Scheduling and data aggregation for periodic data gathering in wireless sensor networks

by
Mario Orne Díaz-Anadón
ornediaz@gmail.com

A Thesis submitted in fulfillment of the requirements for the degree of
Doctor of Philosophy of University of London and
Diploma of Imperial College

Communications and Signal Processing Group
Department of Electrical and Electronic Engineering
Imperial College London
University of London
2011

Declaration

I declare that the content embodied in this thesis is the outcome of my own research work under the guidance of my thesis adviser Prof Kin K. Leung. Any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline. The material on this thesis has not been submitted for any degree at any other academic or professional institution.

M. O. Díaz Anadón, Date

Abstract

Wireless Sensor Networks consist of multiple energy-constrained sensing devices called sensor nodes. Typically, the sensor nodes generate data that must reach a single data sink across multiple hops. The data from neighboring nodes are usually correlated and can be aggregated and compressed inside of the network. This process is referred to as data aggregation and requires the development of data aggregation functions.

Before data aggregation functions can be developed, uncompressed measurements need to be collected. To gather such measurements, we propose a TDMA protocol that assigns time slots very efficiently in networks in which the traffic pattern and the topology change slowly.

Data aggregation requires significant coordination between the sensor nodes because the packets to be aggregated must lie at the same node at the same time, thereby affecting the routing and MAC layers. At the routing layer, we propose a protocol to quickly obtain a routing tree for data aggregation in a network where sensor nodes sleep for long periods of time. At the MAC layer, we provide a TDMA protocol in which transmissions are arranged in order suitable for data aggregation. Our protocol outperforms the existing protocols in terms of energy consumption, reliability, and spatial reuse. For networks with unreliable links, we propose a protocol to decide which packets to transmit and which packets to discard in order to balance the energy consumption of the nodes while satisfying data fidelity constraints.

The protocols proposed in this thesis control access to the wireless channel, decide where to aggregate data, and handle packet losses. They use different kinds of data aggregation functions, operate under various propagation environments, and achieve important benefits in terms of energy and delay. Together, these protocols enhance our understanding on TDMA and data aggregation for periodic data gathering.

Acknowledgment

First of all, I am greatly indebted to my thesis supervisor, Prof Kin K. Leung, for his guidance, advice and support. He brought to my attention aspects that I had to improve and that I continue to work on. He brought me into contact with other members of the WINES project, which helped provided a focus in my research. He also gave me the opportunity to present my work in conferences and visit three research groups in American universities. He showed interest in my well being and progress.

I am grateful to all the members of the WINES project, from Cambridge University and from Imperial College. I am also grateful to my hosts in my American visit, particularly Tom La Porta, Sharanya Eswaran, Fangfei Chen, Raju Kumar, Matthew P. Johnson, Wei Wei, and Don Towsley.

I am very happy to have met many friendly and intelligent students at Imperial College. I have particularly enjoyed my time with David Looney, Nikoletta Sofra, Patryk Mazurkiewicz, Archontis Giannakidis and Fernando Martinez. My time at London could not be understood either without mention to William Temple House, the place where I lived for three years and made most of my friends.

I am very grateful to my sisters, who made me feel understood and valued when I needed it the most. I appreciate my grandmother's interest in me, and thank my brother and parents, who have done so much for me.

Contents

List of Figures

List of Tables

Publications from this thesis

- Journal papers
  1. Mario Orne Díaz-Anadón and Kin K. Leung, "Efficient data aggregation and transport in Wireless sensor networks", *Wiley Wireless Communications and Mobile Computing*, 2009.
  2. Mario Orne Díaz-Anadón and Kin K. Leung, "EATP: a schedule-unaware TDMA protocol for sensor networks operating under irregular wireless environments", to be submitted to *IEEE Transactions on Parallel and Distributed Systems*.
  3. Mario Orne Díaz-Anadón and Kin K. Leung, "Transmission and packet-discard scheduling algorithms for data aggregation in irregular and unreliable wireless sensor networks", submitted to *Elsevier Computer Communications Journal*.
- Conference papers
  4. Mario Orne Díaz-Anadón and Kin K. Leung, "A test-based scheduling protocol (TBSP) for periodic data gathering in wireless sensor networks", 3rd *International Workshop on Multiple Access Communications (MACOM)*, 13-14 September 2010, Barcelona, Spain.
  5. Mario Orne Díaz-Anadón and Kin K. Leung, "Randomized scheduling algorithm for data aggregation in wireless sensor networks", *IEEE European Wireless Conference*, Lucca, Italy, April 2010.
  6. Mario Orne Díaz-Anadón and Kin K. Leung, "Dynamic data aggregation and transport in wireless sensor networks", *IEEE Symposium on Personal, Indoor, Mobile and Radio Communications (PIMRC)*, Cannes, France, September 2008.

1 Introduction

1.1 MAC Protocols for Wireless Sensor Networks

Wireless Sensor Networks (WSNs) consist of devices called *sensor nodes* that use radio communication. Each sensor node contains a sensing module, a radio transceiver, a microcontroller, and a battery. The purpose of WSNs is to collect measurements from multiple locations within a certain area. The measurements can be of multiple types, including temperature, humidity, strain, displacement, acceleration, sound and image. The monitored environments are also diverse [1].

This thesis is motivated by the WINES project [2], which researches the application of WSNs to monitor bridges, tunnels and water supply systems. The focus is on large, multihop networks with a single data sink. The sensor nodes collect high data rate signals such as acceleration and sound. They share a single wireless channel and their batteries need to last for over a year.

Most of the components of the sensor nodes are quickly becoming cheaper and better due to technological improvements, but battery technology and energy harvesting systems are not progressing as quickly. Therefore, energy-efficient operation is a key design goal in many WSNs [3, 4, 5, 6]. The transceiver is typically the most energy consuming component of the sensor nodes. It can operate in four modes: transmit, receive, idle listen, and sleep. On current hardware, the first three modes consume approximately 60 mW when the spacing between nodes is around 20 m. The sleep

mode consumes approximately one thousand times less energy than the other three modes [7]. If a node operates continuously in any of the first three modes, it depletes typical batteries within a couple of weeks. Therefore, it needs to operate in sleep mode as much as possible.

A network's operational lifetime greatly depends on the medium access control (MAC) layer. Protocols in this layer typically try to reduce the following causes of inefficiency. First, *idle listening*, which means listening for packets when none are sent. Second, *overhearing*, which means listening for packets that are addressed to other nodes. Third, *packet collisions*, which occur when several nodes sharing the same wireless channel interfere with each other. Fourth, *backoff periods*, which are delays introduced before packet transmissions to avoid collisions.

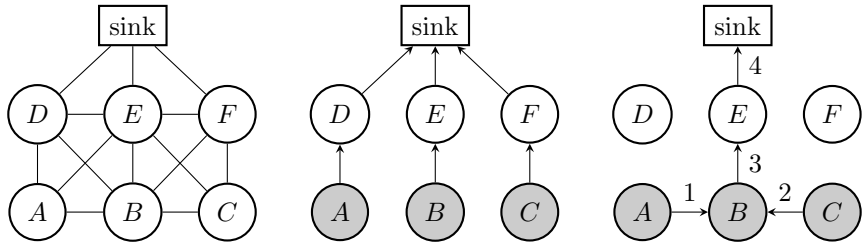
A MAC protocol is said to be *contention-based* if it requires contention before each transmission, and *frame-based* or TDMA-based if it divides time into periodic TDMA frames that contain time slots that can be reserved in advance. Reservation of time slots is typically executed during a certain period of the TDMA frames. A correct reservation system prevents packet collisions and idle listening.

1.2 In-Network Data Aggregation

Typically, as the distance between two sensor nodes decreases, the cross-correlation of their measurements increases. This cross-correlation enables a process referred to as *in-network data aggregation*. It consists in having neighboring nodes transmit their data to another neighboring node. This node aggregates and compresses the data before relaying them, thereby reducing the data volume that travels to the data sink.

Figure 1.1 illustrates data aggregation in a small network with a single data sink and three data sources: A, B and C. Panel (a) shows the connectivity graph, and

the two other panels show possible routing trees. In the connectivity graph, two nodes are linked with an edge if they can communicate directly with each other. Since there are no edges between every pair of nodes, the network is multihop.



(a) Connectivity graph (b) SPT (c) MST

Figure 1.1: A simple network (a) and two routing trees (b, c).

The two routing trees in Figure 1.1 are rooted at the data sink. They indicate the path followed by the information from A, B and C as it travels to the data sink. In these trees, every node receives data from its children nodes and transmits packets to its parent node. The tree in Figure 1.1b is a shortest path tree (SPT), which is a tree with minimal number of hops from each node to the data sink. The tree in Figure 1.1c is a minimum Steiner tree (MST), which is a tree that connects all the data sources with the data sink with the minimum number of links.

The amount of in-network aggregation depends on the routing tree. An example of tree that does not enable in-network data aggregation is the SPT in Figure 1.1b. This tree does not enable aggregation because the earliest common ancestor of the data sources is the data sink. An example of a tree that enables in-network data aggregation is the MST in Figure 1.1c. In this tree, B is an *aggregation point* which aggregates and compresses the packets from A, B and C.

Figure 1.1c shows a TDMA schedule for data aggregation that consists of four time slots. In the first time slot, A transmits a packet to B. In the second time slot, C transmits a packet to B. Then, a brief period is needed for B to aggregate the

packets from A, C and itself into a single packet. Finally, in the third and fourth time slots, the aggregated packet travels to the data sink through E.

An *aggregation function* is a function that combines a number of packets into a single packet whose size is smaller than the sum of the sizes of the input packets. The quality of an aggregation function is measured by the size reduction it provides. An aggregation function is *repeatable* if its output can be aggregated again. An example of a repeatable aggregation function is the maximum. The maximum of arbitrarily many numbers is simply a number, and the maximum of this number and other numbers can be computed. An example of an unrepeatable aggregation function is the cross-correlation of two waveforms, which yields a number that cannot be cross-correlated with other waveforms. Aggregation functions can also be classified based on whether they tolerate duplicates [8]. Some aggregation functions exploit temporal correlation [9]. Some domain-specific aggregation functions for acoustic emission signals are presented in [10]. Other aggregation functions are constructed in response to an SQL query, as if the sensor network were a data base [11, 12].

1.3 Phases of Event-Triggered Data Gathering

A network's operation is said to be *event-triggered* if the sensor nodes hardly generate any data until an unpredicted event occurs. For example, in a network monitoring bridge vibrations [13], the sensor nodes gather acceleration measurements continuously, but only report to the data sink the measurements indicating large vibrations.

This thesis considers event-triggered applications with and without data aggregation with the following properties. First, the sensor nodes and their environment are mostly static. Second, the network must start reporting the events shortly after they are detected. Third, the duration of the event is long enough to warrant the overhead of constructing a routing tree and a TDMA transmission schedule. Fourth,

while the network is reporting an event, it may need to adapt to minor topological changes efficiently, but not necessarily quickly. Some examples of applications with these properties are vibration and [13] and acoustic-emission [14, 15, 10] monitoring in bridges and tunnels.

The WSN goes through the following phases:

a) Initialization phase

Every node keeps its transceiver on during this whole phase. In this phase The first goal of this phase is to create a routing tree based on every node's *tier*, which is its minimum number of hops to the data sink. This goal is achieved through a process that begins with a packet transmission from the data sink. Every node that receives this packet sets the data sink as its parent node, sets its tier to 1, and reports its parent node and tier to its neighbors. Then, every node without tier that receives packets from nodes in Tier 1 records the identities of these nodes, selects one of them as its parent node, sets its tier to 2, and reports its tier and parent node to its neighbors. This process continues until every node has reported its information to its neighbors and the routing tree has been constructed.

The second goal of the initialization phase is to provide initial time synchronization to the sensor nodes. The synchronization information flows from the data sink to every node using the Flooding Time Synchronization Protocol (FTSP) [16]. Every node receives synchronization information from its parent node and relays it to its children nodes. Time synchronization can be used to timestamp measurements or to schedule active periods during the quiet phase.

b) Quiet phase

While no events occur, the network remains in a quiet phase in which no data needs to be reported. Nevertheless, the sensor nodes need to listen periodically to their neighbors in case their help is needed to relay packets towards the data sink. In

addition, the sensor nodes need to resynchronize themselves using FTSP in order to compensate for clock drifts. If a node does not receive synchronization information, it keeps its transceiver on continuously in case it receives synchronization information at another time, and it periodically transmits packets requesting its neighbors to report its identities.

c) Initial routing phase

The detection of the event triggers the initial routing phase. The operation of this phase depends on whether data aggregation is used. If data aggregation is not used, this phase only requires that the sensor nodes with data to transmit request their parents to listen. If data aggregation is used, this phase should construct a new routing tree optimized for the set of data sources associated with the current event.

d) Initial scheduling phase

This phase obtains an initial TDMA transmission schedule for the data transmission phase. If data aggregation is used and latency is important, the schedule must ensure that the time slot assigned to transmit the aggregate of several packets comes at a later position in the TDMA frame than the slots to receive those packets.

e) Data transmission phase

This phase is used to report the event using the routing tree and the schedule computed in the two previous phases. This phase should be able to adapt to topological and traffic changes. This phase should also sensibly decide which packets to discard if it is necessary due to packet losses.

1.4 Research Objective

The research objective of this thesis is to enhance our understanding on the use of TDMA and data aggregation for periodic data collection. We do so by designing and evaluating new wireless communication protocols, which cover the network phases

described above and the following data aggregation models:

1. *Without data aggregation.* The goal is to provide a new TDMA scheduling protocol that obtains an initial schedule fast and reliably, and that adapts the schedule efficiently during the data transmission phase.

2. *With unrepeatable data aggregation.* The goal is to decide the aggregation points in a large network.

3. *With repeatable aggregation.* The goals are to obtain a routing tree quickly, to obtain a TDMA schedule reliably, and to select which packets to discard.

In Chapter 2, we highlight the drawbacks of existing protocols, elaborate on the research goals, and present the thesis structure.

2 Limitations of Existing Protocols for Periodic Data Gathering

This chapter motivates the study of data gathering in large WSNs without and with data aggregation. We argue that many applications require the deployment of an aggregation-less network before a network with aggregation can be deployed. We also discuss the limitations of the related work. Based on these limitations, we describe the thesis structure and goals.

2.1 Achieving Efficient, Large WSNs

Literature on WSN has been criticized [17] for assuming large networks and proposing complex protocols, whereas most practical deployments are small and use simple protocols. However, simple protocols are insufficient in applications with high data rates. For example, Crossbow's XMesh communication stack [18] suffers frequent packet collisions in bridge vibration monitoring applications generating several kbps [19]. For high data rates and periodic traffic, TDMA greatly improves the energy efficiency.

Data aggregation greatly reduces the amount of data that travels to the data sink and thus prolongs the battery lifetime of the nodes close to the data sink, which are the first to deplete their batteries. Data aggregation requires that the measurements

from neighboring nodes are correlated, which means that the sensor nodes are close to each other. Increasing the node density increases the hardware and deployment cost, but improves the reliability for the following reasons. First, more measurements can be averaged. Second, malfunctioning nodes can more easily be detected by comparing their measurements with those of their neighbors. Third, the measurements are more likely to reach the data sink because the network is more likely to be connected.

Data aggregation functions are often application-specific and hard to develop due to lack of application knowledge. For example, there is no publicly available information to decide which features of an acoustic waveform indicate a fracture within the cable of a suspension bridge. As another example, it is unknown how to use vibration data to assess a bridge's structural health, except, of course, if those vibrations are exceptionally large.

Before data-aggregating functions can be developed, exhaustive, uncompressed measurements must be collected in order to better understand the application. Therefore, the steps needed to develop an efficient data-aggregating WSNs are the following:

- Send exhaustive, uncompressed measurements to the data sink. The batteries of the sensor nodes are quickly depleted with this collection step, but the collected information is necessary for the next step.

- Analyze the exhaustive measurements to decide which events are relevant and how they can be identified and characterized. Based on this information, the sensor nodes can decide in the final system when they need to report their measurements to the data sink.

- Develop aggregation functions to reduce the transmitted data volume to a minimum. For example, most of the relevant information of the long acoustic

waveform detected following a bridge fracture can be summarized into only seven parameters [10].

- Adapt the communication protocols to the aggregation functions.

Efficient in-network data aggregation requires a strong interaction between the layers of the OSI communication stack. Data aggregation strongly depends on the aggregation functions, the node density, the node reliability, and the link reliability.

In this thesis, we contribute to development of data-aggregating WSNs in several ways. First, we propose a protocol that improves the efficiency of the exhaustive, uncompressed data gathering step, which is often necessary. Second, we develop communication protocols for two different data aggregation functions, namely unrepeatable and repeatable.

2.2 Contention-Based MAC Protocols for WSNs

Most contention-based protocols try to keep the sensor nodes' transceivers in sleep mode to save energy. These protocols respond to traffic changes quickly, but suffer frequent packet collisions under high traffic loads. They make the sensor nodes sleep periodically in order to avoid idle listening. The protocols where the sensor nodes do not synchronize their sleep periods are referred to as *random*, and the protocols that synchronize them are referred to as *slotted* [7].

The random protocols force the transmitters to precede their packets with preambles longer than their recipients' sleeping time in order to ensure packet reception [20, 21, 22, 23]. Therefore, the transmitters consume a significant amount of energy in transmitting preambles if the sleep period is long. B-MAC [23] is a popular random protocol that sets the sensors' sleep duration during the deployment and decides whether the wireless channel is idle or busy by taking five channel samples.

The *slotted protocols* [24, 21, 25] are characterized by synchronizing the sleeping times of the sensor nodes, which reduces the preamble length. However, they suffer intense contention and frequent packet collisions under high loads. DMAC [25] is a slotted protocol in which the sleep periods of nodes in different tiers are staggered so that the data can be transmitted quickly from the data sources to the data sink.

