., a single-hop setup to deliver messages from the root to the nodes). This assumption is not very realistic, in particular for WSNs deployed in large areas where obstacles, interference, and other environment factors restrict how far the root's signal can reach. We do not make any assumptions with respect to underling logical tree topology.

A solution proposed by Silberstein et. al. in [59] combines the idea of *temporal* and *spatial* suppression for continuously collecting *all* sensor values from the WSN. Although related, this problem is fundamentally different than the problem that we consider. The solutions proposed in [58] also deal with returning approximate answers. In yet another work [60] Silberstein *et. al.* carried out a detailed investigation of MAX (top-1), which unfortunately cannot be easily generalized. In addition, neither of these proposals deal properly with possible tied values, e.g., only one of many nodes with tied values is returned by these solutions, which may have undesired consequences as mentioned previously. Zeinalipour-Yazti et. al. proposed a novel framework for answering continuous queries based on *top-k views* [79]. In another study [4], details of a graphical tool for monitoring the k highest-ranked answers in a wireless sensor network are presented.

FILA has been up to now the state-of-the-art solution for processing the top-k queries in WSN and, as such, we use it as the basis to compare our proposed solution the top-k queries. FILA's performance is not only significantly better than TAG, it outperforms all other filtering based solutions proposed elsewhere [6]. Thus, for the sake of setting a proper background, we present a more detailed overview of FILA in the following.



(a) Round 1 (b) Round 2 (c) Evaluation (Round 2)

Figure 2.2: Initial rounds of a top-2 query in FILA. Darker circles represent nodes that triggered an update. Rectangles represent the packets transmitted by the corresponding nodes.

2.3.1 A Review of the State-of-the-Art

In the first round FILA works similar to TAG [44], that is all nodes send their observed values, with parent nodes in the SPT performing *en-route* aggregation. At this point the root determines the top-k result and also computes filters for the nodes (which are communicated to them by the root). A filter in FILA is a set of two values, an upper and a lower bound, and is used to control a node's update in the subsequent rounds. After the filters are installed, a node triggers an update, i.e., sends its most recent value, only if it violates its filter, i.e., if its value in not within the filter's specified range.

A scenario of top-2 query processing in FILA is shown in Figure 2.2 in which values of nodes are depicted within the circles. In the first round every node sends its update with aggregation performed *en-route* as shown in Figure 2.2(a). The root finds the top-2 values $\{23, 20\}$ that are produced by nodes $\{$D, C, E$\}$. Based on the top-2 result from the first round, the root computes the filter[1] $\langle 22\ 30 \rangle$ for node D and $\langle 19\ 22 \rangle$ for nodes C and E. The rest of the nodes will have $\langle 1\ 19 \rangle$ as their filter. These filters are then sent (not shown in the figures) by the root to the corresponding nodes at the end of round one. In the second round, only node D violates its filter (as its value, 20, falls out of the range $\langle 22\ 30 \rangle$), and

[1] In this example we used the uniform filter setting [73] rounding off the values. For simplicity we assume that the nodes' values are within 1 and 30.

it triggers an update as shown in Figure 2.2(b). During the validation phase, the root finds that the newly received value of node D, i.e., 20, falls into the filtering window of nodes C and E, i.e., $\langle 19\ 22 \rangle$. In this situation the top-2 result becomes undecided as nodes C and E may have any value from the range $\langle 19\ 22 \rangle$. Therefore, the root must find the current values of nodes C and E to compute the correct top-2 result. For that the root probes the previous top-k nodes, which had not sent an update during the current round, in this example nodes C and E. (For this purpose the root sends a probing message which is not shown in the figures.) In response to the probing message nodes C and E reply back with their current values as shown in Figure 2.2(c). The root can now compute the new top-2 result $\{21, 20\}$ from nodes $\{$C, D$\}$ respectively. Finally the root computes new filters and updates the corresponding nodes, if necessary, at the end of the current round.

2.3.2 Observations

There are several observations that can be made from FILA's example above in particular and filtering based solutions in general. Let us first examine FILA in detail. Recall that we seek the exact top-k unique values and the full set of nodes that observed them in a WSN. In contrast to that FILA may return approximate values of nodes, as long as their values remain within their filtering range. More precisely, FILA may use the nodes' values reported in the previous rounds to compute the top-k result of the current round. Consider again the example of top-2 query processing as shown in Figure 2.2. During the second round, consider that the value of node D is 25 and the values for the rest of the nodes are the same as before, i.e., as shown in Figure 2.2(b). During the second round, none of the nodes triggers its update (as none of them violates its filter), and no further action is taken by the root. In this situation FILA's reported result during the second round will be $\{20, 23\}$, coming from node $\{$D, C, E$\}$, which is different than the actual correct result: $\{21, 25\}$ from nodes $\{$D, C$\}$. We can clearly see in this simple example that FILA's result is often-time just an approximation, not only on the nodes' values, but also on the set of nodes. We note that although in [73] the authors propose a means to control the answer's approximation, it comes at the expense of increased energy cost, hence at a reduced network lifetime (c.f., Figures 15 and 16 in [73].) As we shall see shortly our solution always guarantees exact answers while at the same time processing queries more efficiently than FILA.

Another characteristic of FILA is that it computes $k + 1$ filters, one each for the top-k nodes and one common filter for all other non top-k nodes. Whenever a node's value, regardless of whether in the top-k set or not, enters the filtering window of a top-k node,

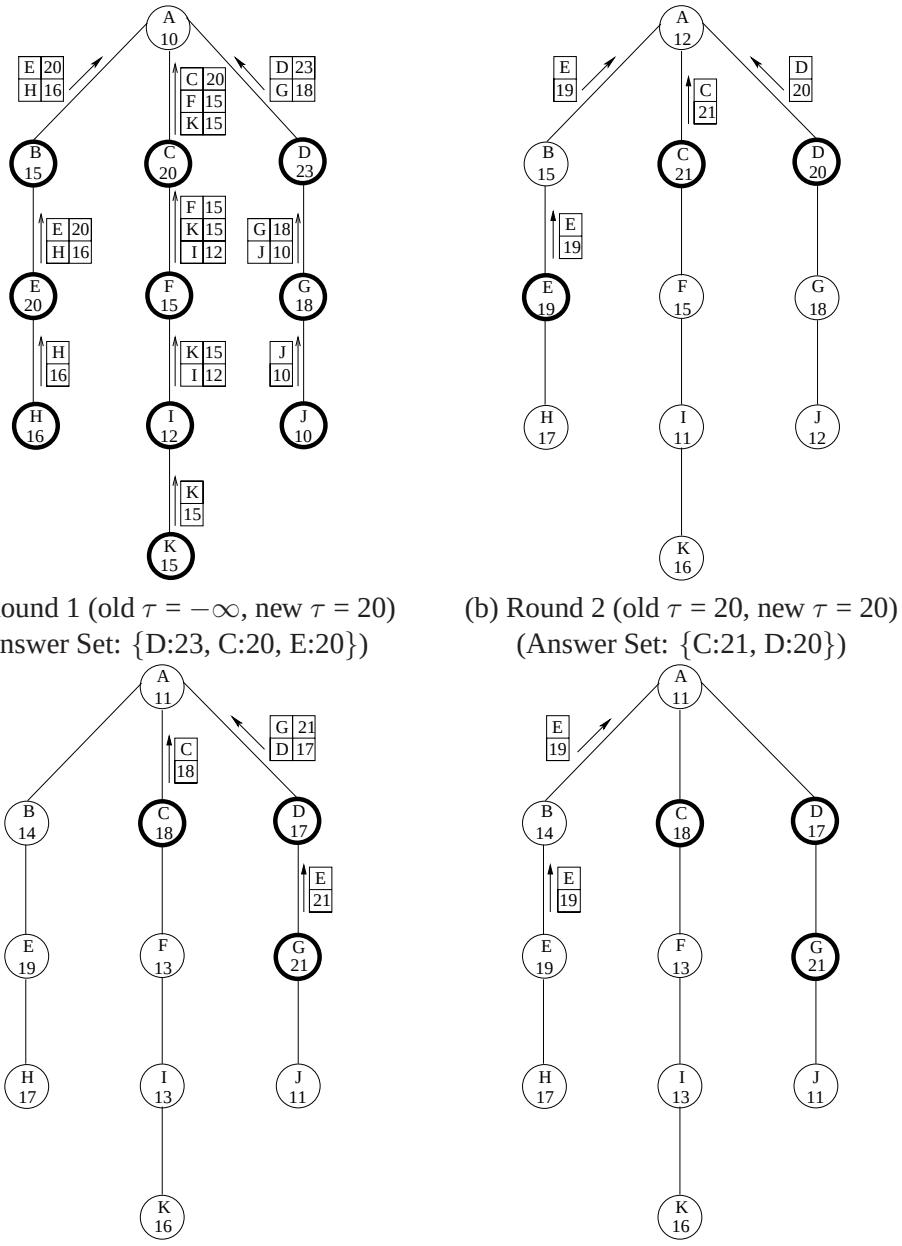
then that top-k node must be probed during the validation phase if it has not triggered its update in the beginning of a round. Note that in the second round as shown in Figure 2.2(b), if nodes C and E had triggered their update they would need not to be probed. In general, if all the top-k nodes trigger their update during every round then the top-k nodes need not be probed at all during the validation phase, which may result in reduced communication cost.

The worst case for FILA is when the root, apart from probing the top-k nodes, needs to probe the non top-k nodes as well during the validation phase (only those non top-k nodes that had not triggered their update in the beginning of a round). Assume that during the second round nodes C and E have a common value of 20, and all other values are as before for the rest of the nodes as shown in Figure 2.2(b). Again only node D triggers its update in the second round (recall that filter values are $\langle 22\ 30 \rangle$ for node D, $\langle 19\ 22 \rangle$ for nodes C and E, and $\langle 1\ 19 \rangle$ for the rest of the nodes). The root probes the values of nodes C and E during the validation phase. Now when the root receives their values, the top-2 result still remain undecided as the root has only top-1 value (as nodes D, C and E have one common value, 20). In this situation the root must inquire for the second top value, which must come from the non top-k nodes. For this, the root needs to send another probe message for the non top-k nodes to which they need to reply back appropriately. In essence FILA has two exclusive probing phases for: (i) the top-k nodes and (ii) the non top-k nodes. We argue that it is beneficial to eliminate the first probing phase altogether while using the second probing phase only when required. As we will show in our proposal that is indeed possible and it results in reduced communication cost.

Overall FILA uses 3 convergecast and 3 broadcast phases in every round to produce the top-k result. Though during a convergecast phase only a limited number of nodes are required to participate, a broadcast phase still needs all non-leaf nodes of the logical tree to forward the messages (recall the example of query broadcasting as shown in Figure 1.3). Furthermore, many convergecast and broadcast phases (6 in total) may not only increase FILA's communication cost, but they may also increase the query *latency*, i.e., the time it takes for the sink to actually answer the query. The above observations form the basis of our proposed algorithm, EXTOK, which stands for EXact TOp-K that we discuss next.

2.4 EXTOK: An Algorithm for EXact TOp-K Queries

In every round EXTOK's execution starts from the leaf nodes and progresses towards the root. In the first round EXTOK works similarly to TAG, in which all nodes send their



(a) Round 1 (old $\tau = -\infty$, new $\tau = 20$) (Answer Set: {D:23, C:20, E:20})

(b) Round 2 (old $\tau = 20$, new $\tau = 20$) (Answer Set: {C:21, D:20})

(c) Round 3 (old $\tau = 20$, new τ is undecided) (Answer set needs to be validated)

(d) Round 3 (old $\tau = 20$, new $\tau = 19$) (Answer set validated: {G:21, E:19})

Figure 2.3: Initial rounds of a top-2 query in EXTOK. Darker nodes denote TM-nodes.

updates, and the root, after collecting values, determines the top-k values. The root also calculates a threshold value, henceforth referred to as τ which is the minimum value of the current top-k values, and that will be sent to the nodes and installed as their filter. At this point nodes enter in one of two operation modes. A node is said to be in a "temporal monitoring" mode (or a TM-node) if it produced one of the current top-k values. TM-nodes are required to report *any* changes to their current value at *every* round as it may yield a

change in the current top-k answer set. Otherwise, a node is said to be in a "filtering" mode (or a F-node) if its value did not contribute to the current answer set. F-nodes are required to report their new values only when they observe a value that could belong to the answer set, i.e., a value that is greater than or equal to τ.

After the first round each subsequent round in EXTOK consists of three stages. In the first stage nodes trigger their update according to their operating mode. After receiving values during the first stage, the root proceeds to validate the current top-k results, and, if necessary, it initiates a validation procedure at the end of the first stage. During the second stage of the algorithm, *some* nodes may reply back in response to the validation procedure invoked by the root. At the end of the second stage the root determines the correct answer to the query. During the third stage the root adjusts the value of τ based on the newly computed result, and informs all other nodes about it, if τ has changed.

We illustrate the execution of EXTOK during the three initial rounds for a top-2 query in Figure 2.3. Values observed by the nodes during a particular round are depicted within the respective nodes. (Note that the nodes' values during the first two rounds are similar to the values used in FILA's example as shown in Figure 2.2.)

In the first round all nodes but the root are TM-nodes and τ is set to the application's minimum meaningful value (which we assume to be = $-\infty$ for simplicity). Then every node (Figure 2.3(a)) sends its update to the root. As the nodes push their values up in the tree, aggregation is performed by parent nodes. For instance, node B, after receiving values from its child E discards the result $\{$(B, 15)$\}$, i.e., its own value and only forwards its local top-2 result, i.e., $\{$(E, 20), (H, 16)$\}$, to its parent A. Since EXTOK considers the unique top-k values, the local top-2 result forwarded by node C is $\{$(C, 20), (F, 15), (K, 15)$\}$, i.e., 2 unique values observed at 3 different nodes. Finally, the root finds the top-2 result and also determines τ, which is the lower bound on the current top-k values (20 in this particular example). The root transmits τ after which only nodes C, D and E become TM-nodes, i.e., if their values change, they must propagate their new values. The rest of the nodes will trigger an update only if their value is greater than or equal to $\tau = 20$ (i.e., they become F-nodes).

During the second round (Figure 2.3(b)) nodes C, D and E change their values and trigger updates. In order to correctly compute the results the root always requires the current values of the TM-nodes that changed during a given round, therefore, their values cannot be aggregated *en-route*. Since the top-2 values received by the root during the second round (i.e., 21 and 20) do not invalidate $\tau = 20$, and also the value of every F-node is less than

20 (otherwise they would have triggered their own update) the root can correctly find the top-2 values in the second round without taking further action. (This is in contrast to FILA's execution which required to trigger the validation phase during the second round as shown in Figure 2.2(b).) Note that at the end of the current round E has not received an update for τ (because it has not changed yet) and since its own value is smaller than τ it becomes a F-node automatically.

Changes to the values in nodes C and D in the third round (Figure 2.3(c)) trigger a propagation of their update. Node G (which was a F-node) also triggers its update because its new value is above τ. At the end of the stage one in the third round the root receives values $\{21, 18, 17\}$ yielding $\{21, 18\}$ as the (temporary) answer set. At this point the root finds that the current top-2's lower bound (18) is lower than the current value of τ (20). This means that other unreported values from F-nodes may now be part of the answer set. To determine the new correct result, the root sends a validation query in the tree seeking values that are greater than or equal to 18. In response to the validation query, node E replies back with its value (Figure 2.3(d)). The root finally determines the (guaranteed) correct top-2 values, i.e., $\{21, 19\}$, and also updates $\tau = 19$, which is propagated down in the tree to update the nodes' filters. Thus after round 3, E and G will become TM-nodes, and C and D will become F-nodes along with the rest of the nodes that are already F-nodes. Next we provide a textual description of the EXTOK algorithm after which we will provide its pseudo-code.

2.4.1 EXTOK's Algorithm

Initialization

In the first round every node but the root sends its update. Aggregation is applied by the non-leaf nodes of the tree. If the total number of values received by a non-leaf node, including its own value, is less than k then that node sends all values to its parent; otherwise it sends the information pertaining to the top-k values only. After receiving values from its immediate children the root can determine the top-k values and the corresponding nodes.

The root computes an arithmetic filter for the non top-k nodes to suppress unnecessary updates in the subsequent rounds. For that the root uses a threshold value, τ set to the minimum value of the top-k values it collected. The root transmits τ so that all nodes can update their filters. (For the sake of efficiency we make the assumption that if a new τ is not transmitted within a round the nodes will behave as if the same current τ had been transmitted. This approach has been adopted in other studies as well, see e.g., [56, 60].) Let

$v_{i,j}$ be the value of node i during the j^{th} round. After τ is announced, a node i becomes a TM-node for round $(j+1)$ if $v_{i,j} \geq \tau$, otherwise it becomes an F-node. After that EXTOK works in three sequential stages detailed next.

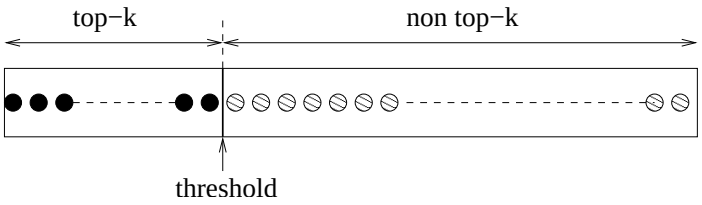
Stage 1

During this stage a TM-node, i, triggers its update only if $v_{i,j} \neq v_{i,j-1}$, while an F-node, i', triggers its update only if $v_{i',j} \geq \tau$. The root requires the new values of all triggering TM-nodes in a given round, therefore their values cannot be aggregated *en-route*. However, the values of F-nodes can and indeed are aggregated *en-route*. After the root has received values from all of its children, it determines the correctness of the current top-k result. As represented in Figure 2.4 there are a few cases that the root needs to consider and which we discuss next. For the sake of explanation we use Figure 2.4(a) to illustrate the set of top-k and the non top-k values separated by the current threshold (τ) value, further we refer to the former as "top-k-space".

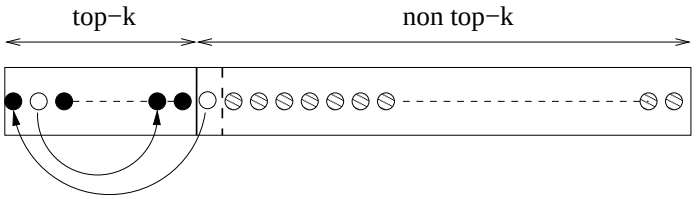
S1a: In this case we consider updates triggered by changes in TM-nodes only, in particular changes that happen *within* the top-k-space (Figure 2.4(b)). The only interesting event is when there is a new and higher value for the lower bound of the current top-k values. In this case a new τ is computed and transmitted to all nodes. Consequently some TM-nodes may now become F-nodes. This scenario has no effect on current F-nodes.

S1b: Next we consider changes triggered by updates from F-nodes only, i.e., changes that happen outside the top-k-space (Figure 2.4(c)). By definition this means that the new values of these nodes are greater than or equal to τ, and also that those nodes may become TM-nodes. It is also possible that a new τ exists and needs to be transmitted, and again depending on that value, some TM-nodes may need to switch to F-nodes and vice-versa.

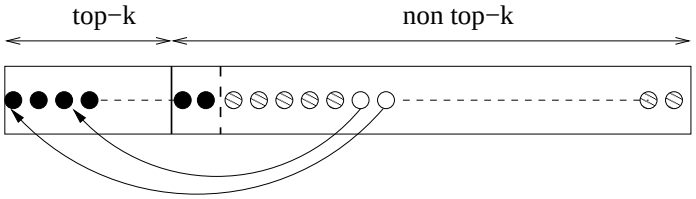
S1c: The last case of interest is more general; values are coming in or leaving the top-k-space (Figure 2.4(d)). The first situation can be triggered by certain updates from either TM- or F-nodes, while the second happens when a TM-node's value that is unique falls below the current τ. When the root receives all updates it will have k' values that are equal or above the current τ value and m values that are below τ. If $k' \geq k$, the root computes the current lower bound τ and, if there is a change, transmits it. As before, nodes may then need to switch their operating modes accordingly. Even if τ does not change, and therefore is not transmitted, nodes may still switch their mode since they know their own values and can presume that τ did not change. A more interesting scenario however, is when the root ends up with $k' < k$ values greater than τ. Then there are two cases to consider depending



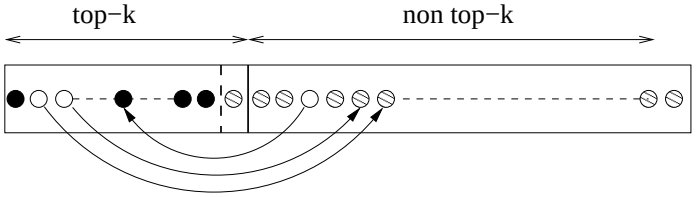
(a) Values in the top-k and non top-k spaces.



(b) Moving values within the top-k space.



(c) Moving values into the top-k space.



(d) Moving values in and out of the top-k space.

Figure 2.4: Various scenarios of changing values that can impact the top-k result and τ setting. The dark-color filled circles represent the top-k values and the circles with a filled pattern represent the non top-k values. The arrows visually represent how the nodes' values may change with respect to the total order of values from one round to the next. A non-filled circle represents a "space" created by a moving top-k or non top-k value. Solid (dashed) vertical lines represent the new (previous) τ values.

on the relation between $k - k'$ and m, in both cases the root determines a suitable *probe value* to be sent in a *validation query* to all F-nodes; their responses are considered in Stage 2 (which we discuss shortly.). The purpose of the probe value is to restrict the subset of the nodes that need to respond to the validation query.

- If $k - k' \leq m$ the root has enough values to complete the answer set, but it is possible that F-nodes that did not trigger an update, and therefore whose values are not known

to the root, should be part of the answer. To solve that potential problem the probe value is set equal to the $(k - k')^{th}$ highest value from the set of m values of nodes that have dropped below τ.

- On the other hand if $k - k' > m$ the root does not have enough values to complete the answer set and is unable to set a sensible bound for the probe value therefore it operates similar to the first "TAG-like" round over the F-nodes (TM-nodes need not be queried as they would have already sent useful updates by themselves), and the probe value is set to $-\infty$.

To illustrate the situations above consider an example where $\tau = 30$, $k' = 8$ and the set of m values below τ at the root is $\{25, 22, 19, 18, 15\}$. If $k = 10$, then the root needs $k - k' = 2$ values to complete its answer set. Only values greater than or equal to 22 need be considered since the root already has values above it to complete the answer set, thus the probe value in the validation message is set to 22, i.e., the 2^{nd} ($(k - k')^{th}$) highest value among the m values available. If $k = 15$ then the root has no means to set a bound on the probe value as it has less values than it needs, thus it sets the probe value to be $-\infty$

Stage 2

In the second stage all F-nodes, which had not triggered their update in the current round, reply back in response to the validation query only if their value is greater than or equal to the probed value. Aggregation is applied *en-route* and can actually be "tightened". Note that the node needs no more than $k - k'$ values, thus if a node receives more than $k - k'$ values, then it forwards only the information about the top $k - k'$ values to its parent. The root may or may not receive any values in response to the validation query. In any case the root can correctly determine the top-k result from the values it has received from the TM- or F-nodes during the first stage, plus the F-nodes that have replied back in the second stage in response to the validation query (if any), which is in addition to the k' values above τ that the root already has. At this point the root can recompute a new value for τ from the newly computed top-k values, and if it is different from the previous τ then it is transmitted.

Stage 3

In this "concluding" stage the root updates the nodes about the new τ, and the nodes can set their mode accordingly. In the absence of a threshold update message during a given

Algorithm 1

1: **procedure** EXTOK-NODE(ξ_i, P_i, R, k)
2: $j \leftarrow 1$;
3: $mode_i \leftarrow TN$;
4: $v_{i,0} = -\infty$;
5: **repeat**
6: $Q = \emptyset$;
7: $Trigger \leftarrow false$;
8: **if** $(mode_i = TN \wedge v_{i,j} \neq v_{i,j-1}) \vee (mode_i = FN \wedge v_{i,j} \geq \tau_j)$ **then**
9: $Trigger \leftarrow true$;
10: $Q = \{\langle v_{i,j}, i\rangle\}$;
11: **if** $\xi_i \neq \emptyset$ **then**
12: **for all** $i' \in \xi_i$ **do**
13: **if** $Receive(Q', i')$ **then**
14: $Q = Q \cup Q'$;
15: **if** $Q \neq \emptyset$ **then**
16: $Send(P_i, Q)$;
17: **if** $Receive(ValidationQuery(v_{q,j}, rVal), P_i)$ **then**
18: $Broadcast(\xi_i, ValidationQuery(v_{q,j}, rVal))$;
19: $Q = \emptyset$;
20: **if** $mode_i = FN \wedge v_{i,j} \geq v_{q,j} \wedge Trigger = false$ **then**
21: $Q = \{\langle v_{i,j}, i\rangle\}$;
22: **if** $\xi_i \neq \emptyset$ **then**
23: **for all** $i' \in \xi_i$ **do**
24: **if** $Receive(Q', i')$ **then**
25: $Q = Q \cup Q'$;
26: **if** $Q \neq \emptyset$ **then**
27: $Send(P_i, Q)$;
28: **if** $(Receive(\tau_{j+1}, P_i))$ **then**
29: **if** $\xi_i \neq \emptyset$ **then**
30: $Broadcast(\xi_i, \tau_{j+1})$;
31: **else**
32: $\tau_{j+1} = \tau_j$;
33: **if** $v_{i,j} \geq \tau_{j+1}$ **then**
34: $mode_i = TN$;
35: **else**
36: $mode_i = FN$;
37: $j \leftarrow j + 1$;
38: **until** $j > R$

Algorithm 2

procedure EXTOK-ROOT(ξ_r, R, k)
  $j \leftarrow 1$;
  repeat
    $Q = \emptyset$;
    for all $i' \in \xi_r$ **do**
      if $Receive(Q', i')$ **then**
        $Q = Q \cup Q'$;
    $S_l = ExtractIDs(Q)$;
    for all $i' \in S_{t,j} \setminus S_l$; **do**
      $Q = \{\langle v_{i',j-1}, i' \rangle\} \cup Q$;
    $S_l = ExtractIDs(Q)$;
    $S_{\tau+,j} = \emptyset$;
    $S_{\tau-,j} = \emptyset$;
    for all $i' \in S_l$ **do**
      if $v_{i',j} \geq \tau_j$ **then**
        $S_{\tau+,j} = i' \cup S_{\tau+,j}$;
      else
        $S_{\tau-,j} = i' \cup S_{\tau-,j}$;
    $k' = |V(S_{\tau+,j})|$;
    $m = |V(S_{\tau-,j})|$;
    if $k' \geq k$ **then**
      $[V(S_{t,j}), S_{t,j}] = FindTopK(V(S_{\tau+,j}), S_{\tau+,j})$;
    else
      if $k - k' \leq m$ **then**
        $v_{q,j} \longleftarrow (k - k')^{th}\ highest\ value\ of\ V(S_{\tau-,j})$;
      else
        $v_{q,j} = -\infty$;
      $rVal = k - k'$;
      $Broadcast(\xi_r, ValidationQuery(v_{q,j}, rVal))$;
      $Q = \emptyset$;
      for all $i' \in \xi_r$ **do**
        if $Receive(Q', i')$ **then**
          $Q = Q \cup Q'$;
      $S_l = ExtractIDs(Q)$;
      for all $i' \in S_{\tau+,j} \cup S_{\tau-,j}$ **do**
        $S_l = i' \cup S_l$;
      $[V(S_{t,j}), S_{t,j}] = FindTopK(V(S_l), S_l)$;
    $\tau_{j+1} = FindMin\,(V(S_{t,j}))$;
    if $\tau_{j+1} \neq \tau_j$ **then**
      $Broadcast(\xi_r, \tau_{j+1})$;
    Output ($[V(S_{t,j}), S_{t,j}]$);
    $j \leftarrow j + 1$;
  until $j > R$

round nodes can safely assume that the threshold value has not changed and they set their (possibly) new mode for the next round based on their current value alone.

2.4.2 EXTOK's Pseudocode

EXTOK's pseudocode for node and root's functionality are given in , respectively. EXTOK-Node takes as input the set of children, ξ_i, and parent, P_i, of node i in the logical tree. Number of rounds, R, and k are other two inputs. EXTOK-Root, which is executed exclusively on the root node, takes ξ_r, R, and k as input, where r represents the root node.

It is worth noting that the *Broadcast* operation used in Algorithm 1 at lines 18 and 30, and in Algorithm 2 at lines 29 and 40 represents a single transmission that is received by the appropriate nodes *locally*, i.e., only neighbors (possibly more than one) of a transmitting node. It differs from the *global* broadcasting (discussed in the next chapter) where the main goal is to send a message transmitted by a particular node to every other node in the network, which is clearly different from sending the message to the neighbors only. Of course the nodes may use a multi-hop setup of the network, e.g., a logical tree structure to disseminate the message in the network. We will examine this problem in detail in the next chapter.

2.4.3 EXTOK's Correctness

The correctness of EXTOK's algorithm in the first round is straightforward to establish, as its behavior is very much similar to TAG's. The following theorem asserts and proves its correctness for subsequent rounds. We assume the availability of two functions: $V(S)$ that returns the number of unique values within a given set of nodes S, and $v_{i,j}$ that returns the value of a given node i at round j.

Theorem 1. *In any given round $j \geq 2$, EXTOK produces a correct top-k result.*

Proof. Given a set S of N nodes, let $S_{t,j}$ and $S_{f,j}$ be the sets of TM-nodes and F-nodes, respectively, from which the root received values in the j^{th} round. Let $S_{c,j}$ = $S_{t,j} \cup S_{f,j}$, be the combined set of TM-nodes and F-nodes. Further, let $S_{\tau+,j} = \{i \in S_{c,j},\ s.t.\ v_{i,j} \geq \tau\}$ and similarly $S_{\tau-,j} = \{i \in S_{c,j},\ s.t.\ v_{i,j} < \tau\}$. τ is set to the minimum value in the current answer set, i.e., it is the smallest of the current top-k values. Finally, let $k' = |S_{\tau+,j}|$, and $m = |S_{\tau-,j}|$. We distinguish two cases: $k' \geq k$ and $k' < k$.

C1: If $k' \geq k$ the root has at least k values that are greater than or equal to τ. By construction, the values that the root does not know must be less than τ, therefore the root must have the exact top-k values.

C2: If $k' < k$, then the root requires additional $k - k'$ values to find the top-k result. Further, $S_{\tau-,j} \neq \emptyset$ (otherwise we would necessarily have $k' \geq k$) and it contains only those TM-nodes whose values have dropped below τ; F-nodes only send updates if their values become greater than or equal to τ. At this stage (S1c, Algorithm 2, line 29), the root sends a validation query containing a probe value $v_{q,j}$, which leads to the following two sub-cases.

C2a: If $k - k' \leq m$, then the probe value, $v_{q,j}$, is the $(k - k')^{th}$ highest value from the set, $V(S_{\tau-,j})$. Assume the root receives k'' values in response to the validation query. By construction all k'' values that the root receives are greater than or equal to $v_{q,j}$ but smaller than τ; otherwise they would have triggered an update and be known to the root already. The root now has k' values that are greater than or equal to τ, and $k - k'$ values that are less than τ but greater than or equal to $v_{q,j}$, and additional k'' values (received in response to the validation query) that are also less than τ, but greater than or equal to $v_{q,j}$. The root now is guaranteed to have at least k values that are greater than or equal to $v_{q,j}$ and all other values in the tree are, again, by construction smaller than $v_{q,j}$. Thus the root must be able to find the correct top-k values.

C2b: If $k - k' > m$, then $v_{q,j} = -\infty$, which means that the root will receive answers, possibly aggregated *en-route* from every F-node that had not triggered its update during the first stage, and clearly there will be enough values (current plus newly received ones) at the root for the correct top-k values to be found. □

2.5 Performance Evaluation

In our simulation study we implemented FILA and EXTOK using the commonly used SPT topology. For FILA we implemented uniform and skewed filter settings, and lazy filter update policy because of its superior performance [73]. A node id and its value are represented by 2 bytes each. A filter in FILA and τ value in EXTOK are characterized by 4 and 2 bytes, respectively. Each message also accounts for 4 bytes as a packet header overhead. In all experiments we assume that messages are delivered using a multi-hop setup. This is in contrast with the experimental setup used in [73] in which only nodes-to-root messages are delivered using a multi-hop setup; while the root is assumed to be capable of communicating with the nodes using single-hop transmissions. For various reasons we believe this assumption cannot be considered to hold in general. Consider e.g., the situation in which the root has limited power to transmit the signals that can be received by a limited number nodes only. Nonetheless, for the sake of fair comparison we also perform experiments

while considering such an assumption for both approaches, i.e., we also consider a setup in which the root is capable of communicating with the nodes using single-hop transmissions in FILA as well as EXTOK.

In order to evaluate our proposal we used both synthetic and real datasets. The synthetic dataset was generated by simulating a network of nodes deployed in a 200m$\times$200m area. Using this dataset we performed experiments by varying five parameters: number of top values sought (k), number of nodes (N), wireless/transmission range (ω), probability that a node's value changes between two consecutive rounds (γ) and percentage of change in node's value (δ). To investigate the impact of randomly changing values (nodes' measurements) on the performance of the algorithms we generated "temperature" values for nodes. The initial value of nodes was randomly set between 1 and 100 and could vary between rounds according to parameter δ (equally likely to be a negative or positive change). Results using the synthetic dataset are based on an average of 20 simulation runs in which each run consists of 200 rounds. In each of these simulation runs the position of the nodes and the root node were chosen randomly. All results presented in this thesis include 95% Confidence Intervals as marked by the vertical error bars.