2.3 Frame-Based Protocols for WSNs

The frame-based protocols [26, 27, 28, 29, 30, 31] are also called TDMA protocols because they divide time in TDMA frames. Each frame contains N Data Blocks (DB) that are labeled $\{\mathrm{DB}_1, \ldots, \mathrm{DB}_N\}$. A DB is the minimum number of time slots that can be assigned to a node. In some protocols, a DB is simply one time slot used for communication in a single direction. In other protocols it consists of two time slots: the first time slot to transmit a DATA packet in a certain direction, and the second time slot to transmit an ACK in the opposite direction.

The frame-based protocols require time synchronization. The synchronization overhead is usually small relative to the total energy consumed in packet transmissions. The typical clock drift is below 10 ppm, which usually requires one resynchronization approximately every minute [5]. If the sensor nodes are engaged in periodic packet transmissions more often than that, the sensor nodes can piggyback the synchronization information in those transmissions. Furthermore, accurate time synchronization is sometimes needed anyway to timestamp the measurements.

The TDMA scheduling problem is to obtain a TDMA schedule that indicates which transmission and reception DBs are assigned to each sensor node. The transmission DBs are used to transmit DATA packets, and reception DBs are used to receive DATA packets. The schedule allows the sensor nodes to save energy by turning off their transceivers in the DBs in which they are not scheduled to transmit or receive.

2.3.1 Limitations of Existing Frame-Based Protocols

Scheduling Failure Probability The DB assigned to a node is said to be *infeasible* if the node cannot communicate during that DB due to excessive interference. The *scheduling failure probability* p_f is the probability that a node is assigned an infeasible slot. The existing protocols use interference models to decide whether the DB assigned to a node is feasible.

The most common interference model is the *k-hop interference model*, which neglects the interference generated more than k-hops away from a receiver. The protocols that use this model with $k = 2$ suffer a significant failure probability p_f, and the protocols that use this model with $k > 2$ achieve very little concurrency, as shown in [27] and Chapter 3. The *concurrency* of a schedule is the average number of nodes that are assigned the same DB.

An uncommon but realistic interference model is the *physical interference model*, which uses the SINR to determine the success of a transmission [32, 33], and in some cases consider Rayleigh fading [34]. However, it incurs high overhead because it requires collecting link quality information about every link in the network and transmitting this information to the data sink.

Energy Consumption Limitations The TDMA scheduling protocols can be classified as centralized or distributed. In the centralized protocols [35, 36, 37, 38], the data sink receives topological information from the network, decides the complete schedule, and transmits to each node the information it needs. In large networks, this method is slow and strains the batteries of the nodes close to the data sink because they act as relays. The current distributed scheduling protocols [39, 27, 31, 30] require every sensor node to know its neighbors' schedules in order to decide which DBs are free. This requirement wastes energy in listening in case of schedule changes.

Therefore, there is a need to develop a distributed TDMA protocol without this requirement.

2.4 Tree-Based Data Aggregation

Tree-based data aggregation requires a routing tree and a timing scheme. The routing tree decides the aggregation points and the path followed by the packets on their way to the data sink. The timing scheme decides for how long to wait for a packet or when to transmit it.

2.4.1 Routing for Tree-Based Data Aggregation

The optimal routing tree varies with the properties of the aggregation functions. An aggregation function is *perfect* if it compresses any number of packets into a single packet with the same size as the biggest input packet. An aggregation function is *costless* if its execution does not incur any cost such as energy or time.

If the aggregation function is perfect and costless, the optimal tree is the unweighted Minimum Steiner Tree (MST); otherwise, the optimal is the weighted MST [40, 38]. Obtaining either kind of MST is an NP-complete problem [41, 42] whose solution can be approximated in polynomial time [43, 44, 45, 46]. The quality of the approximation is particularly good if little compression is possible. This is natural because, if no compression is possible, the optimal tree is the Shortest Path Tree (SPT), which can be computed quickly and efficiently with the distributed Bellman Ford algorithm.

Let the tree construction time T_{constr} be the interval between the time when an event is detected until every node becomes aware of its parent node in the routing tree. The construction time depends on the MAC protocol used. For example, consider a sensor network in which an event is detected in tier K. If S-MAC [47]

is used, which is a MAC protocol that makes the sensor nodes sleep synchronously with period T_{CCA}, the tree construction time T_{constr} is at least $2KT_{\mathrm{CCA}}$ because the tree construction instruction needs to travel from the data sources to the data sink, and then the routing tree information has to travel from the data sink to every node.

Therefore, in order to construct the routing protocol quickly, we need a new protocol that considers the interaction between the MAC and routing layers.

2.4.2 Timing for Tree-Based Data Aggregation

If the wireless links are reliable, low latency is needed, and a TDMA is used, TDMA should arrange DB assignments in such an order that every node receives packets from all its children before it is scheduled to transmit its own packet. This order allows every sensor node to have the aggregate of its children's packets ready to be transmitted in its transmission slot in the current frame, but makes the modification of the schedule more complex. Modifying the schedule becomes unnecessary if the schedule has a very low failure probability. Therefore, a new TDMA schedule protocol with a very low failure probability is needed.

If two packets are expected to be aggregated at a certain node, the first packet to reach the node has to wait for the other packet. In unreliable networks, it is unknown whether the second will arrive if at all, and thus the packet should not wait longer than a certain timeout. The timeout should be short enough to provide low latency, and long enough to avoid missing many aggregation opportunities.

The relationship between the timeouts of nodes in different tiers in the routing tree is important. TAG [48] and cascading timeouts [49] are two protocols that set the timeouts of nodes far away from the data sink before the timeouts of nodes close to the data sink [48, 49]. These protocols are designed for contention-based medium access and are unsuitable for schedule-based access. Furthermore, they

force the nodes with the weakest link to make many packet retransmissions and thus to consume a lot energy. Therefore, there is a need for timing schemes for schedule-based access that balance the energy consumption of different sensor nodes.

2.5 Cluster-Based Data Aggregation

A cluster is a self-coordinated group of neighboring sensor nodes. One cluster member acts as the group coordinator and is referred as *cluster head*. Since the cluster members are close to each other, their data is typically correlated and can be aggregated and compressed. Every sensor node transmits its data to its cluster head, which aggregates the data from all the nodes in the cluster. Data is only aggregated once, and thus cluster-based approaches do not require repeatabl eaggregation functions.

Only a few clustering schemes segment the network into clusters upon detection of the event [50]. Most protocols change the clusters periodically to distribute the energy consumption evenly among the sensor nodes, but do not recompute the clusters after each event [51, 52, 53, 54, 55]. As a result, the same clusters are used for events with different locations and sizes. Fixed clusters are suboptimal because the optimal aggregation point are event-dependent, but perform well for moderate degrees of spatial correlation [44].

Choosing the optimal cluster size involves the following tradeoff. On the one hand, big clusters are beneficial in that they are more likely to contain all the data sources associated with an event, and thus can aggregate all the data before relaying it to the data sink. On the other hand, big clusters are detrimental in that they require transmission of uncompressed data across many hops within each cluster.

The optimal cluster size increases with the cluster's distance to the data sink. This is because the clusters far away from the data sink have to relay less data

than the clusters near the data sink. However, the existing clustering schemes that produce unequal cluster sizes are not scalable and do not consider multi-hop clusters. Therefore, we need new clustering schemes with these properties.

2.6 Other Data Aggregation Approaches

The following data aggregation approaches are not considered in this thesis, but we discuss them as follows for completeness.

First, *multi-path* approaches [56, 57, 58] are those that transmit duplicates of the same information to the data sink across multiple paths. This redundancy allows the data sink to receive all the information regardless of the loss of some duplicates. As the duplicates travel towards the data sink, they can be aggregated and compressed multiple times. For example, in Figure 1.1a, if nodes A, B, and C are data sources, the operation would be as follows. Node D receives and compresses the data from {A,B} and transmits the result to the data sink. Node E receives and compresses the data from {A, B, C} and transmits the result to the data sink. Node F receives and compresses the data from {B, C} and transmits the result. Therefore, the sink receives three times the data from D. However, although compression occurred at three nodes, the energy consumption of B is trebled unless B broadcasts its data simultaneously to {D, E, F}. Therefore, multi-path approaches aggregate and compress data but may not save energy.

Second, *unstructured aggregation* consists in aggregating data packets opportunistically [59, 60]. No routing tree or schedule is constructed for data aggregation. Therefore, lower overhead is incurred, but less packets are aggregated and the data transmission phase is more inefficient. Every sensor node holds its packets for random periods of time before relaying them, which increases probability that two packets that can be aggregated stay at a node at the same time. Every sensor node also

chooses its next hop in order to promote data aggregation. Among all its neighbors lying fewer hops from the data sink than itself, it chooses the neighbor holding the greatest number of packets.

Third, *distributed source coding* [61] is an alternative to data aggregation based on the Slepian-Wolf theorem [62]. This information-theoretic result states that, if two data-generating nodes know the cross-correlation between each other's data, they can independently compress their data as much as if they knew each other's information. If the two nodes encode their data in this way, their packets do not need to go through the same nodes or wait for each other. Therefore, data compression and data transmission become independent. This kind operation is very desirable, but difficult to implement because the cross-correlation of different nodes is typically unknown.

2.7 Thesis Structure and Goals

This thesis addresses data gathering with different levels of data aggregation. Its chapters are organized from no aggregation to more aggregation: Chapter 3 addresses the non-aggregation case, Chapter 4 the unrepeatable aggregation case, and Chapters 5 to 7 the repeatable aggregation case. This is order is followed in the development of may monitoring applications, as discussed in Section 2.1.

Chapter 3 presents our first contribution, namely a TDMA scheduling protocol called EATP. Despite being proposed and evaluated for the non-aggregation case, it can be applied for various aggregation models if minor changes are made. EATP's goals are to obtain an initial schedule quickly and to modify the schedule during the data transmission phase in an energy efficient way. EATP is more efficient that the existing TDMA protocols because it spares the sensor nodes from the burden of keeping track of their neighbors' schedules.

Chapter 4 considers the routing problem in large networks that perform unrepeatable aggregation. For this kind of aggregation, a cluster-based topology is appropriate. We model and solve the problem of choosing the cluster size as a function of the distance from the data sink.

Chapters 5, 6, and 7 deal with repeatable aggregation in tree-based structures. Chapter 5 addresses the routing problem, Chapter 6 the scheduling problem, and Chapter 7 the packet-loss problem. Chapter 5 proposes a protocol called FAT, which is executed during the quiet phase and during the initial routing phase. It is designed for a network where the nodes sleep for long intervals during the quiet phase. By coordinating the sleep intervals of the nodes in different tiers, FAT constructs a routing tree quickly. Chapter 6 discusses the extension of EATP to obtain schedules for data aggregation. Chapter 7 studies a tree-based WSN that uses repeatable aggregation. Each wireless link is assumed to have a constant success probability p_s that varies for different nodes. Therefore, different nodes need to make different numbers of retransmissions and consume different amounts of energy to transmit the same number of packets. The chapter proposes different schemes to balance the energy consumption of different nodes while ensuring that sufficient information reaches the sensor nodes.

2.8 Simulation Methodology

2.8.1 Language Choice

This thesis proposes several protocols and compares their performance with that of the existing protocols. We develop custom simulators in order to compare the performance of the different protocols. The simulator in Chapter 5 is written in Matlab; the simulators in chapters 3, 4 and 6 are written in Python; and the simulator

in Chapter 7 is written in C#.

The reason for using different programming languages is partly historic. I began my doctoral research investigating what would become Chapter 5. For that work I used Matlab because I knew it well, it is an high-level programming language, and it provides interactivity and plotting capabilities. However, I found Matlab insatisfactory for writing event-triggered simulators because its poor support for object-oriented programming was poor; this support has improved since then. Furthermore, Matlab has a flat namespace and requires writing many small functions. Also, not all the Matlab scripts run under all the versions. In addition, Matlab is expensive proprietary software.

This limitations of Matlab pushed me to find an alternative. I chose to use Python [63] because it produces the most readable code I have seen, particularly for large programs. It is open source, cross platform, and comes with an extensive library. For writing an event-triggered simulator, I used the SimPy library [64], which yields very simple code because it requires keeping very little state information. For plotting my results, I used another library called Matplotlib [65], which is good for bidimensional plots. However, I later found it better to produce plots directly in LaTeX using the Pgfplots package [66].

Unfortunately, some of my Python simulations had to run for as long as a month in order to obtain the required accuracy. To speed up the execution time, I decided to use a faster language than Python in Chapter 7. The choices that I considered were C++, C#, and Java. I ruled out C++ because of does not have automatic memory management for preventing memory leaks. Between Java and C# I chose the latter because it has more features [67] . Unfortunately, I later discovered that C# has a poor command line debugger in Linux [68] so I had to use Windows.

2.8.2 Review of Some Existing Network Simulators

We review as follows some of the existing network simulators for wireless sensor networks. A popular simulator is ns-2 [69], which is written in C++ and provides a simulation interface through OTcl, a scripting language barely used elsewhere. Modelling in ns-2 has been criticized for being complex. Many protocols have been implemented in ns-2, but those implementations may be buggy and their code is continuously changing [70]. In order to address some of the defficiencies of ns-2, the ns3 simulator was proposed [71], which uses C++ and Python. A disadvantage of ns-3 is that fewer protocols have been implemented in it.

Omnet++ [72] is a good C++ framework for developing event-triggered simulators. Castalia [73] is an add-on for Omnet++ that provides realistic wireless channel models to be used in wireles sensor networks simulations. However, Castalia is not popular and few protocols have been implemented in it.

Other simulators aim to execute the same code that is going to be deployed in the sensor ndoes. TOSSIM [74] is a simulator for TinyOS [75], a very popular operating system for wireless embedded sensor networks. Avrora is a simulator of sensor nodes that run the AVR microcontroller [76]. SunSPOT [77] is a research effort providing a Java runtime for sensor nodes and simulation environment for it. However, TinyOS is written in the nesC programming language, and its is unclear whether such a non-mainstream language will remain popular as the sensor nodes become more powerful. Avora has the limitation of simulating very specific hardware that may soon become obsolete. SunSPOT has an advantage in this respect because it uses the Java programming language, which is powerful, popular and can run under many platforms.

2.8.3 Rationale for Developing a Custom Simulator

One reason why we chose to develop our own simulator was to simplify the development of new models, which is particularly useful in chapters 4 and 7. The models in these chapters are intentionally simple so as to provide fast algorithms and provide insights about the key factors and their influence. The development simplicity is also important in Chapter 3, where we intentionally discount the energy consumed by some nodes in FlexiTP [27], a competing protocol, in order to show that even without considering this energy consumption, our protocol performs best. Developing this would have been harder in an existing simulator.

Another reason for using a custom simulator is to have full understanding of every part of the simulator. In large and complex simulators as ns-2, it is harder to understand which assumptions or parameters are used by different parts of the simulator. It is also harder to identify which variables most affect the measurements. Furthermore, bugs are continually found in ns-2 [69], so the results obtained with this simulator may not be more reliable than those obtained with our own simulator.

The intelligibility of the simulator is also important in order to reduce the probability of bugs in the program and in order to facilitate that other researchers replicate our results. The languages we used, and Python especially, often result in code that is easy to understand. In addition, there are free implementations of C# and Python under multiple platforms. Our Python and C# simulators are available in [78]. These simulators have far fewer lines of code and are easier to understand than ns-2, at least for those without ns-2 knowledge.

One reason commonly argued to using existing simulators is that those simulators are more reliable. However, even if these simulators are bug free, they may be used in an inappropriate way.

Another reason for using an existing simulator is that some protocols have been

already implemented in them. Reusing protocol implementation is compelling because it avoids some work and provides increased confidence on the protocols implementation. Furthermore, the existing implementation may provide essential information not provided in the original paper. For example, the FlexiTP paper [27] omits some MAC parameters because they are defaults in ns-2.

However, using existing protocol implementations on popular simulators has been proved difficult or impossible in the past. For example, the FlexiTP paper [27] compares FlexiTP with another existing protocol called Z-MAC [79]. The FlexiTP authors used ns-2 because ns-2 is respected, and an ns-2 implementation of Z-MAC already existed. However, the FlexiTP authors claim that the Z-MAC implementation only worked for networks with fewer than 100 nodes, presumably because of a bug. Hence, the existence of a ns-2 implementation of Z-MAC was of little use.

2.8.4 Validation Efforts

In order to validate our simulation results we do the following. First, in chapters 3 and 4 we try to use realistic, popular propagation models. Second, we use high level programming languages to reduce the probability of bugs, and we make much of the code available so that other researches can verify it. Third, we verify the correctness of the implementation in small networks by debugging them step-by-step and verifying that they function properly. Fourth, we check that the simulation results make logical sense. Other forms of validation that we have not performed and that we leave for future are the following: performing more realistic simulations; comparing our simulations results with those obtained with other simulators; implementing the protocols in actual sensor nodes and deploying them in real sites.

3 MAC Scheduling Without Data Aggregation

This chapter considers the problem of scheduling packet transmissions in a network without data aggregation. We propose a TDMA protocol called Efficient Adaptation TDMA Protocol (EATP), which is executed during the initial scheduling phase and the data transmission phase. Its main advantage is that it consumes less energy in slowly-changing networks than comparable protocols.

3.1 Problem formulation

In a multi-hop WSN with multiple data sources and a single data sink, a routing tree rooted in the data sink has been established. A TDMA schedule must be obtained during the initial scheduling phase so that it can be used during the data transmission phase. During the data transmission phase, the network remains mostly stationary, except for infrequent *network changes* such as node additions, node removals, and link failures.

Every TDMA Frame (TF) contains N Data Blocks (DBs) that are labeled DB_1, ..., DB_N. Each DB consists of a DATA slot and an ACK slot. Every sensor node in the routing tree generates one packet per TF, and each packet transmission occupies one DATA slot. Therefore, the number of DBs to be assigned to a node is equal to

its number of descendants plus one.

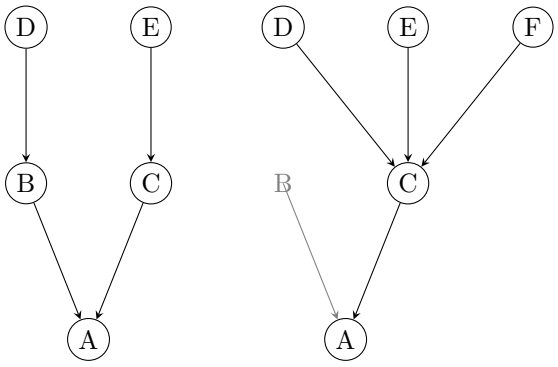
The problem is to design an initial scheduling phase that is fast and reliable, and a data transmission phase that adapts to network changes in an energy-efficient way. The initial scheduling phase must be fast because until it concludes that the data sink cannot receive any data about the event. However, during the data transmission phase, energy efficiency is more important than adaptation speed because the data sink is already receiving some information about the event. The obtained schedule should verify the following properties.

3.1.1 Properties of the Schedule

First, the schedule should have a low *failure probability* p_f, defined as the probability that the DB assigned to a sensor node is *infeasible*, which means that it contains excessive interference. If a node's slot is feasible, the node is said to *tolerate* the other nodes that are assigned that DB. It is possible that a node tolerates a second node but not the other way around.

Second, the schedule should have a high *concurrency* c, which is the average number of nodes that are assigned the same DB. The concurrency c measures the spectral efficiency, and is computed as the quotient between the total number of DB allocations and the schedule length M. The schedule length M is defined as the maximum integer that verifies that at least one node is assigned DB_M. The number N of DBs per TF may be slightly larger than the schedule length M in order to facilitate the extension of the schedule.

Third, the obtained schedule should verify the *precedence property* for the aggregation-less case. This property means that if a node is scheduled to receive a packet from one of its children nodes in the routing tree in DB_i and it is scheduled to forward that packet in DB_j, then j should be bigger than i. This property ensures a low



(a) Tree 1 (b) Tree 2

	Tree 1			Tree 2		
DB	Tx	Rx	Src	Tx	Rx	Src
1	B	A	B	D	C	D
2*	C	A	C	C	A	C
3	D	B	D	C	A	D
4	B	A	D	F	C	F
5*	E	C	E	E	C	E
6*	C	A	E	C	A	E
7				C	A	F

(c) Schedules for Tree 1 and Tree 2

Figure 3.1: An initial tree, an evolution of that tree, and schedules for each tree.

end-to-end upstream latency by enabling the packets from every sensor node to reach the data sink within a single TDMA frame.

Fourth, the schedule should verify the *low-buffering property*, which means that, for every node, $b < B$, where b is the maximum number of packets buffered by that node and B is a network-wide parameter. This property assumes no packet losses. If there are packet losses, this property does not prevent the sensor nodes' buffers from containing more than B packets, but reduces the probability of buffer overflows, which are particularly frequent in large networks because they have to relay many packets. The low-buffering property is enforced in [27].

3.1.2 Example of Schedule Adaptation to Network Changes

Figure 3.1 illustrates the adaptation of a schedule to network changes. It presents Tree 1, Tree 2, and a schedule for each of these trees. Tree 1 is the routing tree of a sensor network whose data sink is node A. Tree 2 is a new routing tree for the same network after it suffers two changes: the removal of B and the addition of F.

The table in Figure 3.1c shows the schedule of Tree 1 and the schedule of Tree 2. The network is too small to allow two nodes to transmit in the same DB. Each

row of the table indicates the packet transmitted in one DB. The `Tx`, `Rx` and `Src` columns of the table indicate the transmitter, the receiver and the source of the data packet, respectively.

The schedule of Tree 1 is used as follows. At the start of each TF, every sensor node has one packet in its buffer, namely the packet that it generated. In DB_1, node B transmits its packet to A and removes it from its buffer. In DB_2, node C transmits its packet to A and removes it from its buffer. The packet from D travels to A through B in DB_3 and DB_4. The packet from E travels to A through C in DB_5 and DB_6. Therefore, no node needs to buffer more than $b = 1$ packets.

When the network changes occur and Tree 1 becomes Tree 2, some of the DB assignments of Tree 1 are maintained, other DB assignments have to be canceled, and new DB assignments are made. In other words, the schedule is modified as opposed to being totally recomputed. In this small network, modifying the schedule is as fast as obtaining a new schedule, but in large networks modifying the schedule is faster.

After Tree 1 becomes Tree 2, the three DBs marked with an asterisk in Figure 3.1c are maintained. However, three DBs become unused: B stops using DB_1 and DB_4 and D stops using DB_3. In addition, four new DBs have to be assigned for the transmission of the packets from D and F: one DB has to be assigned to D, another DB to F, and two DBs to C.

The schedule of Tree 2 is used as follows. In DB_1, D transmits its packet to C. In DB_2, C transmits its packet to A. In DB_3, C transmits D's packet to A. In DB_4, F transmits its packet to C. In DB_5, E transmits its packet to C. In DB_5, E transmits its packet to C. In DB_6, C transmits E's packet to A. Finally, in DB_7, C transmits F's packet to A. Therefore, B is the node that needs to buffer the highest number of packets, and this number is $b = 2$.

3.2 Related Work

Contention-based MAC [20, 21, 22, 23, 24, 21, 25] is simple and flexible: when a node needs to transmit a packet, it simply contends to transmit the packet until it succeeds. There is no need to reserve time slots in advance or keep track of other nodes' transmission intentions. By contrast, the contention-based protocols are more complex and slower because they incur a time slot reservation system. Therefore, if the traffic is short compared to the time slot reservation process, contention-based MAC outperforms schedule-based MAC in terms of energy and delay [80].

However, if traffic is long-running and periodic and the wireless links are static, schedule-based MAC becomes the best approach. The energy and time spent in reserving time slots are outweighed by the more efficient operation during the data transmission phase. During the data transmission phase, the schedule-based protocols are more energy efficient than the contention-based protocols because they suffer no packet collisions, overhearing, or useless back off periods. Furthermore, schedule-based protocols provide greater channel utilization because no time is wasted in packet collisions or back off periods [81, 7].

Based on the discussion above, two things are clear. First, schedule-based MAC is best for long-running periodic traffic in stable networks. Second, contention-based MAC is best for networks where the traffic or the topology are "sufficiently dynamic". However, there remains the problem of quantifying "sufficiently dynamic", i.e., the problem of setting the limit between the networks for which schedule-based MAC is best and the networks for which contention-based MAC is best. This limit can be shifted by the introduction of new protocols, and this is what we do in this chapter: we present a new schedule-based protocol that extends the scope of application of schedule-based protocols to more dynamic networks.

Some protocols are hybrids between contention-based MAC and schedule-based

MAC [79, 82, 83, 84]. These hybrid protocols switch from contention-based MAC to schedule-based MAC when the traffic load grows above a certain threshold. In this way, they seek to combine the benefits of the two channel access methods. However, they are not not always completely successful at doing so. For example, ZMAC [79], a hybrid protocol, has been shown [85] to be less efficient than FlexiTP, a pure TDMA protocol, for periodic traffic.

Although there are many TDMA protocols for wireless sensor networks [39, 31, 30], few of them [27, 37, 36] satisfy the precedence and low-buffering properties. Among these protocols, only FlexiTP [27] is distributed. However, FlexiTP's initial scheduling phase is slow because it uses a token-passing mechanism incapable of assigning slots to several nodes simultaneously. For example, in a network with 400 nodes it requires three hours to execute and consumes 0.6 % of the sensors' batteries [27].

FlexiTP's authors claim that the speed of the initial scheduling phase is unimportant because it is executed only once [27]. We disagree that this speed is unimportant for two reasons. First, the initial phase is executed multiple times in applications where different nodes generate data at different times. Second, a fast initial scheduling phase allows to test network quickly and thus reduces the deployment cost.

The existing TDMA protocols use the k-hop interference model, which can operate for any positive integer k. Section 3.4 shows that different values of k obtain different tradeoffs, but they never obtain low failure probability p_f and a high concurrency c simultaneously. This problem arises from the irregularity of wireless links [86].

TRAMA [30] is a TDMA protocol that divides time in random access slots and scheduled access slots. A node uses the random access slots to advertise its existence to its 2-hop neighbors and to discover its own 2-hops neighbors. The schedule-based slots are used to exchange schedule information and to transmit data. The algorithm to decide whether a node is active in a slot is relatively complex. TRAMA

occasionally makes some nodes unnecessarily active; this is not a problem if traffic is random because this cause of inefficiency is rare, but if traffic is periodic some nodes may suffer this burden periodically. In addition, every node spends a significant amount of energy in exchanging schedule information. Furthermore, TRAMA does not build the schedule to obtain low delay or low buffering. In [31], the delay of TRAMA is shown to be very large.

FLAMA [31] is a TDMA protocol in which scheduled slots are used exclusively for data, whereas in TRAMA they are also used for schedule information. For periodic traffic, FLAMA greatly outperforms TRAMA in terms of delay and energy. However, FLAMA is still inefficient because each node has to either keep track of the priorities of all its two-hop neighbors or perform idle listening at the beginning of several slots to decide whether the highest-priority one-hop flow is an incoming flow. Furthermore, neither TRAMA nor FLAMA verify the precedence or low-buffering properties. FlexiTP [27] and EATP are designed to satisfy these properties, with the difference that FlexiTP restricts the number of buffered packets to $B = 1$, whereas EATP provides greater flexibility by making B a user parameter.

FlexiTP does not update all the schedule information that it should. When a node stops using a DB, FlexiTP removes this slot from the node's neighbors' reception schedules. However, FlexiTP fails to remove the slot from the CSL, which is the list of slots used within 2 hops. Therefore, the sensor nodes claim slots with indices larger than necessary, and the schedule length M increases gradually. TRAMA avoids this problem by periodically updating the schedule information, but this consumes energy.

To our knowledge, every existing distributed TDMA-scheduling protocol for sensor networks requires that each node keeps track of its neighbors' schedules. These protocols force the sensor nodes to spend energy listening for *schedule updates*, which

are packets containing schedule information. EATP is more energy efficient than other protocols because it does not require listening for schedule updates.

In FLAMA and FlexiTP, when a node obtains an infeasible transmission DB to communicate with its parent node, it incorrectly assumes that its parent is unreachable and requests its neighbors' identities to find a new parent. Since its old parent is actually reachable, the node may choose the same parent again, especially in sparse networks. If it selects the same parent and its schedule information has not changed, the node selects the same infeasible DB as it had before. Therefore, the node may never obtain a feasible slot because of the weakness of the protocol.

To summarize, the existing protocols suffer a high failure probability because schedule information may be lost or because the interference model fails. Furthermore, they spend energy in listening for schedule updates. In order to solve these two problems, we propose EATP.

3.3 EATP

EATP is a distributed TDMA scheduling protocol for periodic data gathering in wireless sensor networks that is designed to function with the limited computational speed, energy and memory of the sensor nodes. EATP is the first TDMA protocol for periodic data gathering in sensor networks that does not require the exchange of schedule updates.

EATP consists of the phases described in Section 3.1. If obtaining an initial schedule quickly is unnecessary, the initial scheduling phase can be skipped because the data transmission phase provides its own scheduling mechanism. The internal structure of these phases is shown in Figure 3.2. We describe the initial scheduling phase and the data transmission phase in the next two subsections.

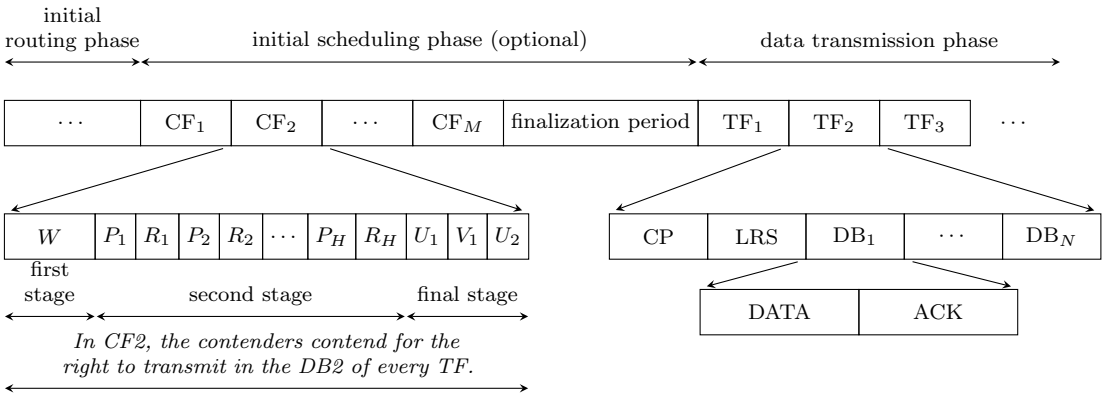


Figure 3.2: Time diagram of EATP. After the initial routing phase, time is divided in slots.

3.3.1 Initial Scheduling Phase

Figure 3.2 shows that the initial scheduling phase consists of M contention frames (*CFs*) and a *finalization period*. The number M is equal to the schedule length, and thus it is denoted by the same symbol. The CFs are labeled $\{\mathrm{CF}_1, \mathrm{CF}_2, \ldots, \mathrm{CF}_M\}$.

In CF_i, a distributed contention process decides which nodes are assigned DB_i. Every node that needs to obtain a DB contends in every DB until it has obtained all the DBs it needs. The nodes that contend during a DB are referred to as *contenders*. The contention process consists of three stages, and only the contenders that succeed in the three stages are assigned DB_i. The successful contenders notify their victory to their respective parents, but not to other nodes because they do not need to know.

In order to reduce the length of the schedule, it is desirable that as many contenders as possible succeed in obtaining a DB in every CF. However, all the nodes that obtain the same DB must tolerate each other to make the DB assignment feasible. These goals are achieved through the following three-stage contention process.

3.3.1.1 The First Stage of the CF

The first stage of the CF occurs within the W slot. Every contender backs off for a random period within the W. The back-off period should be especially short for nodes with a large b or close to the data sink. Such nodes are given priority to obtain a DB because unless $b < B$ they cannot assign DBs to their children.

Every contender performs clear channel assessment (CCA) the end of its back-off period. If the channel is clear, the contender has succeeded in this stage, otherwise it has failed. If the contender has succeeded, it transmits dummy data until the end of the W slot. If it has failed, it withdraws from the contention until the next CF and remains active during the rest of CF in case any of its children are contending.

Every non-contender senses the channel at the end of the W slot. If the channel is idle, the non-contender sleeps until the next CF. Otherwise, the non-contender listens during the rest of the CF in case any of its children are seeking to obtain a DB.

At the end of this stage, it is unlikely that two contenders within transmission range of each other remain in the contention. This is beneficial because such contenders are unlikely to tolerate each other. However, the remaining contenders may not tolerate each other because the interference level at their parent nodes is different that at themselves. This problem is referred as the *hidden terminal problem*, and is addressed in the second and third stages of the CF.

The first stage of the CF can be suppressed without any disruption to the properties of the schedule. The second and third stages do not need to be preceded by the first stage because they alone can decide a group of contenders that tolerate each other. However, if at the start of the second phase the number of contenders is very high, the number H of slot pairs must be set to a very large number. This causes very long CFs, which in turn makes the initial scheduling phase of EATP slow and energy

consuming. We performed some simulations that showed that by introducing the first stage of the CF we can greatly reduce the duration of the second stage of the CF. These simulation showed that the reduction in the duration of the second stage is much greater than the duration of the first phase. Therefore, introducing the first phase greatly increases the energy efficiency of EATP.

3.3.1.2 The Second Stage of the CF

The second stage of the CF consists of the slots $\{P_1, R_1, P_2, R_2, \ldots, P_H, R_H\}$, where typically $H = 7$ as will be discussed in Section 3.4.2.1. Every remaining contender X chooses a random integer h between 1 and H. In slot P_h, X transmits a packet to its parent node Y indicating its identity. If Y receives this packet, it replies in slot R_h with a packet indicating its buffer occupation b and, if $b < B$, it transmits dummy packets in the slots $R_{h+1}, R_{h+1+2}, \ldots, R_H$ in order to provoke the failure of the contenders that X is unlikely to tolerate.

Depending on whether X receives Y's packet in slot R_h and its contents, three scenarios are possible. First, if X receives no reply in slot P_h, X has failed in the second stage and contends again in the next CF. Second, if X receives a packet from Y indicating $b = B$ in slot P_h, node X has also failed in the second stage and refrains from becoming a contender in the next few (in our simulations $4B$) CFs to give Y the time to reduce its b. Third, if X receives a packet from Y indicating $b < B$, X has succeeded in the second stage and transmits dummy packets in the slots $P_{h+1}, P_{h+2}, \ldots, P_H$ in order to provoke the failure of contenders that it may be unable to tolerate.

This phase is designed so that different contenders choose different values of h. The contenders that choose a small h are very likely to succeed, and those with a big h only succeed if they tolerate the nodes that succeeded earlier. However, some

contenders that succeed with a small h may not tolerate the contenders that succeed with a large h. These contenders should not obtain the current DB because it would not be feasible for them. The third stage of the contention eliminates them from the contention.

3.3.1.3 The Third Stage of the CF

The third stage of CF_i consists of the slots U_1, V_1 and U_2, as shown in Figure 3.2. In slot U_1, every remaining contender X transmits a packet indicating its identity to its parent node Y. If Y receives this packet, it replies with an ACK packet in slot V_1. If X receives this ACK, X has succeeded in the last stage of the contention, otherwise it has failed. If X has succeeded, it reports its victory to Y in slot U_2 and adds DB_i to its transmission schedule.

With the above algorithm, node Y is likely to receive the packet from X in slot U_2 because there is at most as much interference as in slot U_1 and in slot U_1 the reception was successful. The successful reception in slot U_2 is desirable because Y must add the corresponding DB to its reception schedule. In noisy environments, Y may not receive the packet in slot U_2. In this case, Y does not add the current DB to its reception schedule and X does not receive acknowledgments during the data transmission phase. This is not a problem because node X can obtain new DBs during the data transmission phase.