2.5.1 Intel Berkeley Research Lab Setup

In all simulation studies presented in this thesis we also used a real sensor network setup from the Intel Berkeley Research Lab [1], which provided us with a dataset consisting of approximately 3.5 million sensor readings from 54 nodes deployed in the approximately 50m$\times$50m lab. There were some missing values from the data that were replaced using linear interpolation. Sensor readings were originally maintained by epochs, a monotonically increasing number for each of the nodes. We organized the sensor readings in such a way that the dataset has 60,000 rounds, each one containing one value for each of the 54 nodes.

Nodes' position were also available with the dataset that we used to create the original physical topology. Since the number of nodes and their positions are fixed in this setup, we varied transmission range (in meters) of nodes from the set $\{8, 10, 12, 14, 16\}$ to create various logical tree topologies. As before the reported results are an average of 20 simulation runs, and during each of those runs the root node was chosen randomly to create a logical tree topology. Note that unlike the synthetic dataset in which we could vary parameters k, ω, γ, δ and N, only parameters k and ω are investigated using the Intel dataset in this chapter. Nonetheless, throughout this research we used this real sensor network setup (and the corresponding data as well whenever it was applicable) to evaluate

Parameter	Values
k (# of top values)	1, 5, **10**, 15, 20
N (# of nodes)	100, 200, **300**, 400, 500
L (length of the square area [m])	200 (Synthetic) 50 (Intel)
ω (transmission range [m])	25, 30, **35**, 40, 45 (Synthetic) and 8, 10, **12**, 14, 16 (Intel)
γ (probability of change)	0.1, 0.2, **0.3**, 0.4, 0.5
δ (change [%])	2, 4, **6**, 8, 10

Table 2.2: Parameter values used in this chapter (default values are shown in bold face).

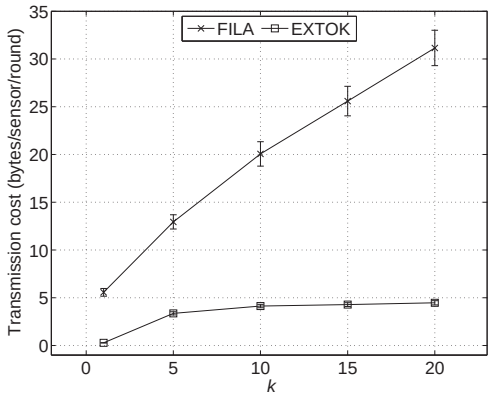
various solutions including the ones proposed in this thesis.

In our experiments we also keep track of the density, $\Psi = \frac{\pi\omega^2 N}{L^2}$, where N is the number of nodes, ω is the transmission range of nodes and L is the length of a square area. In particular, whenever we vary N, L and ω (or a combination of those parameters), we report the changes in Ψ value as well. Note that changing Ψ basically represents the change in the node density of the network, which may impact the underlying logical tree topologies. Table 2.2 summarizes the set of values used for various parameters in our experiments presented in this chapter.

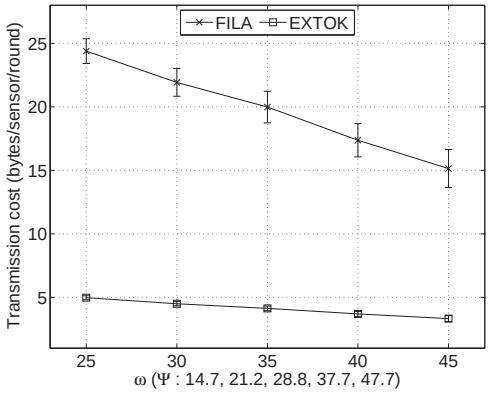
2.5.2 Transmission Cost

Transmission cost is measured as the average number of bytes transmitted by a node per round. Results from our experiments with the synthetic dataset are summarized in Figure 2.5. In the first experiment we evaluate the impact of varying k on the EXTOK's performance (Figure 2.5(a)). The foremost trend that we can see is that EXTOK's transmission cost is consistently smaller than FILA's. Particularly EXTOK incurs 70-80% less cost than FILA. The reason for this behavior is root-to-node communication which occurs more often in FILA (recall its filter updates and two probing phases). In contrast to that EXTOK generates less root-to-node communication (recall that there is a single value threshold updates for filter settings and only one probing phase in EXTOK). As expected, when k increases the costs for EXTOK and FILA increase as well, however, the increase is much faster in the case of FILA as more filters need to be maintained (specifically, k+1 pairs of values). Overall EXTOK offers the best solution which saves up to 80% of the communication cost as compared to the previously best known solution, FILA.

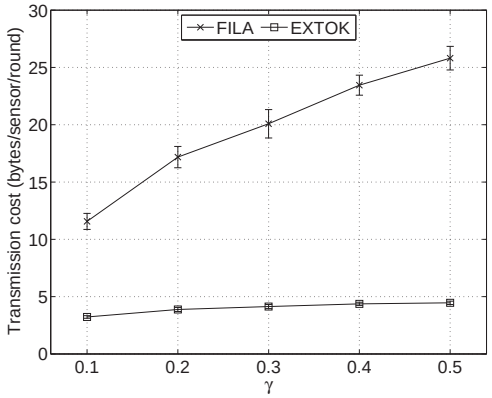
In the second experiment we evaluate the impact of varying ω (Figure 2.5(b)). Note that changing ω is bound to change Ψ, and hence the node degree (number of neighbors) of the



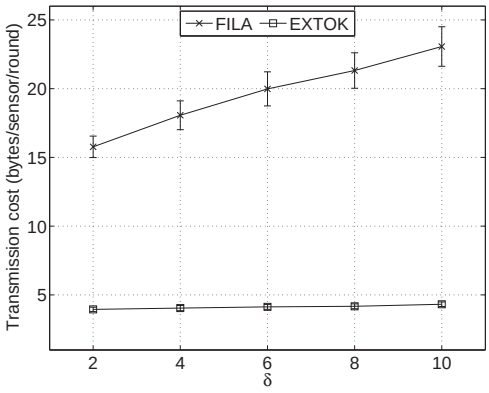
(a) Varying k



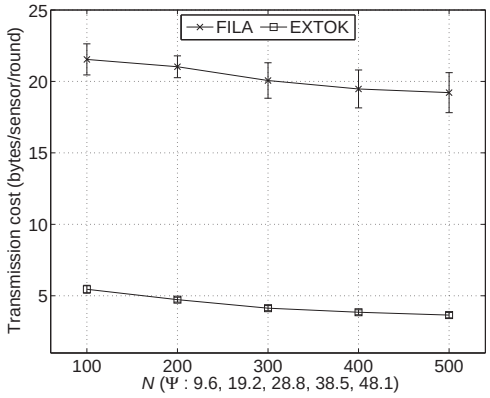
(b) Varying ω - transmission range (m)



(c) Varying γ - probability of change

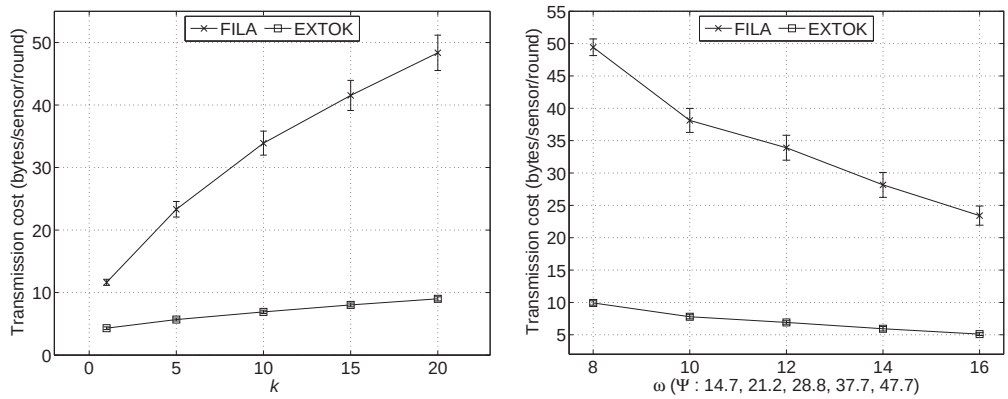


(d) Varying δ - percentage change



(e) Varying N - number of nodes

Figure 2.5: Transmission cost in synthetic dataset.



(a) Varying k

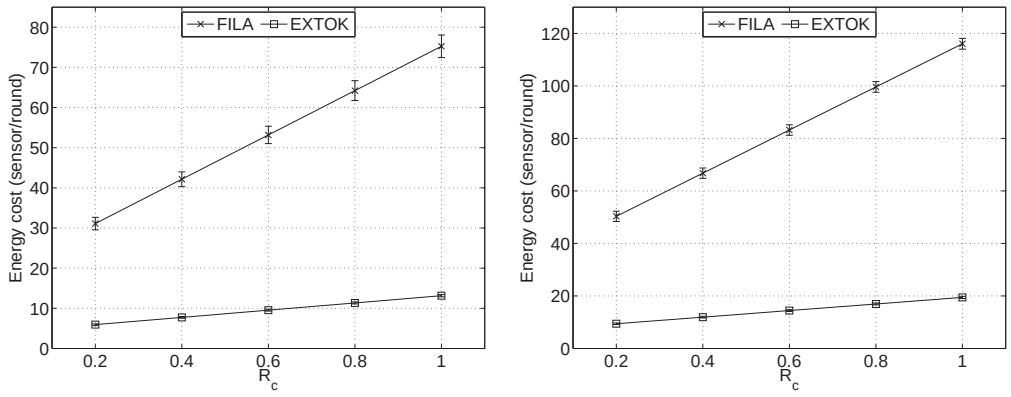
(b) Varying ω - transmission range (m)

Figure 2.6: Transmission cost in the Intel dataset.

nodes. Therefore, along with ω, we also show the changes in value of Ψ. Clearly EXTOK is the best option. The reason for the improved performance with increased ω is that as ω increases the underlying tree becomes shorter, decreasing the number of hops to the root. That results in efficient communication between the root and the nodes, and vice-versa. Here again, EXTOK consistently performs better than FILA.

Varying γ and δ create a scenario that allows observing how the dynamics of the observed values affect the algorithms' performance. Naturally, the more dynamic the observed values, the more updates will be required. In essence this situation creates more communication traffic in the tree. Our experiments in this regard are summarized in Figures 2.5(c) and 2.5(d). Again EXTOK outperforms FILA by a substantial margin. It is interesting to note that the increase in communication traffic clearly impacts FILA's performance, while EXTOK is virtually oblivious to the same. The reason for FILA's behavior is that when values are changed more dynamically filters are violated more frequently, and more communication takes place between the root and nodes. That results in overall increase in the transmission cost of FILA. On the other hand filters are violated in EXTOK as well, however, since the top-k nodes always send their update, the validation phase is used less frequently by the root. Moreover, whenever the threshold is changed, installing new filters at the nodes is much cheaper in EXTOK making it much less affected by the dynamism of values. Overall, EXTOK saved more than 65% of the cost over FILA.

Figure 2.5(e) summarizes the results from the experiments with the synthetic data in which we vary N while keeping all other parameters fixed. Increasing N increases the



(a) Synthetic dataset

(b) Intel dataset

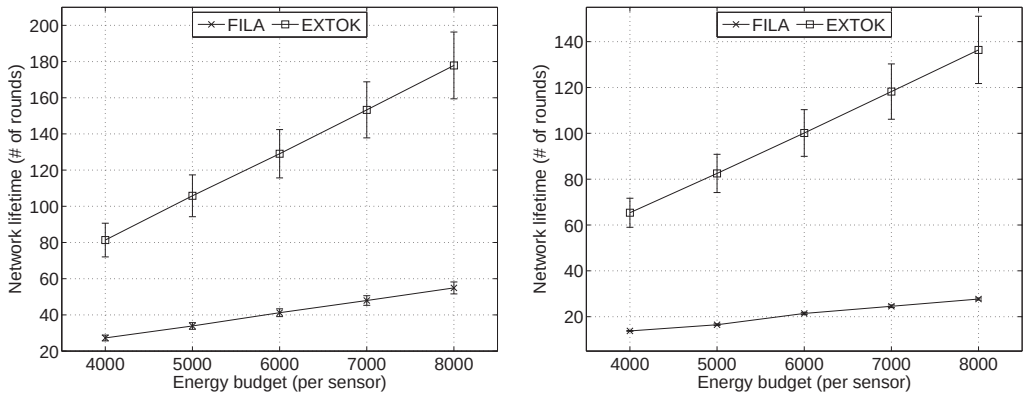
Figure 2.7: Energy cost.

network density, Ψ, which basically increases the node degree of the nodes. As expected EXTOK performs better than FILA. The noticeable trend is that as N increases the performance of all approaches increases, which can be explained by the fact that as the number of nodes increases the amortized per-sensor cost decreases.

Results from our experiments using the Intel dataset are discussed in Figure 2.6. The qualitative behavior is not very different from the case where synthetic data is used. Quantitatively though, there are noticeable differences. This can be explained by the following two observations and their compounded effect. First, the average node degree is now smaller (compare e.g., the node density in Figure 2.6(b) with that in Figure 2.5(b)). Second, the Intel dataset is more dynamic, naturally triggering more updates and consequently more nodes-to-root transmissions, which has resulted in overall increase in the transmission cost, e.g., compare the scale of results in Figures 2.5(a)-(b) and 2.6(a)-(b).

2.5.3 Energy Cost

Each bit received by a wireless transceiver incurs an energy cost, E_{rx}. It is also typical of transceivers used in wireless sensor platforms that receiving one bit requires less than the energy for transmitting one bit (E_{tx}). Models capturing the energy consumption have been proposed and used in various previous studies [43, 60]. In order to make the presented results as technology-neutral as possible, we assume that the unit of energy cost is the energy required for the transmission of a single bit, E_{tx}, and we use a parameter, R_c, to link transmission and reception cost via $R_c = E_{rx}/E_{tx}$. The energy cost of a node is computed as B_t + $B_r.R_c$, where B_t and B_r, respectively, are the number of bytes transmitted and



(a) Synthetic dataset

(b) Intel dataset

Figure 2.8: Network lifetime.

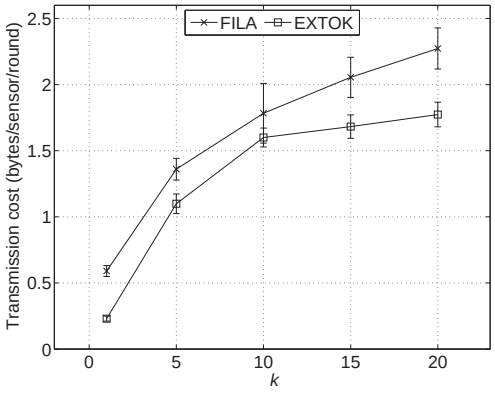
received by the node. In our experiments R_c assumes values from the set $\{0.2, 0.4, 0.6, 0.8, 1.0\}$, and all other parameters are kept at their default values. An increasing R_c value means the cost of reception is increasingly becoming equal to the cost of transmission. For simplicity we do not include the processing cost (which is generally much cheaper than the transmission or reception cost) in our energy consumption model.

The results from synthetic data are summarized in Figure 2.7(a). As the cost of reception increases the overall energy cost increases for all solutions, though the increase is much faster in the case of FILA. The reason is that FILA uses range based filters that nodes receive during the filter updates apart from the two probing phases during which nodes receive probing messages, which accounts for much of the reception cost in FILA. Therefore, when R_c increases the reception cost becomes dominant and increases the overall energy cost in FILA. On the other hand EXTOK has less reception cost to begin with, and therefore, it does not contribute much to the overall energy consumption. Qualitatively similar results were obtained when using the Intel dataset as shown in Figure 2.7(b).

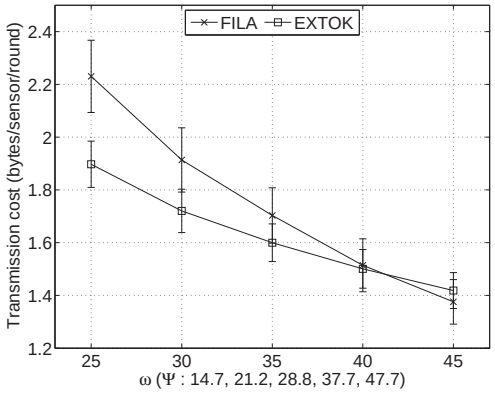
2.5.4 Network Lifetime

We also evaluated the performance with respect to the *network lifetime* that we define as the number of rounds before the first node runs out of its energy. To compute the energy consumption we used an R_c value of 0.6. The initial energy budget for a node was chosen from the set $\{4000, 5000, 6000, 7000, 8000\}$. Results are summarized in Figure 2.8.

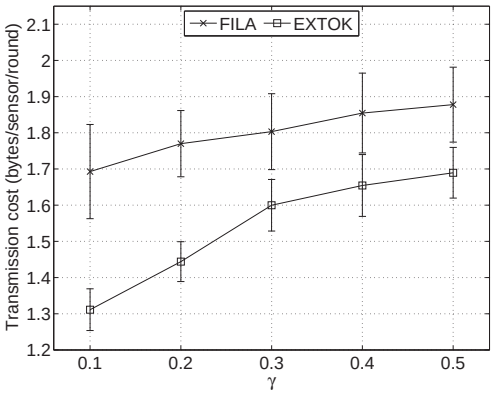
Results from synthetic data reveal that EXTOK extends the network's lifetime significantly as compared with FILA. In particular when the initial budget is 4000 units



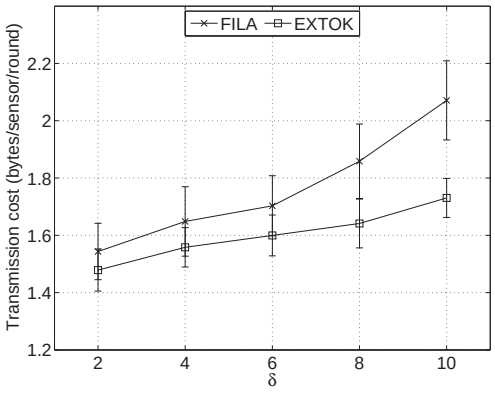
(a) Varying k



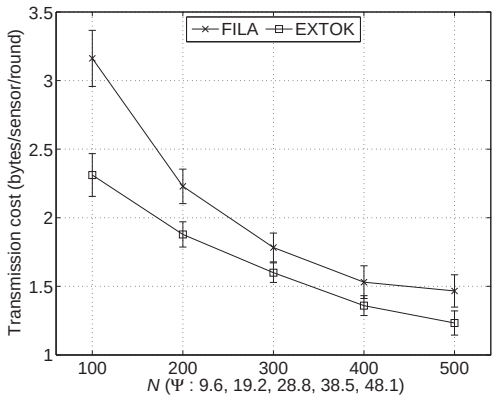
(b) Varying ω - transmission range (m)



(c) Varying γ - probability of change

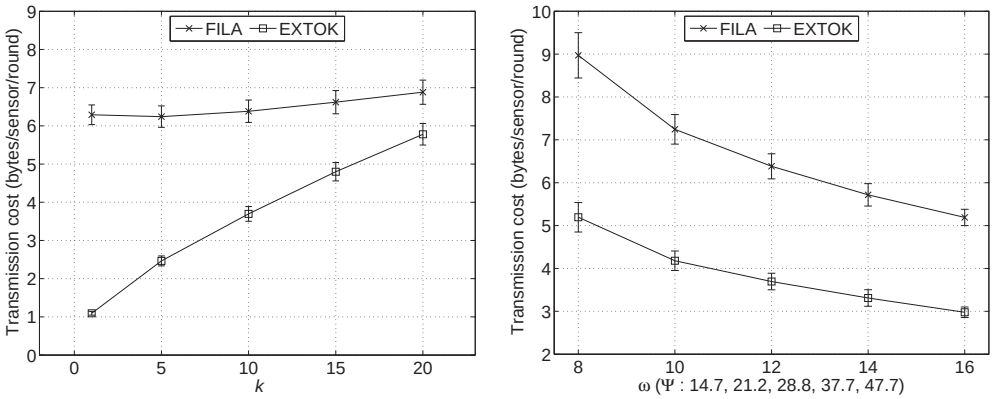


(d) Varying δ - percentage change



(e) Varying N - number of nodes

Figure 2.9: Transmission cost in synthetic dataset (the case of a single-hop broadcast).



(a) Varying k

(b) Varying ω - transmission range (m)

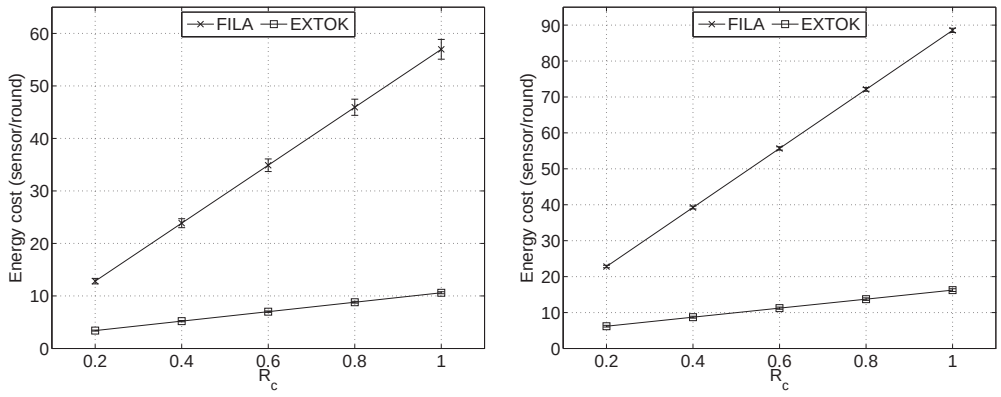
Figure 2.10: Transmission cost in the Intel dataset (the case of a single-hop broadcast).

of energy, EXTOK works for 100 rounds as compared to 28 rounds in FILA. The lifetime in EXTOK is almost doubled to 200 rounds when the energy budget is doubled to 8000 units. Qualitatively similar results were obtained when using the Intel dataset. However, the network lifetime is decreased as the overall per-node energy consumption is higher in the Intel dataset due to which the network lifetime is reduced. This can be verified by comparing the results in Figures 2.7(a) and 2.7(b) in which we can see that the energy cost in the Intel dataset is more than the synthetic dataset.

2.5.5 The Case of a Single-Hop Broadcast

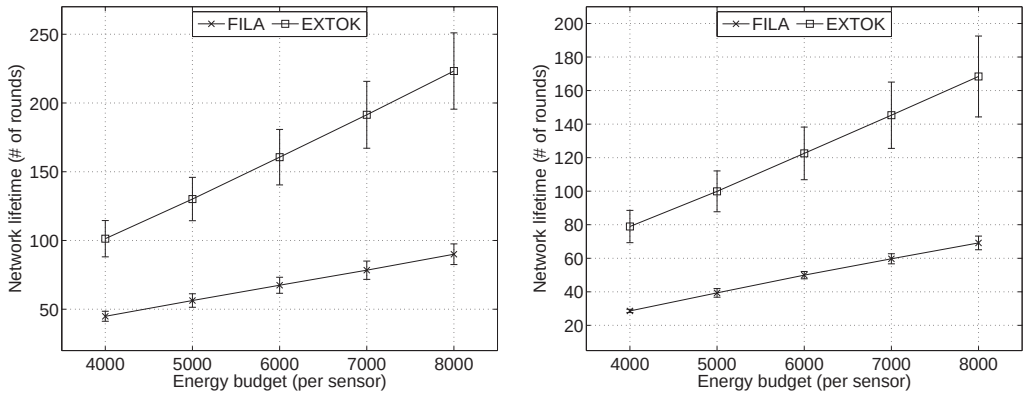
As presented in [73], FILA uses a TAG tree (an SPT) as an underlying logical topology to deliver messages from the nodes to the root (i.e., multi-hop setup for nodes-to-root messages). However, no tree is used to deliver root-to-nodes messages, because it is assumed that root can directly communicate with nodes (i.e., single-hop setup for root-to-nodes messages). This assumption, which was made in [73], is not very realistic, in particular for WSNs deployed in large areas where obstacles, interference, and other environment factors restrict how far the root's signal can reach. Nevertheless, we also examined FILA vs. EXTOK assuming this assumption holds for both approaches.

Results on transmission cost from synthetic and Intel dataset are summarized in Figures 2.9 and 2.10, respectively. As before EXTOK outperforms FILA, however, the margin in performance gain is reduced. This can be attributed to the fact that these results do not account for broadcast messages that are significant in the case of FILA. Recall nodes receive filter updates and probe messages directly from the root without using multi-hop setup.



(a) Synthetic dataset (b) Intel dataset

Figure 2.11: Energy cost (the case of a single-hop broadcast).



(a) Synthetic dataset (b) Intel dataset

Figure 2.12: Network lifetime (the case of a single-hop broadcast).

Results on energy cost and network lifetime are summarized in Figures 2.11 and 2.12, respectively. Once again EXTOK decreases the energy consumption, and therefore extends the network's lifetime by a significant margin. Note that in this setup in which the cost for multi-hop transmission of root-to-nodes messages is not considered, the network's lifetime is increased even further, which is in contrast to the results summarized in Figure 2.8. This can be explained by the fact that nodes do not spend their energy on *relaying* the messages from the root as the root can reach all nodes with single-hop transmissions. (However, the nodes still pay the cost of reception for the messages received from the root, which has been accounted for in all our results.) Nevertheless, even in this setup the network lifetime

while using EXTOK is significantly higher than the lifetime achieved by FILA as shown in Figures 2.12(a)-(b).

2.6 Conclusions

In this chapter we proposed a filtering based solution for efficiently processing the continuous top-k queries in WSNs. Filters allow to suppress a considerable amount of nodes-to-root messages during the convergecast phase. However, that is achieved at an increased amount of root-to-nodes messages in order to get nodes' response to validate the results and also to update the nodes' filter. Note that a filtering based solution requires broadcast phases in every round hence increasing the query latency. In contrast to that, a non-filtering based solution, e.g., TAG, does not need broadcast phase at all (apart from the one during which the query is actually disseminated) in which every node simply sends its value in every round and aggregation is performed *en-route*. Nevertheless, employing filters, as in the case of EXTOK, saves a considerable amount of query processing cost in terms of energy as compared to TAG. Basically there exists a trade-off between the energy cost and query latency for these two types of solutions, i.e., filtering and non-filtering based.

Irrespective of the type of a solution, broadcasting and convergecasting remain two basic operations to process the queries and hence their importance is significant. The efficiency of a filtering based solution for the top-k queries will be highly dependent on how efficiently broadcasting is done, in particular with respect to EXTOK's validation query and threshold updates. This is the precise problem that we discuss in detail in the next chapter. Our intuition is that by effectively exploiting the physical topology one can create a logical tree topology that is better suited than commonly used logical topologies, e.g., SPT, for broadcasting.

Chapter 3

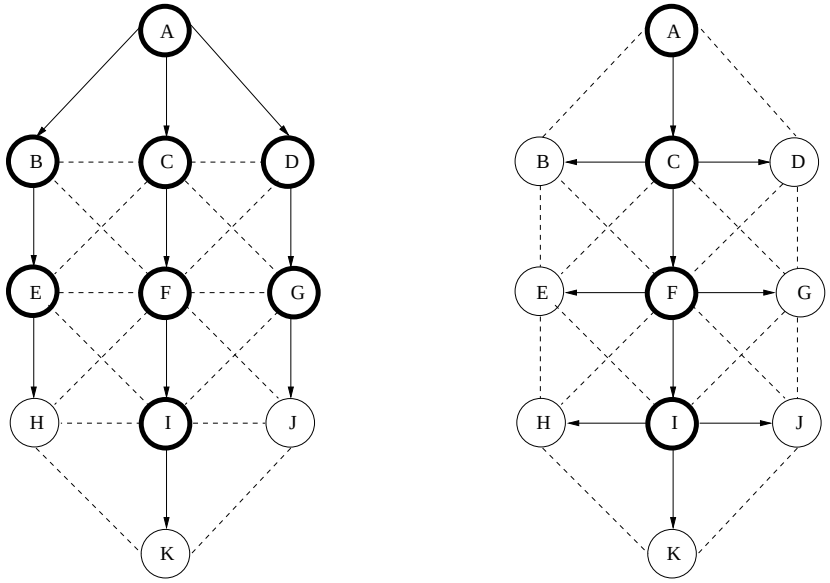
Broadcasting

3.1 Introduction

Broadcasting is a basic operation used in WSNs for disseminating messages in the network. Many applications rely on broadcasting to achieve certain objectives, e.g., to coordinate the distributed computing operations or to update a patch of software through executable code dissemination. In particular, recall the top-k query processing from the previous chapter in which EXTOK and FILA require the root to send the validation query and filter/threshold update messages to the nodes in the network. In this chapter we make no particular assumption about what is the message that needs to be broadcast. We just assume that all nodes, either periodically or during pre–determined periods, need to participate in broadcasting to allow the network–wide dissemination of messages.

Typically, there are two types of broadcasting problems that are considered in the literature, i.e., *One-to-All* and *All-to-All*. In the One-to-All broadcast problem a particular node generates a message that needs to be delivered to every other node in the network, e.g., a validation query issued by the sink. In the All-to-All broadcast problem every node in the network generates a message that needs to be communicated to every other node in the network. All-to-All broadcasting essentially requires multiple *instances* of One-to-All broadcasting. In this thesis, we propose solutions for One-to-All broadcasting problem only, however, the proposed solutions can be extended to the All-to-All broadcasting problem.

As discussed previously in Chapter 1, one naïve solution to broadcasting is flooding in which every node simply forwards the message that it receives from its neighbor. It is trivial to see that flooding will achieve the desired goal, i.e., every node in the network will eventually receive the message originated from the sink. Nevertheless, flooding leads to increased redundancy, contention and collisions in the network, which makes it an inefficient solution [29]. A sensible alternative to flooding is to use a logical tree topology, e.g., commonly



(a) SPT (Total 8 messages used for broadcasting)

(b) DST (Total 4 messages used for broadcasting)

Figure 3.1: Examples of different logical trees used for broadcasting. Solid arrowed lines represent edges of the logical tree. Dashed lines represent edges that are in the graph but not in the logical tree. Arrowed lines also represent messages with the distinction that all arrowed lines coming out from a node represent a single transmission from that node.

used SPT. An obvious problem now is to decide which logical tree is better than others for broadcasting in WSNs. This is the precise problem that we address in this chapter.

It is interesting to note that only the root and non-leaf nodes of a logical tree need to transmit the message during broadcasting. Leaf nodes need to be recipients exclusively. Consider, e.g., the SPT as shown in Figure 3.1(a). It is easy to understand that a smaller set of non-leaf nodes will result in a lesser number of transmissions. Consider, e.g., the Dominating Set Tree (DST) [46] shown in Figure 3.1(b), which is another alternative for the logical tree that can be used for broadcasting instead of the SPT shown in Figure 3.1(a). (Note that both logical trees, i.e., SPT and DST are constructed from the same communication graph.) Clearly the DST has a lesser number of non-leaf nodes as compared to the SPT. Using the DST, only nodes $\{$A, C, F, I$\}$ need to transmit the message, i.e., only 4 messages are used for broadcasting. This is in contrast to the SPT in which 8 messages are used. Therefore, the problem here is to find a logical tree topology rooted at a particular node, i.e., the sink, and which has the smallest possible set of non-leaf nodes. Interestingly, the above problem is similar to the problem of finding a Minimum Connected Dominating Set (MCDS) [69] of a graph. Formally the MCDS problem is defined as following:

Definition 1. The MCDS problem. *Given an undirected graph $G(V, E)$, find a set of vertices $V' \subset V$ such that (i) V' is connected, (ii) $\forall v' \in V - V'$: v' is connected to one of the vertices in V', and (iii) $|V'|$ is minimum.*

The above two problems are similar in the sense that the non-leaf nodes of a tree constructed from a graph constitute a connected dominating set of the same graph, e.g., the non-leaf nodes {C, F, I} of the DST shown in Figure 3.1(b) also constitute the corresponding MCDS of the underlying graph from which that DST has been created. It can be easily verified that nodes {C, F, I} satisfy all three conditions of the MCDS problem, i.e., (i) nodes {C, F, I} are connected, (ii) all nodes other than {C, F, I} are connected to at least one of the nodes from {C, F, I}, and (iii) there is no set of less than three nodes that meet the former two conditions. Unfortunately, finding an MCDS is known to be an NP-hard problem even when specialized to Unit Disk Graphs (UDGs)[1] [16]. The question of a tight approximation to the MCDS remains an area of active research [9].

An MCDS represents the minimum number of transmissions to reach all nodes in a network during broadcasting. Note that an MCDS is not constrained by the fact that a particular node initiates the broadcasting. However, this is in contrast to our specific problem in which it is required that the broadcasting be initiated from a particular node, i.e., the sink. Basically in this case the vertex representing the sink must be contained in the MCDS. Next, we examine the problem of finding an MCDS with the condition that a given vertex is part of the solution. The new problem is defined as following:

Definition 2. The MDST problem. *Given an undirected graph $G(V, E)$ and a particular vertex, $v \in V$, find a set of vertices $V' \subset V$ such that (i) V' is connected, (ii) $v \in V'$, (iii) $\forall v' \in V - V'$: v' is connected to one of the vertices in V' and (iv) $|V'|$ is minimum.*

Theorem 2. *The MDST problem is NP-Hard.*

Proof. Assume there exists an algorithm A that takes two inputs $G(V, E)$ and $v_i \in V$ to solve the MDST problem, and outputs the solution, $V_i \subseteq V$ containing v_i. We can execute A, $|V|$ times, to find every solution with respect to every input vertex $v_i \in V$. Clearly we can check the cardinality of each output set, V_i (from the possible $|V|$ solutions) in a polynomial time. If V_k is the set with the minimum cardinality, then V_k must also be the solution of the MCDS problem (as it does not restrict which set of vertices are selected).

[1] UDGs are frequently used to model the communication of wireless nodes with identical circular ranges, deployed on 2–dimensional space. UDGs are, therefore, considered a "benchmark" class of graphs for the study of wireless algorithm complexity.

It means that if A solves the MDST problem in polynomial time then the MCDS problem is also solvable in polynomial time. However, it is known that the MCSD problem is NP-Hard [69]. Therefore, the MDST problem is not polynomial time solvable unless P = NP, and hence the proof that the MDST problem is NP-Hard. □

3.2 Related Work

Because constructing an optimal broadcast tree is an NP-Hard problem, several approximation solutions have been proposed. Many heuristics including Broadcast Incremental Power (BIP) [72, 77], Iterative Maximum-Branch Minimization (IMBM) [40], Adaptive Broadcast Consumption (ABC) [38] and others [20, 23, 69]. Shortly, we will propose our solution while establishing a lower bound on the number of transmissions generated by a logical tree during broadcasting. As a side note, [69] provides an upper bound on the size of the CDS, i.e., the number of dominating or non-leaf nodes, which will result in equivalent number of transmissions. To be precise their algorithm has an approximation ratio of 8. But from the performance of their algorithm it is evident that this upper bound is far too pessimistic compared to the typical *practical* behavior of their algorithm. We take a different view of trying to squeeze the performance as close as possible to the lower bound corresponding to the broadcasting tree. Nevertheless, for comparison purposes, we will also evaluate the performance of [69] in our simulation study.

There are several other existing proposals that use different logical structures, e.g., clusters, for efficient communication in wireless networks [7, 41, 49, 51, 61]. In particular, Lin and Gerla [41] proposed a clustering based network architecture for multimedia support and Basagni proposed DCA [7], a distributed clustering algorithm for ad-hoc networks. Considering the power constraints of WSNs several energy-efficient solutions have also been proposed in the literature. Heinzelman et. al. proposed a class of adaptive protocols, SPIN, for efficient dissemination of data in WSNs [28]. In [33], a data centric approach called Directed Diffusion has been presented for query processing in WSNs. Younis and Fahmy proposed HEED [51], a distributed clustering protocol for ad-hoc sensor networks. Our work is different from [7, 41, 49, 51, 61] in the sense that we are focused on logical tree topologies for broadcasting.

3.3 Bounds and Tree Construction

A lower bound on broadcasting represents that the minimum number of transmissions that are required to broadcast a message from a particular node to every other node in the network. Consider, e.g., node A that has a message, which needs to be broadcast using either of the two logical tree topologies shown in Figure 3.1. Consider the DST first that is shown in Figure 3.1(b). The depth of nodes {H, K, J} (i.e., their distance from the root, node A) is 4. Note that nodes {H, K, J} have the maximum depth in the DST, i.e., no other node in the DST has depth 4. Clearly nodes {H, K, J} cannot receive the message unless their parent node I transmits the message. Similarly, node I cannot receive the message unless its parent node F transmits the message, and so on. Overall, broadcasting cannot be completed with less than four transmissions while using the DST. Similarly, in the case of the SPT shown in Figure 3.1(a), the depth of node K is 4, which is also the node with the maximum depth in the SPT. Clearly, broadcasting requires more than four transmissions while using the SPT. To generalize this observation, we introduce the following lemma.

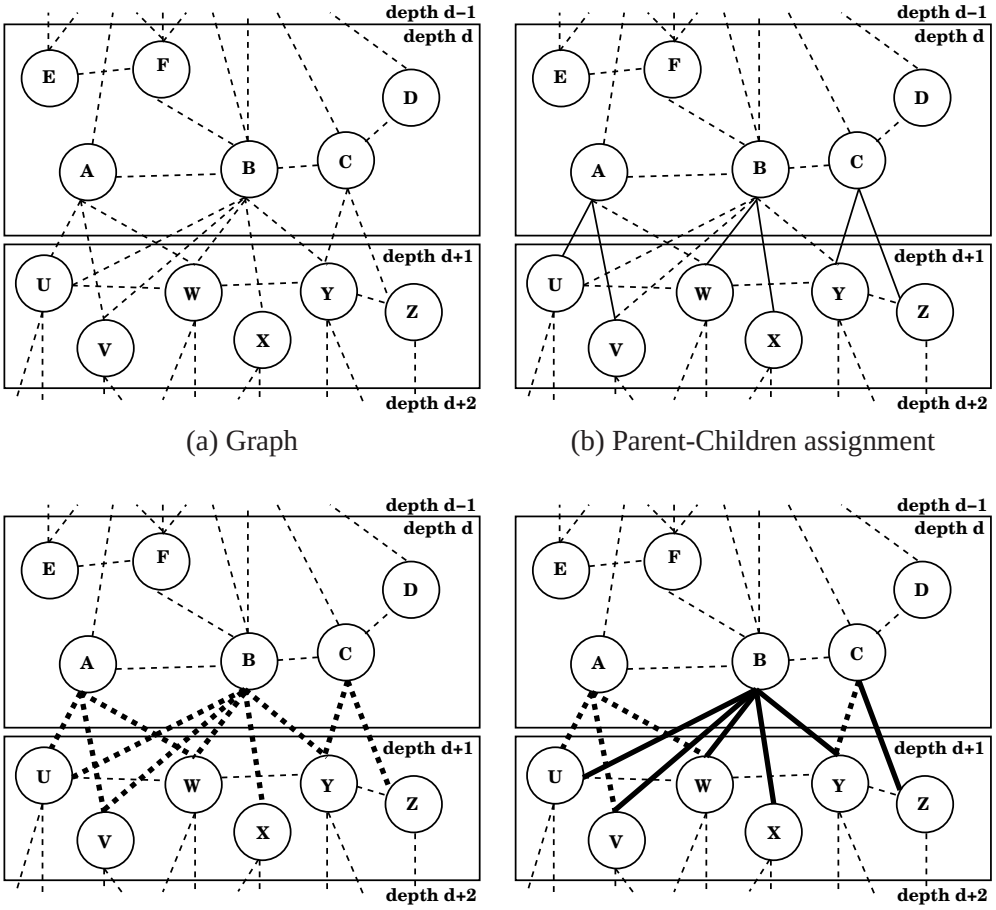
Lemma 1. *Given an undirected graph $G(V, E)$ and a particular vertex, $v \in V$, a lower bound on broadcasting by v, T_{min}, is $max\{d_i : i = 1, 2, ...N\}$, where d_i is the shortest-distance of node i from v in the graph.*

Proof. Let $k \in V$ be a node that has the maximum shortest-distance from v in the given graph, i.e., d_k = $max\{d_i : i = 1, 2, ...N\}$. We will prove by contradiction that $T_{min} \geq max\{d_i : i = 1, 2, ...N\}$ for any given graph.

Assume that $T_{min} < max\{d_i : i = 1, 2, ...N\}$. Recall d_k = $max\{d_i : i = 1, 2, ...N\}$, therefore, $T_{min} < d_k$. However, that is an impossible result as node k will require at least d_k transmissions (by v and possibly some other intermediate nodes) to receive the message. It means that our assumption $T_{min} < max\{d_i : i = 1, 2, ...N\}$ must be incorrect. Therefore, $T_{min} \geq max\{d_i : i = 1, 2, ...N\}$, and hence the proof.

□

We note that a tree construction algorithm, whether for an SPT, a DST or any other kind of tree, ought to be guided by the potential it has for using fewer transmissions for broadcasting. Until now, the tree construction algorithms produced a tree based on topological properties alone, e.g., by finding a Maximal Independent Set (MIS) of a graph [69]. To the best of our knowledge, we are the first to perform the broadcast tree construction in a manner that is "guided" by the lower bound as established in Lemma 1. Intuitively, this



(a) Graph (b) Parent-Children assignment

(c) Bipartite graph (d) Biased matching

Figure 3.2: Parent-children assignment during SPT construction. Rectangles contain nodes at a particular depth (from the root) of the tree under construction. Dashed lines represent the graph (adjacency) edges. Solid lines represent parent-children assignments and also represent potential edges that can be selected in the tree construction. Dashed very-thick lines represent the edges in the bipartite graph formed between two consecutive depths of the tree. Solid very-thick lines represent the optimal semi-matching of the bipartite graph.

approach has the potential to result in a tree that requires the smallest possible number of transmissions needed for broadcasting.

As discussed previously, a tree with fewer non-leaf nodes or conversely with more leaf nodes is better for reducing the number of transmissions needed for broadcasting (recall Definition 2). Therefore, our primary objective in this chapter is to construct a tree that is not only "leafy", but which also meets the criterion as set forth in Lemma 1. To this end, and in contrast to the existing approaches, our solution specifically targets at constructing a leafy tree that also "relaxes" the dependencies of the logical tree, i.e., $max\{d_i : i = 1, 2, ...N\}$. Next, we describe our procedure for constructing a leafy broadcast tree based on the bound established by Lemma 1.

3.4 Biased Shortest Path Tree

It is easy to understand that any SPT constructed from a given graph will minimize the lower bound, T_{min} , established in Lemma 1. However, not all SPTs may have the same number of non-leaf nodes. Therefore, our problem now is to construct an SPT having a minimum possible number of non-leaf nodes. More formally, if ξ_i represents the set of children of node i in a given logical tree, then the problem is to construct an SPT that has the maximum number of leaf nodes, i.e., nodes having $|\xi_i| = 0$. Our intuition behind constructing such a logical structure is that a logical tree that minimizes the lower bound as established by Lemma 1, and that also has the maximum number of leaf nodes may potentially use a smaller number of transmissions for broadcasting.

For every node i, d_i can be minimized by ensuring that every node is connected to the root using a shortest path, i.e., by constructing an SPT. A shortest path tree can be constructed using a standard Breadth First Search (BFS) algorithm. However, minimizing the total number of non-leaf nodes in SPT is non-trivial. Consider the scenario of a shortest path tree construction as shown in Figure 3.2. A set of 12 nodes have been shown in Figure 3.2(a) at two consecutive "depths", d and $d + 1$, of the tree. In particular, nodes from A to F are at distance d, and nodes from U to Z are at distance $d + 1$ from the root (not shown in the figures). A possible scenario of parent-children assignment is shown in Figure 3.2(b). In this example $|\xi_A|$ = $|\xi_B|$ = $|\xi_C|$ = 2. However, this parent-children assignment has resulted in nodes A, B and C being non-leaf nodes. In other words this parent-children assignment has not created any leaf node in the tree. We argue to perform parent-children assignments in such a way that the number of leaf nodes can be maximized. The reason being that by maximizing the number of leaf nodes, we can naturally minimize the number of non-leaf nodes. To that end, our goal is to maximize the "load" on specific parents during parent-children assignments so that other potential parents can be "freed" to become leaf nodes. In general, we have the following sub-problem that needs to be solved.

Definition 3. *Assuming $\mathcal{C}_d$ represents the set of nodes that are at distance d from the root, the parent-children assignment problem is to assign every node at depth $d + 1$, $\mathcal{C}_{d+1}$, a parent from the nodes at depth d, $\mathcal{C}_d$, such that $max\{|\xi_k| : \forall k \in \mathcal{C}_d\}$ is maximum.*

Interestingly, the parent-children assignment problem as defined above is equivalent to the problem of finding maximum-load semi-matchings in bipartite graphs [2] [26]. A bipartite

[2]Harvey et. al. propose solution for optimal semi-matchings that *minimizes* the load in semi-matchings (with respect to L_∞ norm). However, our goal is to *maximize* the semi-matchings load.

Algorithm 3

```
1:  procedure FINDMAXLOADSEMIMATCHING(P, C, E)
2:      E' = ∅;
3:      C' = C;
4:      repeat
5:          for all p ∈ P do
6:              ξ_p ← ∅;
7:              ξ_p = C' ∩ N_p;
8:              W(p) = |ξ_p|;
9:          p_max = FindMaxWeightParent(P, W);
10:         for all c ∈ ξ_{p_max} do
11:             E' = E' ∪ {(p_max, c)};
12:         C' = C' \ ξ_{p_max};
13:     until C' = ∅
14:     return (E');
```

graph formed by the nodes at depth d, i.e., $\mathcal{C}_d$ = {A, B, C}, and nodes at depth $d+1$, i.e., $\mathcal{C}_{d+1}$ = {U, V, W, X, Y, Z}, is shown in Figure 3.2(c). (Since nodes D, E and F do not have any neighbors from the nodes at depth $d+1$, they cannot become parents during the SPT construction and hence they are ignored. However, they will eventually become leaves of the SPT). A maximum-load semi-matching is shown in Figure 3.2(d). In this particular example, $|\xi_A|$ = 0, $|\xi_B|$ = 5, and $|\xi_C|$ = 1. It is optimal with respect to the maximum load assigned to a parent among all possible parents, i.e., $max\{|\xi_A|, |\xi_B|, |\xi_C|\}$ = 5 is maximum. Note that this assignment has resulted in node A being a leaf node. This is in contrast to the parent-children assignment shown in Figure 3.2(b) in which $|\xi_A| = |\xi_B| = |\xi_C|$ = 2, and $max\{|\xi_A|, |\xi_B|, |\xi_C|\}$ = 2, due to which node A did not become a leaf node.

Using the basic idea depicted in Figure 3.2(d) we construct a "special" SPT in which, at every two consecutive depths, maximum-load semi-matchings are obtained by constructing bipartite graphs with nodes at those consecutive depths. Of course that results in the maximization of $max\{|\xi_k| : \forall k \in \mathcal{C}_d\}$ at every depth d of the SPT. Intuitively, maximizing $max\{|\xi_k| : \forall k \in \mathcal{C}_d\}$ will result in the increased number of leaf nodes at every depth of the SPT. We call an SPT for which the parent-children assignments are done using maximum-load semi-matchings as a BIased SPT (BISPT).

The pseudo-code for finding the semi-matchings (with maximum load) in bipartite graphs is presented in Algorithm 3. FindMaxLoadSemiMatching takes as input a bipartite graph $G(P, C, E)$ and returns a subgraph $G'(P, C, E')$ containing semi-matchings. Shortly we will prove that FindMaxLoadSemiMatching produces an optimal solution for finding the semi-matchings having a maximum load, a problem that we will refer as *Maximum-*

Algorithm 4

```
1:  procedure CONSTRUCTBISPT(V, E, r)
2:      P ← {r};
3:      E' ← ∅;
4:      for all v ∈ V do
5:          Mark(v) = False;
6:      repeat
7:          C ← ∅;
8:          for all m ∈ P do
9:              Mark(m) = True;
10:         for all m ∈ P do
11:             for all n ∈ N_m do
12:                 if Mark(n) = False then
13:                     C = C ∪ {n};
14:         G_b ← BipartiteGraph(P, C);
15:         Z ← FindMaxLoadSemiMatching(G_b);
16:         E' ← E' ∪ Z;
17:         P ← C;
18:     until P = ∅;
19:     return (E');
```

Load Semi-Matching problem. The procedure starts by assigning every node, p, from P (which are the potential parents) with a weight that is equivalent to the total number of neighbors, i.e., N_p, that p has in the bipartite graph (lines 5 through 8). (Recall the property of bipartite graphs that $N_p \in C : \forall p \in P$, i.e., all neighbors of a node from set P are in set C). The parent with the maximum weight is then chosen and all its neighbors are assigned as children to that parent (lines 10-11). The assigned children are then removed from the bipartite graph (line 12). This procedure continues until all children are assigned (line 13).

Theorem 3. *Given a bipartite graph, G(P, C, E), FindMaxLoadSemiMatching produces an optimal solution for the Maximum-Load Semi-Matching problem.*

Proof. The very first node chosen by the procedure for children assignment is the node with the maximum number of neighbors in the bipartite graph. (Recall the weight assignment based on the degree of nodes.) Let $\Delta(p_{max})$ be the degree (and its weight) of the first chosen node. It is trivial to see that $\Delta(p_{max}) = \Delta(G)$. Since the maximum possible load for the semi-matchings cannot be greater than $\Delta(G)$, FindMaxLoadSemiMatching is guaranteed to find the semi-matchings having the maximum load, and hence the proof. □

The pseudo-code for BISPT construction is shown in Algorithm 4. It is essentially a breadth first search algorithm with two new additions: (i) a bipartite graph is created from

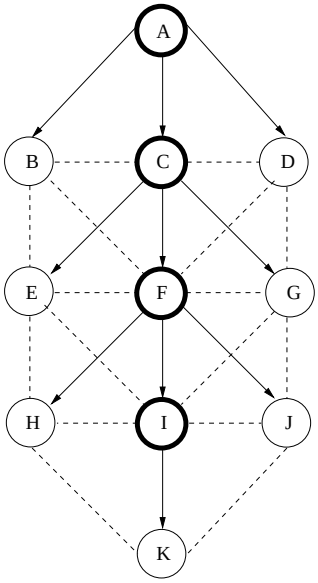


Figure 3.3: A BISPT constructed by the procedure ConstructBISPT. Solid arrowed lines represent edges of the logical tree. Dashed lines represent edges that are in the graph but not in the logical tree.

the nodes of two consecutive depths of the tree as shown at line 14, and (ii) a maximum-load semi-matching is found for the corresponding bipartite graph using the FindMaxLoadSemi-Matching procedure at line 15. Parent-children assignments obtained through the semi-matchings, which eventually become edges of the desired tree, are then added in the edge set of the tree under construction at line 16. This procedure is repeated until all possible parents are exhausted (line 18). Finally the desired tree, BISPT, is produced at line 19.

A BISPT produced by the ConstructBISPT procedure is shown in Figure 3.3. First notice that the constructed BISPT is indeed an SPT, i.e., every node in this tree is connected to the root using a shortest path. Nodes $\{$C, F, I$\}$ constitute the non-leaf nodes of the BISPT, which also forms the MCDS. Notice the difference between the DST shown in Figure 3.1(b) and the BISPT shown in Figure 3.3. In the DST some paths connecting the nodes to the root are not optimal, e.g., see nodes $\{$B, D, E, G, H, J$\}$. We will discuss the importance of this observation during the discussion on convergecasting in Chapter 5.

3.5 Performance Evaluation

To evaluate our proposal we implemented the solutions based on the proposals in [69] and more recently in [23]. In the rest of the section we will refer to the two solutions proposed in [69] and [23] as WAF and GKLRW, respectively, named after the authors' last names.

Parameter	Values
N (number of nodes)	100, 200, **300**, 400, 500 (Synthetic) 54 (Intel)
L (length of the square area [m])	800, 1000, **1200**, 1400, 1600 (Synthetic) 50 (Intel)
ω (transmission range [m])	200 (Synthetic) 8, 10, **12**, 14 16 (Intel)

Table 3.1: Parameter values used in this chapter (default values are shown in bold face).

WAF is a two phase approach for constructing a connected dominating set. In the first phase an MIS is chosen from the given graph and then a spanning tree is constructed to connect the nodes of the chosen MIS. Non-leaf nodes of the constructed tree constitute a connected dominating set. More recently GKLRW proposed an algorithm for constructing a broadcast tree which can eventually be used for broadcast scheduling, a topic discussed in Chapter 4. For the purpose of our study it is sufficient to use the broadcast tree proposed by GKLRW to evaluate the performance of BISPT.

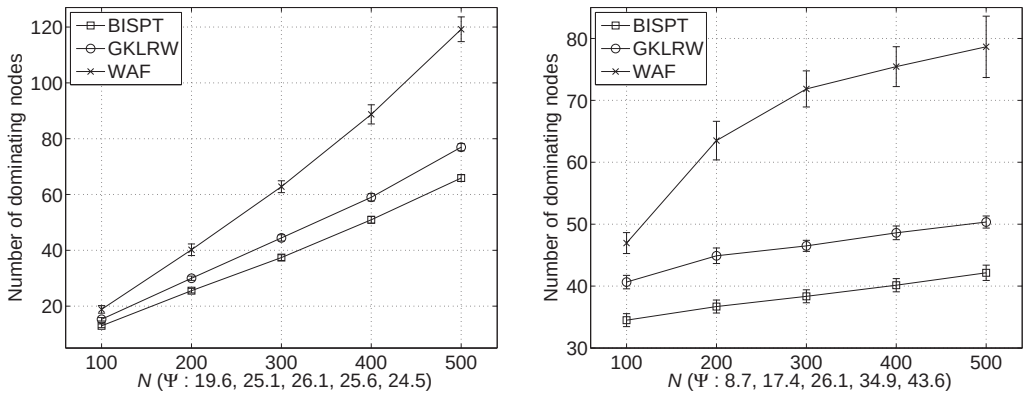
In our study we considered a similar setup as the one presented in [23]. In particular, the authors considered four different setups, i.e., 100, 200, 300, and 400 nodes in a 800m$\times$800m, 1000m$\times$1000m, 1200m$\times$1200m, and 1400m$\times$1400m area, respectively. Transmission range of nodes was fixed at 200m. To test the scalability of the solutions to a larger network we added one additional setup, i.e., 500 nodes in a 1600m$\times$1600m area. We also used the Intel setup discussed in the previous chapter to evaluate our solutions.

We keep track of density, $\Psi = \frac{\pi\omega^2 N}{L^2}$, where N is the number of nodes, ω is the transmission range of nodes and L is the length of a square area. Table 3.1 summarizes the values used for all the parameters used in our experiments in this chapter. All the results from the synthetic and Intel setups are an average of 20 simulation runs. In each run one node is chosen randomly uniformly as the root.

3.5.1 Number of Dominating Nodes

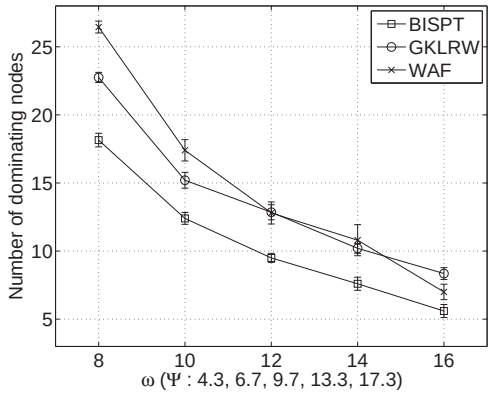
We use the *number of dominating (non-leaf) nodes*, which is basically the size of a connected dominating set (CDS) representing the number of non-leaf nodes of the logical tree, as the metric to evaluate the performance of various solutions, since the fewer the dominating nodes, the fewer the transmissions for broadcasting. In the following experiments we vary N, L and ω to note their impact on the size of the dominating set.

Figure 3.4 summarizes the results from the synthetic dataset. In the first experiment we vary the area size and the number of nodes to observe the scalability of the proposed



(a) Scalability (varying L and N) (b) Density (fixed L and varying N)

Figure 3.4: Performance of various solutions with the synthetic dataset.



Density (fixed L and varying ω)

Figure 3.5: Performance of various solutions with the Intel dataset.

solutions for which the results are presented in Figure 3.4(a). In the second experiment we keep the area fixed at 1200m$\times$1200m and change the number of nodes from 100 to 500. This experiment creates the scenario of an increasing node density in which an increasing number of nodes are "packed" in a fixed area. The results from this experiment are presented in Figure 3.4(b). The foremost trend that can be observed from the results obtained through these experiments is that BISPT's approximation of MCDS is better than all other solutions. In particular, the size of the CDS produced by BISPT is 5 to 20% less than the size of the CDS produced by other solutions.

As shown in Figure 3.4(a), when the network scales up (in terms of the area and the number of nodes), the size of the CDS increases as expected. The increase is much faster in the case of WAF as compared to GKLRW and BISPT. Nevertheless, the performance of BISPT with respect to GKLRW increases as the network scales up. In the case of the

increasing node density, the CDS size increases as well, but by a smaller margin. The reason is that when more nodes are "packed" in a fixed area the size of the underlying tree grows. However, that does not necessarily increase the number of dominating nodes as many of the nodes become leaf nodes in the logical tree.

Figure 3.5 summarizes the results from the Intel dataset. The main change in these results with respect to the synthetic setup is that the trend of the performance curves is reversed. The reason is that in the case of the Intel setup we can only vary the transmission range to change the node density. Due to this the number of neighbors per node is increased. That results in a "short" and "fat" underlying tree due to which the size of the CDS (number of dominating nodes) is reduced to a trivially small value.

3.6 Conclusions

In this chapter we proposed BISPT, a logical tree topology for efficient broadcasting in WSNs. Simulation results reveal that BISPT outperforms other well known solutions in terms of better approximation for MCDS or MDST. It also means that BISPT is a tree with a smaller set of non-leaf nodes, which will incur lower transmission cost for broadcasting. Clearly this will reduce the query processing cost for solutions that require broadcasting as an elementary operation, e.g., recall EXTOK and FILA. In the next chapter, we propose solutions for scheduling broadcast. It is a problem in its own right, which needs to take into account the unique features of a WSN, mainly its distributed environment and the wireless nature of nodes' transmissions.

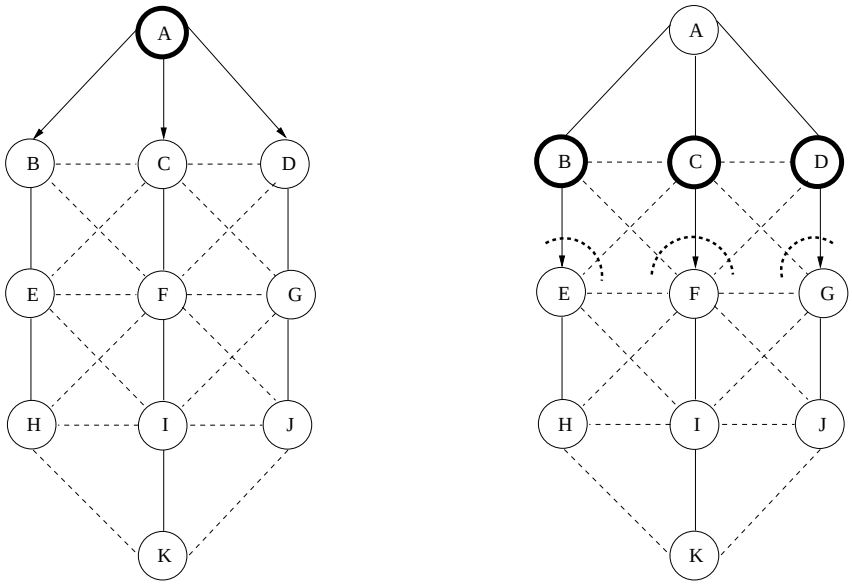
Chapter 4

Broadcast Scheduling

4.1 Introduction

In the previous chapter we studied the problem of broadcasting in WSNs. In particular, we highlighted the importance of an underlying logical tree topology during the broadcasting operation. We proposed a simple solution for constructing an efficient logical tree topology that is more suitable than the existing logical tree topologies for broadcasting. Essentially, the problem of broadcasting is the problem of determining which nodes in the network should repeat (relay) a message and which ones should not, such that all nodes receive a message sent by a particular node. The former are called *non-leaf nodes* (or *relays*) and the latter are called *leaf nodes* in recognition of the fact that the dependency of transmissions forms a logical tree. (A non-leaf node cannot transmit before it receives the message from another "upstream" non-leaf node, i.e., its parent in the logical tree.) The objective here is to minimize the number of transmissions, hence the number of non-leaf nodes of the logical tree topology. This objective led to the BISPT algorithm presented in the previous chapter.

An obvious problem now is to efficiently schedule a given logical tree, e.g., BISPT, which is the main focus of this chapter. Determining the logical tree is not sufficient because it does not determine *when* the non-leaf nodes need to transmit. Consider the scenario of broadcasting using a commonly used logical tree topology, SPT, as depicted in Figure 4.1. As shown in Figure 4.1(a), the root starts broadcasting by transmitting a message which is received by the root's children $\{$B, C, D$\}$. Next, the root's children are required to forward the message. As all of the root's children are ready to transmit the message, possibly at the same time, there may be interference if they transmit the message concurrently. In the worst case nodes $\{$B, C, D$\}$ transmit the message simultaneously as shown in Figure 4.1(b). Because of the wireless medium, transmissions from nodes $\{$B, C, D$\}$ interfere with each other causing "collisions" at the receiving nodes $\{$E, F, G$\}$ due to which none of the nodes



(a) Transmission from the root. (No collisions at root's children B, C, D)

(b) Transmission from the root's children. (Collisions at nodes E, F, G marked by arcs.)

Figure 4.1: Interference in SPT during broadcasting. (Solid lines represent edges of the logical tree. Arrowed lines coming out from a node represent a single transmission from that node. Dashed lines represent edges that are in the graph but not in the logical tree. Highlighted circles represent nodes that are transmitting a message.)

from $\{$E, F, G$\}$ are able to receive the message from their respective parents.

One solution here is to rely on an underlying Medium Access Protocol (MAC) such as CSMA/CA [66] to handle contention and re-transmissions in the event of collisions. The main problem with a contention-based MAC protocol is that nodes do not know when to expect the messages that are *intended* for them. In this situation, the transceivers need to be turned on to continuously *listen* for the intended messages, which may consume a significant amount of the nodes' energy. There exist compelling reasons to avoid a contention-based MAC protocol, and to create a time schedule instead, e.g., a Time Division Multiple Access (TDMA) like protocol [66]. More specifically: (*a*) it makes no sense to try to minimize the number of transmissions which, among other things, reduces the latency to complete the broadcast, only to introduce latency caused by contention arbitration mechanisms, but more importantly, (*b*) we are interested to reduce the time a node needs to listen on the medium and hence waste energy while idle listening [29]. In other words, a schedule allows us to prescribe *exactly* when a node has to listen and when it has to transmit (if at all), keeping its transceiver inactive at all other times. Naturally, the schedule must also ensure that, (*c*) there are no collisions (from the perspective of each intended recipient) caused by transmissions scheduled to occur in the slot that the recipient is listening.

There are two kinds of solutions that exist for TDMA-based broadcast scheduling. A common approach of most of these solutions [23] is to decompose the problem into two independent subproblems: (i) a logical tree construction, and (ii) scheduling of transmissions along the constructed tree. Other existing solutions, e.g., [76], propose a tree-scheduling algorithm alone, which obviously needs an already constructed tree as input. One common objective of both kinds of solutions is to reduce the broadcast latency, i.e., the number of time slots used for scheduling the broadcast. Before further discussion, we describe the broadcast scheduling problem in detail.

4.2 Problem Statement

Throughout this thesis, we assume a slotted system and that all nodes have adequate synchronization capabilities to follow a slot-by-slot schedule. The purpose of the broadcast (and convergecast, discussed in Chapter 5) schedule is to instruct each node when to: receive, transmit, or deactivate the transceiver (thus avoiding idle listening). The schedule is constructed for a single round/epoch and then the same schedule is repeated for each round. This mode of operation is consistent with the execution model of continuous queries and their solutions, e.g., TAG, FILA and EXTOK. Extensions to multiple (pre-computed) schedules are possible but beyond the scope of this thesis.

Given a network of N nodes, the problem is to minimize the total number of time slots, T, that are required for broadcast scheduling, subject to the following constraints:

- *C1:* only a subset of nodes, including the sink, are allocated one of the time slots from the set $\{1,2,...,T\}$ to transmit the broadcast message only once.
- *C2:* every node (except the sink) is allocated one of the time slots from the set $\{1,2,...,T\}$ to receive the broadcast message only once. (The sink is assumed to be the originator of the message, therefore, it need not be scheduled to receive).
- *C3:* the reception slot of any transmitting node (except the sink) is earlier than its transmission slot, i.e., all transmitting nodes (except the sink) are scheduled to receive before they are actually scheduled to transmit. This also means that the reception and transmission slots of any transmitting node cannot be the same, i.e., every transmitting node is scheduled to receive and transmit using two different slots to enforce the half–duplex operation, i.e., a node can either receive or transmit during a given slot.

- *C4:* there is no interference by colliding transmissions at any of the recipient nodes during its assigned reception slot.