The third stage is very likely to ensure that all the DB allocations are feasible. This is because it was proved in the U_1 slot that X's parent receives X's packets correctly in the presence of the interference from the other nodes that are assigned the same transmission slot as X, and because it was proved in the slot V_1 that X can receive the acknowledgments from its parent node in the presence of the interference from the parents of all the nodes that are assigned the same transmission slot as X.

3.3.1.4 Finalization Period

The purpose of this period is to propagate a packet called *finalization packet* from the data sink to every sensor node in the network. The finalization packet contains information that all the nodes in the network should have, such as the start time of the first Transmission Frame (TF), the period T_f between TFs and the number N of DBs per TF. The number N should be at least as big as the number M of CFs elapsed until the data sink created the finalization packet, but a larger N is preferable to enhance the schedule's extensibility. The data sink initiates the finalization period by transmitting the finalization packet to its children.

Our performance evaluation in Section 3.4.2 does not simulate node failures and uses static channels without external interference. Under this conditions, every node obtains a time slot eventually. Our simulations also assume that the data sink knows the routing tree, which means that, when the data sink has assigned as many DBs as data sources are present in the routing tree, the data sink knows that all the nodes have obtained all the DBs they need and thus the finalization phase should begin. Therefore, in our performance evaluation, the data sink does not need any timeout to begin the finalization period.

If there are node failures, the wireless channel is dynamic, or there is external interference, the data sink sink needs a timeout indicating the maximum amount of time to wait before starting the finalization period. When all the nodes have obtained all their DBs or the timeout period elapses, the data sink initiates the finalization period.

We have not developed an algorithm to decide the timeout period because this timeout is unnecessary in our simulations. However, we now discuss the choice of the timeout for more realistic settings. The timeout should increase with the number of nodes in the routing tree because this number increases the number of DBs that have

to be assigned. The timeout should also increase with the node density because if the nodes are close to each other fewer contenders can share a DB. The timeout should also increase as the node failure probability decreases because if failures are rare a full schedule is likely to be found eventually. The timeout should also increase if the data sink has high tolerate to scheduling delay or if the number of nodes providing information about the event is critical. We suggest the use of simulations to choose the timeout as a function of all the variables above.

In our simulations, we implement the finalization period by dividing it into M time slots using the inverse of the schedule obtained in the scheduling phase. We illustrate the concept of inverse of a schedule by describing the inverse of the schedule in Figure 1.1c. Since in the original schedule node E transmits in the last time slot, in the inverse schedule, which is used in the finalization period, node E receives the finalization packet in the first time slot. The rest of the inverse schedule is as follows. In the second time slot, node E transmits the finalization packet to node B. In the third time slot, node B transmits the finalization packet to node C. In the fourth time slot, node B transmits the finalization packet to node A. All these transmissions succeed in our simulations because the original schedule is collision free. .

For more realistic settings, we suggest to implement the finalization period as follows. Rather than dividing the finalization period into time slots, we use CSMA. Every node transmits the finalization to its children, and it retransmits the packet until it receives an acknowledgment from all its children. The propagation of the finalization packet occurs from parent to child along the routing tree. In this way, all the nodes receive the finalization packet.

Note that if the sensor nodes do not know the timeout that the data sink uses to trigger the finalization phase, it is possible that they continue contending for CFs during the finalization period, and thus that they obtain DBs with indices greater

than N, which are invalid DBs. In order to handle this problem, the sensor nodes keep their transceiver active continuously from the start of the initial scheduling phase. If at any time they receive the finalization packet, they take it as a sign that the finalization phase has begun. Then, the nodes give up any slots that they have obtained that are invalid according to the value of N indicated in the finalization packet. The nodes can attempt to gain additional DBs during the data transmission phase.

3.3.2 The Data Transmission Phase

Figure 3.2 shows that the data transmission phase consists of TDMA Frames (TFs). Each TF consists of a CSMA Period (CP), a Listening Request Slot (LRS) and M Data Blocks (DBs). Every sensor node transmits its packets in its transmission DBs, and keeps its receiver active during the CP, the LRS, its reception DBs. Each DB consists of a DATA slot and an ACK slot. The packets transmitted in the ACK slot include synchronization information. The nodes that do not receive ACK packets have to request and receive synchronization information during the CP.

The CP is also used by *orphans*, which are nodes that need to select a new parent, which is necessary after certain network changes. For example, in Figure 3.1, after the network changes, nodes D and E become orphans. Every orphan contends during the CP to transmit an information request packet to its neighbors. The neighbors contend to reply during the CP with a packet containing their identities and their hop distance to the data sink. Using this information, the orphan selects the node with the shortest hop distance as its parent.

Network changes may cause a node to stop receiving packets from a child in a certain DB. This may happen because the child died, has no more data to transmit, or can no longer communicate during that DB. In any case, the parent simply

removes the DB from its reception list.

3.3.2.1 DB-Acquisition Overview

Following network changes, some nodes need to obtain new DBs. Each of these nodes is referred to as *contender*. A simplification of the DB acquisition process is as follows.

The LRS is only long enough for the contenders to contend briefly and for one successful contender to transmit a packet called a listening request packet (LRP). The successful contender indicates in the LRP the DB that it tries to obtain, which is referred to as its *target DB*. Next the successful contender transmits a packet in its target DB called *testing packet*. If it receives an ACK in response, the contender has obtained its target DB. Otherwise it selects another target DB and repeats the process until successful.

Existing DB allocations are unprotected and may become unusable due to the interference from new DB allocations. In other words, *expulsions* may occur. A node is said to have suffered an expulsion if it has to find a new DB because a node that recently started using that DB is causing excessive interference. We study the influence of expulsions in Section 3.4.3.3.

Every contender chooses its target DB randomly among the DBs in which it senses the channel idle. Due to the hidden terminal problem, its target DB may be infeasible, in which case it simply chooses another DB and tries again in the next TDMA frame. This algorithm works best if the traffic changes slowly, because then the interference in the DBs of two consecutive TFs is very likely to be similar. The rationale of this DB assignment process is to spare the sensor nodes from the burden of keeping track of their neighbors' schedules. A more detailed description of the algorithms to assign and obtain DBs is as follows.

3.3.2.2 Parent's Algorithm

Here, *parent* refers to a node Y that receives a LRP from a child X during a LRS. Node X transmits the LRP in order to report its target DB to Y. Node Y reacts to the LRP by listening during the DATA slot of X's target DB. If Y does not receive any packet during this slot, it does not take any other action. On the other hand, if Y receives a packet from X, it replies with an acknowledgment. This acknowledgment indicates the number b of packets stored in Y's buffer. If this number b is smaller than B, Y adds the current DB to its reception schedule.

Whether X adds the current DB to its transmission schedule depends on its successful reception of Y's acknowledgment. Therefore, X might not add the current DB to its transmission schedule, but this is not a problem because Y does not receive packets in this DB and removes it from its transmission schedule.

3.3.2.3 Contender's Algorithm

The algorithm executed by every contender X in order to obtain a DB is as follows.

Contender X selects a random integer w between 1 and N_w. At the end of each TDMA frame it decrements this counter. When w reaches 1, the node selects its *target DB*.

The first step that X takes to select its target DB, DB_k, is to set a variable N_r with a random integer between 0 and N_s, where N_s is 1 in our simulations. Then, X sets k to the maximum of two integers. The first integer is 0 if the purpose of the desired DB is to transmit an own packet, and $i + 1$ if the purpose of the DB is to relay another node's packet that is received in DB_i. The second index is the index of the last DB where it tried to obtain a DB, unless that DB was transmitted a long time ago.

Node continues X selecting its target DB by checking whether the channel is clear

in the two time slots of DB_k. Three cases may arise. First, if the channel is clear and N_s is zero, the contender has obtained its target DB: DB_k. Second if the channel is clear and N_s is positive, the contender decrements N_s, increments k, and repeats the channel check operation in the new k. Third, if the channel is busy, the contender increments k and repeats the channel check operation in the new k.

Therefore, within one TF, namely the TF in which X's w is 1, X selects its target DB, DB_k. At the end of the TF, as at the end of every other TF, X decrements w, which reaches zero. During the LRS, X contends to transmit a LRP indicating k. If X fails to gain access to the medium, it resets w to a random integer between 1 and N_w and repeats the whole process.

After X manages to transmit a LRP, it transmits a testing packet to its parent node Y during the data slot of its target DB. Then, X listens for an acknowledgment during the ACK slot of its target DB. Next, three possible situations may arise.

First, if X does not receive an acknowledgment during its target DB , X has failed to obtain its target DB. Then, X resets w to an integer between 1 and N_w and repeats the whole process. Second, if X receives an acknowledgment that indicates that Y's value of b is smaller than B, X has obtained its target DB, so it adds its target DB to its transmission schedule. Third, if X receives an acknowledgment that indicates that Y's value of b is B, X resets w to an integer between BN_w and $(B+1)N_w$ in order to give time Y to empty its buffer.

3.4 Performance Evaluation

In order to evaluate the performance of EATP, we use a custom simulator written in the Python programming language. The simulator can be downloaded from [78] with all its parameters. It uses FlexiTP [27] as a benchmark for EATP because it is the only existing distributed and adaptive TDMA scheduling protocol for periodic

data gathering in randomly deployed networks that ensures the precedence and low-buffering properties. We implement two versions of FlexiTP: FlexiTP$_2$ and FlexiTP$_3$. FlexiTP$_2$ is the original version [27], and FlexiTP$_3$ is a new version that we propose that differs from FlexiTP$_2$ in that it propagates schedule updates across 3 hops instead of across only 2.

Both EATP and FlexiTP consist of an initial scheduling phase followed by a data transmission phase during which the initial schedule can be modified. The major difference between the two protocols is that in EATP the sensor nodes do not require knowledge of their neighbors' schedules. Another difference is that during the initial scheduling phase EATP only assigns the same DB to nodes that have been proved to tolerate each other's interference, which increases the probability of obtaining a feasible schedule.

3.4.1 Simulation Scenario

The FlexiTP paper [27] does not detail the MAC parameters to transmit schedule updates. In order to perform a fair comparison between FlexiTP and EATP, we simulate the two protocols for slotted CSMA. The simulation parameters common to all simulations in this chapter are shown in Table 3.1.

Time is divided in 24 ms time slots. At the beginning of each time slot, every node wishing to transmit backs off for a short random period between 0 and 200 μs. At the end of this back off period, it performs a clear channel assessment and only transmits a packet if the channel is clear. The process of checking the channel and starting a packet transmission requires 50 μ. We neglect the propagation time of the radio waves. With these parameters, there is a small chance that two neighbors within transmission range of each other transmit simultaneously. Similar parameters are used in [27].

parameter	value
◦ duration of the W slot in EATP	10 ms
◦ duration of the slots $\{P_1, R_1, \ldots, P_H, R_H, U_1, V_1, U_2\}$ in EATP	24 ms
◦ duration of the slots to claim DBs and propagate schedule information in FlexiTP	24 ms
◦ range of the random back off period during which a node needs to sense the channel continuously clear before transmitting a packet	0—200 μs
◦ time required to perform a clear channel assessment and start a packet transmission	50, μs
◦ power consumption in sleep mode	60 μW
◦ power consumption in receive mode	63 mW
◦ power consumption in transmit mode	63 mW
◦ power consumption in idle power	63 mW
◦ attenuation exponent of the wireless channel	3.5
◦ standard deviation of the log-normal shadow fading	8 dB
◦ path loss at 100 m	80 dB
◦ noise figure of the receiver	4.8 dB
◦ minimal at the receiver SINR for a transmission to be successful	13 dB
◦ transmission range t when there is no fading or interference	74 m

Table 3.1: Simulation parameters in EATP.

The transmit, receive and idle power are all equal to 63 mW, and the sleep power is 60 μW. Such values are typical over short distances with current hardware [7]. The attenuation exponent of the channel is 3.5 and the path loss at 100 m is 80 dB. The channel suffers log-normal shadow fading with a standard deviation of 8 dB. Since different values of shadow-fading attenuation are applied to every link, there is no such a thing as a transmission range. The noise figure of the receiver is 4.8 dB. Some of these parameters are extracted from [87].

The success of a packet transmission is determined based on the joint interference from all concurrent transmitters. A transmission is treated as successful if the SINR at the receiver exceeds 13 dB. We define t as the maximum distance across which two nodes can communicate if there is no fading and interference; the parameters

parameter	value
◦ number H of slot pairs per CF in EATP	7
◦ node density ρ	6–15
◦ maximum fraction of disconnected nodes in the network	10 %
◦ normalized network size $\bar{x}$	3–6
◦ number of nodes in the network	69–172
◦ number of simulation runs for each node density and network size	100

Table 3.2: Simulation parameters in the initial scheduling phase.

presented above yield $t = 48\,\mathrm{m}$.

The monitored area is a square of side $\bar{x}t$, where $\bar{x}$ is referred to as the *normalized network side*. The data sink is deployed in the middle of one the square sides, and L nodes are deployed randomly within the square. The node density ρ is defined as $L(\pi t^2)/(\bar{x}t)^2$. It is the number of neighbors within transmission range per node when there is no fading and interference. All nodes are data sources and are linked to the data sink through the shortest path tree. We discard the deployments in which more than 10 % of the nodes are unable to reach the data sink through one or multiple hops, which is rare for $\rho \geq 7$.

3.4.2 Performance of the Initial Scheduling Phase

In this section, the maximum normalized network side $\bar{x}$ is 6 and the maximum node density ρ is 15, which yield a maximum of 172 nodes. We do not choose values of ρ smaller than 6 because then the network would be disconnected with high probability. We do not choose values of ρ larger than 15 in order to keep the simulation time low. Furthermore, a high node density is energy and bandwidth inefficient and can typically be improved by reducing the transmit power. Every simulation setting, which is given by a normalized network size $\bar{x}$ and a node density ρ, is simulated 100 time. The parameters used in this section are shown in Table 3.1 and Table 3.2.

3.4.2.1 Choice of H in EATP

An important parameter in EATP is H, which is the number of slot pairs in the second stage of every CF, as shown in Figure 3.2. The choice of H has to consider the following factors. Increasing H makes the CFs longer, but fewer CFs are necessary, and thus the impact of H on the execution speed is uncertain. Increasing H reduces the number of CFs because it increases the probability of contenders transmitting at different times during the second stage of a CF. The following example illustrates why an increase in this probability is beneficial.

Suppose that only two contenders, X and Z, reach the second stage of a CF, and that X and Z do not tolerate each other. If X and Z choose the same h, they transmit simultaneously, suffer each other's interference, do not to receive a reply, and fail in the CF. By contrast, if X chooses a smaller h than Z does, X suffers no interference in its first transmission of the second phase and reaches the third stage. When Z transmits its packet, it receives no reply due to the interference from X or X's parent. When X transmits in the third stage, it suffers no interference and becomes a winner. Therefore, fewer CFs are needed than if X and Z choose different h than if they choose the same h.

After H is sufficiently large, all the contenders are likely to choose different h. Therefore, further increasing H barely reduces M, as shown in Figure 6.2. Increasing H from $H = 5$ to $H = 15$ only reduces the schedule length M by 7 % independently of the node density ρ. Since $H = 7$ obtains a fast execution speed (the CFs are short, and few CFs are needed) and a high concurrency (H is small), we use it in the rest of the simulations.

In order to justify why the same H is sufficient for all simulated densities, consider Figure 3.4. For simplicity, we assume that there is a uniform transmission range r_t, which is the maximum distance within which two nodes can communicate with each

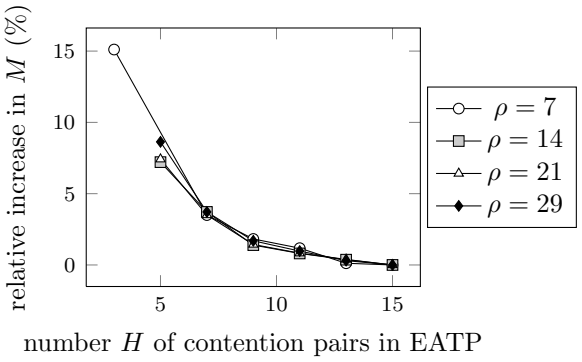


Figure 3.3: Schedule length M for different values of H relative to M for $H = 15$.

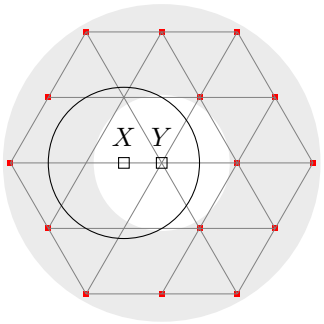


Figure 3.4: Simplified model to justify that the number of hidden terminals does not increase indefinitely with the node density ρ.

other or detect each other's transmissions. We also assume that there is a uniform interference range $r_i = 2r_t$, defined as the maximum distance within which a node can interfere with other nodes.

In Figure 3.4, node X is contending for a DB to communicate with its parent Y. Centered at Y, there is a ring with internal radius r_t and external radius $r_i = 2r_t$. Therefore, the nodes within the white area around Y can communicate with Y, and the nodes within the shaded area can interfere with Y but not communicate with it.

Suppose that X reaches the second stage of a CF. The circle centered at X has radius r_t, and thus the other contenders that reach the second phase are outside this circle. Among all these contenders, we refer to those that can interfere with Y as *incompatible contenders* of X.

The incompatible contenders of X are inside the circle of radius $2r_t$ centered at Y and outside the circle of radius r_t centered at X. The maximum number of incompatible contenders that fit into this area is limited by the minimum distance between two contenders, which is r_t after the first stage of the CF. Therefore, the number of contenders does not grow with ρ, and thus H does not need to increase with ρ either.