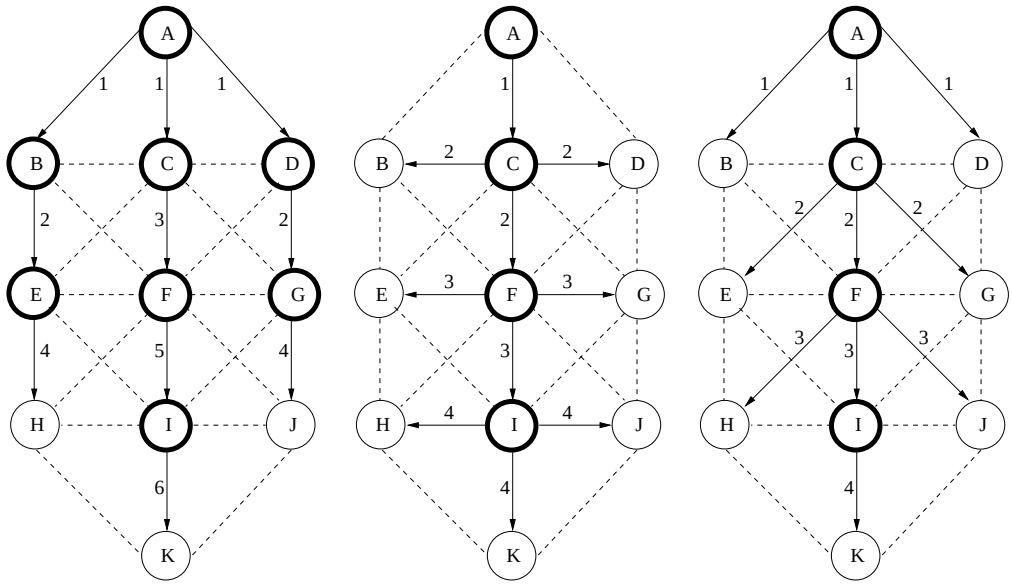
Unfortunately, the problem of finding a schedule that satisfy all the above constraints has been shown to be NP-Complete by Gandhi et. al. in [24]. Therefore, the existing solutions are heuristic approximations, which are reviewed in the next section.

4.3 Related Work

Broadcast scheduling requires resolving interference introduced by the physical (wireless) topology of the network while minimizing the broadcast latency, i.e., the number of slots that are required to schedule the transmissions. Several heuristic solutions have been proposed in the literature to solve the above NP-Complete problem. In particular, Gandhi et. al. proposed their solution that achieved a constant approximation, i.e., the performance ratio of their proposed solution with respect to an optimal solution is constant. Unfortunately, their broadcast latency approximation bound is greater than 400, which is quite high. Recently they proposed another solution in [23], which improved the approximation ratio to 12. Huang et. al. also proposed a 16-approximation algorithm in [31]. However, as noted in [23] their algorithm has a hidden cost in the order of $O(R)$, where R is the longest shortest path from the sink in the network. Chen et. al. proposed yet another constant approximation algorithm in [14], which has been found to be at best 16-approximation only in special cases [23].

Existing solutions typically take the approach of first constructing a logical tree topology and then scheduling the nodes of the constructed tree. Some other solutions only focus on minimizing the broadcast latency through efficiently scheduling the nodes of a logical tree. All of the solutions proposed in [23, 24, 31, 76] take one of the two approaches mentioned above. In particular, [23] proposes a solution that contain a tree construction phase and a scheduling phase, and [76] only proposes a tree scheduling algorithm. In yet another study [78], Zadoronzhny et. al. propose solutions for constructing logical trees for query processing and data delivery in mobile sensor networks.

Our work is different from [23, 24, 31, 76] in many ways. First, the scheduling solutions proposed in these works are not independent of tree construction, due to which their usefulness is not obvious in certain situations (discussed in section 4.5). We make the tree scheduling independent of the tree construction, while still paying due attention to the quality of the constructed tree in terms of the number of time slots that the tree may use for



(a) SPT (b) DST (c) BISPT

Figure 4.2: Various logical trees and their corresponding schedules. (Solid arrowed lines represent edges of the logical tree. Dashed lines represent edges that are in the graph but not in the logical tree. Arrowed lines also represent messages with the distinction that all arrowed lines coming out from a node represent a single transmission from that node. Each arrowed edge is also annotated with a time slot in which it will be activated.)

broadcasting (examples to follow.) Towards that end our scheduling algorithm tends to maximize the number of concurrent transmissions in order to reduce the number of time slots used. This is in contrast with approaches, such as the one in [76], where the schedule is constructed through a traversal of the supplied tree without any concern as to whether such a traversal allows the maximum number of concurrent transmissions to be scheduled in every time slot. Yet another important difference, which distinguishes our work from [23, 24, 31, 76], is that unlike previous solutions we strive to make broadcasting more reliable in the event of failures which are common in WSNs, an issue that we will address in detail in Chapter 6.

4.4 Proposed Solution

Our tree usage for broadcast scheduling is marked by the observation that only the root and non-leaf nodes of a logical tree need to be scheduled as only these nodes in the logical tree need to be the transmitters and all leaf nodes need to be the recipients only. (Of course all nodes, except the root, need to be scheduled for reception as well.) Intuitively, a smaller set of non-leaf nodes will require a lesser number of time slots for scheduling the transmissions.

Therefore, the problem is to ensure that only the smallest possible set of non-leaf nodes transmit the message.

It is trivial to see that the BISPT presented in the previous chapter meets the above criterion. To put forth our case more concretely we consider the examples of various logical trees, BISPT as well as SPT and DST, and their corresponding schedules as shown in Figure 4.2. Clearly the schedules meet all the constraints C1 through C4 set forth in section 4.2. The total length of the broadcast schedule, T, for the SPT is 6 with nodes $\{$A, B, C, D, E, F, G, I$\}$ scheduled to transmit the message. Note that the set of nodes $\{$B , D$\}$ and $\{$E, G$\}$ are scheduled for concurrent transmissions during slots 2 and 4, respectively, as doing so does not violate the collision-free constraint. In the case of the DST only nodes $\{$A, C, F, I$\}$ are scheduled to transmit during slots 1, 2, 3 and 4, respectively, as shown in Figure 4.2(b). The total length of the schedule in this case is 4 as compared to 6 in the case of the SPT. Similarly, the total length of the schedule used by the BISPT is 4 as shown in Figure 4.2(c). Clearly in these examples of the DST and BISPT, which have a smaller set of non-leaf nodes as compared to the SPT, indeed a lesser number of slots are used for scheduling. Now an obvious question is, after forming a logical tree topology, how to construct a schedule that meets the constraints set forth in in section 4.2. Towards that end we present the Weighted Incremental ScHeduling (WISH) algorithm for broadcasting.

4.4.1 WISH for Broadcast Scheduling

WISH takes as input a spanning tree over a graph $G(V, E)$ which represents the logical topology over which broadcasting is to be performed. Let us denote by r ($r \in V$) the sink and the root of the broadcast tree. For convenience, let us denote as V' all nodes of V except for r, i.e., $V' = V \setminus \{r\}$. The tree can be described by the parent-of relation denoted by P_v which indicates which node is the parent of node v. The set of parent-of relations $\mathcal{P}$ is defined as $\mathcal{P} = \{P_v | v \in V'\}$. The inverse of the parent-of relation is the children-of set, ξ_v, which indicates the set of children of node v (as per the logical tree topology), that is $\xi_v = \{u | P_u = v \wedge u \in V'\}$. Finally, let us capture the *physical* topology of $G(V, E)$ through the neighborhood set, N_v, that includes all nodes connected to node v in the physical topology, i.e., $N_v = \{u | (v, u) \in E\}$. Collectively, the set of neighbor sets for all nodes in V' will be denoted by $\mathcal{N}$, i.e., $\mathcal{N} = \{N_v | v \in V'\}$. The pseudocode for WISH is presented in Algorithm 5.

The algorithm maintains a set of nodes that are eligible to transmit, denoted by F. However, only a subset of nodes in F will be allowed to transmit concurrently because

Algorithm 5

```
1:  procedure WISH(r, V', 𝒩, 𝒫)
2:      j ← 0;
3:      F ← {r};
4:      repeat
5:          j ← j + 1;
6:          for all e ∈ F do
7:              w_e ← |ξ_e|;
8:          S_j ← ∅;
9:          R_j ← ∅;
10:         for all e ∈ F in decreasing w_e do
11:             NoCollA ← true;
12:             NoCollB ← true;
13:             for all s ∈ S_j do
14:                 for all c ∈ ξ_e do
15:                     if c ∈ N_s then
16:                         NoCollA ← false; break;
17:             if NoCollA = true then
18:                 for all r ∈ R_j do
19:                     if r ∈ N_e then
20:                         NoCollB ← false; break;
21:             if NoCollA ∧ NoCollB then
22:                 S_j ← S_j ∪ e;
23:                 R_j ← R_j ∪ ξ_e;
24:         F ← ∅;
25:         for all q ∈ ∪_{i=1}^{j} R_i do
26:             if q ∉ ∪_{i=1}^{j} S_i then
27:                 if |ξ_q| > 0 then
28:                     F ← F ∪ {q};
29:     until |∪_{i=1}^{j} R_i| = |V'|
30:     𝒮 ← {S_v | v ∈ {1, ..., j}};
31:     return (𝒮);
```

they need to satisfy the requirement that no collision is caused (from the viewpoint of the receivers) with any other transmissions happening in the same slot. Trivially, in the first slot only the root r is eligible to be scheduled (line 3). Subsequently all of its children become the recipients of its transmission and become eligible to transmit in the second slot but only a subset of them may be allowed to transmit concurrently.

The basic idea is to rank all eligible nodes based on their *weight* and then consider them in decreasing order based on their weight (line 10). We check whether the transmission of an eligible node, e, can be scheduled in the current slot in a collision-free manner. This test considers two possible types of collisions: (A) collisions perceived by *any* child of e

due to concurrent transmission in the same slot from a node already scheduled in the same slot (lines 13-16), and (B) collisions that could be caused if e transmits in the current slot because it would collide at the children of nodes already assigned to transmit in the same slot (lines 17-20). If a node does not cause either type of collision, it is scheduled for transmission and its children are marked as receiving the node's transmission (lines 21-23). This procedure continues with the new set of eligible nodes (lines 24-28) until all nodes have received the transmission (line 29). WISH returns the sets of nodes that are assigned to transmit in each slot (lines 30-31).

The ability of this process to generate good schedules in terms of length depends on the way the weights are assigned. We have experimented with many alternative weights and the one that we have found as consistently producing good results is the cardinality of the children-of set, i.e., $|\xi_v|$ (lines 6-7). The higher weight gives a higher relative priority to a node to be scheduled in the current slot over other eligible nodes. The intuition behind why this works best is based on the observation that the earlier we schedule a node with many children, the larger the increase in the eligible set (because the children become subsequently eligible for scheduling). The larger the eligible set earlier in the schedule, the higher the chances that several eligible nodes can transmit concurrently. By enhancing the number of concurrent transmissions, we are reducing the number of slots it takes to schedule all the nodes.

Lemma 2. *A schedule produced by WISH is collision-free.*

Proof. Every slot in WISH is allocated in an incremental fashion, i.e., S_j and R_j are initialized to empty sets and then eligible transmitters and receivers are added into them incrementally. Since a node is added to S_j and its children to R_j only if it meets the collision-free criteria, S_j will contain the set of nodes that do not interfere with each other's transmissions. Because this procedure is repeated for every slot, $j \geq 1$, S_j and R_j will always contain nodes that are collision-free, and hence the proof. □

4.5 Performance Evaluation

To evaluate our proposal we implemented the solutions proposed in [23] and [76]. In the rest of the section we will refer to the two solutions proposed in [23] and [76] as GKLRW and YMV, respectively, named after the authors' last names. Note that the scheduling phase of GKLRW is not independent of its tree construction phase, therefore, GKLRW cannot be used for scheduling "any" tree that is given as input to the algorithm. In particular, GKLRW

Parameter	Values
N (# of nodes)	100, 200, **300**, 400, 500 (Synthetic) 54 (Intel)
L (length of the square area [m])	800, 1000, **1200**, 1400, 1600 (Synthetic) 50 (Intel)
ω (transmission range [m])	200 (Synthetic) 8, 10, **12**, 14 16 (Intel)

Table 4.1: Parameter values used in this chapter (default values are shown in bold face).

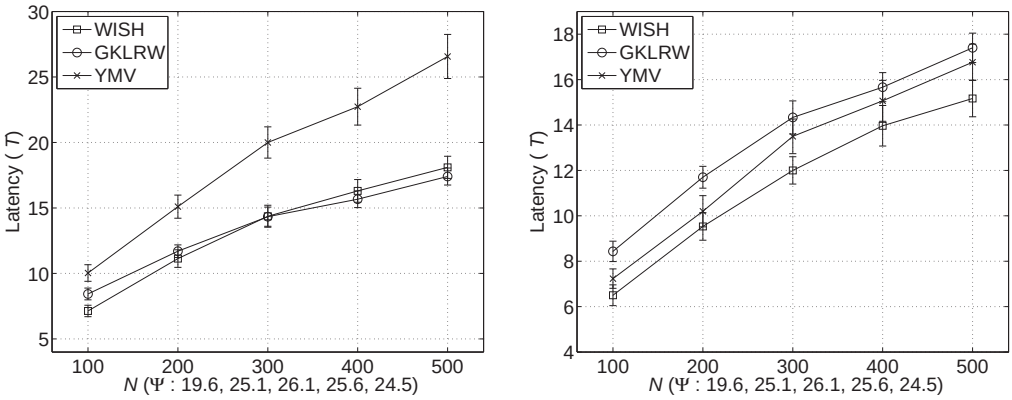
requires an SPT as an input. In contrast to that, YMV is a tree scheduling algorithm, which means that it is independent of the input tree, and can schedule any tree that is given as input. Unlike GKLRW and like YMV, our tree scheduling algorithm, WISH, is independent of the tree construction phase, which means it can be used to schedule any logical tree. This feature of a solution, to independently schedule a tree, is particularly useful in situations where a specific tree is given to be scheduled to achieve certain other application objectives. In our experiments we will use this feature of a solution to evaluate GKLRW, YMV and WISH using various scenarios.

GKLRW is currently the-state-of-the-art solution for broadcast scheduling. GKLRW has been shown to outperform solutions proposed previously in [24, 31]. Interestingly, YMV has not been compared against these solutions including GKLRW. Our simulation study "fills" this gap and evaluates GKLRW with respect to YMV as well as our own solutions. In the simulation study presented in this section, we considered a similar setup as presented in the previous chapter. However, for the reader's convenience we present the main details of the setup again.

Five different simulation setups are considered in this chapter. In particular, 100, 200, 300, 400 and 500 nodes in a 800m$\times$800m, 1000m$\times$1000m, 1200m$\times$1200m, 1400m$\times$1400m, and 1600m$\times$1600m area, respectively. Transmission range of nodes was fixed at 200m. We also used the Intel setup discussed in Chapter 2 to evaluate our solutions. We keep track of the node density, $\Psi = \frac{\pi\omega^2 N}{L^2}$, where N is the number of nodes, ω is the transmission range of nodes and L is the length of a square area. All results presented here are an average of 20 simulation runs. Table 4.1 presents a summary of the values that we used for different parameters in our experiments presented in this chapter.

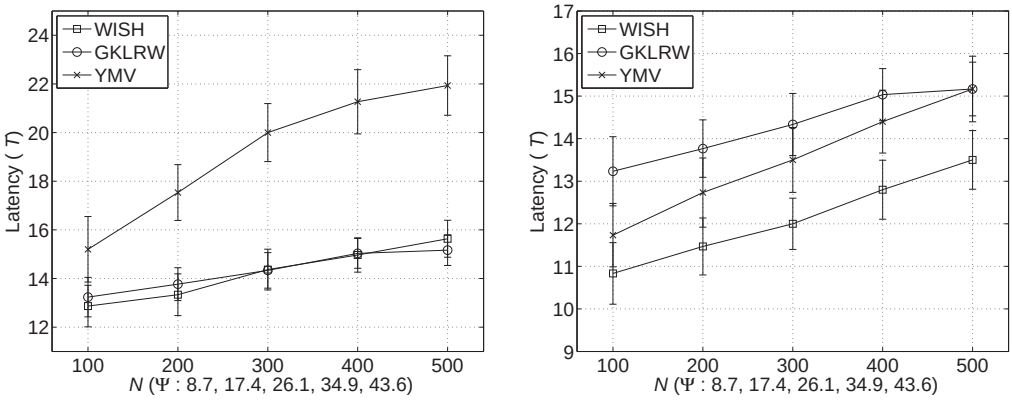
4.5.1 Broadcast Latency

We use *broadcast latency*, which is the total number of slots used for a given schedule, as a metric to evaluate the performance of various solutions. In the first set of experiments



(a) With GKLRW-tree for all solutions (b) With BISPT for WISH and YMV

Figure 4.3: Scalability (varying L and N) in the synthetic dataset.

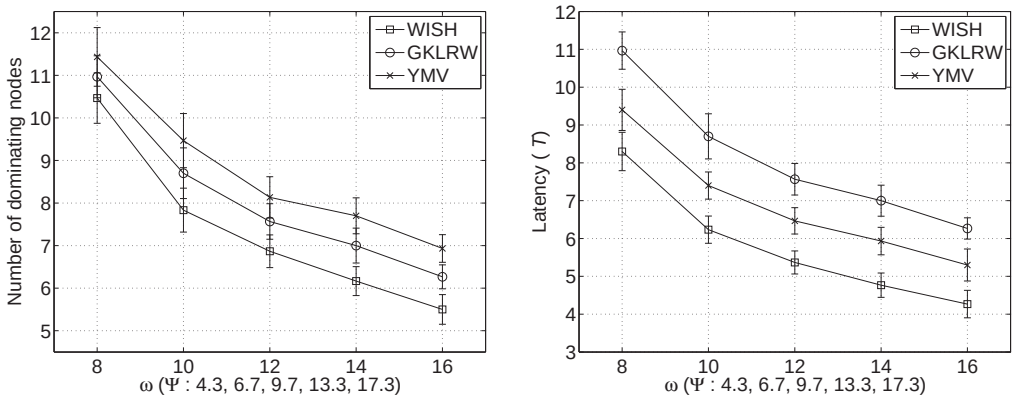


(a) With GKLRW-tree for all solutions (b) With BISPT for WISH and YMV

Figure 4.4: Density (fixed L and varying N) in the synthetic dataset.

we keep the underlying logical tree same for GKLRW, YMV and WISH. In particular, we constructed and used the same tree that is proposed for GKLRW. We will refer to the tree proposed in GKLRW as GKLRW-tree. Since YMV and WISH can independently schedule this tree, we can evaluate the latency of the schedule produced by these solutions independent of the underlying logical tree. The results are summarized in Figure 4.3(a). We can see that for a smaller network with less than 200 nodes WISH can outperform GKLRW by 5 to 15% and YMV by 15 to 30%. As the network scales up, the performance gap between WISH and GKLRW shrinks and becomes insignificant. However, the performance of YMV degrades quite rapidly when the network size is increased because, as we remarked in section 4.3, YMV misses opportunities to concurrently schedule nodes that are eligible in other parts of the tree because of its strict traversal strategy of the input tree.

In the second set of experiments we kept the original tree for GKLRW (as that is the only



(a) With GKLRW-tree for all solutions (b) With BISPT for WISH and YMV

Figure 4.5: Density (fixed L and varying ω) in the Intel dataset.

option it has) and supply WISH and YMV with BISPT as the underlying tree. Our results are summarized in Figure 4.3(b). Obviously the performance of GKLRW is not expected to change as compared to Figure 4.3(a) and indeed that is the case as shown in Figure 4.3(b). However, the performance of WISH and YMV is improved, though slightly for WISH and considerably for YMV. As a matter of fact, YMV with BISPT has outperformed GKLRW. This result highlights the importance of the underlying logical tree topology.

In the above set of experiments we tested the scalability of solutions by varying the network area and the number of nodes. In the next set of experiments we keep the network area fixed at 1200m$\times$1200m and vary number of nodes from 100 to 500. These experiments create the scenario of an increasing node density in which an increasing number of nodes is "packed" in a fixed area. The results from these experiments are presented in Figure 4.4. Similar trends can be noticed here as observed in Figure 4.3, i.e., WISH performs similar to GKLRW when supplied with the common logical tree topology. However, when BISPT is supplied as the input tree, the performance of WISH improves.

Another interesting trend that we can notice with WISH as well as GKLRW and YMV is that when the nodes are more "densely packed", the schedule length decreases. Compare, e.g., Figure 4.3(a) with Figure 4.4(a) and Figure 4.3(b) with Figure 4.4(b). Notice that WISH schedules 500 nodes that are spread over 1600m$\times$1600m area using 18 time slots (Figure 4.3(a)), however, when 500 nodes are deployed within 1200m$\times$1200m area the total slots used are reduced to less than 16 (Figure 4.4(a)). This behavior can be attributed to the fact that when nodes are densely packed the underlying logical tree "shrinks" while having a lesser number of non-leaf nodes, which can be scheduled using fewer slots.

Figure 4.5 summarizes the results from the Intel setup. The main difference in these

results with respect to the synthetic setup is that the trend of the performance curves is reversed. The reason is that in the case of the Intel setup we vary the transmission range due to which the underlying logical tree becomes "shorter" resulting in the decreased number of non-leaf nodes. Recall that only non-leaf nodes need to be scheduled for broadcasting. As the non-leaf nodes decrease it is natural that the total number of slots used for their scheduling is also decreased. There is not much difference among the performances of WISH, GKLRW and YMV as shown in Figures 4.5(a). However, an interesting trend, which did not appear in the synthetic setup, can now be seen in Figures 4.5(b). This result is from the setup in which we supply WISH and YMV a different tree than GKLRW. We can clearly see that GKLRW is outperformed not only by WISH but also by YMV. This result shows the sensitivity of GKLRW towards a particular type of network topology (as in the case of the Intel dataset that is restricted to produce a certain type of the logical tree topologies). Nonetheless, in all cases WISH outperformed all other solutions.

4.6 Conclusions

Broadcasting is an important operation in wireless networks. In this chapter we addressed the problem of scheduling *one-to-all* broadcasting. Towards that end we proposed an efficient broadcast scheduling algorithm, WISH, that outperformed the current state-of-the-art solution. Admittedly, the results show that the reduction in the scheduling length (for a single broadcast schedule) is marginal, however, overall that may account for a significant reduction if a query is processed for a large number of rounds, which could be the case in many applications. The time that is gained due to short schedules, i.e., by reducing the broadcast latency, may actually improve the response time of the sink that answers the query. In the next chapter we study another important operation of in-network query processing in WSNs, i.e., convergecasting.

Chapter 5

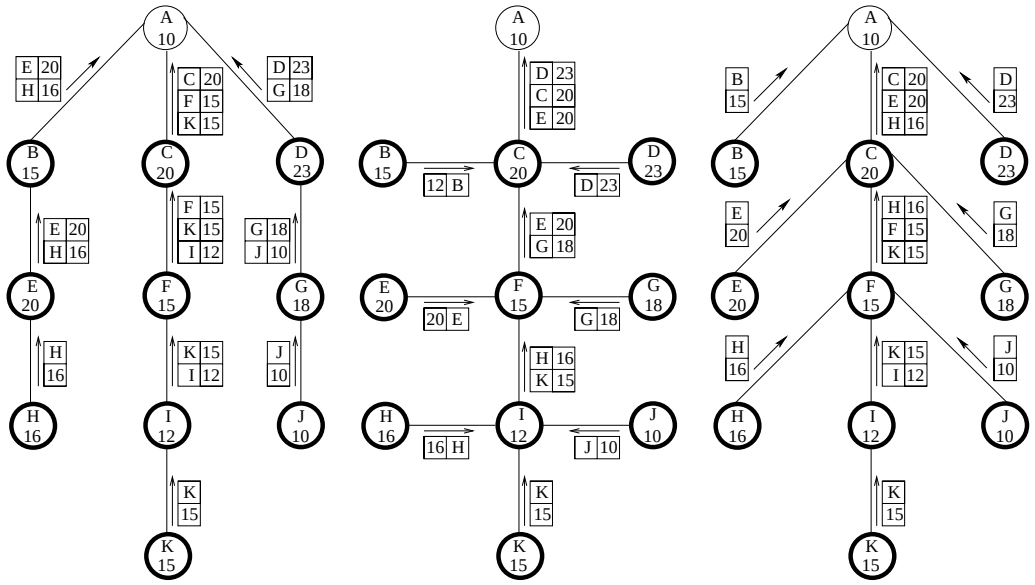
Convergecast Scheduling

5.1 Introduction

Data gathering is a basic capability expected of any WSN. In the context of in-network query processing the usual means of performing data gathering is to have all nodes send their measurements (possibly over multiple hops) to the sink. To that end a logical tree topology is used for collecting and forwarding data to the sink, which becomes the root of the logical tree. The corresponding many-to-one "funnel" type of communication is called *convergecast*. Convergecast is the basic building block for various solutions proposed in the literature for data gathering and query processing including TAG, FILA and EXTOK that we discussed in the previous chapters.

A scenario of convergecasting is presented in Figure 5.1 in which a top-2 query is processed using TAG on top of SPT, DST, and BISPT. Note that in TAG every node participates during the convergecast by forwarding its own value or aggregated values. It is trivial to see that, because of the different structures of the logical trees, the same query is processed differently. Hence, the convergecast cost (in terms of bytes transmitted/received by the nodes) is also different for different trees. For example, if a sensor's value and ID are represented by 2 bytes each, then the total bytes transmitted to find the top-2 results while using SPT, DST and BISPT are 76, 56 and 60, respectively. Similarly, the cost of processing EXTOK on top of SPT, DST, and BISPT may vary during any given round as shown in Figure 5.2.

Finding a tree that minimizes the convergecast cost is a non-trivial problem. Interestingly, this problem can be treated as a Minimum Steiner Tree (MST) problem, which is well known to be NP-Hard [39]. Filtering based solutions complicate the problem even further. Recall that in EXTOK the threshold value may change due to which F-nodes and TM-nodes may change accordingly during every round. It means that unlike the MST problem, *source* nodes cannot be determined in advance while using a filtering based solution. Basically,



(a) SPT (b) DST (c) BISPT

Figure 5.1: Convergecasting (TAG) using various logical topologies. (Darker circles represent nodes that triggered an update. Rectangles represent the packets transmitted by the corresponding nodes.)

a new instance of the MST problem is created during every round in the case of filtering based solutions. In this situation, constructing an optimal or near-optimal logical tree topology during every round is impractical and costly due to the overheads, which will become significant because of multiple rounds. Therefore in this thesis, we focus on convergecast scheduling rather than investigating logical tree topologies for convergecasting.

Several algorithms proposed for multi-hop wireless convergecast scheduling can be used for WSNs (see e.g., [37] and the references there in). A common objective of scheduling algorithms is to use the least number of time slots. As with broadcast scheduling, a common trait of most of those algorithms is the decomposition of the problem into two independent subproblems: first a logical tree construction, followed by the scheduling of transmissions along the constructed tree. We will see a similar approach is followed in *aggregation convergecast*, a problem that we address in this chapter.

5.2 Problem Statement

Aggregation convergecast is described as the routing and the en-route aggregation of data as they travel to the sink (plus of course interference constraints as in regular convergecast). Aggregation is a means to achieve energy efficiency by reducing the transmitted traffic volume, recall e.g., TAG. In its simplest definition, aggregation operates by ensuring that

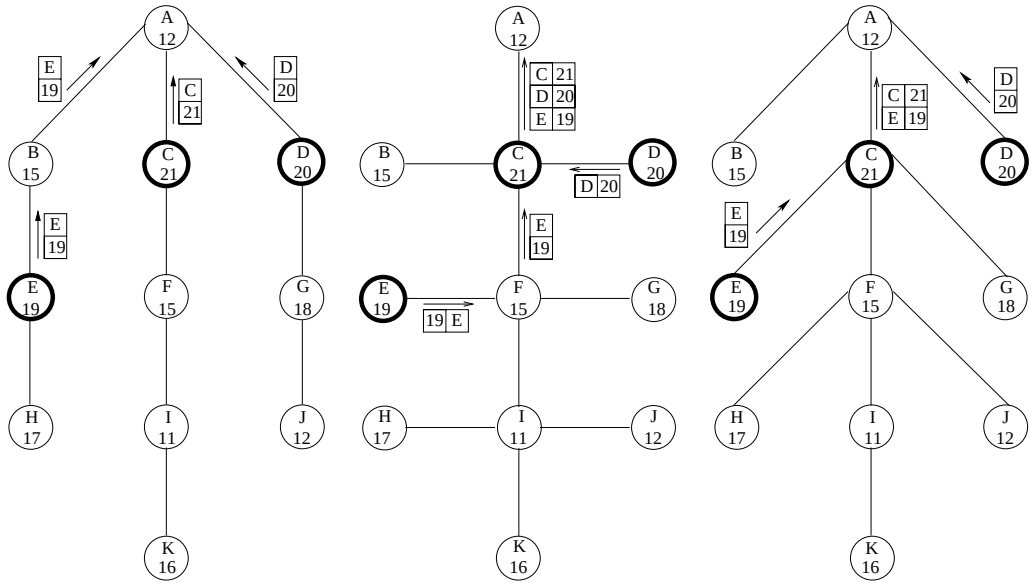


Figure 5.2: Convergecasting (EXTOK) using various logical topologies. (Darker circles represent nodes that triggered an update. Rectangles represent the packets transmitted by the corresponding nodes.)

a node receives a specific number of incoming messages (from a correspondingly specific subset of its neighbors), then combines the received data along with its own to generate a *single* output message that describes collectively the received and its own data together. Naturally, this definition can be applied recursively all the way to the sink.

The important characteristic of aggregation convergecast is that a node (except the root) transmits only once per round. This transmission can happen only after the node has collected data from other nodes on which to perform aggregation (including its own data). The aggregated data is subsequently received by another node that performs further aggregation along with those from other nodes, and so on, until the aggregated data arrives at the sink. We assume that a single slot period is sufficient to transmit data in its original (source) form or in its aggregated form. For example, in the case of top-k query processing some nodes may forward exactly k values and some nodes may forward less than k values depending on the number of values collected. The slot size should be tailored to "fit" the worst case size of k values (inclusive of header and other overheads). For the purpose of this thesis we will assume that all slots have the same fixed duration, which is sufficient to transmit/receive the *largest* packet/message required by the application.

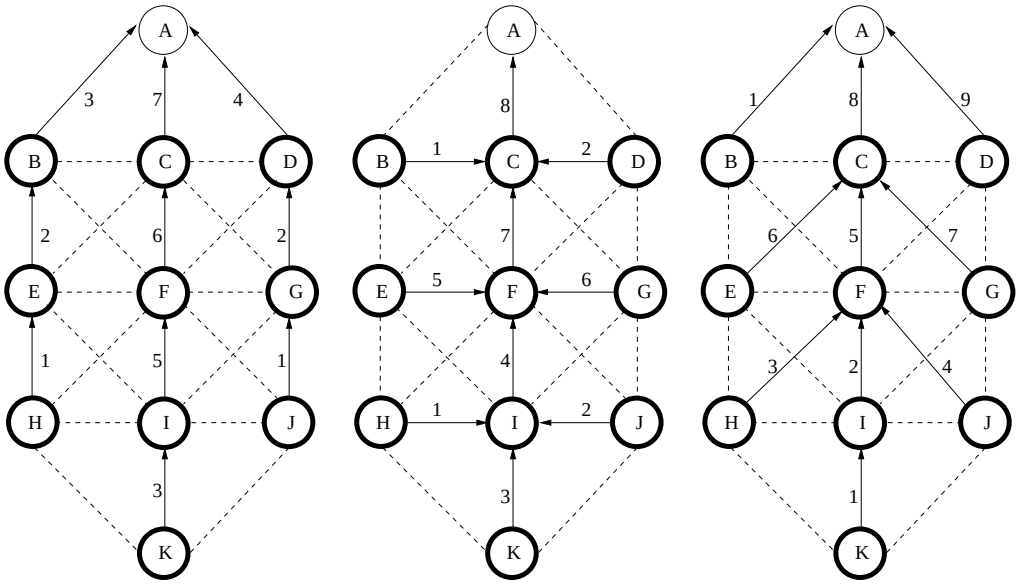
Given a network of N nodes, the problem is to minimize the total number of time slots, T, that are required to schedule transmissions, subject to the following constraints:

- *C1:* every node, except the sink, is allocated a single time slot from the T time slots to transmit.
- *C2:* a subset of nodes, including the sink, are allocated a subset of time slots from the set $\{1, 2...T\}$ to receive transmissions.
- *C3:* once a node transmits, it can no longer be scheduled to receive the transmissions from any other node.
- *C4:* a recipient cannot transmit in the same slot in which it has been assigned to receive, i.e., half–duplex operation.
- *C5:* there is no interference by colliding transmissions at any of the recipient nodes during its assigned reception slot.

Unfortunately, the problem described above is known to be NP-complete even if restricted to Unit Disk Graphs (UDGs) [13]. Therefore, the existing solutions in the literature are heuristic approximations, and will be reviewed in the next section.

Before reviewing the existing work it is worth noting that the combination of C1 and C2 means that the activated edges (i.e., those edges of the underlying network topology graph that are used for communication, regardless of when they are activated or used) form a subgraph of the network topology graph, which is a tree. This property follows from the fact that this subgraph (a) contains exactly $N - 1$ edges (to satisfy C1) and (b) is directed and acyclic (to satisfy C2). As it is well-known, a subgraph satisfying those two characteristics must be a tree. That means a byproduct of solving the scheduling problem is a tree that represents a dependency of transmissions dictating the order in which aggregation convergecast is performed. In other words, no matter what solution we adapt for the scheduling problem a tree must be used to schedule the nodes.

The above fact has been recognized by the existing solutions that structure their heuristics as consisting of two phases: first a phase to build a dependency tree (based on a variety of criteria), and then a second phase to construct a schedule. To put these solutions in some concrete perspective, we will consider the example of Figure 5.3 to depict alternative aggregation/dependency trees and the corresponding schedules. Specifically, scheduled transmissions are represented as directed edges from a transmitter, i, to the recipient P_i. Each directed edge is also annotated with the time slot t_i in which it will be activated (by the transmitter i) in order for dependency and interference constraints (as described in



(a) SPT (b) DST (c) BISPT

Figure 5.3: Various logical trees and their corresponding schedules. (Solid arrowed lines represent edges of the logical tree. Dashed lines represent edges that are in the graph but not in the logical tree. Each arrowed edge is also annotated with a time slot in which it will be activated, by a node marked as a dark circle, to transmit the message.)

this section) to be met. Next, we review the existing solutions in detail while making some observations that lay the foundation of our proposed solutions.

5.3 Related Work

There are two types of solutions that exist for aggregation convergecast scheduling. One approach is to schedule a given aggregation tree irrespective of how that tree is constructed. (Recall that constraints C1 and C2 enforce any solution to have a logical tree.) Naturally, if the scheduling algorithm is defined independently of the aggregation tree, then the performance will vary greatly depending on the aggregation tree supplied. Such is the case of the scheme called PAS by Yu et. al. [76]. Another (and more common) approach is for a scheme to prescribe both the tree construction as well as the scheduling algorithm. Within this category, there are those algorithms that retain the tree constructed in the first phase, and others that do not. In the latter category we find SDA by Chen et al. [13] and First–Fit by Huang et al. [32]. Unfortunately, the algorithm proposed in [32] produces schedules with possible collisions. This leaves SDA as the main example in this category.

SDA constructs an SPT in the first phase. It then incrementally schedules the nodes but also assigns them a parent, possibly different than the one in the original SPT. The examples

shown in Figures 5.3(a) and 5.3(c) are in fact two possible (distinct) trees/schedules that may be generated by the SDA algorithm with the input of SPT shown in Figure 5.3(a). Note the change of parents for nodes E and D in Figure 5.3(c) that is different than the original parents as shown in Figure 5.3(a). As we will later see, SDA is an efficient algorithm but its drawback is the "distortion" caused by the scheduling phase, which means we cannot apply SDA's scheduler when we need to retain the aggregation tree exactly as supplied.

In contrast to SDA, the scheme called DAS by Yu et al. [75] retains the tree constructed in the first phase. The shortcoming of DAS is that its first phase constructs a DST (based on [69]) An example of such a tree is shown in Figures 5.3(b). Unfortunately, a DST is not necessarily a good selection for producing a better schedule. Compare e.g., the schedules of the SPT and DST shown in Figures 5.3. In particular, the DST used 8 slots as compared the SPT, which is scheduled using 7 slots only. The reason is that a DST *organizes* the nodes of a network in the form *clusters* in which many those nodes eventually become leaf (children) nodes of the tree, with the rest of the nodes being a (smaller) set of cluster-head become internal (parent) nodes. That leads to the formation of a "bushy" tree having a large number of dependency constraints. Note that a cluster-head (parent) cannot perform aggregation before it receives data from its cluster-members (children), and therefore its transmission cannot take place earlier than the transmissions from all (typically many) of its children. Overall, the result is long schedules.

Finally, other examples of two-phase approaches are the scheme by Wan et. al. [70] (DST-based via Maximal Independent Set construction), and the scheme by Annamalai et al. [5] (SPT-based) which has been exceeded in performance by [13]). Certain other efforts, such as [81], result in schedules that are potentially *not* collision free, and have been left out of consideration for obvious reasons.

5.4 Bounds and Tree Construction

As we pointed out earlier, DST (as well as BISPT) is not necessarily providing a short schedule compared to SPT. This is partly to be expected because the nodes in DST and BISPT are "clustered" to create more leaf nodes (conversely to decrease non-leaf nodes for the purpose of decreasing the broadcasting transmissions). On the other hand, given sufficiently rich connectivity, there also exist multiple alternative SPTs for the same physical topology. Hence, we would like to know what features make an SPT better than another SPT; a question that we address in this section.

We note that a tree construction phase, whether SPT, DST or any other kind of tree, ought to be guided by the potential it has to generate a short schedule for convergecasting. Until now, the two–phase schemes produced a tree based on topological properties alone. To the best of our knowledge, we are the first to perform the tree construction in a manner that is "informed" by the potential it has to result in a short schedule. (Recall that we had followed a similar principle to tackle the broadcast scheduling problem, which has resulted in improved performance.) To this end, and in contrast to the existing approaches, our tree construction specifically targets at "relaxing" the logical dependency constraints of the tree. The intuition being that the less constrained in terms of dependencies is the aggregation tree, the fewer the additional constraints (on top of the collision freedom constraints), hence the potentially shorter the schedule will be.

We start by establishing, for a given tree, a lower bound on the schedule length. Then, we restrict ourselves to SPT trees and produce an SPT that follows this lower bound. As a side note, [70] provides an *upper* bound on the schedule length. But from the performance of their algorithm it is evident that this upper bound is far too pessimistic compared to the typical *practical* behavior of their algorithm. We take a different view of trying to squeeze the performance as close as possible to the lower bound corresponding to the constructed tree. Nevertheless, for comparison purposes, we will also evaluate the performance of [70] in our performance study Section 5.6.

A lower bound specifies the minimum number of slots that are required for convergecasting. Chen et. al. proposed the lower bound to be $max\{h, log_2 N\}$, where h is the longest path in a logical tree [13]. Similarly, Huang et. al. also argue that the data aggregation latency cannot be less than the network radius [32]. It basically means that the lower bound is the longest shortest path between the sink and a node in the network, i.e., h. Unfortunately, these lower bounds are loose. Consider the DST shown in Figure 5.3(b). Chen's and Huang's lower bound for this particular DST is 4, i.e., this particular DST cannot be scheduled with less than four slots. However, it is clear from the DST structure that node I will need at least 6 slots to send its data to the root. In particular, three different slots are required by its children, i.e., nodes H, J and K. (That is because all of them have the same parent and therefore, they cannot transmit concurrently without having collisions.) Node I will need one additional slot to transmit to its parent F. Finally, since F is two hops away from the root, it will need at least two more slots (after receiving data from its children) to send across the aggregated data to the root, e.g., node F can use the fifth slot and node C can use the sixth slot. To generalize this observation, we introduce the following lemma.

Lemma 3. *Given a logical tree, an aggregation convergecast schedule length lower bound, T_{min}, is $max\{|\xi_i| + d_i : i = 1, 2, ...N\}$, where ξ_i and d_i, respectively, are the children set and depth (distance from the root) of node i in the given tree.*

Proof. Let k be a node with the maximum sum of the children-count and distance, i.e., $|\xi_k| + d_k$ = $max\{|\xi_i| + d_i : i = 1, 2, ...N\}$. We will prove by contradiction that $T \geq max\{|\xi_i| + d_i : i = 1, 2, ...N\}$ for any schedule[1], $\mathcal{T}$, of the given tree.

Assume that $T < max\{|\xi_i| + d_i : i = 1, 2, ...N\}$. If k has been assigned the slot number $t(k)$, then it must be less than or equal to the total number of slots used i.e., $t(k) \leq T$. Since our assumption is $T < max\{|\xi_i| + d_i : i = 1, 2, ...N\}$, therefore, $t(k) < max\{|\xi_i| + d_i : i = 1, 2, ...N\}$. It also means that $t(k) < |\xi_k| + d_k$, which we get by replacing $max\{|\xi_i| + d_i : i = 1, 2, ...N\}$ with $|\xi_k| + d_k$. (Recall that k is the node with the maximum sum of the children-count and distance, i.e., $|\xi_k| + d_k$ = $max\{|\xi_i| + d_i : i = 1, 2, ...N\}$.) However, $t(k) < |\xi_k| + d_k$ contradicts the fact that k needs at least $|\xi_k|$ slots for its children (for them to transmit first) and another d_k slots to send across its data to the root. It means that our assumption $T < max\{|\xi_i| + d_i : i = 1, 2, ...N\}$ must be incorrect. Therefore, $T \geq max\{|\xi_i| + d_i : i = 1, 2, ...N\}$, and hence the proof that T_{min} = $max\{|\xi_i| + d_i : i = 1, 2, ...N\}$. □

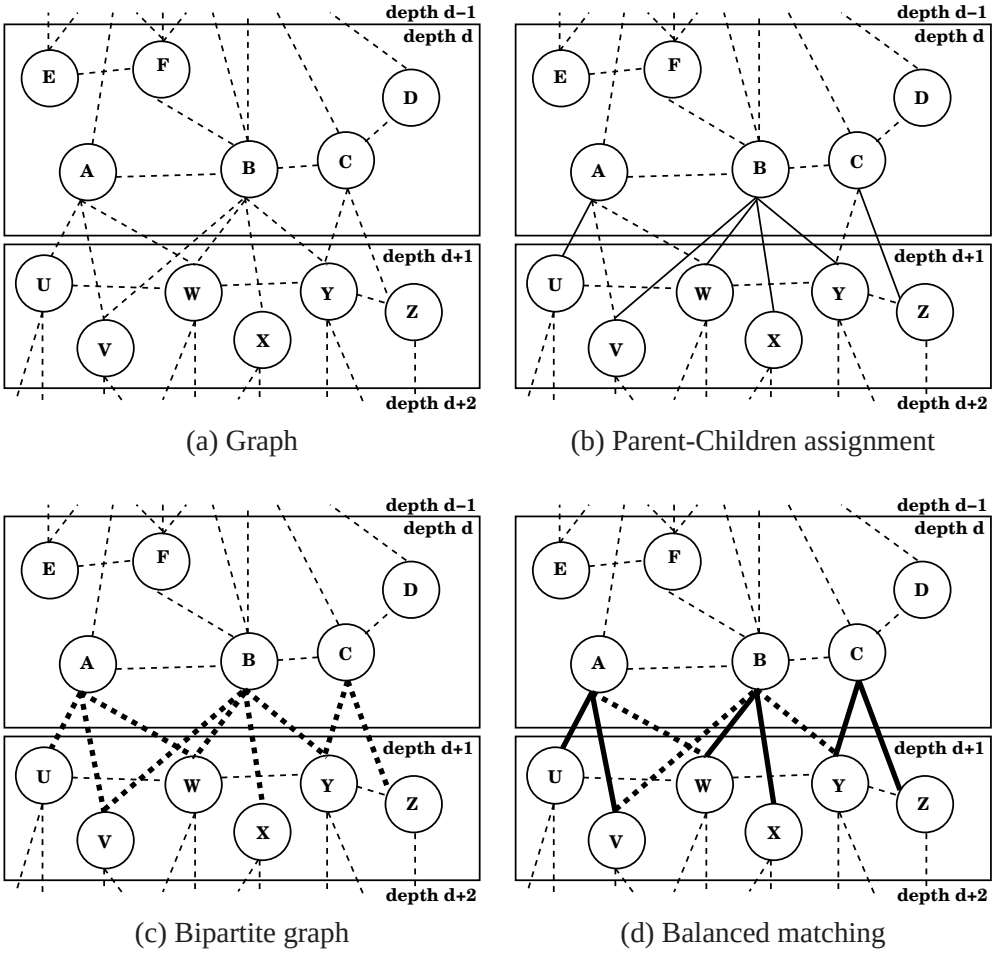
Next, we describe our procedure for constructing the aggregation tree based on the bound established by Lemma 3.

5.4.1 Balanced Shortest Path Tree

A tree that minimizes the lower bound potentially uses a lesser number of slots for scheduling N nodes. Consider the SPT, DST and BISPT shown in Figure 5.3 that have a lower bound of 4, 6 and 5 slots, respectively, as computed according to Lemma 3. In accordance with their respective lower bound, the SPT used 6 slots as compared to 8 and 9 slots used by the DST and BISPT. This scenario exemplifies our observation that a tree, which "relaxes" the logical constraints, i.e., ξ_i and d_i, can indeed reduce the total number of slots that are required for scheduling the nodes. An obvious question now is how to construct such a tree that minimizes $max\{|\xi_i| + d_i : i = 1, 2, ...N\}$.

For every node i, d_i can be minimized by ensuring that every node is connected to the root using a shortest path, i.e., by constructing an SPT. A shortest path tree can be constructed using a standard Breadth First Search (BFS) algorithm. However, minimizing

[1]That meets the criteria set-forth in Section 5.2.



(a) Graph (b) Parent-Children assignment

(c) Bipartite graph (d) Balanced matching

Figure 5.4: Parent-children assignment during SPT construction. Rectangles contain nodes at a particular depth (from the root) of the tree under construction. Dashed lines represent the graph (adjacency) edges. Solid lines represent parent-children assignments and also represent potential edges that can be selected in the tree construction. Dashed very-thick lines represent the edges in the bipartite graph formed between two consecutive depths of the tree. Solid very-thick lines represent the optimal semi-matching of the bipartite graph.

$|\xi_i|$ is non-trivial. Consider the scenario of a shortest path tree construction as shown in Figure 5.4. A set of 12 nodes have been shown in Figure 5.4(a) at two consecutive "depths", d and $d+1$, of the tree. In particular, nodes from A to F are at distance d, and nodes from U to Z are at distance $d+1$ from the root (not shown in the figures). A possible scenario of parent-children assignment is shown in Figure 5.4(b). In this example children-count for nodes B and C have been minimized by assigning only one child to each one of them, i.e., nodes U and Z, respectively. However, this parent-children assignment has resulted in the assignment of four children to node B. In fact, the parent-children assignment shown in Figure 5.4(b) has maximized the "local" lower bound at depth d of the tree, i.e., $max\{|\xi_A|, |\xi_B|, |\xi_C|\}$ = 4. (All the nodes at depth d are at same distance (hop) from the

root, i.e., $d_A = d_B = d_C$, therefore, their respective children set will actually determine the lower bound at depth d of the tree.) An increase in the lower bound at depth d of the tree may indeed result in the increased "global" lower bound of the tree. In general, we have the following sub-problem that needs to be solved.

Definition 4. *Assuming $\mathcal{C}_d$ represents the set of nodes that are at distance d from the root, the parent-children assignment problem is to assign every node at depth $d+1$, $\mathcal{C}_{d+1}$, a parent from the nodes at depth d, $\mathcal{C}_d$, such that $max\{|\xi_k| : \forall k \in \mathcal{C}_d\}$ is minimum.*

Interestingly, the parent-children assignment problem as defined above is equivalent to the problem of finding an optimal[2] semi-matching in a bipartite graph [26]. A bipartite graph formed by the nodes at depth d, i.e., $\mathcal{C}_d$ = {A, B, C}, and nodes at depth $d+1$, i.e., $\mathcal{C}_{d+1}$ = {U, V, W, X, Y, Z}, is shown in Figure 5.4(c). (Since nodes D, E and F do not have any neighbors from the nodes at depth $d+1$, they cannot become parents and hence they are ignored. However, they will eventually become leaves of the shortest path tree). An optimal semi-matching is shown in Figure 5.4(d). It is optimal with respect to the number of assigned children, i.e., $max\{|\xi_A|, |\xi_B|, |\xi_C|\}$ = 2 is minimum.

The BSPT construction is the construction of a "special" SPT in which optimal semi-matchings are obtained by constructing bipartite graphs with nodes at every two consecutive levels of an SPT. Of course that results in the minimization of $max\{|\xi_k| : \forall k \in \mathcal{C}_d\}$ at every depth d of the SPT, which we will prove shortly in this section. We call an SPT for which parent-children assignments are balanced using optimal semi-matchings a Balanced SPT (BSPT).

Algorithm 6 describes the operation of the BSPT. It is essentially a breadth first search algorithm which, as it progresses, (i) creates successive instances, G_b, of bipartite graphs from the nodes at two consecutive levels of the tree (line 14), and, (ii) invokes and solves optimal semi-matchings on each bipartite graph using the algorithm by Harvey et. al. [26] (line 15). The obtained semi-matchings (expressed as edge set Z in line 15) are collected to produce the edge set, E', of the BSPT.

Lemma 4. *Given a graph, $G(V, E)$, and a root node r, ConstructBSPT outputs an SPT with a minimum lower bound (as defined in Lemma 3) across all SPTs of G.*

Proof. It is trivial to see that by virtue of the breadth first traversal the number of hops to reach each node from the root is minimum, i.e., the tree is an SPT. An optimal semi-

[2] In this thesis whenever we mention an optimal semi-matching, we mean that the semi-matching is optimal with respect to L_∞ norm. Note that the optimal semi-matching, which minimizes the minimum load, is a different problem than the Maximum-Load Semi-Matching problem discussed in the previous chapter.

Algorithm 6

```
1:  procedure ConstructBSPT(V, E, r)
2:      P ← {s_r};
3:      E' ← ∅;
4:      for all v ∈ V do
5:          Mark(v) = False;
6:      repeat
7:          C ← ∅;
8:          for all m ∈ P do
9:              Mark(m) = True;
10:         for all m ∈ P do
11:             for all n ∈ 𝒩(m) do
12:                 if Mark(n) = False then
13:                     C = C ∪ {n};
14:         G_b ← BipartiteGraph(P, C);
15:         Z ← FindMinLoadSemiMatchings(G_b);
16:         E' ← E' ∪ Z;
17:         P ← C;
18:     until P = ∅;
19:     return (E');
```

matching with respect to the L_∞ norm minimizes the maximum load [26]. It basically means that if $\mathcal{C}_d$ is the set of nodes at depth d of the tree, then $max\{|\xi_k| : \forall k \in \mathcal{C}_d\}$ is minimum. Because the distance (from the root) of every node at depth d is the same, therefore, $max\{|\xi_k| + h_k : \forall k \in \mathcal{C}_d\}$ is also minimum at depth d of the SPT. Since optimal semi-matching is performed at every depth of the SPT, $max\{|\xi_i| + d_i : i = \{1, 2, ...N\}\}$ is minimum for the SPT produced. □

It is worth noting that for a given graph, ConstructBSPT generates an optimal SPT that has a lower bound as defined in Lemma 3, which is guaranteed to be minimum with respect to all possible SPTs that can be generated from that given graph. However, it does not guarantee a minimum lower bound (as defined in Lemma 3) with respect to trees that are *not* SPTs. It is conceivable that a tree can be constructed in which the paths of the nodes can be *elongated*, hence increasing their hop-count (compared to the shortest possible path) while possibly *decreasing* their children-count to potentially achieve an optimal lower bound as defined in Lemma 3. A more detailed study on this topic is part of our future study.

5.5 A Ranking Based Scheduling Algorithm

In this section we present an algorithm for scheduling the tree produced in section 5.4. As a matter of fact our algorithm can be used for any given tree to produce a collision free schedule. Unlike some other proposals, e.g., SDA [13], our algorithm retains the structure of the input tree. First, we introduce some additional notation.

Let us denote as V' all nodes of V except for the root r, i.e., $V' = V \setminus \{r\}$. For convenience, in addition to the child-of relationship ξ_v we introduced earlier, we denote by Ξ the set of all child-of relationships, i.e., $\Xi = \{\xi_v | v \in V\}$, and the complementary parent-of relation denoted by P_v. P_v indicates which node is the parent of node v, and $\mathcal{P}$ is the set of P_v, i.e., $\mathcal{P} = \{P_v | v \in V'\}$. Also, we denote by N_v the neighborhood set of node v in the network topology, i.e., $N_v = \{u | (v, u) \in E\}$. Collectively, the set of neighbor sets for all nodes in V will be denoted by $\mathcal{N}$, i.e., $\mathcal{N} = \{N_v | v \in V\}$.

On the surface, our scheduling algorithm (Algorithm 7) is fairly simple as compared to many other proposals. It takes as input a tree (as captured by $\mathcal{P}$ and Ξ) and the network topology (as captured by V and $\mathcal{N}$) and constructs a convergecast schedule that leads to the root r. At each step, i.e., for each time slot j (starting with timeslot 1) it considers the set of nodes, F, that are *eligible* to be scheduled. Initially, only the tree leaf nodes are eligible. Subsequently, nodes become eligible if all their children have been scheduled in earlier slots. However, only a subset of eligible nodes can actually be selected for that particular slot in order to meet the collision-freedom constraint (C4 in Section 5.2). The basic idea is that in each slot we go through a ranked list of eligible nodes. The ranking is in terms of *decreasing* weight $w(i)$. As we work through this list from highest to lowest weight, we skip nodes that are not possible to schedule due to violation of collision constraints with transmissions already scheduled for the same slot. The transmissions scheduled in a slot can make more nodes eligible for transmission starting in the next slot. The schedule is the collection of which nodes, S_j, are to transmit in the j–th slot and a corresponding set of receivers (their parent nodes) R_j.

Naturally, the ability of this process to generate good (short) schedules depends on the way the weights $w(i)$ are assigned to each eligible node $(i \in F)$ in a given slot. We have experimented with many alternative weights and have found that the weight assignment that gave the best results was the cardinality of the *non-leaf neighbors*, $\eta(i) \subseteq N_i$ of an eligible node, i, which are yet to be scheduled. The intuition behind this choice is as follows. Think of the act of scheduling an eligible node as "removing" it from the tree. Removal of a node

Algorithm 7

```
1: procedure WIRES(r, V, 𝒩, Ξ, 𝒫)
2:     V' ← V \ {r};
3:     j ← 1;
4:     repeat
5:         F ← ∅;
6:         for all v ∈ V' do
7:             if ξ_v = 0 then
8:                 F ← F ∪ {v};
9:         for all e ∈ F do
10:            η(e) ← ∅;
11:            for all e' ∈ N_e do
12:                if e' ∈ V' and ξ_e' ≠ 0 then
13:                    η(e) ← η(e) ∪ {e'};
14:            w(e) ← |η(e)|;
15:        S_j ← ∅;
16:        R_j ← ∅;
17:        FlagC4a ← True;
18:        FlagC4b ← True;
19:        for all e ∈ F in decreasing w_e do
20:            for all s ∈ S_j do
21:                if P_e ∈ N_s then
22:                    FlagC4a ← False; break;
23:            if FlagC4a = True then
24:                for all r ∈ R_j do
25:                    if e ∈ N_r then
26:                        FlagC4b ← False; break;
27:            if FlagC4a = True ∧ FlagC4b = True then
28:                S_j ← S_j ∪ e;
29:                R_j ← R_j ∪ P_e;
30:                V' ← V' \ {e};
31:                E' ← E' \ {(e, P_e)};
32:        𝒮 ← {S_v | v ∈ {1, ..., j}};
33:        j ← j + 1;
34:    until V' = ∅
35:    return (𝒮);
```

changes the set of leaves in the tree. Hence scheduling a node is equivalent to creating a *new* tree in which the non-leaf neighbors of nodes may change compared to the original tree. If $G_t(V_t, E_t)$ is the graph representing the new tree, then $\eta(i) = \{j : j \in N_i \wedge j \in V_t \wedge \xi_j \neq \emptyset\}$. F is the list of eligible nodes in decreasing $|\eta(i)|$ order. Hence by scheduling first the nodes with higher values of $|\eta(i)|$, we are essentially scheduling the most "constrained" nodes first, hence enabling many other nodes to become eligible nodes for subsequent slots.

The name we use in the rest of this thesis to identify the framework for ranking and incremental scheduling is Weighted Incremental Ranking for convergEcast with aggregation Scheduling (WIRES). Note that a concrete implementation of WIRES requires one to define a specific weight (priority) assignment strategy to eligible nodes. In Algorithm 7 the set of eligible nodes for the j^{th} slot, and their corresponding weights are computed at lines 6 through 14. Eligible nodes are considered in the decreasing order of their weight at lines 19 through 31. Only those eligible nodes are finally allocated the j^{th} slot (line 28) that do not violate the collision-free criteria (line 27). The eligible nodes that are finally scheduled are removed from consideration at lines 30–31. This procedure continues until no nodes remain to schedule (line 34).

Lemma 5. *The schedule produced by WIRES is collision–free.*

Proof. Every slot in WIRES is allocated in an incremental fashion, i.e., S_j is initiated as an empty set and then eligible nodes are added into it incrementally. Since any node is added to S_j only if that node meets the collision-free criteria, S_j will contain the set of nodes that do not interfere with each other's transmissions. Lines 20 to 22 ensure that no nodes already scheduled to transmit in the current slot are neighbors of the parent of the eligible node, e, currently being considered. Lines 24 to 26 ensure that the considered node is not a neighbor of the nodes already scheduled to receive in the current slot. Because this procedure is repeated for every slot, $j \geq 1$, S_j will always contain nodes that do not interfere with each other's transmissions. □

5.6 Performance Evaluation

To evaluate our proposal we implemented SDA [13], PAS [76], DAS [75], SAS [70] and First-Fit [32] algorithms to compare their performance with WIRES-BSPT. We discovered that the First-Fit algorithm does not produce a collision-free schedule, which has also been noted in [75]. Therefore, in our evaluations we omit the results of the First-Fit algorithm.

It is interesting to note that all existing proposals have been tested under drastically different simulation setups. For example, Chen et. al. [13] assumed a network of 100 nodes in a 200m$\times$200m area. Various topological scenarios were created by varying the transmission range of nodes between 21.7m and 40m. In contrast, Yu et. al. [75] used 1000 to 2000 nodes, with a radio range of 25m, in an area of 200m$\times$200m area. These two scenarios exemplify the extreme variations in the simulation setups being used by various studies. Choosing a particular "representative" setup for our study was challenging as one

Parameter	Values
N (# of nodes)	200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000 (Synthetic) 54 (Intel)
L (length of the square area [m])	200 (Synthetic) 50 (Intel)
ω (transmission range [m])	20 (Synthetic) 8, 10, 12, 14 16 (Intel)

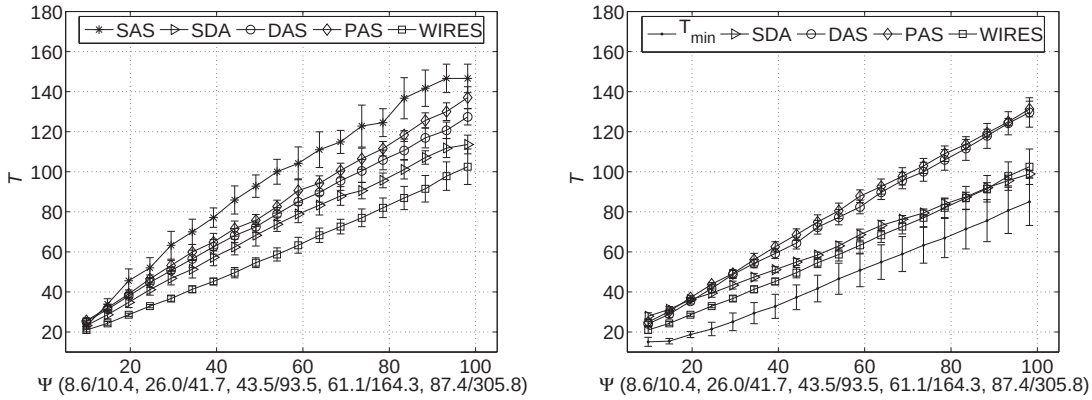
Table 5.1: Parameter values used in this chapter (default values are shown in bold face).

particular setup may not have revealed the actual performance of all other solutions. To address this problem we used the *density* metric (also used in the previous chapters of this thesis) to provide a "common platform" to test all algorithms as fairly as possible. Recall that we define the density to be: $\Psi = \frac{\pi\omega^2 N}{L^2}$, where N is the number of nodes, ω is the transmission range of nodes and L is the length of a square area. By varying density we have essentially captured the "essence" of various setups being used in other studies.

In our experiments reported in this chapter we kept ω and L fixed at 25m and 200m, respectively, while varied N to change the density. Table 5.1 summarizes the set of values used for the different parameters in our experiments presented in this chapter. The reported results are an average of 20 simulation runs. A node is chosen uniformly randomly as the root in each of these runs. We also performed experiments while using the Intel setup as described in Chapter 2.

5.6.1 Convergecast Latency

We use *convergecast latency*, which is the total number of slots used for a given schedule, as a metric to evaluate the performance of various solutions. The first set of results, shown in Figure 5.5, are for the synthetic dataset. As shown in Figure 5.5(a) we can see that WIRES-BSPT outperforms all other solutions by 10 to 30%, which means that our solution will require that much less time for its schedule. More interestingly, as the density increases, the performance of WIRES-BSPT improves compared to other approaches. The reason for the improvement is that, as the density increases, the underlying tree, BSPT, becomes more "bushy". It also means that, on average, the number of children per parent increases substantially. The way BSPT is constructed it tends to spread the load (children) among the parents evenly which in turn "relaxes" the logical constraint for these parents. This results in WIRES taking advantage of the tree to: (1) make nodes eligible for scheduling



(a) With different (original) underlying trees. (b) With a common (BSPT) underlying tree.

Figure 5.5: Latency (synthetic dataset). In the "( )" with Ψ we have provided average-degree/degree-variance of the nodes.

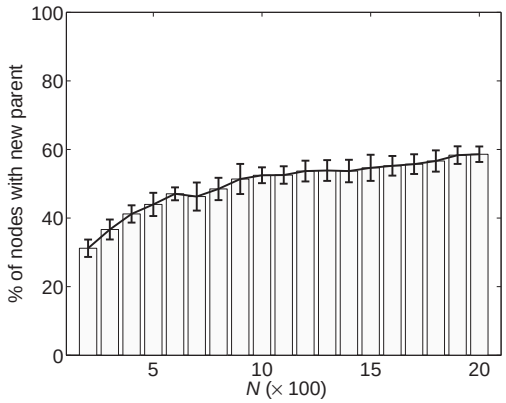


Figure 5.6: Changes inflicted by SDA (synthetic dataset).

earlier in the schedule, and, (2), as more parents become eligible, to significantly increase the "pool" of eligible nodes providing more opportunities for concurrent transmissions and hence spatial reuse.

Since SDA, DAS and PAS can work independently as standalone algorithms for scheduling, we ought to evaluate their performance with respect to BSPT. We provided BSPT as input tree to each one of these algorithms. The results are shown in Figure 5.5(b), and they confirm that the choice of the tree is important. We can see that SDA, DAS and PAS are now able to produce schedules using a smaller number of slots. Their performance is improved by replacing with BSPT the tree with which they were originally proposed and evaluated. In addition, WIRES still performs better than all other solutions. However, as Ψ increases the performance gap between WIRES and SDA shrinks. At a very high density, i.e., $\Psi \geq 45$ the difference between WIRES and SDA becomes negligible. However, SDA

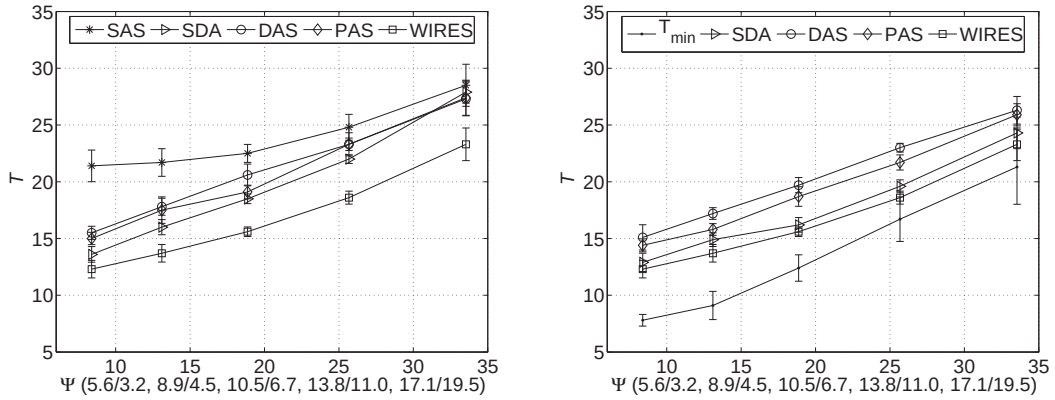
has its own limitation, i.e., it does not retain the input tree. (Shortly, we will present the results in that context). Figure 5.5(b) also presents the results computed using the lower bound, T_{min} as described in Lemma 3. Of course, changes to the logical tree across simulation runs changes also the corresponding lower bound (hence T_{min} values are surrounded by errorbars to represent their range of values).

Figure 5.6 summarizes the "side effects" of SDA-BSPT by illustrating the average number of nodes that have been assigned new parents, different from the ones they had in the tree provided as input to SDA. The main trend that we can observe is that when the node density increases (as the number of nodes increases) the total number of nodes with the new parents also increases. Overall, approximately 30% to 60% of the nodes are assigned a new parent. These results suggest that the changes inflicted by SDA can be substantial.

The second set of results using the Intel dataset are summarized in Figure 5.7. The qualitative behavior of the results remains the same as seen in the results for the synthetic dataset. However, their quantitative behavior has changed. As shown in Figure 5.7(a), WIRES outperforms all other solutions but by slightly less margin, i.e., by 10 to 15%. An interesting result that did not appear in the synthetic dataset is that the performance gain of SDA with respect to DAS and PAS is insignificant as shown in Figure 5.7(a). This experiment reveals SDA's sensitivity to particular topologies. However, WIRES consistently performs better than all other solutions. Figure 5.7(b) summarizes the performance of the algorithms when BSPT is used for all scheduling algorithms. Here again, the results of SDA, DAS and PAS are improved as compared to the results shown in Figure 5.7(a).

Figure 5.8 summarizes the performance of SDA with respect to the changes in the aggregation tree. As shown in these results, SDA changes approximately 45% of the original parents in the output tree when the transmission range (ω) is 8m. An interesting trend here is that as ω increases the number of "newly" assigned parents is decreased. This behavior can be explained by the fact that when ω increases the underlying logical tree becomes "bushy" since the topology becomes and almost complete connected graph. It also means that the number leaf nodes are increased while reducing the number of non-leaf nodes that can actually become parent. (Recall that the number of nodes is fixed in the Intel dataset.) Because the "pool" of nodes that can possibly become parent is reduced, the probability of selecting a new parent is also reduced. It is worth mentioning here that SDA does not adopt any specific mechanism for allocating original parents (from the input tree) to the nodes, which is the main cause of the changes inflicted by the SDA's scheduling process.