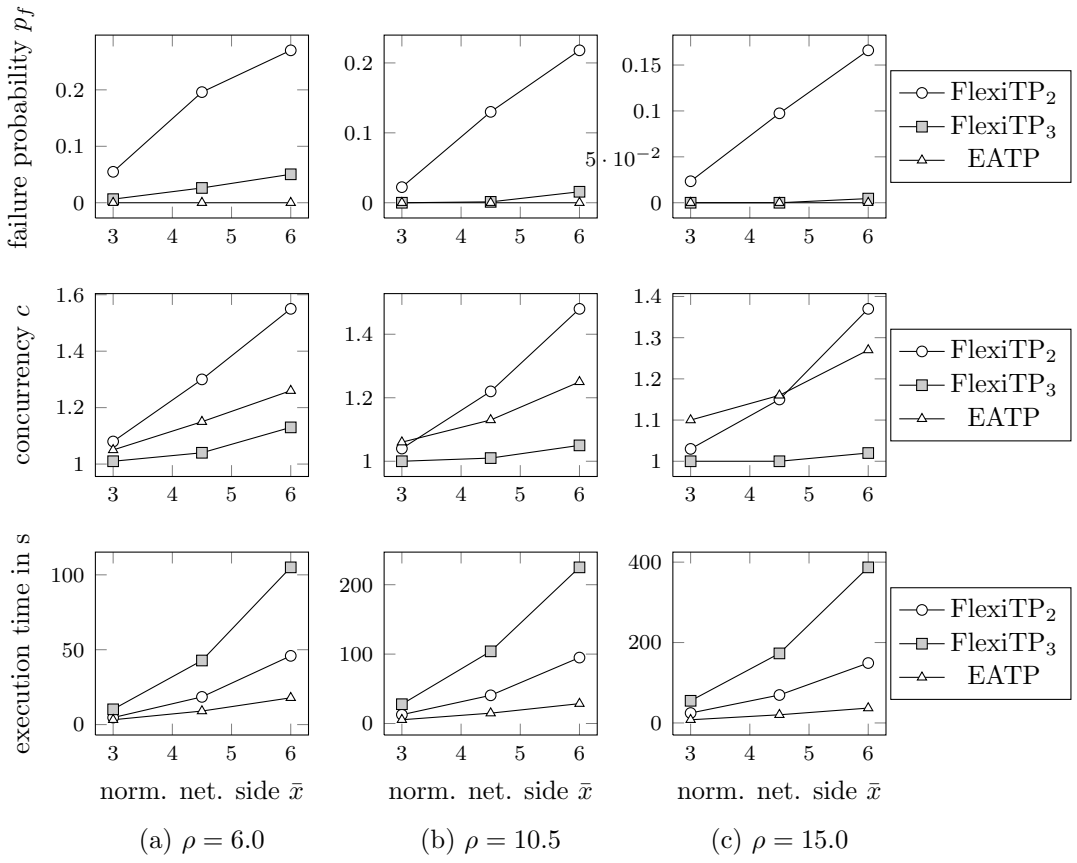
Figure 3.4 shows a distribution of incompatible contenders of X that contains almost as many contenders as possible. The incompatible contenders are represented by small squares and arranged in an hexagonal grid. In this grid, every node has six neighbors at a distance of r_t.

3.4.2.2 Performance Results

Figure 3.5 presents the results of the initial scheduling phase in matrix form. It consists of three rows, each of which presents one performance metric. Each column refers to a different node density ρ. In each graph, the x-axis represents the normalized network side $\bar{x}$.

The first row of Figure 3.5 shows that EATP greatly outperforms FlexiTP$_k$ in terms of the failure probability p_f. EATP achieves $p_f = 0$ because a set of nodes are only assigned the same DB if they have been proved empirically to tolerate each other's interference. By contrast, FlexiTP$_2$ and FlexiTP$_3$ suffer a positive p_f because they are subject to the problem of radio irregularity [86].

FlexiTP$_2$ has a failure probability p_f as high as 0.27 for large and sparse networks. FlexiTP$_3$ performs much better because it imposes a larger hop distance between concurrent transmitters. As the node density ρ increases, the failure probability of FlexiTP$_k$ decreases because the hop distance becomes more strongly correlated with the physical distance and more conditions are examined before assigning the same



(a) $\rho = 6.0$ (b) $\rho = 10.5$ (c) $\rho = 15.0$

Figure 3.5: Scalability of the initial scheduling phase with the node density ρ and the normalized network side.

DB to two nodes. As the normalized network side $\bar{x}$ increases, the failure probability p_f increases because the nodes far from the border of the monitored area have more neighbors and interferers than the nodes in the center of the monitored area.

The second row of Figure 3.5 shows that the concurrency c of the three protocols increases with the normalized network side $\bar{x}$. This is because in small networks the sensor nodes are too close to each other to share the same DBs, and thus the concurrency c is 1. As the node density ρ increases, the concurrency of FlexiTP$_k$ decreases because more conditions are used to decide whether two nodes interfere with each other. EATP enjoys much higher concurrency than FlexiTP$_3$. The relative performance of EATP and FlexiTP$_2$ depends on ρ and $\bar{x}$, but FlexiTP$_2$ is not practical due to excessive p_f.

The third row of Figure 3.5 shows that EATP is much faster than FlexiTP$_k$. FlexiTP$_k$ uses a token-passing mechanism that only allows one node to gain a DB at a time, and thus its execution time is roughly proportional to the number of nodes. By contrast, EATP can assign several DBs at a time if the network is sufficiently large. Therefore, the speed advantage of EATP increases with the two variables that control the number of nodes, which are the network side $\bar{x}$ and the node density ρ.

3.4.3 Performance of the Data Transmission Phase

In this section, 1000 simulation runs are averaged. The normalized network side $\bar{x}$ is 3 and the maximum node density is $\rho = 22$, which yield a maximum of 63 nodes. We use a smaller number of simulated nodes in this section than in Section 3.4.2 because our simulator [78] is slower in the data transmission phase than in the initial scheduling phase. The maximum number of packets to be buffered per node is $B = 5$. We choose $B > 1$ in order to test the functioning of the protocol, and we do not choose a value of B larger than 5 in order to keep the upstream latency low. The

parameter	value
◦ node density ρ	7–22
◦ maximum fraction of disconnected nodes in the network	10 %
◦ normalized network size $\bar{x}$	3
◦ number of nodes in the network	20–64
◦ maximum number B of packets that a node can buffer	5
◦ number of simulation runs for each node density and network size	1000

Table 3.3: Simulation parameters in the data transmission phase.

simulation parameters are shown in Table 3.1 and Table 3.3.

The goal of the data transmission phase is to adapt to infrequent network changes in an energy-efficient way. In our simulations, each network is modified by four *change sets*, each of which consists in removing two randomly selected nodes and deploying two new nodes at random locations. Every change set is instantaneous, and different change sets occur at different times. A change set is executed only after the network has fully adjusted to the previous change set.

The number of DBs that have to be assigned after a change set depends on the network size. For example, suppose that a node X with q children is h hops away from the data sink. Also suppose that X's children are data sources and childless, and that they have other neighbors h hops away from the data sink besides X. Then, if X is removed from the network, $q(h+1)$ new DBs need to be allocated.

The energy consumption of the sensor nodes depends on the speed of change of the network. In order to quantify it, we define the *DB demand period* T_a as the quotient between the number of DBs required due to network changes over a certain period of time, and the number of TDMA frames that elapsed over this period. The DB demand period T_a is inversely related to the speed of change of the network.

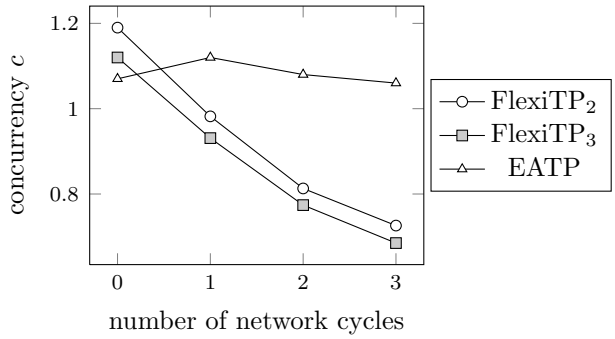


Figure 3.6: Concurrency c as a function of the number of change sets.

3.4.3.1 Concurrency

Figure 3.6 shows that, as the number of network cycles increases, the concurrency of FlexiTP$_k$ is severely degraded, whereas that of EATP is preserved. This is because the nodes in FlexiTP may have outdated information about their neighbors' schedules, whereas the nodes in EATP use no such information.

To explain FlexiTP$_k$'s degradation, consider a node X that stops using DB$_i$ to communicate with its parent Y. Since node Y stops receiving packets in DB$_i$, it removes DB$_i$ from its reception schedule. However, the rest of the neighbors of X are not notified that DB$_i$ has become available. In particular, those neighbors do not remove DB$_i$ from their CSL, which is the list of DBs being used by their neighbors. Since FlexiTP$_k$ fails to update the CSLs, the indices of the DBs claimed by the sensor nodes are unnecessarily large, which reduces the concurrency c over time.

In order to preserve FlexiTP$_k$'s concurrency over time, we propose to modify FlexiTP$_k$ as follows. Every sensor node propagates its schedule periodically to its neighbors and removes from its CSL the DBs not confirmed to be in use. We do not implement this modification of FlexiTP$_k$ because it increases the energy consumption.

3.4.3.2 Eventual Assignment of a DB

The probability that a node obtains an infeasible DB in the transmission phase is similar to the probability that it obtains such a DB in the initial scheduling phase, which is shown in the first row of Figure 3.5. FlexiTP$_k$, TRAMA [30] and other TDMA protocols do not respond appropriately to infeasible slot allocations because they do not make the sensor nodes keep track of the DBs that they have tried to obtain. As a result, the nodes obtain the same DBs repeatedly, thereby increasing the energy consumption and the delay. When computing the energy consumption and the delay, the simulator intentionally ignores the FlexiTP nodes that are assigned infeasible slots. Therefore, the energy advantage of EATP that we will show in Section 3.4.3.4 still holds if FlexiTP is modified to avoid infeasible allocations, which we propose to do as follows.

Suppose that a node has been assigned a DB but repeatedly fails to receive ACKs from its parent in that DB. The node removes the DB from its transmission list and waits for a random interval. Then, it randomly selects a new DB among the DBs that it has not tried before, and claims that DB. The node only changes its parent node if it cannot communicate with it during the CP. This algorithm differs from the original FlexiTP algorithm in that it keeps track of past attempts and in that it introduces randomness. Randomness prevents the deadlock that occurs when two neighbors simultaneously and repeatedly claim the same slots.

3.4.3.3 Scheduling Delay Metrics

Scheduling delay refers to the slowness of the protocol in assigning DBs. We define two metrics to characterize it: the acquisition delay l_a and the postponement ratio r_p. Since the scheduling delay and the energy consumption have to be considered simultaneously, we discuss them together in Section 3.4.3.4.

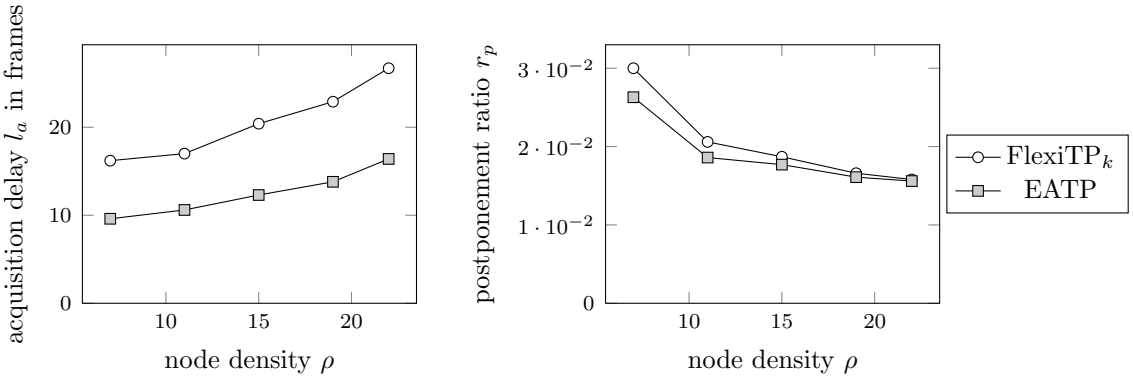
The *acquisition delay* l_a is defined as the number of TDMA frames elapsed from the time when a node needs a DB until the time when it obtains the DB. There are two reasons why a high acquisition delay may be detrimental. First, it postpones the time when the data sink receives the information, which may reduce its usefulness. Second, it forces the nodes to store more packets in their buffers, which may cause buffer overflows.

The *postponement ratio* r_p is defined as the probability that a node's need to transmit a packet in the current TDMA is unmet for one of two reasons. First, the node transmitted the packet but the packet was not received correctly. Second, the node has not obtained yet a DB to transmit the packet. The postponement ratio is useful because it captures two aspects that the acquisition delay does not: the packet losses suffered in EATP due to collisions with testing packets, and the greater number of expulsions (c.f.suffered by EATP .

3.4.3.4 Energy Consumption

FlexiTP$_k$ and EATP can operate at multiple points in the energy-delay space. In FlexiTP$_k$, the operation point depends on the duration T_u of the FTS, which is the slot used to propagate schedule updates. Shortening T_u reduces idle listening and thus the energy consumption, but increases the scheduling delay because more TDMA frames are needed to propagate schedule updates. Similarly, in EATP, removing the LRS from some of the TFs (see Figure 3.2) and disallowing the transmission of testing packets in those TFs reduces the energy consumption, but increases the adaptation latency.

In this section, we prove that EATP can simultaneously enjoy lower scheduling delay and energy consumption than FlexiTP$_k$. We set T_u to a value so small that FlexiTP$_k$'s acquisition delay l_a is 70 % higher than EATP's. This value l_a is chosen



(a) acquisition delay l_a (b) postponement ratio r_p

Figure 3.7: Scheduling delay metrics in the data transmission phase.

so that FlexiTP$_k$ also has a higher postponement ratio r_p. The higher performance of EATP for the chosen l_a in terms of the two scheduling delay metrics is shown in Figure 3.7. Note that our choice of a small l_a increases FlexiTP$_k$'s scheduling delay, but it reduces its energy consumption. Despite giving FlexiTP$_k$ this advantage, we show in Figure 3.8 that EATP still consumes less energy.

Figure 3.7a shows that the acquisition delay l_a of EATP grows with the node density ρ. This is because the node density ρ increases the number of contenders and the level of interference, thereby intensifying the contention and decreasing the probability that testing packets are received. By contrast, Figure 3.7b shows that the postponement ratio r_p decreases with ρ. This is because, as ρ increases, the number of nodes in the network grows faster than the number of nodes affected by the change sets. Figure 3.7 indicates that EATP performs better in terms of acquisition delay l_a than in terms of postponement ratio r_p. This is because EATP suffers expulsions and packet losses due to testing packets (cf. Section 3.4.3.3).

Figure 3.8 displays the energy consumed in scheduling operations per node and TDMA frame. This energy metric excludes the energy consumed in data transmissions, which is the same for all the protocols. It can be seen that EATP consumes around 20 % less energy than FlexiTP$_2$ and less than 65 % less energy than FlexiTP$_3$.

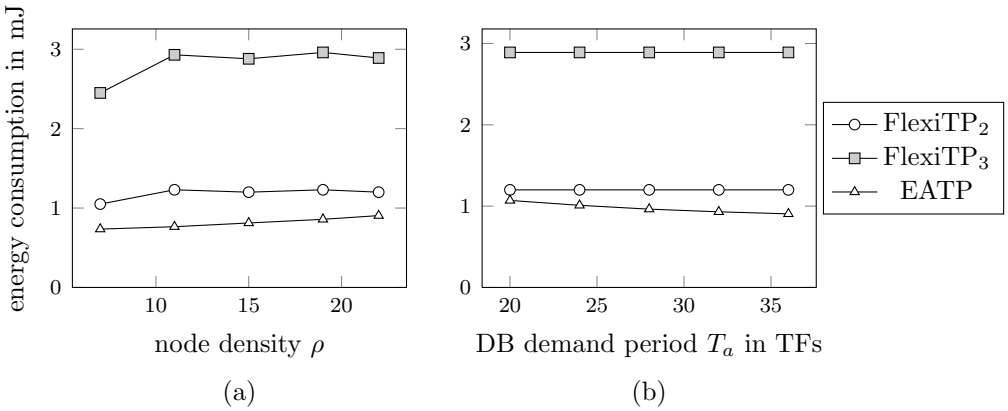


Figure 3.8: Energy consumed per node and TDMA frame.

This is because the main cause of energy consumption in EATP is idle listening in LRS, the main cause of energy consumption in FlexiTP$_k$ is idle listening in the FTS, and the LRS is shorter than the FTS.

Figure 3.8a is obtained for a DB demand period of $T_a = 36$. It reveals an increase of the energy consumption with the node density ρ. In the case of FlexiTP$_k$, this is because T_u needs to increase with ρ in order to be able to propagate schedule updates to an increased number of neighbors. In the case of EATP, this is because a greater ρ increases the interference level and the probability that a testing packet is not received. FlexiTP$_2$ consumes much less energy than FlexiTP$_3$ because it can achieve the scheduling delay of Figure 3.7 with shorter T_u than FlexiTP$_3$ can.

Figure 3.8b is obtained for a node density of $\rho = 24$. It reveals that the energy consumption of FlexiTP$_k$ is independent of the DB demand period T_a. This is because obtaining DBs does not increase the time that the nodes keep their transceivers active. Rather, it only changes the nodes' trasnceiver status, from idle to either transmit or receive. Since we have assumed that these three modes consume equal power, obtaining DBs does not increase the energy consumption.

Figure 3.8b shows that the energy consumption of EATP decreases as T_a increases. This is because a higher T_a reduces the speed of change of the network and the

number of DBs that have to be obtained. Obtaining DBs consumes energy for the following reasons. First, the contenders have to listen in multiple DBs in order to select their target DBs. Second, testing packets need to be transmitted and listened to. Third, packets lost due to collisions with testing packets need to be retransmitted.

3.5 Conclusions

We have proposed a TDMA protocol for uncompressed data gathering called EATP. EATP has a much lower failure probability p_f than other protocols because it only assigns the same DB to a set of nodes if they have been proved empirically to tolerate each other's interference. By contrast, the existing protocols rely on unreliable interference models such as neglecting the interference originated more than 2 hops away.