To summarize, if the underlying tree is not important and the only objective is to reduce



(a) With different (original) underlying trees. (b) With a common (BSPT) underlying tree.

Figure 5.7: Latency (Intel dataset). In the "( )" with Ψ we have provided average-degree/degree-variance of the nodes.

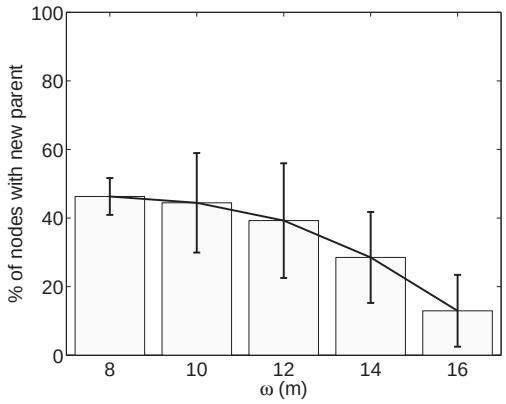


Figure 5.8: Changes inflicted by SDA (Intel dataset).

the number of slots, then WIRES-BSPT performs better in most of the cases. At higher densities, WIRES-BSPT and SDA-BSPT offered equally good solutions. Nevertheless, if the underlying tree is important (i.e., if we have constructed a tree that is important from the application's perspective), then WIRES is the most efficient solution. Figures 5.5(b) and 5.7(b) also show the results of the lower bound on BSPT computed in accordance with Lemma 3, which suggest that the practical behavior of WISH is near to that of an optimal solution since the optimal schedule has to be between the lines for T_{min} and WIRES.

5.7 Conclusions

Previous aggregation convergecast scheduling solutions rely on ad-hoc approaches to create the aggregation dependency tree before applying their scheduling algorithms. Some of the proposed solutions even change the tree, hence their usefulness is not obvious to ap-

plications that wish to retain the aggregation tree intact. Some of the previously proposed solutions are not even collision-free. We have presented several contributions in this chapter. First, we proposed a tighter lower bound to the tree scheduling problem, and proved its correctness. Second, we proposed an algorithm to construct a logical tree, BSPT, guided by the lower bound, allowing the generation of schedules with fewer slots. Third, we proposed a ranking/priority–based scheduling algorithm, WIRES, that produces schedules, that are guaranteed to be collision-free. Our proposal was evaluated extensively using synthetic and real datasets. Our proposed algorithms are efficient and can save up to 15% of the scheduling time, and hence reduces the convergecast latency.

An important conclusion here is that a tree that is better for convergecasting in terms of reducing the convergecast latency is not necessarily better in reducing the transmission cost. Basically, a tree that reduces the convergecast latency may actually increase the query processing cost in terms of energy. We will evaluate our proposal in this context in Chapter 7. Before that in the next chapter we address another important issue, i.e., failures in WSNs, that is equally likely to effect the performance of broadcasting as well as convergecasting.

Chapter 6

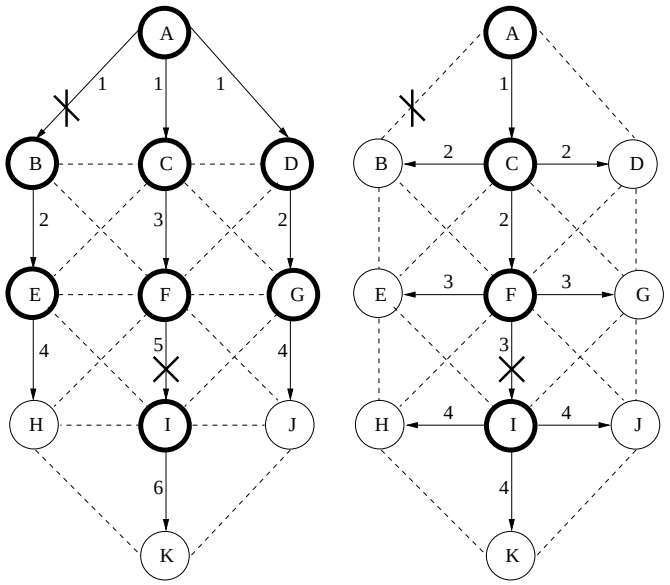
Opportunistic Failure Recovery

6.1 Introduction

Sensor networks, because of energy depletion, and also by virtue of sometimes being deployed in hostile environments, are prone to node failures. Certain node failures could result in a network partition, in which case nothing (short of replacing them with new nodes or changing the network topology) can be sensibly done to reconnect the network. More insidiously, there could be link failures, e.g., when a physical obstruction is introduced between two nodes and all of a sudden they are devoid of any communication, whereas previously they were able to communicate directly. In this latter case, a node that was supposed to forward its message might not be able to get its transmission across, leading to some nodes not receiving the message. In the context of logical topologies the impact of link failures could be more serious.

Consider, e.g., the scenario of broadcasting using the SPT shown in Figure 6.1(a) in which the link between nodes A and B has failed. Node B will not receive the message from A, and hence it will not forward the message. In this situation even nodes E and H will not be able to receive the message, though they are still connected to their respective parents. Similarly, nodes I and K will not receive the message because the link (F, I) has failed. In contrast to the SPT, failure of the link (A, B) has no impact on the DST shown in Figure 6.1(b), as the link (A, B) is not used by the DST. However, the failed link (F, I) has prevented nodes {H, I, J, K} from receiving the message. Link failures are sometimes transient, i.e., it is possible that they do not persist for long. Hence, an attempt to consider an update of the overall logical topology may not be called for, when temporary measures could mitigate the impact of a transiently failing link.

Similarly, failures may impact data collection during convergecast. Consider, e.g., the scenario of convergecasting using SPT and DST as shown in Figure 6.2. In particular, the



(a) SPT (b) DST

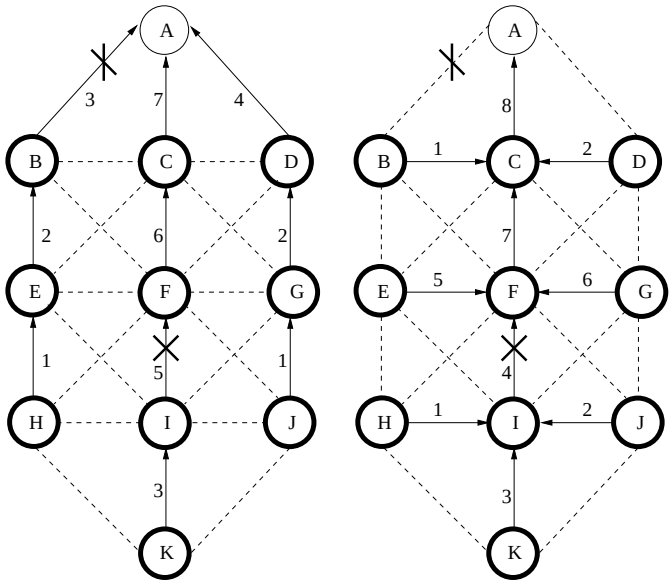
Figure 6.1: Impact of failures (marked with bold "X") on logical topologies during broadcast. (Solid arrowed lines represent edges of the logical tree. Dashed lines represent edges that are in the graph but not in the logical tree. Arrowed lines also represent messages with the distinction that all arrowed lines coming out from a node represent a single transmission from that node. Each arrowed edge is also annotated with a time slot in which it will be activated by the transmitting node marked as dark circle.

failed link (A, B) has no impact on the DST shown in Figure 6.2(b). However, because of the same link failure in the case of the SPT shown in Figure 6.2(a), node B will not be able to send across its data to the root. On the other hand, the failed link (F, I) has more severe impact on the DST as compared to the SPT. More precisely, nodes $\{$H, I, J, K$\}$ will not be able to send across their data in the case of the DST, while in the case of the SPT only nodes K and I are effected by the failure of the link (I, F).

In summary, link failures are equally likely to effect the underlying logical topologies and hence the performance of broadcasting as well as convergecasting. In this chapter, we propose opportunistic schemes that effectively exploit broadcasting and convergecasting schedules for failure recovery in WSNs. Before presenting our schemes, we first discuss some related work.

6.2 Related Work

A commonly used solution for the nodes to recover from the failures is to re-transmit the message if they do not receive an acknowledgment (ACK) from the expected recipients of the message. Consider e.g., the scenario of broadcasting using the SPT shown in Fig-



(a) SPT (b) DST

Figure 6.2: Impact of failures (marked with bold "X") on logical topologies during convergecast. (Solid arrowed lines represent edges of the logical tree. Dashed lines represent edges that are in the graph but not in the logical tree. Arrowed lines also represent messages with the distinction that all arrowed lines coming out from a node represent a single transmission from that node. Each arrowed edge is also annotated with a time slot in which it will be activated by the transmitting node marked as dark circle.

ure 6.1(a) in which every node is expected to send an ACK after they have received the broadcast message successfully. Because of the failed link (A, B) (as node B did not receive the message and hence node A did not receive an ACK from B) node A is triggered to re-transmit the message. This process (which can be controlled by user specified parameters) can be repeated until node A receives an ACK from node B. It is possible that node B receives the message successfully, but node A fails to receive an ACK, due to which node A re-transmits the message, i.e., uni-directional link failures may also occur. Nonetheless, it is trivial to see that such an approach will require an excessive amount of messages (including re-transmissions and ACKs) to recover from the failures, which could be expensive for WSNs. There are several other approaches that use the basic idea of re-transmissions/ACKs to build more sophisticated solutions, for instance, [52, 68].

Reliable broadcasting [52] and convergecasting [55, 64] have been addressed in the past, but in isolation from scheduling. In particular, they do not exploit the opportunities arising from a schedule that is created based on completely different objectives, e.g., the minimization of schedule length. In this thesis, we take a holistic approach where resilience to failures is seen as an attribute of a TDMA schedule. To the best of our knowledge this is

the first proposal that exploits a TDMA schedule for failure recovery in wireless networks.

We will first address the problem of failures in broadcasting for which we will propose our solutions and then we will extend the same proposal to address the failure issues in convergecasting. The basis of our proposal is that a TDMA schedule allocates only one slot per node to receive a message during broadcast. Recall constraint *C2* from Section 4.2 of Chapter 4 that enforces the nodes to receive only once. We argue that it is possible to relax this constraint to effectively exploit a TDMA schedule for nodes to recover from the link failures in a network. We propose to *inject* "redundancy" in terms of reception to increase the chance of successful reception of messages by the nodes during broadcast. It is key to note that in all schemes described here there are no redundant transmissions (recall that due to the scheduling constraints every node gets only one slot per round to transmit its message), and the redundancy is with respect to receptions only. As we will show in this chapter, that is indeed possible without using any additional messages in the network by simply exploiting a constructed schedule. We name this approach RIBS to stand for Reception Redundancy Injection in Broadcast Scheduling. We will use a similar approach to recover from the failures during convergecast, which we called Redundancy Injection in Convergecast Scheduling or RICS in short. We start with a detailed description of RIBS in the next section.

6.3 RIBS for Reliable Broadcasting

Our basic idea is that nodes can opportunistically exploit a TDMA schedule to successfully receive messages during broadcast. It is trivial to notice that if a downstream node was supposed to receive from its upstream node at a particular time slot, but was unable to do so due to link failure there could be more opportunities for it to do so later, by switching ON its receiver during contingent ("backup") time slots. Consider node B in Figure 6.1(a) that is scheduled to receive from node A during slot 1. Since the link (A, B) has failed, node B did not receive node A's transmission. However, node B can receive the message during time slot 3, which is essentially meant for transmission from node C to F. Note that node B still "missed" its allocated transmission slot 2, because it had not received the message earlier but eventually received it from node C at time slot 3. Hence, node E cannot receive the message from node B during slot 2. Nevertheless, node E (having not heard from node B in slot 2) can also opportunistically listen in slot 3 to receive the message from node C. Since node H will be able to receive the message from its parent node E during its allocated

Algorithm 8

```
1: procedure RIBS_BACKUPS(V', N, P, S)
2:     for all v ∈ V' do
3:         W_v ← ∅;
4:         for all t ∈ {1, ..., |S|} do
5:             b ← N_v ∩ S_t;
6:             if |b| = 1 ∧ P_v ∉ b then
7:                 W_v ← W_v ∪ b;
8:     W ← {W_v | v ∈ V'};
9:     return (W);
```

slot 4, it does not need to use slot 5 to receive from node F. In contrast to nodes B, E and H, node I does not have the opportunity to receive from nodes E and G during slot 4. The reason being that node I receives concurrent transmissions from nodes E and G resulting in a collision at node I. Because of node I's inability to transmit the message, node K does not receive the message either.

The situation is somewhat different in the case of the DST as shown in Figure 6.1(b) because node I has no other non-leaf node in its neighborhood except for its own assigned parent node E (the link from which has already failed), and hence node I cannot recover from the failure. Nevertheless, nodes H and J can still use slot 3 to opportunistically receive from node F. Overall, in the example, in both SPT and DST, the impact of certain link failures could be recovered by exploiting the schedule. While there is no guarantee that 100% of failures are recovered, it is important to note that it comes at virtually no additional cost, except for the need to switch a receiver ON, and *only as needed*, in more than one slot. This is in contrast to ACK/RTX/CTX based solutions, which will consume more energy as the transmission cost is considered to be more expensive than the reception cost [29].

An obvious question now is to decide, for each node, which slots can be used for reception. Some nodes may be able to hear from non-leaf nodes other than their own assigned parent, but not all transmissions from "other" non-leaf nodes may be useful, as node I in Figure 6.1(a) illustrates because it was unable to opportunistically use slot 4 due to the potential for collision (from I's perspective). Hence, the problem for nodes is to select only those slots for reception that are collision-free. Stated differently, the "backup" nodes, W_v, of node v are the non-leaf neighboring nodes whose transmissions are scheduled in a slot in which no other neighboring node transmits. For example, in the SPT shown in Figure 6.1(a) nodes C and F are the backup nodes of node B, i.e., $W_B = \{C, F\}$. The pseudocode of the algorithm that determines the backup nodes for broadcast is presented in Algorithm 8.

The input to RIBS consists of a broadcast schedule ($\mathcal{S}$), the logical tree ($\mathcal{P}$), and the topology information about the neighborhood of each node ($\mathcal{N}$). At line 3 every node's backup set is initialized to the empty set. At lines 4 to 7, the transmitting nodes in S_t (recall that S_t is the set of nodes that are scheduled to transmit in slot t), that are also neighbors of v become backup nodes, provided there exists only one such node (else the two or more transmissions could collide at receiver v) and that this node is not the parent of v. RIBS operates by forcing each node to listen to the transmissions from its assigned parent, P_v, as well as its backup nodes, W_v, (if any) during their corresponding slots. A node stops listening in subsequent slots if it successfully received the message in an earlier slot. Notice that the application of the constraint $|b| = 1$ eliminates the potential for using backup transmissions that are concurrent and colliding, even if v could potentially correctly receive one of them (due to the capture effect). Such strict application of the requirement for no–collision could often result in $W_v = \emptyset$ for many nodes.

Next we present our scheduling based solution for failure recovery in convergecast. Note that RIBS, which finds the backup nodes for failure recovery in broadcast is not applicable to convergecast because the corresponding schedules (for broadcast or convergecast) enforce an order of message transmissions, and this order is particular to the logic of aggregation performed in convergecast. Hence, a scheme applicable to convergecast should also honor the aggregation logic. Towards that end we present, RICS, a scheduling based solution for failure recovery in convergecast.

6.4 RICS for Reliable Convergecasting

RICS works on the same principle of allocating "backup" nodes for failure recovery in convergecast as in the case of RIBS for recovery in broadcast. However, unlike RIBS, backup nodes in RICS play an additional role. More precisely, backup nodes in RICS not only receive messages opportunistically but they also redundantly re-aggregate their data together with what they have received as scheduled as well as opportunistically (which are essential to the correctness of the aggregation data) in a logical tree to recover from the failures. Consider, e.g., the failure scenario in the SPT as depicted in Figure 6.2(a). As the link (A, B) fails node B is not able to send its message to the root. However, node C that is scheduled after node B, can opportunistically receive from node B. During its scheduled transmission at slot 6, node C can now send data that represents (as aggregated) B's data along with its own (if any). It is trivial to see that node C will be forced to send a message

containing the impact of B's data, if node C did not have any data to send during its allocated time. Furthermore, node C will send a message containing B's data regardless of the fact that "some" other nodes might be doing the same to recover B's data. Nevertheless, B's data will be successfully delivered to the root in this particular example of the SPT shown in Figure 6.2(a).

A much more interesting scenario is for node I with respect to the failed link (I, F) in the SPT shown in Figure 6.2(a). Notice that all neighbors of node I (except its parent F with which its link has failed) are scheduled before it due to which node I's data (if transmitted) cannot be recovered. Nodes {E, G, H, J} can opportunistically receive the message from node I during its allocated slot 3, but nodes {E, G, H, J} cannot forward node I's data due to their scheduling constraints, i.e., the transmitting slots of nodes {E, G, H, J} are earlier than the transmitting slot of node I. Hence, none of those nodes can become backup node for node I. It also means that there is no 100% guarantee that every node gets the opportunity for its data to be recovered via redundantly re-aggregating them.

Note that no new transmissions are added to the schedule hence collision freedom is maintained. Though there is no guarantee that every node has a backup node to recover from the failures, recovery also depends upon the underlying logical trees and their corresponding schedules. Consider, e.g., the DST now as shown in Figure 6.2(b). While using the DST, node I's data is now recoverable because the schedule allows nodes E and G (that are scheduled to transmit after node I's transmission) to receive and transmit node I's data successfully, which is in contrast to the SPT shown in Figure 6.2(a). Node F now receives the messages from nodes E and G, and aggregates the data (to remove redundancy as nodes E and G are essentially sending the same data) before forwarding the message to its parent, i.e., node C. The failed link (A, B) has no impact on the DST.

It is worth noting that both nodes, E and G, act as backup nodes for node I while using the DST shown in Figure 6.2(b). However, only one of them is sufficient for successful recovery of node I's data. Unfortunately, deciding which one of those nodes should actually be assigned as a backup node is non-trivial. It is possible that they could take turns over successive rounds to save energy. A more detailed study on this topic is part of our future research work. Also note that, a node may become a backup node for more than one node (which is different than the case in which one node has more than one backup nodes, as discussed above). Consider, e.g., the SPT shown in Figure 6.2(a) in which node C is a backup node for node B as well as node D. Similarly, in the DST shown in Figure 6.2(b) node K is a backup node for node H as well as node J. Hence some nodes may be intrinsically more

Algorithm 9

1: **procedure** RICS_BACKUPS($V', \mathcal{N}, \mathcal{P}, \mathcal{S}$)
2: **for all** $v \in V'$ **do**
3: $W_v \leftarrow \emptyset$;
4: **for all** $t \in \{t_v + 1, \ldots, |\mathcal{S}|\}$ **do**
5: $B \leftarrow N_v \cap S_t$;
6: **for all** $b \in B$ **do**
7: **if** $\{N_b \setminus v\} \cap \{S_{t_v} \setminus v\} = \emptyset \wedge b \neq p_v$ **then**
8: $W_v \leftarrow W_v \cup b$;
9: $\mathcal{W} \leftarrow \{W_v | v \in V'\}$;
10: **return** ($\mathcal{W}$);

"helpful" as backup nodes. Unfortunately, any advantages have to be seen under the light of uncertainty about knowing which link will fail and for how long. Hence the approach taken here is the simplest, which does not require any a-priory information other than the constructed schedule.

Like RIBS, not all slots in RICS are useful for opportunistic receptions. Consider, e.g., nodes H and J in Figure 6.2(a), which cannot have any backup node for failure recovery during convergecast. In particular, nodes H and J have three potential candidates for backup recovery, i.e., node F, I and K. However, concurrent transmissions from nodes H and J during slot 1 may collide at nodes F, I and K, due to which they cannot receive the message from either of them. Overall, the problem in RICS is to find the backup nodes that are not only scheduled after a particular node (for which the backup nodes are being sought) but which can receive the messages in a collision-free manner.

The pseudocode of the algorithm that determines the backup nodes for convergecast is presented in Algorithm 9. The input to RICS consists of a convergecast schedule ($\mathcal{S}$), the logical tree ($\mathcal{P}$), and the topology information about the neighborhood of each node ($\mathcal{N}$). At line 3 every node's backup set is initialized to the empty set. At lines 4 to 8, the backup nodes for every node v are determined by exploring all those nodes that are scheduled after v, i.e., considering the nodes from the sets $S_{t_v} \ldots S_T$ (recall that S_t is the set of nodes that are scheduled to transmit in slot t and $\mathcal{S} = \cup_{t=1}^{T} S_t$). Naturally only those nodes can be considered that are neighbors of v (lines 5 and 6). Furthermore, only those nodes can actually become backup nodes, which can receive collision-free transmission from v during slot t_v (line 7).

RICS operates by forcing a parent, P_v and backup nodes, W_v to receive the transmission of node v. Essentially, v transmits only one message, which is received by multiple nodes, i.e., the parent and backup nodes. Notice that the application of the constraint $\{N_b \setminus v\} \cap$

Parameter	Values
N (# of nodes)	100, 200, **300**, 400, 500 (Synthetic) 54 (Intel)
L (length of the square area [m])	800, 1000, **1200**, 1400, 1600 (Synthetic) 50 (Intel)
ω (transmission range [m])	200 (Synthetic) 8, 10, **12**, 14 16 (Intel)
R_c	0.2, 0.4 **0.6**, 0.8, 1.0
PER (packet error rate)	.05, .10, **.15**, .20, .25, .30

Table 6.1: Parameter values used in this chapter (default values are shown in bold face).

$\{S_{t_v} \setminus v\}$ eliminates the potential for using some backup nodes. In particular, not all nodes may transmit during their allocated slot, e.g., when filters are installed, leaving some nodes not being used as backup nodes. Such strict application of the requirement for no–collision could often result in $W_v = \emptyset$ for some nodes. Recall, e.g., nodes H and J in Figure 6.2(a) that will not have any backup nodes. Also, note that W_v is the set of recipients that will incur additional energy cost. Hence, while we do want W_v to be small we do not want it to be $\emptyset$, although in some cases it is.

6.5 Performance Evaluation

In this section we evaluate the performance of RIBS. The effectiveness of RICS will be evaluated in the next chapter for which we implement RICS as well as RIBS on top of EXTOK. To evaluate RIBS we implemented the solutions GKLRW [23] and YMV [76]. Basically we generated broadcast schedules using GKLRW and YMV as well as our own broadcast scheduling solution WISH proposed in Chapter 4. Recall that the scheduling phase of GKLRW is not independent of its tree construction phase, therefore, GKLRW cannot be used for scheduling "any" tree that is given as input to the solution. Contrary to that WISH and YMV can schedule any given tree. Therefore, in the following simulations, to fairly evaluate the performance of RIBS with different trees and their corresponding schedules, we use the tree proposed in GKLRW (which we will refer as GKLRW-tree) as well as BISPT. The schedule for GKLRW-tree is constructed through WISH, YMV and of course GKLRW. The schedule for BISPT is constructed through WISH and YMV only as GKLRW is not tree-independent. To check the effectiveness of RIBS, we first implemented GKLRW, YMV and WISH independently, and then evaluated these solutions against GKLRW-RIBS, YMV-RIBS and WISH-RIBS, i.e., when RIBS is implemented on top of each of them.

Five different simulation setups are considered in this chapter. In particular, 100, 200,

300, 400, 500 nodes in a 800m×800m, 1000m×1000m, 1200m×1200m, 1400m×1400m, and 1600m×1600m area, respectively. Transmission range of nodes was fixed at 200m. We also used the Intel setup discussed in Chapter 2 to evaluate our solutions. We keep track of density, $\Psi = \frac{\pi\omega^2 N}{L^2}$, where N is the number of nodes, ω is the transmission range of nodes and L is the length of a square area. All results presented here are an average of 20 simulation runs. Table 6.1 presents a summary of the values that we used for different parameters in our experiments presented in this chapter.

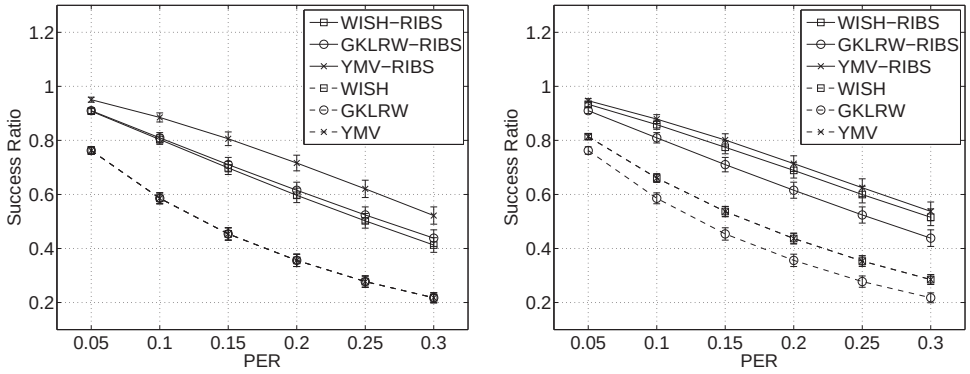
In the following experiments we collected results for two metrics, i.e., *success ratio* and *energy cost*, to evaluate the performance of the competing solutions. Recall that RIBS is a recovery mechanism employed by the nodes to recover the messages in the event of failures. Naturally, when failures occur some nodes in the network do not receive the broadcast message that is being propagated in the network. To be precise the success ratio is defined as the fraction of nodes that successfully receive a broadcast message. Thus the success ratio measures the overall impact of the failures to evaluate the performance of the recovery mechanisms, e.g., by comparing WISH with WISH-RIBS.

In our experiments we introduce link failures to vary Packet Error Rate (PER) in the range of $\{.05, .10, .15, .20, .25, .30\}$. PER is defined as the ratio of the number of messages transmitted to the number of messages that are received successfully. Though a PER of 0.25 to 0.30 is considered relatively high, we are interested to know if RIBS can actually withstand that severe testing.

6.5.1 Success Ratio

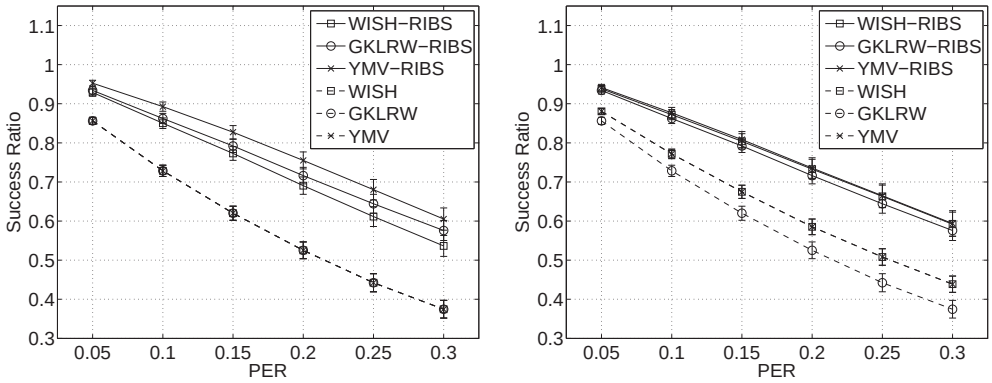
The results of RIBS when implemented with WISH, GKLRW and YMV are shown in Figure 6.3 and Figure 6.4. The foremost trend that can be seen in these results is that irrespective of the scheduling solutions (Figure 6.3(a)) and underlying trees (Figure 6.3(b)) RIBS increases the success ratio significantly. That can also be verified from the results shown in Figures 6.4 that are obtained through the Intel dataset.

In all of the above results, as PER increases the success ratio decreases, which is expected. As far as the improvement is concerned, we can clearly see that the performance of WISH-RIBS is improved by more than 50%, i.e., when RIBS is applied on top of WISH, 50% of nodes that were not able to receive the message previously, are now able to receive the messages. Note that this significant improvement is achieved without using any additional transmissions in the network. Even at a high PER of 0.30 more than 40% of the nodes are able to receive the messages as compared to 20% in the case when RIBS is not applied.



(a) With GKLRW-tree for all solutions (b) With BISPT for WISH and YMV

Figure 6.3: Success ratio vs. PER (Synthetic dataset)

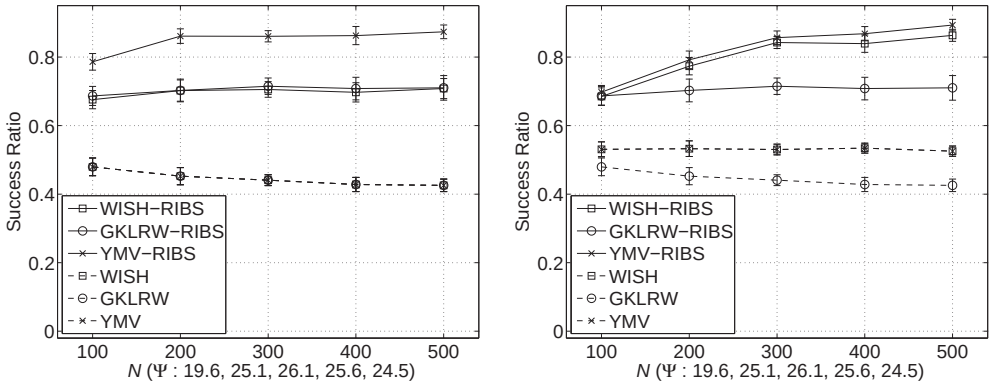


(a) With GKLRW-tree for all solutions (b) With BISPT for WISH and YMV

Figure 6.4: Success ratio vs. PER (Intel dataset)

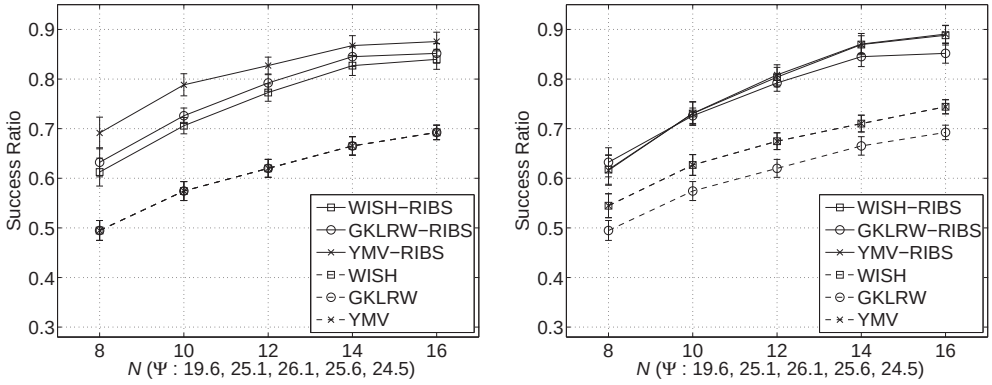
An interesting observation here is that YMV-RIBS does better than all other solutions. The reason for its improved performance is that on average YMV produces a “lengthier” schedule (recall latency results). Because of more slots used in YMV there are more opportunistic slots that are available to nodes which they may use during failure recovery. This result highlights the fact that a lengthier schedule may actually do better for failure recovery while using RIBS.

Also note that when RIBS is not applied then the success ratio is same for all solutions, i.e., WISH, GKLRW and YMV, as shown in Figure 6.3(a) and Figure 6.4(a). The reason is that all of those solutions are using a common logical tree, i.e., GKLRW-tree. On the other hand GKLRW’s success ratio is different than that of WISH and YMV as shown in



(a) With GKLRW tree for all solutions (b) With BISPT for WISH and YMV

Figure 6.5: Success ratio vs. N (Synthetic dataset).



(a) With GKLRW tree for all solutions (b) With BISPT for WISH and YMV

Figure 6.6: Success ratio vs. ω (Intel dataset).

Figure 6.3(b) and Figure 6.4(b). The reason for this behavior is that WISH and YMV are using BISPT, while GKLRW is using it's own logical tree.

In another set of experiments with the synthetic dataset we evaluate the performance of RIBS when N is increased while keeping the PER fixed at 0.15. The results are summarized in Figure 6.5. The foremost trend that can be seen from these results is that RIBS improves the performance of all solutions by up to 85%. Another trend that we can see in these results is that as N increases, the performance of RIBS is improved even further. This behavior can be explained by the fact that when N is increased, Ψ is also increased, which basically increases the node-degree of the nodes. Overall, that increases the number of backup nodes for the nodes in the network while increasing the robustness of the solutions even further.

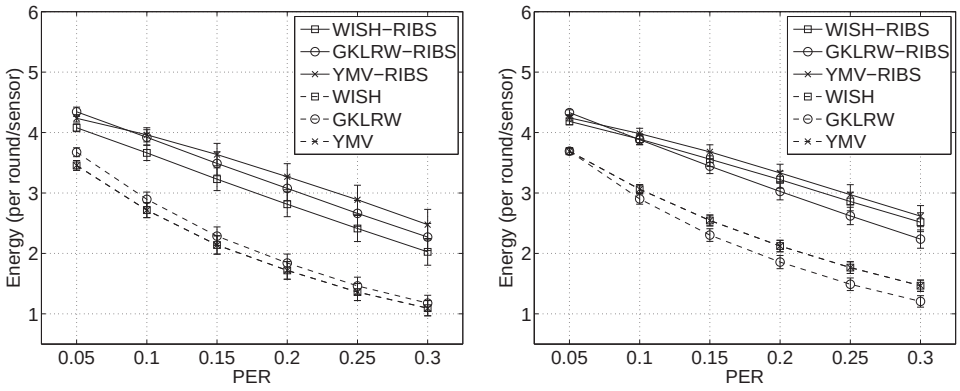
Similar trends can be seen in the results obtained from the Intel dataset that are summarized in Figure 6.6. RIBS improves the performance of all solutions. An interesting trend that was not visible with the synthetic dataset can now be observed with the Intel dataset. Notice that when ω is increased the performance of the underlying logical trees (without RIBS) is also improved. This can be verified when a common tree is used (Figure 6.6(a)), and also when two different logical trees are used (Figure 6.6(b)). The improvement in the performance of the underlying logical trees can be attributed to the fact that when ω is increased the logical trees become short. It also means that nodes become more "closer" to the root. When the nodes' distance (hop-count) from the root is reduced, the probability that the nodes successfully receive the message is increased, and hence the success ratio is increased. These results suggest that the "shorter" logical trees are more robust to failures, which can actually be verified from the results summarized in Figure 6.6(b). Note that the GKLRW-tree (without RIBS) is outperformed by the BISPT (WISH and YMV without RIBS). Recall that, unlike GKLRW-tree, BISPT is a "special" kind of shortest path tree.

6.5.2 Energy Cost

To consider the energy consumption we used a message consisting of 6 bytes that needed to be broadcast in the network. The cost of an unsuccessful message reception, either because of a failed link or "idle" listening, is considered to be 1 byte. Recall that during broadcasting a schedule is in place, and the nodes are "awake" at specific time slots to receive the broadcast message. However, some nodes may not receive the message either because of their failed links or the message is not transmitted by their parent at all in which case the nodes will incur the idle listening cost.

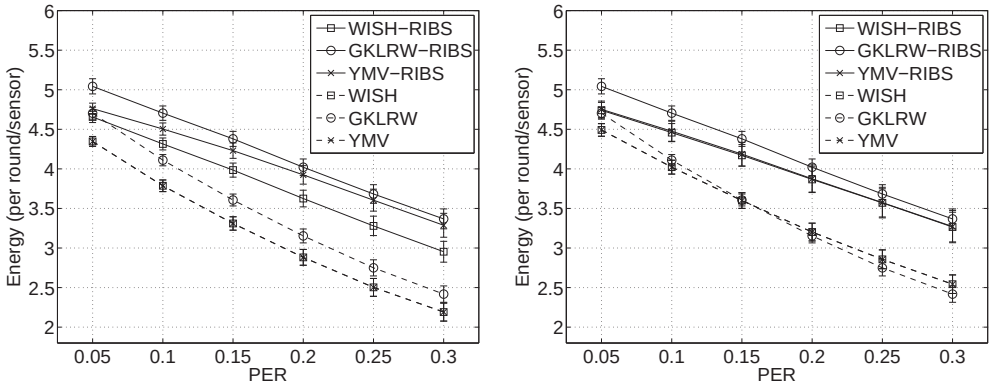
The energy cost of a node is computed as B_t + B_r.R_c, where B_t and B_r, respectively, are the number of bytes transmitted and received by the node. Recall that $R_c = E_{rx}/E_{tx}$, where E_{tx} and E_{rx}, respectively, are the energy cost to transmit and receive a single bit. In our experiments we tried various values of R_c from the set $\{$0.2, 0.4, 0.6, 0.8, 1.0$\}$. An increasing R_c value means the cost of reception is increasingly becoming equal to the cost of transmission. In the following experiments we fixed R_c at 0.6 for which the results are summarized in Figures 6.7 and 6.8.

The trend seen in Figures 6.7 and 6.8 is that as PER increases the energy cost decreases. The reason for this behavior is that as more packets are lost many nodes are unable to forward the message along the tree. That further triggers the nodes at higher levels of the tree to not participate in the broadcasting. In this situation many nodes do not incur



(a) With GKLRW tree for all solutions (b) With BISPT for WISH and YMV

Figure 6.7: Energy cost (Synthetic setup).



(a) With GKLRW-tree for all solutions (b) With BISPT for WISH and YMV

Figure 6.8: Energy cost (Intel setup).

transmission cost. As expected, when RIBS is implemented all solutions now pay additional reception cost which increases their overall energy cost as shown in Figures 6.7 and 6.8. Furthermore, as PER increases RIBS incurs more cost as the number of opportunistic slots that a node uses is also increased. Nevertheless, RIBS improved the performance of all solutions (i.e., irrespective of the quality of the schedules produced by them) by paying less than an average of 2 units per node of additional energy cost.

6.6 Conclusions

In this chapter we addressed the problem of link failures in broadcasting and convergecasting that are often neglected by the existing scheduling solutions. Towards that end we proposed a novel failure recovery framework consisting of RIBS and RICS that exploits TDMA scheduling to effectively deal with failures. Our extensive simulation study reveals that our proposed solutions are not only reliable but they are also energy efficient. In particular, we evaluated the performance of different logical tree topologies under various conditions of PER, N, ω and scheduling solutions. An important conclusion that can be drawn from the results presented in this chapter is that not all logical tree topologies are equally effected by failures, i.e., some logical tree topologies are more robust than others. In the next chapter all of our proposed solutions culminate to make a case for efficient and reliable top-k query processing in WSNs. In particular, through EXTOK we evaluate the performance of proposed logical trees, scheduling and failure recovery solutions for effective top-k query processing in WSNs.

Chapter 7

Efficient and Reliable EXTOK

7.1 Introduction

In this chapter we evaluate the performance of our various solutions with respect to top-k query processing using EXTOK. In particular, we evaluate the performance of EXTOK while using various combinations of logical trees i.e., SPT, DST, BISPT, and BSPT. Recall that a logical tree topology plays a crucial role in broadcasting and convergecasting, which are two elementary operations for query processing. To that end we are interested in using various combinations of logical trees for different phases of EXTOK. It is interesting to note that we can use either a single tree, e.g., BISPT, or a combination of trees, e.g., BISPT and BSPT, for broadcast and convergecast phases of EXTOK. The idea is to use a "specialized" tree for a particular phase. For example, we have already seen that BISPT is more efficient than other logical tree topologies for broadcasting, but not necessarily for convergecasting. Therefore, it might be useful to use "another" tree for convergecasting while still using BISPT for broadcasting. In this chapter, we use the following combinations of logical trees, as mentioned in Table 7.1, for top-k query processing using EXTOK. Later we will use the notation X-Y to represent the logical trees X and Y that are used as broadcast and convergecast trees, respectively, during EXTOK's broadcast and convergecast phases.

Broadcast Tree	Convergecast Tree	Combination Name
SPT	SPT	S-S
DST	DST	D-D
BSPT	BSPT	B-B
BISPT	BISPT	BI-BI
BISPT	SPT	BI-S
BISPT	BSPT	BI-B

Table 7.1: Combinations of various logical trees that are evaluated.

Many existing solutions use underlying logical tree topologies for performance gains without paying much attention to the quality of such logical topologies. It is beyond the scope of this thesis to study all logical trees that are possible for processing the top-k queries in WSNs. Nevertheless, our goal here is to demonstrate that the choice of an underlying logical tree does matter. As we will show later in this chapter that is indeed the case as an underlying logical tree significantly influences the query processing cost.

We are also interested in evaluating the performance of our failure recovery solutions, RIBS and RICS, for broadcasting and convergecasting with respect to the top-k query processing using EXTOK. More precisely, we implemented RIBS and RICS on top of EXTOK to make it resilient to failures in WSNs. Again we use various combinations of logical topologies with EXTOK and note the impact of failures. Note that in order to use RIBS and RICS we need broadcasting and converecasting schedules. Towards that end we use WISH and WIRES for generating broadcast and convergecast schedules, respectively. Overall in this chapter, various components of a query processing solution are put together to present a case of efficient and reliable in-network query processing in WSNs. Next, we present the results of our detailed simulation study. The simulation setup remains the same as discussed in the second chapter. However, for reader's convenience we present the main details of our simulation setup again.

7.2 Simulation Setup

Our proposal is evaluated using both synthetic and real datasets. The synthetic dataset was generated by simulating a network of nodes deployed in a 200m$\times$200m area. Using this dataset we performed experiments by varying five parameters: number of top values sought (k), number of nodes (N), wireless/transmission range (ω), probability that a node's value changes between two consecutive rounds (γ) and percentage of change in node's value (δ). Table 7.2 shows the values used for these parameters. To investigate the impact of randomly changing values (nodes' measurements) on the performance of the algorithms we generated "temperature" values for nodes. The initial value of nodes was randomly set between 1 and 100 and could vary between rounds according to parameter δ (equally likely to be a negative or positive change). Results using the synthetic dataset are based on an average of 20 simulation runs in which each run consists of 200 rounds. In each of these simulation runs position of the nodes and the root node were chosen randomly.

The experiments with real data was performed using the Intel Berkeley dataset [1],

Parameter	Values
k (# of top values)	1, 5, **10**, 15, 20
N (# of nodes)	100, 200, **300**, 400, 500
ω (transmission range [m])	25, 30, **35**, 40, 45 (synthetic data) and 8, 10, **12**, 14, 16 (Intel data)
γ (probability of change)	0.1, 0.2, **0.3**, 0.4, 0.5
δ (change [%])	2, 4, **6**, 8, 10
R_c	0.2, 0.4 **0.6**, 0.8, 1.0
PER	0.05, 0.10, **0.15**, 0.20, 0.25
E_{max}	4000, 5000, **6000**, 7000, 8000

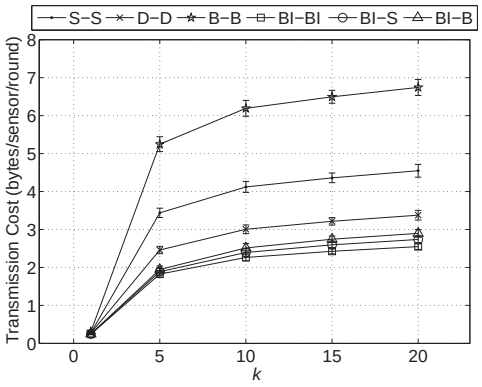
Table 7.2: Parameter values used in this chapter (default values are shown in bold face).

which consists of approximately 3.5 million readings from 54 nodes deployed in the Intel Berkeley Research lab. Missing values in the data were replaced using linear interpolation. Sensor readings were originally maintained by epochs, a monotonically increasing number for each of the nodes. We organized the sensor readings in such a way that the dataset has 60,000 rounds, each one containing one value for each of the 54 nodes. Note that in the Intel dataset only parameters k and ω are investigated using the original placement of the nodes, and having one such node randomly chosen as the root node for each run. As before the reported results are an average of 20 runs.

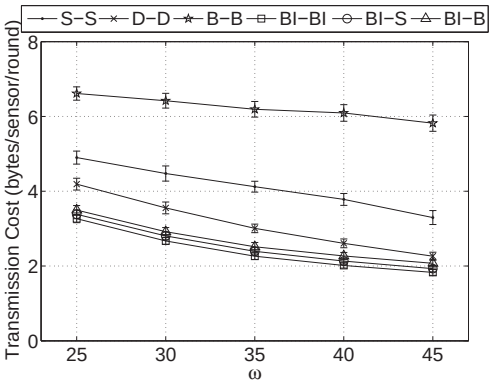
A node id and its value are represented by 2 bytes each. The τ value in EXTOK is characterized by 2 bytes. Each message also accounts for 4 bytes as a packet header overhead. Energy cost is the energy required for the transmission and reception of a single bit represented by E_{tx} and E_{rx}, respectively. We use R_c to link transmission and reception cost via $R_c = E_{rx}/E_{tx}$. The energy cost of a node is computed as B_t + B_r.R_c, where B_t and B_r, respectively, are the number of bytes transmitted and received by the node. The values for R_c are chosen from the set mentioned in Table 7.2. We also evaluated the performance with respect to the *network lifetime* that we define as the number of rounds before the first node runs out of its energy. The initial energy budget, E_{max}, for a node is chosen from the set mentioned in Table 7.2.

7.3 Transmission Cost

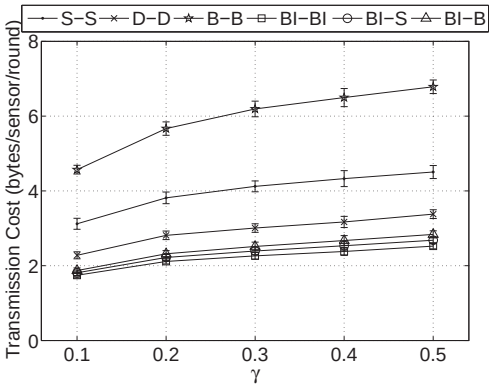
Transmission cost is measured as the average number of bytes transmitted by a node per round. Results on the transmission cost are summarized in Figure 7.1. The first trend that we can notice is that each combination of the trees has incurred a different cost. As expected, when k increases the cost for all combinations of logical trees also increases as



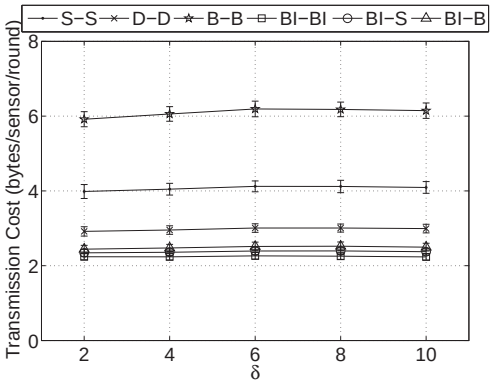
(a) Varying k



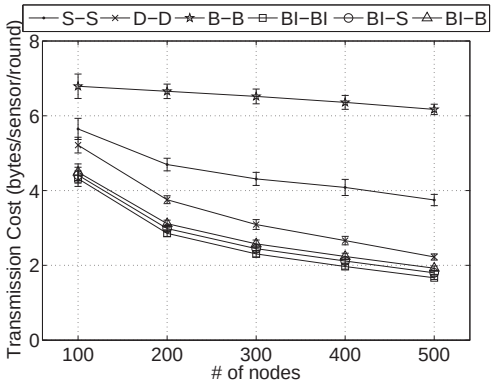
(b) Varying ω - transmission range (m)



(c) Varying γ - probability of change



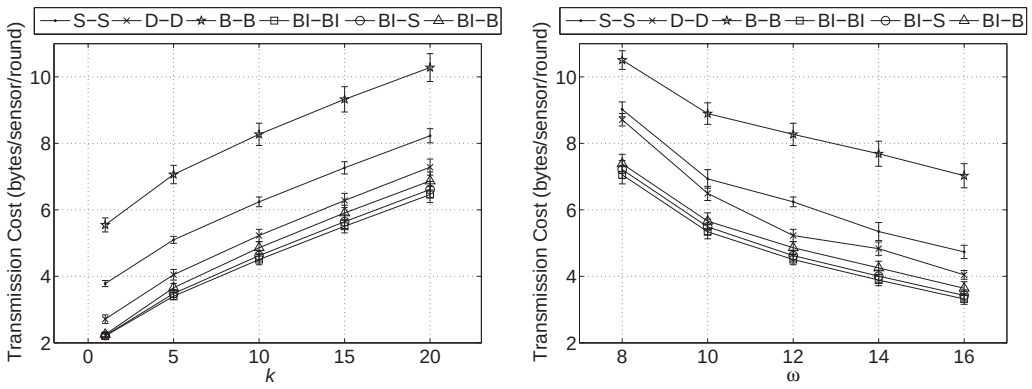
(d) Varying δ - percentage change



(e) Varying N - number of nodes

Figure 7.1: Transmission cost in the synthetic dataset.

shown in Figure 7.1(a). This can be explained by the fact that when k increases more data is needed for the root to answer the query. When transmission range (ω) increases as shown



(a) Varying k

(b) Varying ω - transmission range (m)

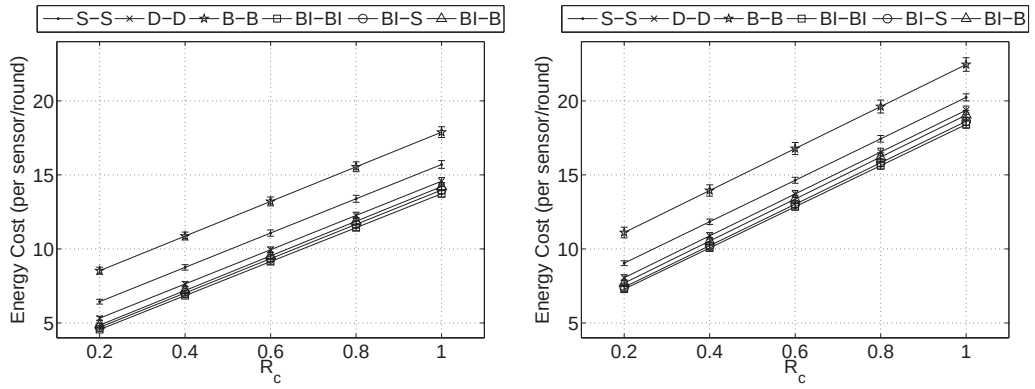
Figure 7.2: Transmission cost in the Intel dataset.

in Figure 7.1(b), the cost decreases for every combination of trees. The reason is that when ω increases the underlying logical trees "shrink" while reducing the number of non-leaf nodes, hence decreasing the communication cost. When communication traffic increases the communication cost is affected slightly as shown in Figure 7.1(c) and Figure 7.1(d). As expected, when N increases the cost decreases as shown in Figure 7.1(e).

Out of all the combinations, BI-BI, BI-S and BI-B have incurred less transmission cost. This can be attributed to the fact that these combinations use BISPT as broadcast tree, which we have already seen is efficient for broadcasting. Because of filters, the transmission cost during convergecast is reduced significantly, therefore the convergecast tree in these combinations plays less significant role compared to the broadcast tree. Combination D-D also incurs less transmission cost, which is due to the DST that is efficient as a broadcasting tree. A more interesting result, however, is that the communication cost is reduced by up to 50%, just by replacing the commonly used logical tree topology, SPT with the BISPT or DST, which can be seen while comparing S-S with D-D, BI-BI, BI-S and BI-B. Similar trends can be noticed in the results shown in Figure 7.2, that are obtained from the Intel dataset. The magnitude of results, however, has changed.

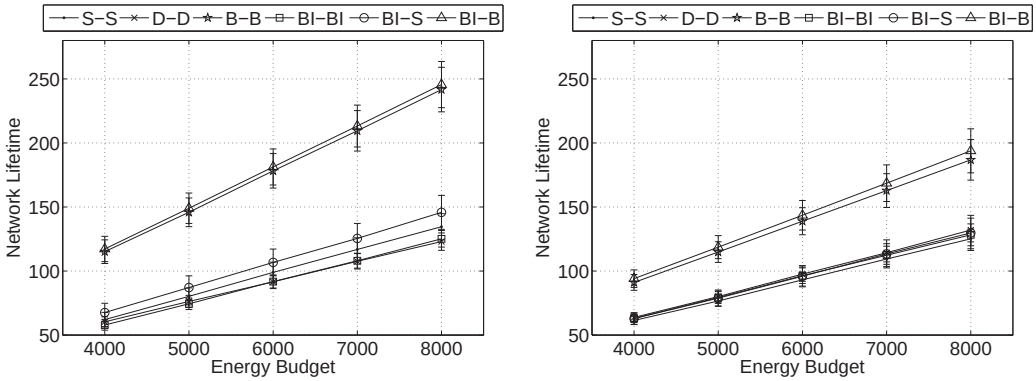
7.4 Energy Cost

The results for the energy cost are summarized in Figure 7.3. As shown in Figure 7.3(a) the energy cost for combinations D-D, BI-BI, BI-S and BI-B is less than S-S. This reflects the efficiency of DST and BISPT for disseminating messages. In particular BI-BI is able



(a) Synthetic dataset (b) Intel dataset

Figure 7.3: Varying R_c in the synthetic and Intel datasets.



(a) Synthetic dataset (b) Intel dataset

Figure 7.4: Varying energy budget in the synthetic and Intel datasets.

to reduce the energy cost by 15% as compared to S-S and more than 23% as compared to B-B. Similar results can be seen in Figure 7.3(b) that presents the result from the Intel dataset. The magnitude of the results, however, has changed. Here also, the combination BI-BI outperformed all other solutions in reducing the energy consumption.

7.5 Network Lifetime

Results for the network lifetime are summarized in Figure 7.4. Some interesting observations can be made from these results. BI-BI performs consistently well in reducing the transmission as well as energy cost. However, BI-BI's performance with respect to the network lifetime is not as good as B-B or BI-B. Recall that BI-BI uses BISPT-BISPT com-

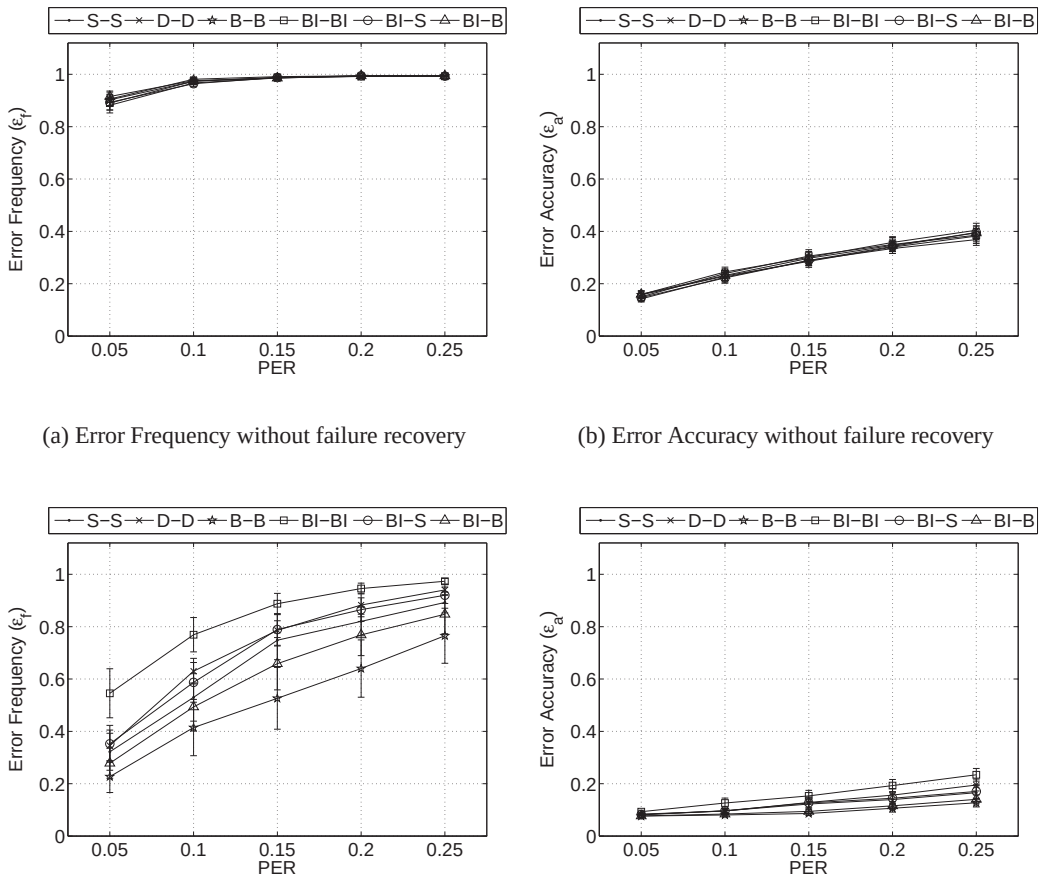
bination for broadcast and convergecast. BISPT is a tree with the least number of non-leaf nodes (as compared to all other trees used here), i.e., it is good in reducing the transmission cost and hence the energy cost. However, that results in an increased amount of "load" on the non-leaf nodes to process and forward the data for the leaf nodes in the tree. Because of this increased load "one" non-leaf node dies quickly resulting in the reduced network lifetime. (Recall our definition for the network lifetime that is defined to the number of rounds before the first node runs out of its energy.)

In contrast to that, B-B, which is a combination of BSPT-BSPT, and which did not perform well in reducing the transmission and energy costs, actually improved the network lifetime significantly compared to other combinations. This behavior can be attributed to the fact that BSPT is a "balanced" tree (recall the BSPT construction). Though the communication and energy cost accumulated by the BSPT is higher but it tends to balance that cost. This is in contrast to BISPT in which "some" nodes are given more communication and processing load. The combination BI-B (BISPT-BSPT) performs even better than B-B, though slightly. Note that BI-B uses two different logical trees, i.e., BISPT and BSPT for broadcast and convergecast, respectively, which helps in further increasing the network lifetime by "distributing" the energy cost evenly. However, the advantage of B-B is that only one structure (logical tree) needs to be maintained within the network.

These results suggest that using a "leafy" tree (e.g., BISPT) for broadcasting while using a "non-leafy" or "balanced" tree for convergecasting (e.g., BSPT) is the most useful combination for extending the network lifetime. This can be verified from the performance of BI-B as shown in Figure 7.4. Even using a single, but balanced tree (for both phases), i.e., BSPT-BSPT, is useful in extending the network lifetime, which is confirmed by the results from B-B as shown in Figure 7.4. All other combinations are found to be less useful. Nevertheless, the combination of our proposed trees BISPT-BSPT is able to extend the network lifetime by more than 60%, which is achieved by just replacing the most commonly used underlying logical tree topology, i.e., SPT-SPT combination. This confirms our intuition that the underlying logical tree topologies indeed play a significant role in query processing.

7.6 Failures and Recovery

In this section we evaluate the effectiveness of EXTOK in the presence of link failures, which may cause the root to report an incorrect result. We assume the top-k result to be



(a) Error Frequency without failure recovery

(b) Error Accuracy without failure recovery

(c) Error Frequency with failure recovery

(d) Error Accuracy with failure recovery

Figure 7.5: Failure recovery in the synthetic dataset.

incorrect if one or more of the sensor *ids* is missing or its value is not correct in the reported top-k result set. Recall that $S_{p,j}$ is the set of sensors *actually* producing the p^{th} highest value, and let us denote $S'_{p,j}$ as the set of sensors that were reported as producing the p^{th} highest correct value; both with respect to the j^{th} round. Then we can define the error accuracy ratio, $\epsilon_a = |\cup_{p=1}^{k} S_{p,j} \setminus \cup_{p=1}^{k} S'_{p,j}| / |\cup_{p=1}^{k} S_{p,j}|$. To better understand the effect of failures we also compute the error frequency, ϵ_f, which is the fraction of rounds that produced an incorrect top-k result having $\epsilon_a > 0$.

In the following experiments we set different values for the probability that a node, during a round, cannot communicate with a specific neighbor, i.e., a *link* failure. We restrict our attention of course only to pairs of nodes (i.e., edges/links) that belong to the underlying physical topology. Each link fails independently of all the other links. Moreover, link failures are assumed to be independent from one round to the next. Through link failures

we vary Packet Error Rate (PER) in the range of $\{.05, .10, .15, .20, .25\}$. As mentioned previously PER is defined as the ratio of the number of messages transmitted to the number of messages that are received successfully.

Results for failure recovery in the synthetic dataset are summarized in Figure 7.5. As shown in Figure 7.5(a) error frequency (ϵ_f) of EXTOK with any tree combination is around 90% when PER is 0.05. It means that during 90% of the rounds EXTOK reports "some" error in the returned results when links fail with a probability of 5%. ϵ_f increases to 100% as PER is increased beyond 0.15.

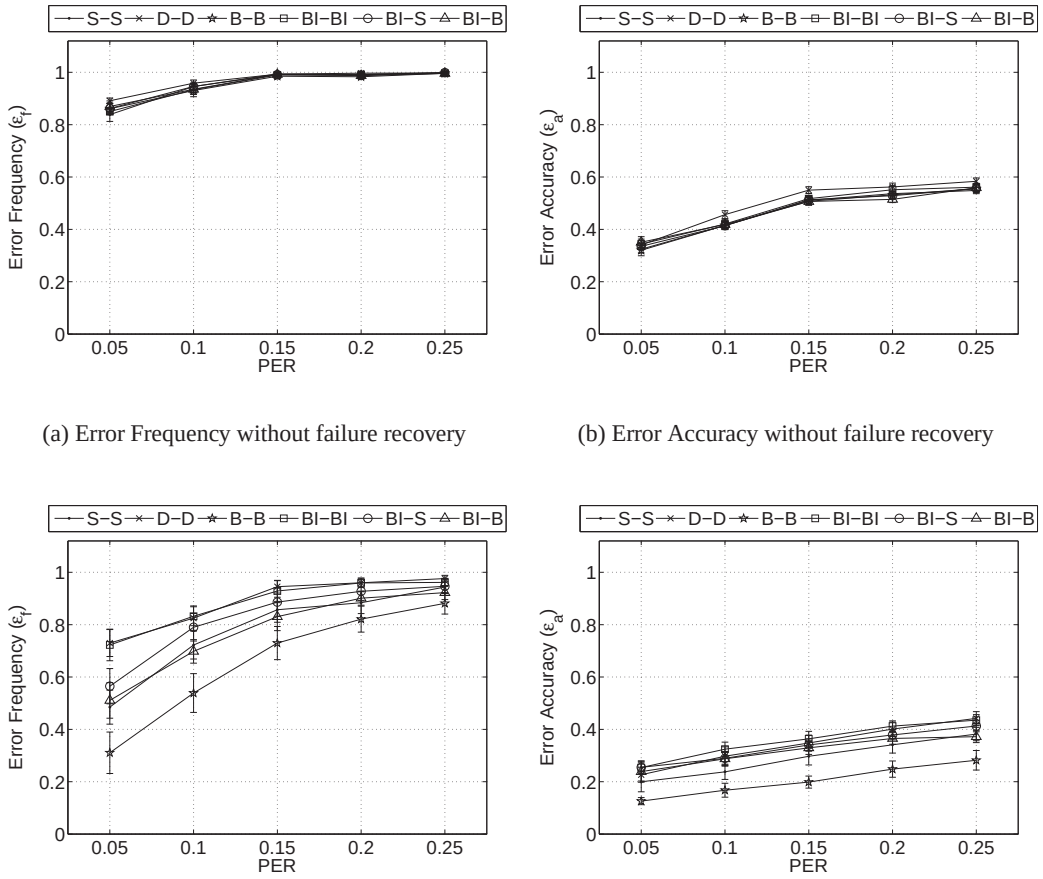
Figure 7.5(b) summarizes the "magnitude" of the error in the reported results, i.e., ϵ_a. In particular, when links fail with a probability of 5%, then 18% of the values as reported by EXTOK are incorrect. More specifically, less than 2 values are incorrect from the top-10 results (recall that the default value for k is 10). As expected, when PER is increased to 0.25 ϵ_a is also increased. In particular, around 40% of the values were found to be incorrect from the top-k values.

Figure 7.5(c) summarizes the results on ϵ_f when failure recovery solutions are applied, i.e., when RIBS and RICS are implemented on top of EXTOK. In this setting all combinations of trees respond differently, but overall the performance of EXTOK is improved significantly as shown in Figure 7.5(c). In particular, at 0.05 PER ϵ_f is reduced from 90% to 20% for B-B, to 40% for S-S and to 55% for BI-BI. The B-B combination consistently resists failures even at high PER, e.g., when PER is 0.25 ϵ_f for B-B is still less than 80%.

Figure 7.5(d) summarizes the results on ϵ_a when failure recovery solutions are applied. ϵ_a is reduced from 20% to less than 10% when PER is 0.05. Here also, among all combinations, EXTOK with B-B shows improved robustness to failures. Even at a high PER of 0.25 EXTOK with B-B returns less than 2 incorrect values.

Another observation from these results is that some tree combinations are not as robust as other tree combinations, e.g., BI-BI (BISPT-BISPT). This behavior can be explained by the fact that the trees, which are more "leafy" as compared to others are more prone to failures. The reason is that, since there are less non-leaf nodes in a "leafy" tree, any failure of such nodes leave many nodes disconnected resulting in the increased error.

Results on failure recovery from the Intel dataset are summarized in Figure 7.6. Similar trends can be observed in these results that we noticed with respect to the synthetic dataset, however, the magnitude of results is different. EXTOK with different tree combinations responded differently to failures. Nonetheless, RIBS and RICS improved the robustness of EXTOK with S-S by decreasing ϵ_f from 90% to up to 25%, and ϵ_a from 30% to up to 10%.



(a) Error Frequency without failure recovery

(b) Error Accuracy without failure recovery

(c) Error Frequency with failure recovery

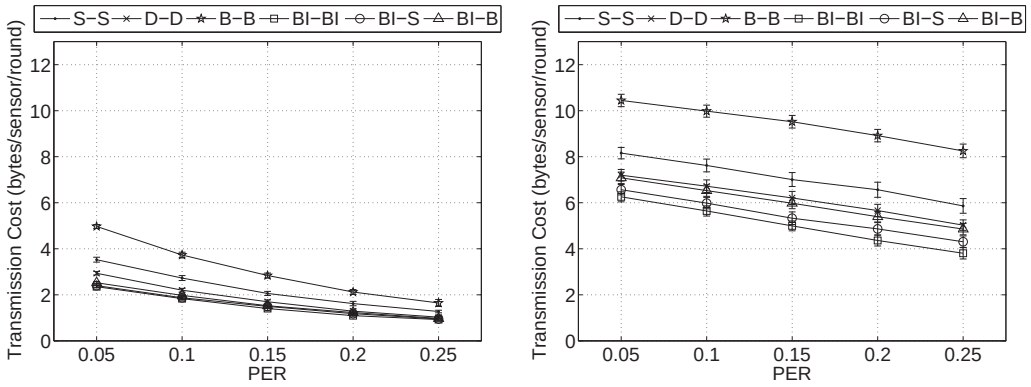
(d) Error Accuracy with failure recovery

Figure 7.6: Failure recovery in the Intel dataset.