During the initial scheduling phase, EATP is faster than FlexiTP$_k$. EATP's speed advantage grows with the network size because EATP can perform simultaneous DB assignments only if there are nodes sufficiently far from each other. EATP's speed advantage also increases with the node density ρ because it selects the owners of a certain DB during a CF and the size of a CF is independent of ρ.

During the data transmission phase, EATP is not very fast, taking as many as 15 TFs to assign a DB. However, if this delay is tolerable and the network does not change very fast, EATP can save more than 20 % of energy. This result was obtained when giving FlexiTP$_k$ several advantages. First, allowing FlexiTP$_k$ to suffer a longer scheduling delay than EATP. Second, not considering FlexiTP$_k$'s energy consumption in keeping its CSL update, which causes a decrease of the spectral use. Third, ignoring the effect of infeasible DB allocations on FlexiTP$_k$'s energy consumption.

These properties make EATP suitable for event-triggered, periodic data gathering

in slowly-changing wireless sensor networks. Note that the merit of a fast initial scheduling phase does not contradict the assumption of a high tolerance to scheduling delay during the data transmission phase. During the initial scheduling phase, speed is important because the data sink is receiving no information about the event yet. During the data transmission phase, scheduling delay can be tolerated because sufficient information of the event is already being reported.

4 Network Segmentation for Unrepeatable Data Aggregation

This chapter is the first to address the data aggregation case. It considers networks in which the data aggregation functions are *unrepeatable*, which means that each packet is only aggregated once. Unrepeatable data aggregation functions are easier to develop but less efficient than repeatable aggregation functions, which are considered in later chapters. We assume a cluster-based topology because it is suitable for unrepeatable aggregation, and propose an algorithm to divide the network into clusters. The proposed cluster segmentation is such that the cluster size increases with the distance to the data sink.

4.1 Introduction

Consider a large, cluster-based network used to report randomly-located events. The clusters are constant over time and are used for multiple events. Each event has an associated *event area*. Every sensor node within the event area becomes a *data source*, which means that it generates data about the event.

Every sensor node belongs to exactly one cluster, and each cluster contains one cluster head. The data sources associated with the same event may belong to different clusters. Every data source transmits its data to its cluster head. Every sensor node

has a fixed transmission range, and it may have to communicate with its cluster head across multiple hops.

The packet transmissions from every data source to its cluster head constitute the *internal traffic* of the cluster. The internal traffic is not aggregated in any way. When a cluster head receives all the information of the data sources of its cluster, it aggregates and compresses this information together.

After compressing the information from its cluster, the cluster head transmits the result to the data sink. These packet transmissions need to traverse other clusters. To those clusters, the packets from other clusters that they have to relay are referred to as *external traffic*. The external traffic is relayed by the normal sensor nodes, not only by the cluster heads, and it may need to travel multiple hops to reach the data sink.

The problem of dividing the network into clusters of unequal sizes has to consider the following tradeoff related to cluster sizes. On the one hand, reducing the cluster size reduces the internal traffic because the uncompressed packets from the data sources travel across fewer hops to the cluster head, where they are aggregated. On the other hand, reducing the cluster size increases the external traffic because more clusters generate information about each event, and the packets from different clusters are not aggregated.

The optimal cluster size varies with the distance between the cluster and the data sink. In particular, the clusters close to the data sink should be smaller than the clusters far way from the data sink. This is because clusters far away from the data sink need to relay little external traffic and thus have more energy than clusters close to the data sink. The extra energy of the clusters far away from the data sink is best employed in increasing their size, which reduces the external traffic that clusters close to the data sink have to relay.

In this chapter, we formulate and solve the problem of dividing the network into clusters so as to minimize the maximum energy consumption in the network. We only consider the energy consumed in packet transmissions. We neglect the energy of compressing data, idle listening, and packet collisions. We divide the network into concentric, non-overlapping rings. Each ring is divided in sectors, and each ring sector is a cluster. In other words, we use location to decide the clusters. This kind of segmentation assumes large, multi-hop clusters with densely deployed sensor nodes. The clustering problem is to select the number of rings, the size of each ring, and the size of each sector.

4.2 Related Work

Most clustering protocols [51, 52, 53, 54, 55] assume that the cluster heads communicate directly with the data sink. In large networks, this is inefficient [88] or impossible because some nodes may have very poor links to the data sink. SCT [55] segments the network in rings and sectors, like we do. However, the SCT paper focuses on clusters that are small enough for single-hop communication. Furthermore, every ring in SCT has the same thickness, and thus the sensor nodes close to the data sink consume their batteries sooner than the rest of the sensor nodes.

UCS [89] and EEUC [90] are two protocols to obtain clusters of unequal size. Similar to our algorithm, they increase the cluster size with the distance to the data sink. However, they both assume that every sensor communicates directly with its cluster head. This constraint is particularly harmful in large networks. This is demonstrated through our simulation results in Section 4.7, which show that clusters consisting of more than four hops are desirable. Additionally, UCS's clustering algorithm is not scalable because it considers many parameters. For this reason, its performance is evaluated mostly in two-ring networks, with only very brief mention

to a three-ring network. By contrast, in our simulation section we consider networks with up to 35 rings.

EEUC is more scalable than UCS because its clustering mechanism is based on contention between neighbors. However, it assumes that each node can estimate its distance to the data sink by the measuring the strength of the signals from the data sink, which is unrealistic in irregular propagation environments. Furthermore, EEUC assumes that every cluster head transmits its compressed packets across one hop to the next cluster head.

4.3 Assumed Network Topology

The network is large and contains only one data sink. The monitored area is limited by two concentric circles of radii r_a and r_b centered at the data sink, and is divided in N_L non-overlapping rings centered at the data sink, as depicted in Figure 4.1. The rings are numbered from 1 to N_L. The internal radius of Ring i is r_i, and its thickness is $h_i = r_{i+1} - r_i$. The sum of the thicknesses of every ring must be equal, to $r_b - r_a$, which can be expressed as $\sum_{i=1}^{N_L} h_i = r_b - r_a$.

Ring k is divided into

$$q_K = \lceil 2\pi r_k / h_k \rceil \tag{4.1}$$

equal clusters. We approximate these clusters by squares of side h_i, which is reasonable if $r_a \gg h_i$.

Within each cluster, the cluster head is the node lying the closest to the desired position, which is the middle of the closest edge of the cluster lying the closest to the sink. In this chapter we assume that the cluster head lies exactly in its desired position.

Under realistic propagation conditions, our assumed distribution in clusters based

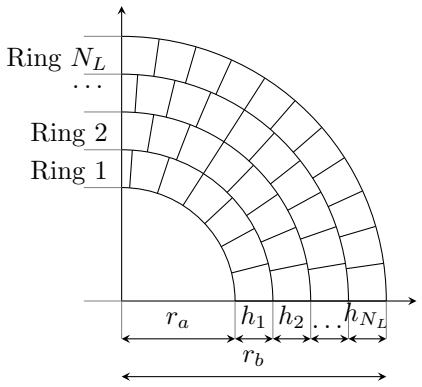


Figure 4.1: Segmentation of the network in layers and clusters. The sink is at the center and the clusters are approximately squares.

on node position is reasonable because typically the minimum transmission range in the network is at least 25 % of the maximum transmission range [81]. The assumption that clusters are square is particularly good if r_a is large, which is the case if the data sink has a very sensitive receiver. The assumption of the cluster head location is good if clusters contain many hops. The assumption about the topology are only realistic in case that the monitoring area is more or less circular. They are unrealistic if the deployment is essentially linear, for example sparse deployment in bridges or tunnels.

The node density is ρ and the number of nodes in the network is $A\rho$, where A is the area of the network and is given by

$$A = \pi \left(r_b^2 - r_a^2 \right) \tag{4.2}$$

Every node has transmission range r_{tx}. The nodes within $r_a + r_{tx}$ from the data sink can reach it directly. This assumption is reasonable because the sink may have a more sensitive receiver or an antenna with a higher gain. We define the *normalized network size* as

$$\beta = \frac{r_b - r_a}{r_{tx}}. \tag{4.3}$$

The event area is a square of side s, two of whose sides are approximately parallel to two radii centered at the sink. The center of the events are randomly located within an area

$$A_f = \pi \left((r_b - s)^2 - r_a^2 \right). \tag{4.4}$$

Specifically, the minimum distance of a corner of the event area to the data sink may vary between r_a to $r_b - s$. The *normalized event size* is defined by

$$\gamma = \frac{s}{r_{tx}} \tag{4.5}$$

Therefore, if an event occurs at a random location, it is detected on average by

$$n_k = \frac{q_k \left(h_k + s \right)^2}{A_f} \tag{4.6}$$

clusters in Ring k, where q_k is given by (4.1).

In this chapter, we use the *hop count approximation*, which we define as follows: if two nodes lie a distance d away from each other, the number of hops between is

$$\zeta \approx \omega d / r_{tx}, \tag{4.7}$$

where ω is a constant that depends on the node density ρ. Constant ω is 1 in extremely dense networks and increases as the node density decreases.

4.4 Cost Analysis

We assume that each data source generates one uncompressed packet per event. Therefore, if a cluster is fully contained within the event area, the number of uncompressed packets that the cluster head aggregates is $h_k^2 \rho$, where k is the cluster

location. Then, the cluster generates one compressed packet that it transmits to the data sink across multiple hops. Obviously, the problem would be very similar if each data source generated a certain number of uncompressed packets and the cluster head generated an unequal number of compressed packets.

We assume that the transmission of every uncompressed packet has equal cost, which is denoted by E_r. Similarly, the transmission of every compressed packet has also equal cost, which is denoted by E_c. The packet size of the compressed packet is independent of the number of uncompressed packets whose information it combines. The parameter that controls the amount of compression is

$$\sigma = \frac{E_r}{E_c}, \tag{4.8}$$

For a node in Ring k, the expected energy consumed in transmitting internal traffic is denoted by E_k^i, and that consumed in transmitting external traffic by E_k^i.

The total energy consumed by a node in Ring k is

$$E_k^t = E_k^i + E_k^i. \tag{4.9}$$

The external to total ratio is defined by

$$\varepsilon = \frac{E_k^i}{E_k^t}. \tag{4.10}$$

4.4.1 Internal Traffic

If the clusters are approximated by squares, the expected distance from a random location in a cluster in Ring k to its cluster head is

$$h_k^{-2} \int_0^{h_k} \int_0^{h_k} \sqrt{(x - h_k/2)^2 + y^2}\, \mathrm{d}x\, \mathrm{d}y \approx 0.76 h_k. \tag{4.11}$$

Therefore, using (4.7), the expected number of hops of the internal traffic is $\zeta = 0.76\omega h_i/r_t$. The probability that a node is contained within the event area is s^2/A. The number of reporting nodes grows with the number of nodes in the cluster m_k, but this load is also divided among the m_k nodes. Therefore, the expected internal cost per event for a node in Ring k is

$$E_k^i = \frac{0.76\omega s^2}{A} E_r h_k. \tag{4.12}$$

4.4.2 External Traffic

Ring k has to relay all the compressed packets that are generated from Ring $k+1$ to Ring N_L. The total number of external packets that traverse Ring k per event is $\sum_{j=k+1}^{N_L} n_j$, where n_j is given by (4.6). Each packet from those rings has to travel across $\omega h_k/r_t$ hops in Ring k. There are $2\pi r_k h_k \rho$ nodes in Ring k, and we assume that their load is uniformly distributed among them. Therefore, the expected external cost per event for a node in Ring k is

$$E_k^i = E_c \left(\omega \frac{h_k}{r_t}\right) \frac{\sum_{j=k+1}^{N_L} n_j}{2\pi r_k h_k \rho}. \tag{4.13}$$

4.5 Optimization Problem

Considering all factors, the expected energy consumption of a node in Ring k when an event occurs is

$$E_k^t = \frac{E_c \omega}{2\pi r_t r_k \rho} \sum_{j=k+1}^{N_L} \frac{2\pi r_j \left(h_j + s\right)^2}{h_j A_f} + \frac{0.76\omega s^2}{A} E_r h_k. \tag{4.14}$$

The optimal layer-size distribution can be obtained by solving the following

optimization problem:

$$\begin{aligned} &\underset{\{h_i\}_{i=1}^{N_L}}{\text{minimize}} \max_{k=1,\ldots,N_L} \left\{E_k^t\right\} \\ &\text{subject to } \sum_{i=1}^{N_L} h_i = r_b - r_a. \\ &h_k \gg r_t \end{aligned} \tag{4.15}$$

The constraint $h_k \gg r_t$ is needed so that the hop count approximation of (4.7) become reasonable.

4.6 Approximation Algorithm

The optimization problem (4.15) has N_L continuous variables and is non-convex, so it is difficult to solve [91]. Figure 4.2 presents a heuristic algorithm to approximate the solution. The algorithm starts with all the layers of equal size. In each iteration, it extends by Δ the size of the most energy consuming ring and shrinks by Δ the size of the leastp energy consuming ring by that same amount. The parameter Δ does not need to be very small because in a real deployment precise geographic information is unavailable.

The above heuristic approximates the optimal distribution of sizes given N_L. In order to select N_L, we use the following process. We start with a small value of N_L, and obtain $\{h_i\}$ and the maximum energy consumption with our heuristic. Then, we increase N_L and obtain the new maximum energy consumption. If the new maximum energy consumption is smaller than the old one, we repeat the process. The process ends when the new maximum energy consumption is bigger than the old one. The final N_L is the current N_L minus one.

Data: Number of layers N_L; topological parameters: r_a, r_b; step size Δ; ω; threshold γ
Result: $\{h_i\}_{i=1}^{N_L}$
foreach $k \in \{1, \ldots, N_L\}$ **do**
  compute E_k^t using Equation 4.14;
newval = $\max(E_1^t, \ldots, E_{N_L}^t)$;
repeat
  oldval = newval;
  q = index k that maximizes E_k^t;
  $h_q = h_q - \Delta$;
  q = index k that minimizes E_k^t;
  $h_q = h_q + \Delta$;
  foreach $k \in \{1, \ldots, N_L\}$ **do**
    compute E_k^t using Equation 4.14;
  newval = $\max(E_1^t, \ldots, E_{N_L}^t)$;
until *newval - oldval* $< \gamma$;

Figure 4.2: Algorithm to compute the thickness h_i of every ring.

4.7 Performance Evaluation

We perform our performance evaluation by using the equations in this chapter, not by using a simulator that simulates actual packet transmissions. Our simulation parameters are shown in Table 4.1. We discuss the choice of these parameters as follows.

We set the *internal radius* r_a to $4r_{tx}$. We do not set r_a to smaller values because it would make the approximation that the clusters in Ring 1 are squares very poor. We do not take larger values of r_a because would make the assumption that the data sink can communicate with nodes in Ring 1 unreasonable.

We set the *network size* β to values between 10 and 160. We set values larger than 10 because we are interested in large networks with multiple rings. We do not consider values larger than 160 because a network with 160 nodes is already very large and contains and because if the number of rings in the network.

The *node density* ρ is the average number of neighbors per node. We set ρ to 10 because this value is sufficiently large to yield a connected network with high

parameter	value
internal radius r_a of the monitored area	$4r_a$
node density ρ (average number of neighbors per node)	10
network size β	10 – 160
normalized event size γ	1 – 30 (4 default)
compression coefficient σ	4 – 40 (15 default)
increment Δ in the algorithm of Figure 4.2	$0.05r_{tx}$

Table 4.1: Simulation parameters for the cluster segmentation algorithm.

probability.

We set the *normalized event size* γ to values between 1 and 30. We do not set values of γ smaller than 1 because then the probability of the event being contained within a cluster would be very high. This would remove any motivation for large clusters, and thus clusters would become very small. In turn, this would cause the hop count approximation (defined in Eq. 4.7 to become very poor. We do not choose values of γ larger than 30 because this would lead to a small number of rings, which would make the approximation that the clusters in Ring 1 are square-shaped very poor.

We set the *compression coefficient* σ to a value between 4 and 40 for the following reasons. The value of σ has to be bigger than 1 because otherwise there is no compression. The σ larger than 40 because if σ is very large, the external traffic becomes negligible, the cluster size becomes so small that the hop count approximation (defined in Eq. 4.7) becomes very poor.

We set the *increment* Δ in the algorithm of Figure 4.2 to $0.05r_{tx}$. A much larger Δ would not allow to demonstrate the advantage of unequal clusters sizes. A much smaller Δ would be unrealistic because it would imply very high granularity in the number of nodes per cluster, whereas in practice the granularity of this number is limited because the number of nodes per cluster must be an integer.