7.6.1 Transmission Cost for Failure Recovery

In this section we evaluate the overhead in terms of transmission cost that comes with the failure recovery solutions. Recall that RIBS and RICS increase the reception cost by receiving multiple messages, and also the transmission cost by injecting the messages that are essential for failure recovery. In these experiments, first, we evaluate the transmission cost when no recovery solutions are applied, i.e., EXTOK without RIBS and RICS. Then we compare the results when RIBS and RICS are applied. Results from the synthetic dataset are summarized in Figure 7.7.

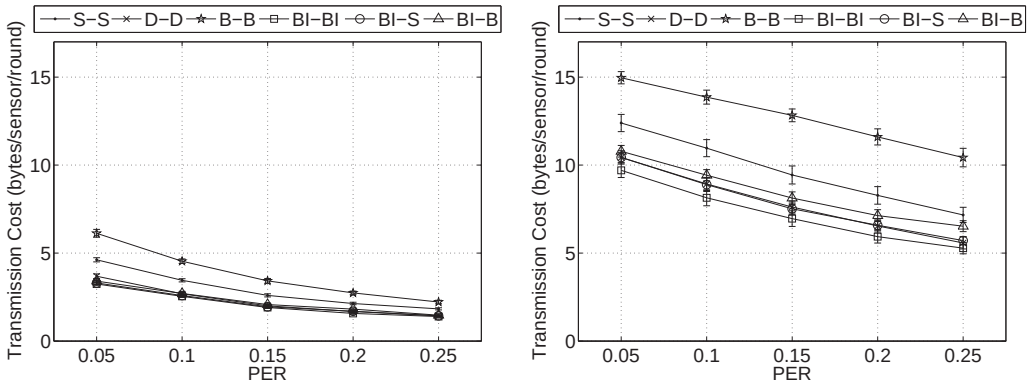
There are two observations that can be made from the results summarized in Figure 7.7(a) that are obtained without applying RIBS and RICS. The first observation is that all tree combinations have different transmission cost, which was quite expected. The second, more interesting observation is that as PER is increased the transmission cost is decreased



(a) Without failure recovery

(b) With failure recovery

Figure 7.7: Transmission cost for failure recovery in the synthetic dataset.



(a) Without failure recovery

(b) With failure recovery

Figure 7.8: Transmission cost for failure recovery in the Intel dataset.

(for all combinations of trees). This behavior can be explained by the fact that as more messages are dropped due to failures many nodes are either unable to forward the messages along the tree (during broadcasting) or they are missing data partially (during convergecast), which reduces the number of bytes being transmitted. In the case of broadcasting the effects are more severe as the children nodes under the subtree of a failed link do not participate in the broadcasting at all. In this situation many nodes do not incur the transmission cost.

Another interesting trend with respect to the tree combinations is that the combination B-B (BSPT-BSPT) that we found more robust among all other combinations incurred more cost than others. This is because BSPT is a "balanced" tree due to which it is more resilient to failures. Because many nodes are still able to participate in convergecasting and

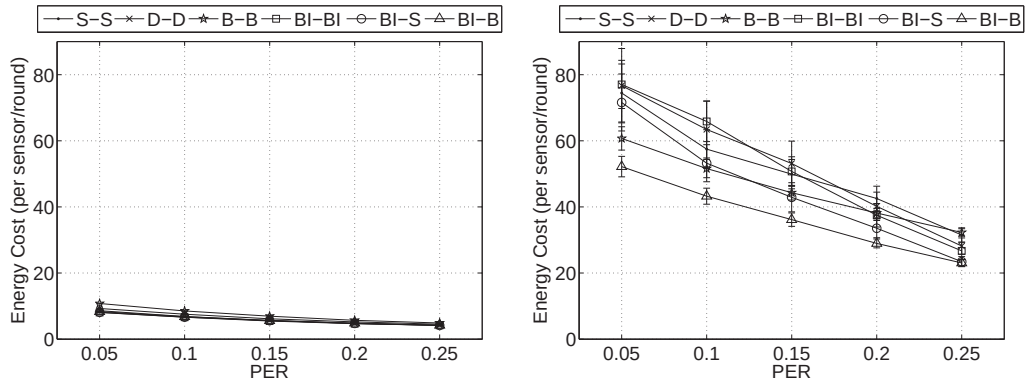
broadcasting, its transmission cost is more as compared to other tree combinations.

Figure 7.7(b) summarizes the results when RIBS and RICS are implemented on top of EXTOK with various tree combinations. The first trend that can be observed here is that the transmission cost has increased for all solutions. This was expected as nodes that were unable to send across their messages are now able to send their messages because of recovery schemes. Overall, the transmission cost is almost doubled when RIBS and RICS are implemented on top of EXTOK. One way in which this result can be interpreted is that when failures occur in the network, EXTOK reduces the chance of reporting a wrong result by paying more communication cost. Here also, when PER increases the transmission cost decreases for the same reason as explained above.

Similar trends can be observed in the results obtained from Intel dataset. As shown in Figure 7.8, different combinations of the trees incurred different transmission costs. As PER increased the transmission cost decreased (with and without recovery solutions) because of the reasons that we explained previously. The combination B-B (BSPT-BSPT) incurred more cost than other because of its robustness to failures.

7.6.2 Energy Cost for Failure Recovery

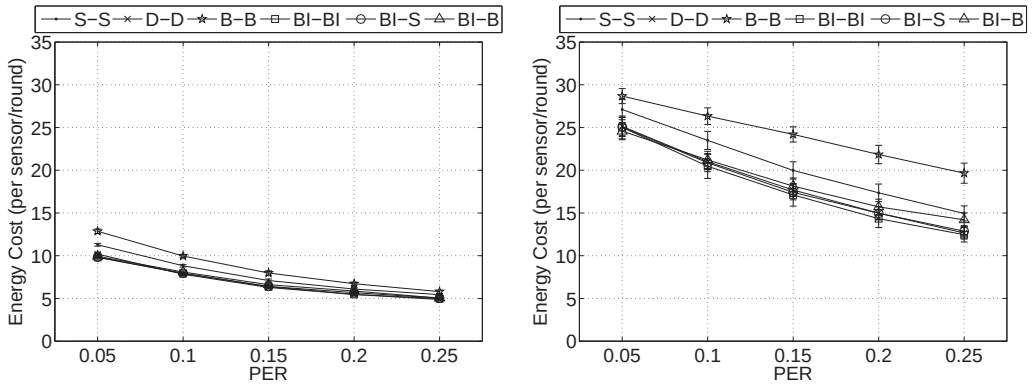
Results of the failure recovery overhead in terms of the energy cost are summarized in Figure 7.9. As expected when PER is increased the energy cost for all solutions is decreased. As explained previously, transmission cost is decreased as the PER is increased, which results in the reduced energy cost. As shown in Figure 7.9(a), B-B incurs slightly more energy than other tree combinations when no failures are recovered. When RICS and RIBS are implemented on top of EXTOK the energy consumption increases for all the solutions as shown in Figure 7.9(b). The reason for the increased energy cost is that as more nodes are able to participate in query processing more transmission as well as reception cost is incurred by the nodes. Recall that some nodes that were not able to transmit their message at all or were able to send their partial data only (which reduced their transmission cost) are now able to send their messages as well as redundantly re-aggregated data due to failures recovery. Also, note that RIBS and RICS require that nodes *listen* during backup slots. That results in the overall increased reception and transmission cost. Especially in the case of RICS many backup nodes may trigger messages for a node increasing the reception as well transmission cost. Since every node may have many backup nodes, the reception cost dominates the transmission cost in RICS. This is the reason that, when RIBS and RICS are applied, the energy cost has four to six fold increase as compared to the energy cost when



(a) Without failure recovery

(b) With failure recovery

Figure 7.9: Energy cost (R_c = 0.6) for failure recovery in the synthetic dataset.



(a) Without failure recovery

(b) With failure recovery

Figure 7.10: Energy cost (R_c = 0.6) for failure recovery in the Intel dataset.

failure recovery scheme is not used. This is in contrast to the transmission cost shown in Figure 7.7 in which we noticed less than three-fold increase. As expected, the energy cost decreases when PER increases. That is partly due to many nodes not participating in the query processing as they do not get "updated" because of the failures.

The results from the Intel dataset are summarized in Figure 7.10. Similar trends can be seen in these results as observed during the synthetic dataset, however the magnitude of results has changed. Here again, when RIBS and RICS are applied we notice an increase in the energy cost, however the increase is around three-fold as compared to a six-fold increase in the case of the synthetic dataset.

7.7 Conclusions

In this chapter we evaluated the performance of EXTOK with respect to a number of combinations of various logical trees. We evaluated EXTOK's performance in the event of failures as well while using our failure recovery solutions, RIBS and RICS. These results revealed two important observations: (1) Logical topologies are important not only for query processing cost but also for dealing with failures. In particular, some logical topologies are not only better suited than others in reducing the query processing cost, but they are also more robust to failures even at a high rate of failures. (2) The robustness of a query processing solution can be improved significantly but at the expense of an increased energy cost. In particular, RIBS and RICS reduced the chance of EXTOK giving an incorrect result (in the event of failures) from 90% to upto 20%. However, that reduction is achieved at the expense of three to six fold increase in the energy expenditure. In the next chapter we summarize this thesis while outlining some future directions in which the work presented in this thesis can further be extended.

Chapter 8

Conclusions and Future Directions

In this thesis we addressed the problem of in-network query processing in WSNs. We started this research while investigating the issues dealing with efficient processing of MAX queries in WSNs. We discovered that a particular logical tree topology, DST, significantly influenced the query processing cost for MAX queries [46]. To be precise we showed that by simply replacing a commonly used logical tree topology, SPT, with the DST (and without any changes to the original algorithms) one can reduce the query processing cost up to 50%, depending upon the type of the algorithm used for MAX queries.

Subsequently, our investigation moved beyond that work to identify several basic problems that were not only concerned with MAX queries but they were also applicable to several other in-network query processing problems in WSNs. Since considering every type of query to make a generic case for in-network query processing problems was beyond the scope of this thesis, we eventually focused on the top-k queries to take our investigation to the next level. Top-k queries are an important class of aggregation queries that are useful in many applications. Through a systematic study of the top-k query processing in WSNs we shared our findings in this thesis, which are applicable not only to the top-k queries, but also to other in-network query processing problems in WSNs. Towards that end this thesis offered several contributions that are listed as following.

- In this research we relied upon a filtering based mechanism for efficiently processing top-k queries in WSNs. The fundamental idea of filters was proposed in [50], however, we are first to propose a filtering based algorithm for processing exact top-k queries in WSNs. EXTOK does not only return exact answers but it also takes into account the tied values, which were neglected by existing solutions. Furthermore, previous proposals often "detached" the properties of a tree from the semantics of a query and the specialized techniques used, e.g., arithmetic filters, while designing

their solutions. EXTOK, however, is built upon this observation, and effectively exploits the semantics of a top-k query, filtering constraints and the underlying logical tree topology for processing the top-k queries efficiently.

- We proposed a new logical tree topology, BISPT, that is better suited than other existing logical tree topologies for one-to-all broadcasting in WSNs. In particular, BISPT outperformed SPT, as well as DST and GKLRW-tree in terms of reducing the number of messages that are required to disseminate the message in a network. However, BISPT has its own disadvantage. Because BISPT is a "leafy" tree it reduces the number of non-leaf nodes in a logical tree topology. It means that the non-leaf nodes have more communication and computation "load", which might leave the non-leaf nodes depleted of energy faster. This could become a problem in WSNs where nodes are battery powered. Nevertheless, BISPT could be more useful in static networks where energy is not an issue or in networks where the non-leaf nodes are assumed to be rich in resources. We also established a theoretical bound for the broadcasting problem, which actually guided us to discover the BISPT.

- In this research we propose efficient TDMA-based solutions for the broadcast and convergecast scheduling problems. Our approach was to, first, establish the theoretical bounds for those problems and then look into solutions, which can perform as close as possible to the established bounds. Towards that end we formalized the theoretical bound for broadcast scheduling, and presented a new "tighter" bound for convergecast scheduling. We found in our research that the broadcast and convergecast scheduling problems (with their particular constraints) enforce the construction of a logical tree topology for scheduling the nodes in a network. Towards that end we identified certain logical topologies that offer better solutions (in terms of the established theoretical bounds) for the scheduling problem. In particular, we used the BISPT for the convergecast scheduling problem, and proposed a new logical tree, BSPT, for the convergecast scheduling problem. In addition to those contributions we proposed two new scheduling algorithms, WISH and WIRES, for broadcast and convergecast scheduling, respectively. WISH and WIRES are topology-independent algorithms that can be used for scheduling any given logical tree.

- Link failures are part of any wireless network, and the solutions proposed in this thesis, e.g., algorithms and logical topologies, are not immune to failures either. Furthermore, unlike many other wireless networks, nodes in WSNs have limited resources

to deal with failures. Therefore, dealing with failures in WSNs is a challenging problem. We wanted to avoid existing re-transmissions/ACKs based solutions for energy efficiency in WSNs. Furthermore, our TDMA based scheduling solution, in particular for convergecast, seemed to have a "gap". Recall that in EXTOK many nodes do not reply during convergecast due to their filtering constraints. In this situation many nodes do not use their allocated slot during convergecast. This also means that the time slots that we allocated to many nodes are "unused" while unnecessarily increasing the convergecast latency. RICS filled this "gap" by utilizing the "unused" time slots through the backup nodes for failure recovery. With RICS we are able to avoid a costlier solution for failures based on re-transmissions/ACKs while exploiting a convergecast schedule effectively, which was a natural choice while considering energy efficiency. (Note that inefficient utilization of the time slots is not an issue during broadcast as every node participates during that operation.)

Our work in this thesis has addressed some important problems in WSNs. The solutions presented in this thesis are simple yet effective as revealed by our extensive simulation studies. Our proposed solutions are applicable, of course with some modifications, to some other problems not addressed in this thesis, e.g., extending our *one-to-all* broadcasting solution to solve the problem of *all-to-all* broadcasting in WSNs is one such possibility. Within the context of the contributions offered in this thesis, next, we list some more future directions in which our work can be extended.

- Broadcasting remains a fundamental problem in many distributed wired and wireless networks. Further research on BISPT in this context could be useful. Another direction here is to extend the BISPT based solution to non UDGs based topologies. Recall that in this thesis we applied BISPT in UDGs only, which is generally used to model a WSN. One can also use a non-UDG to model a WSN, which basically means that nodes may have different transmission ranges in that particular setup. In this situation, it remains to be seen that BISPT is a "good" solution.

- Filtering and aggregation remain two important techniques for reducing the communication cost in WSNs. Filtering is mainly used for aggregation queries in WSNs, e.g., the top-k query processing. It might be useful to survey other types of queries such as join and nearest neighbor queries for which filtering and aggregation based solutions can be designed. Towards that end another important direction is to investigate which logical topologies are better suited to "execute" the new solutions.

Obviously, if the new solutions also require broadcast and convergecast operations, we can use the solutions proposed in this thesis. However, there may be other requirements and objectives for which one may need "special" logical structures including trees and clusters.

- Another interesting direction of our future work is the processing of multiple queries. Continuous execution of a single query, e.g., top-k as considered in this thesis, may not be sufficient for some applications that seek to extract more complex information than just the top-k sensors/values. To that end, it is possible that users may want to process a combination of multiple queries during various rounds. In this situation, one can compute (in advance) multiple logical trees (built upon the same physical topology) to be used for various queries during specific rounds. Recall that the "backbone" nodes of a given logical tree propagate most of the communication traffic causing faster dissipation of energy from those nodes, which may overall reduce the network lifetime. Therefore, it might be useful to employ multiple logical trees during various rounds on "rotation" basis to evenly distribute the energy cost of the nodes to prolong the network lifetime.

- Recent advances in MIMO technologies may allow single-radio nodes to cooperate on data transmission and/or reception [19]. What that means basically in the context of this thesis is that there could be transceivers that could receive multiple transmissions concurrently, which may change the scheduling problems addressed in this thesis. In this new scenario, scheduling constraints may change as well. An important research in this direction is to construct schedules that take into account the ability of the radios to receive multiple transmissions concurrently.

- In the context of failure recovery and RICS, an important problem is to control the "redundancy" injected by the backup nodes. Recall that in RICS there may be multiple backup nodes that may forward multiple messages in a logical tree for a given node. Overall, that leads to increased energy consumption in WSNs. Unfortunately, it is not trivial for the backup nodes to know which "one" of them should trigger a message in order to recover a failure. Solving this problem in an energy-efficient manner could be a challenging issue. In the context of failures it might be interesting to explore for new techniques to deal with failures that were not recovered opportunistically.

- In this thesis, we addressed the problem of semi-matchings (with two different criteria) in bipartite graphs for constructing "special" logical trees. This work can further be extended to find other criteria for semi-matchings to discover new logical tree topologies. It might be useful to find applications (other than the logical tree constructions), e.g., in social network analysis where Maximum-Load Semi-Matching can be applied.

- BSPT proposed in Chapter 5, does not guarantee a minimum lower bound as defined in Lemma 3 with respect to trees that are *not* SPTs. It might be an interesting problem to find a logical tree topology that is optimal with respect to Lemma 3. Intuitively, a tree can be constructed in which the paths of the nodes can be *elongated*, hence increasing their hop-count (compared to the shortest possible path) while possibly *decreasing* their children-count to potentially achieve this optimal lower bound.

Bibliography

[1] Intel lab data. http://db.csail.mit.edu/labdata/labdata.html.

[2] K. Akkaya and M. Younis. A survey of routing protocols in wireless sensor networks. *Ad Hoc Networks*, 3(3):325–349, 2005.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, 2002.

[4] P. Andreou, D. Zeinalipour-Yazti, M. Vassiliadou, P. K. Chrysanthis, and G. Samaras. KSpot: Effectively monitoring the k most important events in a wireless sensor network. *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE'09)*, pages 1503–1506, 2009.

[5] V. Annamalai, S.K.S. Gupta, and L. Schwiebert. On tree-based convergecasting in wireless sensor networks. *Proc. of the IEEE Conf. on Wireless Communications and Networking (WCNC'03)*, 3:1942–1947, 2003.

[6] B. Babcock and C. Olston. Distributed top-k monitoring. *Proc. of the ACM Conf. on Management of Data (SIGMOD'03)*, pages 28–39, 2003.

[7] S. Basagni. Distributed clustering for ad hoc networks. *Proc. of the Intl. Symp. on Parallel Architectures, Algorithms and Networks*, pages 310–315, 1999.

[8] J. P. Benson, T. O'Donovan, P. O'Sullivan, U. Roedig, C. Sreenan, J. Barton, A. Murphy, and B. O'Flynn. Car-park management using wireless sensor networks. *Proc. of the 31st IEEE Conf. on Local Computer Networks*, pages 588–595, 2006.

[9] J. Blum, M. Ding, A. Thaeler, and X. Cheng. Connected dominating set in sensor networks and manets. In *D.-Z. Du, P. Pardalos (Eds.), Handbook of Combinatorial Optimization*, pages 329–369. 2005.

[10] N. M. Boers, D. Chodos, J. Huang, E. Stroulia, P. Gburzynski, and I. Nikolaidis. The Smart Condo: Visualizing independent living environments in a virtual world. *Proc. of the 3rd Int. Conf. on Pervasive Computing Technologies for Healthcare*, pages 1–8, 2009.

[11] R. R. Brooks, P. Ramanathan, and A. M. Sayeed. Distributed target classification and tracking in sensor networks. *Proc. of the IEEE*, 91(8):1163–1171, 2003.

[12] W. P. Chen, J. C. Hou, and L. Sha. Dynamic clustering for acoustic target tracking in wireless sensor networks. *IEEE Trans. on Mobile Computing*, 3(3):258–271, 2004.

[13] X. Chen, X. Hu, and J. Zhu. Minimum data aggregation time problem in wireless sensor networks. *Lecture Notes in Computer Sciences*, 3794:133–142, 2005.

[14] Z. Chen, C. Qiao, J. Xu, and T. Lee. A constant approximation algorithm for interference aware broadcast in wireless networks. *Proc. of the 26th Conf. on Computer Communications (INFOCOM'07)*, pages 740–748, 2007.

[15] Y.-H. Cho, J. Son, and Y.-D. Chung. POT: An efficient top-k monitoring method for spatially correlated sensor readings. *Proc. of the 5th VLDB Workshop on Data Management for Sensor Networks (DMSN'08)*, pages 8–13, 2008.

[16] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Math*, 86(1-3):165–177, 1990.

[17] A. Coman, J. Sander, and M. A. Nascimento. Adaptive processing of historical spatial range queries in peer-to-peer sensor networks. *Distributed and Parallel Databases*, 22(2-3):133–163, 2007.

[18] Crossbow Technology Inc. *Imote2*.

[19] S. Cui, A. J. Goldsmith, and A. Bahai. Energy-efficiency of mimo and cooperative mimo techniques in sensor networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1089–1098, 2004.

[20] A.K. Das, R.J. Markas, M. El-Sharkawai, P. Arabshahi, and A. Gray. Minimum energy broadcast trees for wireless networks: Integer programming formulations. *Proc. of the 22nd Conf. on Computer Communications (INFOCOM'03)*, 2:1001–1010, 2003.

[21] M. Duarte and Y. H. Hu. Vehicle classification in distributed sensor networks. *Jour. of Parallel and Distributed Computing*, 64(7):826–838, 2004.

[22] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Jour. of Computer and System Sciences*, 66(4):614–656, 2003.

[23] R. Gandhi, Y.-A. Kim, S. Lee, J. Ryu, and P.-J. Wan. Approximation algorithms for data broadcast in wireless networks. *Proc. of the 28th Conf. on Computer Communications (INFOCOM'09)*, 2009.

[24] R. Gandhi, A. Mishra, and S. Parthasarthy. Minimizing broadcast latency and redundancy in ad hoc networks. *IEEE/ACM Transactions on Networking*, 16(4):840–851, 2008.

[25] R. Gupta and S. R. Das. Tracking moving targets in a smart sensor networks. *Proc. of the IEEE 58th Vehicular Technology Conf. (VTC Fall'03)*, 5:3035–3039, 2003.

[26] N. J. A. Harvey, R. E. Ladner, L. Lovász, and T. Tamir. Semi-matchings for bipartite graphs and load balancing. *Jour. of Algorithms*, 59(1):53–78, 2006.

[27] M. Hefeeda. Forest fire modeling and early detection using wireless sensor networks. Technical Report 2007-08, School of Computing Science, Simon Fraser University, 2007.

[28] W. H. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. *Proc. of the 33rd Annual Hawaii Int. Conf. on System Sciences*, 2:1–10, 2000.

[29] W. H. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. *Proc. of the 33rd Annual Hawaii Intl. Conf. on System Sciences*, 2:1–10, 2000.

[30] M. Hua, J. Pei, A. W. C. Fu, X. Lin, and H.-F. Leung. Efficiently answering top-k typicality queries on large databases. *Proc. of the 33rd Conf. on Very Large Data Bases (VLDB'07)*, pages 890–901, 2007.

[31] S. C.-H. Huang, P.-J. Wan, X. Jia, H. Du, and W. Shang. Minimum-latency broadcast scheduling in wireless ad hoc networks. *Proc. of the 26th Conf. on Computer Communications (INFOCOM'07)*, pages 733–739, 2007.

[32] S. C.-H. Huang, P.-J. Wan, C. T. Vu, Y. Li, and F. Yao. Nearly constant approximation for data aggregation scheduling in wireless sensor networks. *Proc. of the 26th Conf. on Computer Communications (INFOCOM'07)*, pages 366–372, 2007.

[33] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE Trans. on Networking*, 11(1):2–16, 2003.

[34] J.Heidemann, R. Govindan, F. Silva, D. Estrin, C. Intanagonwiwat, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. *Proc. of the 18th ACM Symp. on Operating Systems Principles*, pages 146–159, 2001.

[35] B. Jiang, J. R. Smith, M. Philipose, S. Roy, K. Sundara-Rajan, and A. V. Mamishev. Energy scavenging for inductively coupled passive RFID systems. *IEEE Transactions on Instrumentation and Measurement*, 56(1):118–125, 2007.

[36] H. Karl and A. Willig. A short survey of wireless sensor networks. Technical Report 03-018, Telecommunication Networks Group, Technical University of Berlin, Germany, October 2003.

[37] A. Kesselman and D. R. Kowalski. Fast distributed algorithm for convergecast in ad hoc geometric radio networks. *Journal of Parallel and Distributed Computing*, 66(4):578–585, 2006.

[38] R. Klasing, A. Navarra, A. Papadopoulos, and S. Perennes. Adaptive broadcast consumption (ABC), a new heuristic and new bounds for the minimum energy broadcast routing problem. *Proc. of the 3rd IFIP-TC6 International Networking Conference (LNCS)*, 3042:866–877, 2004.

[39] B. Krishnamachari, D. Estrin, and S. Wicker. Modeling data-centric routing in wireless sensor networks. *Proc. of the IEEE Conf. on Computer Communications (INFOCOM'02)*, pages 1–11, 2002.

[40] F. Li and I. Nikolaidis. On minimum-energy broadcasting in all-wireless networks. *Proc. of the IEEE Conference on Local Computer Networks (LCN'01)*, pages 193–202, 2001.

[41] C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Jour. on Selected Areas in Communications*, 15(7):1265–1275, 1997.

[42] H. Luo, F. Ye, J. Cheng, S. Lu, and L. Zhang. TTDD: Two-tier data dissemination in large-scale sensor networks. *Proc. of 8th Int. Conf. on Mobile Computing and Networking (MOBICOM'02)*, pages 148–159, 2002.

[43] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. *Proc. of the 5th symposium on Operating systems design and implementation (OSDI'02)*, 36:131–146, 2002.

[44] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD'03)*, pages 491–502, 2003.

[45] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. *Proc. of the ACM Int. Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, pages 88–97, 2002.

[46] B. Malhotra, M. A. Nascimento, and I. Nikolaidis. Better tree - better fruits: Using dominating set trees for max queries. *Proc. of the 5th VLDB Workshop on Data Management for Sensor Networks (DMSN'08)*, pages 1–7, 2008.

[47] B. Malhotra, I. Nikolaidis, and J. Harms. Distributed classification of acoustic targets in wireless audio-sensor networks. *Computer Networks*, 52(13):2582–2593, 2008.

[48] B. Malhotra, I. Nikolaidis, and M. A. Nascimento. Distributed and efficient classifiers for wireless audio-sensor networks. *Proc of the 5th Int. Conf. on Networked Sensing Systems (INSS'08)*, pages 203–206, 2008.

[49] B. McLaughlan and K. Akkaya. Coverage-based clustering of wireless sensor and actor networks. *Proc. of IEEE Int. Conf. on Pervasive Services (ICPS'07)*, pages 45–54, 2007.

[50] C. Olston, B. Loo, and J. Widom. Adaptive precision setting for cached approximate values. *Proc. of the ACM Conf. on Management of Data (SIGMOD'01)*, pages 355–366, 2001.

[51] O.Younis and S. Fahmy. HEED: A hybrid, energy-efficient, distributed clustering approach for ad-hoc sensor networks. *IEEE Trans. on Mobile Computing*, 4(4):366–379, 2004.

[52] S.-J. Park, R. Vedantham, R. Sivakumar, and I. F. Akyildiz. A scalable approach for reliable downstream data. *Proc. of the ACM Int. Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'04)*, pages 78–89, 2004.

[53] I. C. Paschalidis, K. Li, R. M. Estanjini, Y. Lin, and D. Guok. Intelligent forklift dispatching in warehouses using a sensor network. *Proc. of the 17th Mediterranean Conference on Control and Automation*, pages 112–114, 2009.

[54] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[55] Y. Sankarasubramaniam, O. B. Akan, and I. F. Akyildiz. ESRT: Event-to-sink reliable transport in wireless sensor networks. *Proc. of the 4th ACM Int. Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'03)*, pages 177–188, 2003.

[56] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Tina: A scheme for temporal coherency-aware in-network aggregation. *Proc. of the 3rd ACM Int. Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'03)*, pages 69–76, 2003.

[57] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Balancing energy efficiency and quality of aggregate data in sensor networks. *Jour. of Very Large Data Bases (PVLDB)*, 13(4):384–403, 2004.

[58] A. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. *Proc. of the 22nd IEEE Conf. on Data Engineering (ICDE'06)*, pages 68–78, 2006.

[59] A. Silberstein, R. Braynard, and J. Yang. Constraint-chaining: On energy-efficient continuous monitoring in sensor networks. *Proc. of the ACM Conf. on Management of Data (SIGMOD'06)*, pages 157–168, 2006.

[60] A. Silberstein, K. Munagala, and J. Yang. Energy-efficient monitoring of extreme values in sensor networks. *Proc. of the ACM Conf. on Management of Data (SIGMOD'06)*, pages 169–180, 2006.

[61] S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. *Proc. of the IEEE Int. Conf. on Communications (ICC'01)*, 2:472–476, 2001.

[62] S. Spieker and C. Rhrig. Localization of pallets in warehouses using wireless sensor networks. *Proc. of the 16th Mediterranean Conference on Control and Automation*, pages 1833–1838, 2008.

[63] J. A. Stankovic. *Lecture Notes on Wireless Sensor Networks.* Dept. of Computer Science, University of Virginia, USA, 2006.

[64] F. Stann and J. Heidemann. RMST: Reliable data transport in sensor networks. *Proc. of the Int. Workshop on Sensor Net Protocols and Applications (SNPA)*, pages 102–112, 2003.

[65] D. Tacconi, D. Miorandi, I. Carreras, F. Chiti, and R. Fantacci. Using wireless sensor networks to support intelligent transportation systems. *Ad Hoc Networks*, 8(5):462–473, 2010.

[66] A. S. Tanenbaum. Computer networks–4th ed. pages 243–343, 2003.

[67] Y. C. Tseng, S. P. Kuo, H. W. Lee, and C. F. Huang. Location tracking in a wireless sensor networks by mobile agents and its data fusion strategies. *The Computer Journal*, 47(4):448–460, 2004.

[68] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy. PSFQ: A reliable transport protocol for wireless sensor networks. *Proc. of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 1–11, 2002.

[69] P. J. Wan, K. M. Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, 9(2):141–149, 2004.

[70] P.-J. Wan, S. C.-H. Huang, L. Wang, Z. Wan, and X. Jia. Minimum-latency aggregation scheduling in multihop wireless networks. *Proc. of the 10th ACM Int. Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'09)*, pages 185–194, 2009.

[71] A. Wheeler. Commercial applications of wireless sensor networks using ZigBee. *IEEE Communications Magazine*, 45(4):70–77, 2007.

[72] J.E. Wieselthier, G.D. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. *Proc. of*

the 19th Conf. on Computer Communications (INFOCOM'00), pages 585–594, 2000.

[73] M. Wu, J. Xu, X. Tang, and W.-C. Lee. Top-k monitoring in wireless sensor networks. *IEEE Trans. on Knowledge and Data Engineering*, 19(7):962–976, 2007.

[74] H. Yang and B. Sikdar. A protocol for tracking mobile targets using sensor networks. *Proc. of IEEE workshop on Sensor Network Protocols and Applications*, pages 71–81, 2003.

[75] B. Yu, J. Li, and Y. Li. Distributed data aggregation scheduling in wireless sensor networks. *Proc. of the 28th Conf. on Computer Communications (INFOCOM'09)*, 2009.

[76] X. Yu, S. Mehrotra, and N. Venkatasubramanian. Sensor scheduling for aggregate monitoring in wireless sensor networks. *Proc. of the 19th Int. Conf. on Scientific and Statistical Database Management (SSDBM'07)*, page 24, 2007.

[77] D. Yuan. Computing optimal or near-optimal trees for minimum-energy broadcasting in wireless networks. *Proc. of the 3rd Int. Symp. on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt'05)*, pages 323–331, 2005.

[78] V. I. Zadoronzhny, P. K. Chrysanthis, and A. Labrinidis. Algebraic optimization of data delivery patterns in mobile sensor networks. *Proc. of the DEXA Int. Workshop on Mobility in Databases and Distributed Systems*, pages 668–672, 2004.

[79] D. Zeinalipour-Yazti, P. Andreou, P. K. Chrysanthis, and G. Samaras. MINT Views: Materialized in-network top-k views in sensor networks. *Proc. of the Int. Conf. on Mobile Data Management (MDM'07)*, pages 182–189, 2007.

[80] D. Zeinalipour-Yazti, Z. Vagena, D. Gunopulos, V. Kalogeraki, and V. Tsotras. The threshold join algorithm for top-k queries in distributed sensor networks. *Proc. of the 2nd VLDB Workshop on Data Management for Sensor Networks (in conjunction with VLDB 2005)*, pages 61–66, 2005.

[81] J. Zhu and X. Hu. Improved algorithm for minimum data aggregation time problem in wireless sensor networks. *Jour. of Systems Science and Complexity*, 21(4):626–636, 2008.