Figure 4.3 evaluates the algorithm performance as a function as a function of three

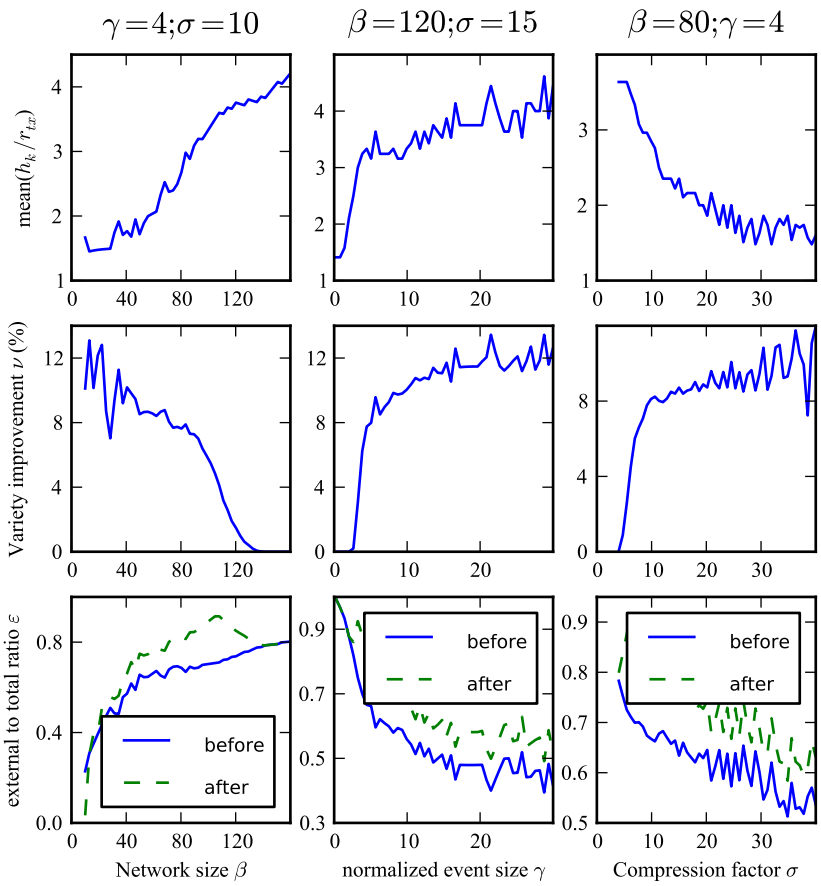


Figure 4.3: Influence of the network size β, the event size γ and the compression factor σ in our system.

parameters: the network size β, the normalized event size γ, and the compression factor σ. The figure is structured as a 3×3 matrix of panels. In each matrix column, two of the three are constant, and the other parameter varies. The values of the constant parameters are shown at the top of the column, and the varying parameter is shown in the X axis at the bottom of the column.

Each matrix column consists of three panels, one in each row.

The first row of Figure 4.3 shows the average normalized ring thickness, defined by mean(h_k/r_{tx}), of our chosen cluster size distribution. This thickness is approximately the maximum number of hops from a node to its cluster head, and it varies between

2 and 4 for my simulated parameters according to Figure 4.3.

The second row of Figure 4.3 shows a metric called *variety improvement*, denoted by ν. We define it as the relative lifetime improvement of our unequal cluster-size heuristic compared to the lifetime obtained with an uniform cluster size. The desired ν is desired to be large because one of the objectives of this chapter is to show the usefulness of using different cluster sizes according to the distance to the data sink. Figure 4.3 shows that lifetime increases of up to 12 % when compared to the uniform distribution.

The third row of Figure 4.3 show the external to internal ratio, ε, defined in (4.10), for the most energy consuming ring. This metric is the quotient between the energy consumed in the external traffic and the total traffic. This metric is shown for the network before and after applying our algorithm.

Figure 4.3 shows that the value of ε is a good predictor of the energy gain ν. In particular, when ε is small, ν is high. This is because if ε is low, the most energy consuming ring consumes most of its energy in internal traffic, which can be reduced by reducing the ring's thickness. By contrast, if ε is high, the most energy consuming layer consumes most of its energy in external traffic, which can hardly be reduced by an unequal cluster size distribution.

The first column of Figure 4.3 shows the influence of the network size β. As the network grows, the number of external packets grows, further straining the nodes close to the gateway. To reduce the number of external packets, the average layer width, mean(h_k), increases, but not as much to avoid an increase in the external to total traffic ratio ε, and thus the gain ν decreases.

The second column of Figure (4.3) analyzes the effect of changing the event size γ. As the event area grows, the clusters grow in order to prevent the growth of the number of clusters that detect each event. The increase in the cluster size increases

the internal traffic and reduces ε, which allows a higher gain ν.

The third column of Figure (4.3) analyzes the effect of the compression factor σ. As compressibility increases, reducing the number of clusters that detect each event becomes less important than reducing the distance from the sensor nodes to their cluster heads. As a result, the average ring thickness decreases, but not as to prevent the decrease in ε, and thus the variety improvement ν grows.

4.8 Conclusions

We have formulated the problem of dividing the network in fixed clusters to report randomly located events. A piece of information can be compressed only once, namely in a cluster head. This kind of data aggregation is unrepeatable, which is the first level of aggregation of this thesis. Our algorithm is unique in that it obtains multi-hop clusters of unequal sizes.

Our algorithm extends the network lifetime by up to a 12 % when compared to the existing algorithms. This gain is small, but comes at very little cost because the clusters are changed infrequently. The gain of our algorithm is the highest when the most energy consuming nodes consume at least 30 % of their energy in their internal traffic. This is because such a fraction of internal traffic allows the network lifetime to be increased by reducing the clusters near the data sink. As a result, the advantage of a heterogeneous cluster size distribution grows as the network size in hops decreases, as the event size increases, and as the compression factor increases.

The energy gains in real networks may not be as high as reported here because some of its assumptions are inaccurate. For example, the closest node from a cluster's desired cluster head position may be far from that position, or it may be impossible to modify a cluster's size by very small amount. However, the algorithm is useful because it can quickly give an upper bound on how much gain can be achieved by

choosing clusters of different sizes. The proposed algorithm has been evaluated in very large networks, larger than 160 hops in radius.

5 Fast Construction of Routing Trees for Repeatable Aggregation

This chapter begins the use of second level of data aggregation in this thesis: repeatable aggregation over routing trees. This model is also used in the next two chapters. The network phases presented in Section 1.3 are initialization, quiet phase, initial routing phase, initial scheduling phase, and data transmission phase. This chapter is concerned with the quiet phase and the initial routing phase. Its goal is to construct a routing tree for repeatable aggregation quickly.

5.1 Introduction

Consider a network operating in the quiet phase. Most of the time, the sensor nodes keep their transceivers turned off in order to save energy. However, they need to use their transceivers to obtain synchronization information and to listen for potential packets. Suddenly, some sensor nodes detect and event and become data sources that need to start reporting their measurements quickly, either to avoid a buffer overflow or because the data sink needs the information quickly. The first step to report the event is to construct the routing tree.

The time to construct the tree depends on the relative timing of the sensor nodes' sleep intervals. The existing protocols fail to consider this and their tree construction

time is long and dominated by the sensor nodes' sleep intervals. This chapter proposes the Fast Aggregation Tree (FAT) protocol, which is a cross-layer protocol that reduces the tree construction time.

Among the network phases from Section 1.3, the initial scheduling phase is skipped in this chapter for two reasons. First, the initial scheduling phase is addressed in Chapter 6. Second, it incurs excessive overhead for the short-lived events for which FAT is designed.

5.2 System Model

5.2.1 Data Generation and Compression Model

Events occur at random locations with average period T_{event}. Each event is detected by a different set of n_s data sources. Every data source generates n uncompressed packets, each one about a period of time called the *reporting interval*. Each uncompressed packet requires a time T_1 to be transmitted. Multiple packets from different data sources can be aggregated if they refer to the same reporting interval. Aggregation is repeatable, which means that the compressed packets can be aggregated again with other packets. The time to transmit a packet containing information from n data sources is

$$T_n = T_1(1 + (1 - \alpha)(n - 1)), \tag{5.1}$$

where α is a compression coefficient between 0 and 1. If $\alpha = 0$, data is aggregated without size reduction. If $\alpha = 1$, any number of packets from the same reporting interval are compressed into a single packet with the same size as an uncompressed packet from a single node.

5.2.2 Delay Constraint

Every event must have been reported to the data sink no later than a time $D_{\max}$ after it occurs. Within this time, the routing tree must be constructed and the data must be transmitted across this tree. We denote the time to construct the tree by T_{constr}, and the time to transmit the data over the tree by T_{trav}. Therefore, the delay constraint that we impose is

$$T_{\text{constr}} + T_{\text{trav}} < D_{\max}. \tag{5.2}$$

Every sensor node keeps its transceiver in sleep mode most of the time, mostly turning it on to listen to its neighbors with period T_{CCA}. The tree construction time is started by the data sources. The data sources cannot communicate with their neighbors until their neighbors become active, which may be a time T_{CCA} later. In turn, these neighbors have to wait until their own neighbors become active. Therefore, in large networks that use a long sleep period T_{CCA}, the tree construction time T_{constr} depends on the number of sleep periods encountered during the tree construction.

5.2.3 Total Power Consumption

In addition to performing channel check-operations with period T_{CCA}, every sensor node also turns on its transceiver with period T_{maint} for *maintenance operations*, which consist in exchanging synchronization information and verifying the links with its neighbors. We denote the energy consumed in channel-check operations by E_{CCA}, and the energy consumed in maintenance operations by E_{maint}.

The sensor nodes consume energy in checking for packets from their neighbors, performing maintenance operations, constructing the routing tree, and transmitting

data over the tree. Therefore, the average total power consumption per node is

$$P_{\text{tot}} = \frac{E_{\text{CCA}}}{T_{\text{CCA}}} + \frac{E_{\text{maint}}}{T_{\text{maint}}} + \frac{E_{\text{constr}} + E_{\text{trav}}}{T_{\text{event}}}, \tag{5.3}$$

where E_{constr} is average energy consumed per node during the tree construction, and E_{trav} is the average energy consumed per node during the data transmission phase.

We assume that maintenance operations are rare and their contribution to P_{tot} is small. The channel-check period T_{CCA} is high, and thus it is an important factor in the time to construct the tree T_{constr}. The events are very rare (e.g. T_{event} is longer than an hour) and short (T_{trav} is in the order of seconds).

5.2.4 Problem Formulation

The problem is to design the quiet phase and the initial routing phase of the network so as to minimize the average total power consumption, given by (5.3), while satisfying the delay constraint, given by (5.2). For efficient operation, the nodes must use a long T_{CCA}, but this increases the T_{constr} and thus the total delay. Therefore, to provide a delay lower than $D_{\max}$, both T_{constr} and T_{trav} must be reduced. The tree construction time T_{constr} can be reduced by arranging the sleep periods of different nodes in a clever way. The time to transmit data across the tree T_{trav} can be reduced by using a routing tree that aggregates data intensively close to the data sources. In short, the problem is to obtain an efficient routing tree for data aggregation quickly.

5.2.5 Related Work

Energy efficient operation during the quiet phase requires the nodes to sleep for long periods. Pure TDMA is unsuitable for the quiet phase because there are no data packets to transmit. Some random access protocols such as B-MAC [23] are

simple and support long sleep periods, but consume much energy in transmitting long preambles. The slotted MAC protocols [24, 21, 25] are probably the best solution for the quiet phase. They consume very little energy when no events occur and they do not need long preambles. Two such protocols are S-MAC [24] and T-MAC [21], which make the sensor nodes sleep synchronously.

Another slotted MAC protocol is DMAC [25]. It groups the nodes in tiers according to their hop distance to the data sink. The nodes i hops away from the data sink constitute Tier i and check the channel state synchronously with period T_{CCA}. The nodes in Tier $i-1$ check the channel state slightly later than the nodes in Tier i. This schedule enables the packets from a data source to reach the data sink in little more than T_{CCA}. The timing diagram of DMAC is very similar to that of FAT, our proposed protocol. However, the two protocols have different functions. DMAC is a MAC protocol that does not enable data aggregation, whereas FAT is a cross-layer MAC and routing protocol for data aggregation.

Most of the existing protocols to construct trees for data aggregation are centralized or do not consider the influence of the sleep period in the tree construction time [43, 44, 45, 46]. The tree that minimizes the energy consumption is the Steiner Tree. Since obtaining it is NP hard [42], the existing protocols obtain approximations. Oceanus [38] is one of several centralized tree construction heuristics that obtain a good approximation of the optimal tree.

The centralized tree construction heuristics operate in three steps. First, the data sources report themselves to the data sink. Second, the data sink computes the routing tree. Third, the data sink informs the sensor nodes of the new routing tree. Therefore, there are two data flows in opposite direction. The centralized tree construction heuristics do not specify the MAC protocol they use to transmit these two data flows. We make the following analysis of their time to transmit their data

flows. Suppose K is the number of tiers in the network. If these heuristics operate on top of S-MAC [24] T-MAC [21] , which have globally synchronous sleep intervals, each of the two flows lasts for a time KT_{CCA}. Therefore, the total routing process lasts for at least $2KT_{\mathrm{CCA}}$. If these heuristics operate on top of DMAC, the duration of the first data flow is approximately T_{CCA}, which is very short, and the duration of the second data flow is almost KT_{CCA}. Therefore, the total is approximately $(K+1)T_{\mathrm{CCA}}$. By contrast, FAT constructs the tree in approximately T_{CCA}.

DB-MAC [59] and DAA+RW [60] are two aggregation protocols that do not require any initial routing phase, and thus $T_{\mathrm{constr}} = 0$. In DB-MAC, every node overhears its neighbors transmission to determine how many packets they hold. When choosing its next hop, it chooses its neighbor with the greatest number of packets in order to promote data aggregation. In DAA+RW, several nodes contend to relay a packet, and nodes with more packets enjoy better chances of winning the contention. In both DB-MAC and DAA+RW, the nodes hold their packets for random periods to increase the probability of data aggregation. This technique is insufficient to guarantee aggregation and introduces delays and overhearing in each packet transmission. Therefore, if the data sources generate a sufficiently high number of packets, DB-MAC and DAA+RW suffer high T_{trav} and E_{trav}. Another protocol, ToD [92], combines DAA+RW with a fixed aggregation structure. ToD outperforms DAA+RW in big networks as it guarantees aggregation after a number of hops, but it is not efficient for big data volumes in those few hops because it aggregates data opportunistically.

5.3 The Fast Aggregation Tree (FAT)

We propose the FAT protocol to construct an aggregation tree after each event. FAT is a distributed, cross-layer protocol to obtain a routing tree for data aggregation.

The operation of the network starts with the initialization phase. In this phase, the sensor nodes obtain their initial time synchronization and obtain their *tier*, which is their minimum number of hops to the data sink.

5.3.1 Quiet Phase

The next network phase is the quiet phase, which is depicted in Figure 5.1 and continues until an event occurs. The sensor nodes' transceivers are in sleep mode most of the time, and they are turned on with period T_{CCA} to perform a clear channel assessment (CCA). All the sensor nodes in the same tier perform the CCA simultaneously, and the nodes in different tiers perform it at different times. Specifically, the schedule of Tier i is T_{tier} in advance with respect to schedule of Tier $i-1$. Here, T_{tier} is an amount of time sufficient to transmit approximately a dozen packets.

5.3.2 Initial Routing Phase

When an event occurs, some sensor nodes become data sources and initiate the tree construction as follows. Every data source chooses its parent node, and this parent node chooses its own parent node, and this process is continued until the full tree is constructed. If a node X is in Tier i, it chooses its parent node among its neighbors in Tier $i-1$ within its transmission range, which are referred as X's *potential parents*. We now describe the timing of the process in more detail.

5.3.2.1 Listening for Parent Requests

Let X be a sensor node from Tier i. If X senses the channel idle during its scheduled CCA time, it sleeps until its next scheduled CCA time. However, if it senses the channel busy, it keeps its transceiver active for a period T_{tier}. During this period,

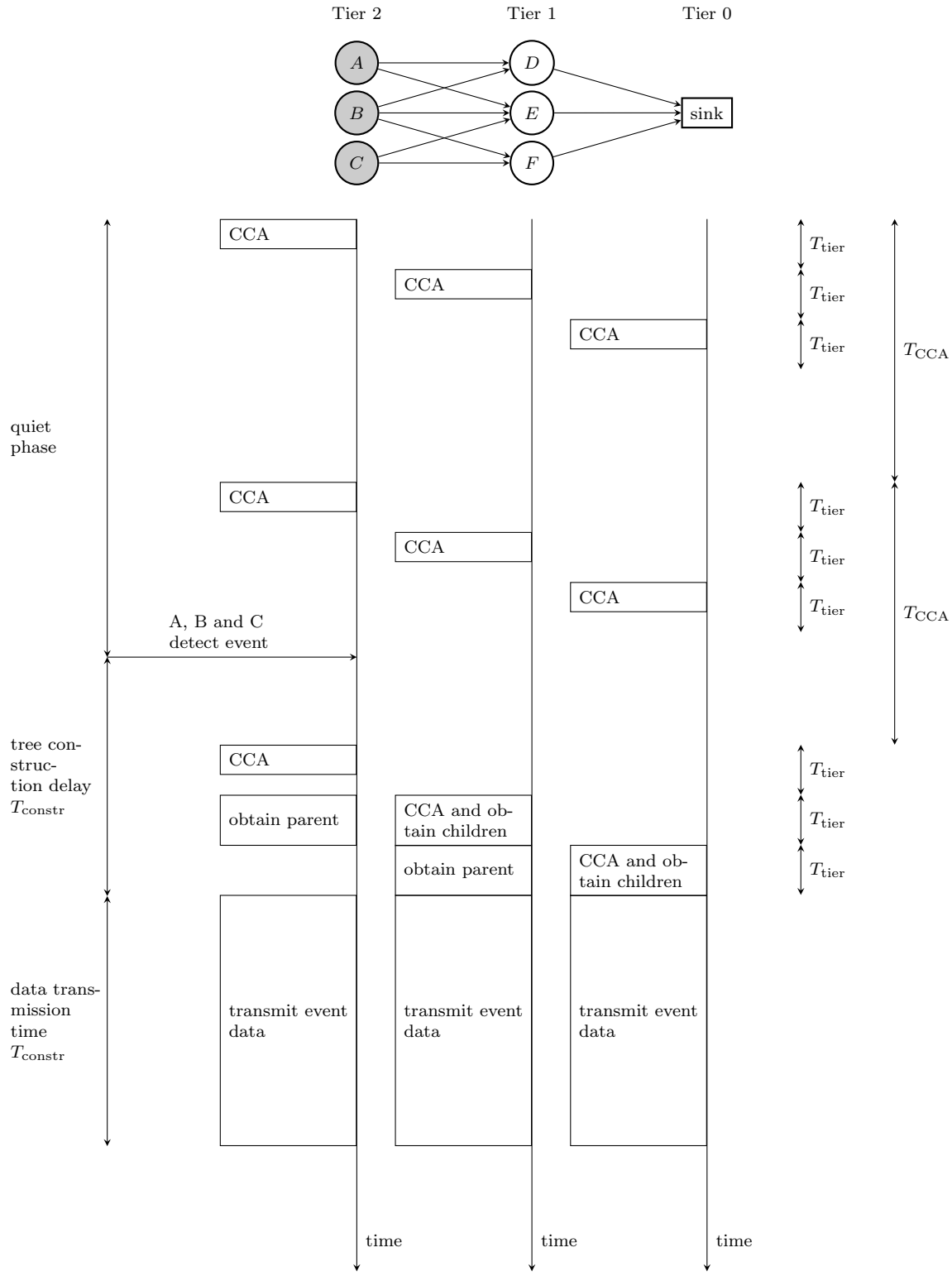


Figure 5.1: Staggered clear channel assessment (CCA) times of nodes in different tiers.

X may receive packets from its neighbors from Tier $i+1$. These packets are called *parent requests*. If X receives a parent request from a certain node Z, X adds Z to its list of children in the routing tree. In addition, X contends immediately to transmit a packet to Z. This packet is called *parent confirmation*, and it serves to confirm X's acceptance of Z as a child node. Node X only contends to transmit this packet once. If X fails the first time, it removes Z from its children list and does not contend again to transmit a parent confirmation unless it receives a new parent confirmation.

5.3.2.2 Obtaining a Parent Node

Let X be a sensor node from Tier i that is a data source or has obtained a child node. Therefore, X needs to find a parent node, and it does so as follows. It waits until the scheduled CCA time of Tier $i-1$, and at this time it transmits a very short dummy packet in order to make the channel busy. With this packet, X requests its neighbors in Tier $i-1$ to keep their transceivers active for a period of duration T_{tier}.

After transmitting the dummy packet, X contends to transmit a parent request. In order to do so, it chooses a backoff period and it senses the channel until the end of this period. If X senses the channel idle during this period, it transmits a parent request to the node that it wishes to have as a parent. This node is referred to as X's *preferred parent*. Node X randomly chooses its preferred parent among its neighbors in Tier $i-1$. After transmitting the parent request, X listens for a parent confirmation from its preferred parent. Node X only listens for a short time for this packet because the back off period to transmit a parent confirmation is smaller than the back off period to transmit a parent confirmation. The back off periods are shown in Table 5.1. If X receives the parent confirmation, X has obtained a parent.

Node X fails its first attempt to obtain a parent if either it does not manage to

transmit a parent request or it transmits a parent request but receives no confirmation. Either way, node X keeps trying to transmit parent requests until it receives a confirmation or the period of duration T_{tier} reserved for parent acquisitions of nodes in Tier i has expired. After each failed attempt to obtain a parent, node X modifies the lower and upper boundaries that the random back off period can take. Specifically, it multiplies each of these two boundaries by two in order to obtain a exponential back off.

Until X obtains a parent confirmation, it keeps its transceiver on. This way, it can overhear some packets addressed to other nodes. In particular, it is interested in the parent confirmations addressed other nodes. Based on these parent confirmations, X becomes aware of some of the children of its potential parents. Node X uses this information to choose its preferred parent. When X manages to get access to the medium, it selects its preferred parent among its potential parents with the greatest number of children. This choice promotes data aggregation.

5.3.3 Examples of Network Operation

Suppose that in the network in Figure 5.1, nodes A, B and C detect an event and become data sources. The data sources wait until the CCA time of nodes in Tier 1, which may be as late as T_{CCA} after the event occurrence. At the CCA time of Tier 1, the data sources transmit a dummy packet. As a result, nodes D, E and F from Tier 1 sense the channel busy, and interpret this as a request to remain active for a period of duration T_{tier}. During this period, they listen for parent requests from nodes in Tier 2.

During the period where nodes D, E and F remain active, nodes A, B and C contend to transmit parent requests. Suppose that A is the first node to transmit a parent request to its preferred parent. Node A randomly chooses its preferred parent

among its potential parents, which are D and E. Suppose that A chooses E as its preferred parent. Node A transmits a parent request to E. If E receives this request, it adds A to its child list. Next, E replies with a parent confirmation to A. If A receives this confirmation, A has obtained a parent.

Suppose that nodes B and C overhear the parent confirmation from E to A. Nodes B and C continue to contend to transmit a parent request. Suppose that C is the next node to transmit a parent request. Before transmitting its request, C selects its preferred parent. Node C can choose its preferred parent among E and F according to Figure 5.1. However, in this case it does not choose randomly between E and F. From the parent request it overheard, C knows that E has a child, whereas C is not aware of F having any child. Therefore, C transmits its parent request to E. When C receives the confirmation from E, C has obtained a parent. Next, B transmits its parent request to E because it knows that E has two children.

At the end of the period of duration T_{tier}, node E has obtained three children. Now E is responsible for relaying other nodes' packets, and thus it needs to find a new parent node. Therefore, it contends to transmit a parent request to the data sink. It transmit such requests until it receives a parent confirmation.

5.4 Discussion of Properties of FAT

5.4.1 FAT's Tree Construction Time T_{constr}

The worst case for the tree construction time T_{constr} occurs when the data sources lie in Tier K, and when the event is detected just after the beginning of the CCA period of the nodes in Tier $K-1$. In this case, the data sources have to wait for a time as long as T_{tier} until their potential parents (which lie in Tier $K-1$) become available again. Then, during the period of duration T_{tier} starting in the CCA period

of Tier $K-1$, the data sources obtain a parent. Next, during the period of duration T_{tier} starting in the CCA period of Tier $K-2$, the nodes in Tier $K-2$ obtain a parent. This process continues until the full tree has been constructed. All in all, the tree construction time is

$$T_{\text{constr}} = T_{\text{CCA}} + (K-1)T_{\text{tier}}. \tag{5.4}$$

Recall that the fastest protocol discussed in Section 5.2.5, is $T_{\text{constr}} = KT_{\text{CCA}}$. Therefore, if T_{CCA} is much larger than T_{tier}, FAT is much faster than the fastest existing protocols. This tree construction speed is FAT's main advantage.

5.4.2 Suboptimality of the Obtained Tree

In the example in Section 5.3.3, nodes A, B and C obtain E as their parent node, and E's parent is the data sink. Therefore, the constructed tree has four edges. Since no tree can connect the data sources to the data sink with fewer edges, the constructed tree is a Minimum Steiner Tree (MST), which is the optimal solution.

Although in the example in Section 5.3.3 FAT obtains an optimal tree, this is not always the case. In particular, if all the parent requests and parent confirmations are received correctly, FAT only obtains an optimal tree in Figure 5.1 if the recipient of the first transmitted parent request is E. This is because in order to obtain the MST, {A,B,C} have to select E as a parent node.

5.5 Choice of the offset T_{tier} between tiers

The network-wide parameter T_{tier} is important. If T_{tier} is too short, some nodes cannot find a parent because they do not have time to transmit a parent request. If T_{tier} is too long, the tree construction time T_{constr} increases according to (5.4).

Therefore, a balance must be found when choosing T_{tier}.

In the particular case in which T_{CCA} is much larger than T_{tier}, T_{tier} has little influence on the tree construction time T_{constr} unless K is very large.

Parameter T_{tier} has to grow with the node density. This is because as the node density increases, more nodes contend to transmit parent requests and more collisions occur.

5.6 Extensions for Enhanced Reliability

We discuss as follows two potential extensions to FAT that we do not implement in our simulations.

5.6.1 Improved Selection of Preferred Parent

There is a problem in the original preferred parent selection algorithm. In order to illustrate this problem, consider a node X seeking to obtain a parent. Suppose that X has recorded nodes Y and Z as its neighbors in the next tier, which means that Y and Z are the potential parents of X. Suppose that node Y has failed but node Z has not. Also suppose that X does not have any neighbors in the same tier contending for a parent. If node X chooses node Y as its preferred parent, it will repeatedly transmit parent requests without receiving an answer and will never obtain a parent. This is a poor choice of the preferred parent because if X had chosen Z as a parent, it would have obtained a parent node.

The improvement of the preferred parent selection algorithm that we propose is as follows. If a node repeatedly transmits parent requests to a node and fails to receive parent confirmations, it changes its preferred parent. The number of failed attempts that trigger a change in the preferred parent should be determined based on the

node failure probability and the packet collision probability, which depends on the node density and the average numbers that detect an event.

This improvement raises a minor problem that we illustrate through the following example. Suppose that a node X transmits a parent request to a node Z. Node Z replies with a parent confirmation, but X does not receive it. Node X transmits another parent request to another node Y, and in this case X receives a parent confirmation. This situation is problematic because Z considers X as its child but X does not consider Z as its parent. However, this problem can be solved easily during the data transmission phase. After Z spends a certain amount of time without receiving DATA packets from X, it removes X from its child list.

5.6.2 Emergency Construction of the Tree

If any node X seeking to obtain a parent does not succeed within the period of duration T_{tier}, the tree construction of FAT is said to have failed. This failure may occur for one of three reasons. First, X could not find a parent because the period T_{tier} was too short. Second, X could not find a parent because the improvement of Section 5.6.1 was not implemented and X chose a faulty node as its preferred parent. Third, none of the potential parents of X are functional, possibly because the division of nodes in tiers is outdated due to an excessively long maintenance period T_{maint}.

We suggest, but do not implement, an extension to FAT, which consists of providing an *emergency construction method*. This method is slower and more energy consuming than the *normal construction method*, which is the one described until now. Thus, the emergency construction is only meant as a backup. We describe the emergency construction as follows.

5.6.2.1 The Emergency Slot

The extension does not affect the quiet phase, only the tree construction phase. One of the changes affects the period labeled "CCA and obtain children" in Figure 5.1, which will be referred to as *contention period*. In the original version, the contention period consists of two slots. The first slot is referred to as *activation slot*, and is used for transmitting dummy packets whose purpose is to make the channel busy. The second slot is referred to as *exchange slot*, and is used for transmitting parent requests and parent confirmations. The nodes remain active during the exchange slot only if the channel was busy during the exchange slot. In the modified version, the contention period contains a third slot in addition to the two slots of the original version. This third slot is referred to as *emergency slot* and is as short as the activation period.

5.6.2.2 The Emergency Signal

Every node that hears the channel busy during the activation period remains active during both the exchange and emergency slots. The emergency slot is used in the same way as the activation slot, i.e., dummy packets are transmitted with the goal of making the channel busy. However, a busy channel has a different meaning in the activation slot than it has in the emergency slot. In the emergency slot, a busy channel is considered as an *emergency signal*. The emergency signal indicates that the normal construction has failed and that the emergency construction should begin. Any node receiving the emergency signal should propagate the emergency signal to its neighbors and remain active until the tree construction has concluded.

5.6.2.3 Generation and Propagation of the Emergency Signal

Le X be a node contending for a parent. Suppose that at the end of the exchange slot X has not obtained a parent. Then, X transmits a dummy packet during the emergency slot in order to make the channel busy. Over the next period of duration $KT_{\mathrm{CCA}} - T_{\mathrm{tier}}$, node X transmits dummy packets during the activation, and emergency slots of the contention periods of every tier. Since there are $K-1$ such periods, X transmits $2(K-1)$ dummy packets in total since the event occurrence. When X concludes the transmission of all these dummy packets, it remains active.

If a node Y senses the channel busy during an emergency slot, it propagates the emergency signal by transmitting $2(K-1)$ dummy packets in the same way as X did. Then, Y starts listening. The transmission of dummy packets propagates the emergency signal across the network. If the network is connected, it reaches the data sink eventually. The time until the emergency signal reaches the data sink will usually be smaller than KT_{CCA}, but it may be as long as LT_{CCA}, where L is the number of nodes in the network.

5.6.2.4 The Tree Construction

When the emergency signal reaches the data sink, the data sink waits for some time in order to give more time to other nodes that may not have received that signal yet. Such nodes are unprepared for the tree construction because they keep the schedule of the quiet phase and thus spend most of the time in sleep mode. The waiting time of the data sink would have to be as high as LT_{CCA} in order to guarantee that all nodes have received the emergency signal. However, such a long waiting time would be cause excessive delay and energy consumption. Therefore, we recommend a waiting time such as KT_{CCA}. A the end of the waiting time, the data sink is ready to initiate the construction of the routing tree. This construction is implemented

through the protocol of the initialization phase, which was described in Section 1.3.

5.7 Simulation Results

FAT's advantage is a low T_{constr}, which allows use of a large channel check period T_{CCA} and still verify (5.2). A large T_{CCA} is beneficial because it reduces the energy consumption according to (5.3).

One disadvantage of FAT is that the routing tree it obtains is not as efficient as that of other centralized heuristics such as Oceanus [38]. This means that the tree traversal time T_{trav} of FAT is higher than that of the centralized heuristics. However, Section 5.7.1 shows that this difference is small.

FAT performs best when every sensor node is in the appropriate tier, as discussed in Section 5.6.2. However, keeping each node in the appropriate tier requires periodic maintenance operations if the channel changes over time. Section 5.7.2 studies FAT's tolerance to outdated topological information.

5.7.1 Tree Traversal Time T_{trav}

The tree traversal time T_{trav} is a measure of the ability of the routing tree to transmit data. Here, we compare the tree obtained by FAT with the trees obtained with the following algorithms:

- *Shortest Path Tree (SPT).* It is Dijkstra's algorithm using the hop distance as the cost metric. It is unaware of the set of sources and does not try to maximize data aggregation.

- *Dijkstra1.* It is a modification of SPT designed to promote aggregation. When choosing the node to add to the tree among different choices, it considers the following criteria in this order. First, it prefers nodes with a smaller hop

distance to the data sink. Second, it prefers data sources over other nodes. Third, it prefers nodes some of whose descendants are data sources.

- *Centralized1.* This algorithm has several steps. First, it identifies the node X lying the closest to the event. Second, it constructs a tree rooted at X using the Dijkstra1 algorithm. Third, it removes all the nodes that are neither sources nor in the path from a source to node X. Fourth, it links X with the sink through the shortest path.

Centralized1 is a modification of Oceanus [38]. In Oceanus, the node X is chosen randomly, whereas in Centralized1 X is chosen as the closest node to the event. The event center is unknown in practice, and thus Oceanus is more practical. However, the purpose of this section is to evaluate the quality of the routing tree, and in this matter Centralized1 is better than Oceanus.

We develop a custom simulator in Matlab with the parameters in Table 5.1. The power consumption parameters are taken from the Chipcon CC2420 transceiver [93]. The simulator takes the average of 400 different networks. Each networks consists of $L = 50$ nodes randomly deployed over an $x \times y$ rectangle, where $x = y = 200\,m$. Every sensor node has a transmission range $r_t = 60\,\mathrm{m}$ and an interference range of 150 m. The node density ρ is defined by $\rho = L\pi r_t^2/x/y$, and thus is 14 in this case. The sink is at one corner of the rectangle. The event to be reported occurs in the diagonally opposite corner and the $n_s = 12$ nodes closest to the event location detect the event in some way and thus become data sources. Each source node generates $n = 10$ packets. The transmission of an uncompressed packet is $T_1 = 8\,\mathrm{ms}$.

During the data transmission phase, the sensor nodes use CSMA. Every time that a node receives a packet, it replies with an ACK. Every sensor node waits from all its children's packets before aggregating the result and relaying it to its parent node. If a node does not receive any packet for a duration of $4T_1$, it requests a retransmission

parameter	value
◦ number of nodes L	50
◦ width x of the monitored area	200 m
◦ height y of the monitored area	150 m
◦ transmission range r_t	60 m
◦ interference range	150 m
◦ node density $\rho = L\pi r_t^2/x/y$	14
◦ number n_s of nodes that detect the event	12
◦ number n of packets generated per node	10
◦ interval T_{CCA} between CCA operations during the quiet phase	120 s
◦ time T_{tier} during which the sensor nodes in a tier can contend for a parent during the tree construction	300 ms
◦ time to perform a CCA and start a packet transmission	10 μs
◦ range of the random back off period during which a node needs to sense the channel continuously clear before transmitting a parent request	100—200 μs
◦ range of the random back off period during which a node needs to sense the channel continuously clear before transmitting a parent confirmation	0—100 μs
◦ maximum of the random back off period during which a node needs to sense the channel continuously before transmitting a data packet	0.05 T_1
◦ transmission time a dummy packet in order to make the channel busy during the CCA period of the next tier	1 ms
◦ transmission time of a parent request	1 ms
◦ transmission time of a parent confirmation	1 ms
◦ transmission time T_1 of an uncompressed packet	8 ms
◦ transmission time of the acknowledgment of a data packet	1 ms T_1
◦ range of the random back off period during which a node needs to sense the channel continuously clear before transmitting a parent request	100—200 μs
◦ range of the random back off period during which a node needs to sense the channel continuously clear before transmitting a parent confirmation	0—100 μs
◦ time without hearing any packet in the data transmission phase after which a node requests a packet retransmission from its children nodes	6 T_1
◦ sleep power	60] μW
◦ receive power	63 mW
◦ transmit power	57 mW

Table 5.1: Simulation parameters for the cluster segmentation algorithm.

from the next packet that it expects.

Figure 5.2 shows the reduction in tree traversal time T_{trav} of FAT and Centralized1 with respect to SPT. The curve for Dijkstra1 is not shown because it is very similar to that of FAT. The X axis shows the aggregation coefficient α.

A low α represents a low degree of compression. For $\alpha = 0$, the optimal tree is the SPT. The best possible performance is to achieve a 0 % reduction, and FAT achieves exactly that. By contrast, Centralized1 performs very poorly for $\alpha = 0$. According to Figure 5.2, Centralized1 performs a 24 % worse than SPT. This is because the performance of Centralized1 greatly depends on the aggregation coefficient α.

A high α represents a high degree of data compression. Figure 5.2 shows that for $\alpha = 1$, FAT is 8 % better than SPT, and Centralized1 is 14 % better than SPT. Therefore, FAT aggregates significantly more packets than SPT does. The advantage of Centralized1 over FAT is not very large, and is often compensated by FAT's shorter tree construction time T_{constr}.

FAT is less efficient than Centralized1 for $\alpha = 1$ for two reasons. First, because FAT constrains a node's possible parents to its neighbors in the next tier. Second, because the node does not make an optimal choice among its possible parents. The first reason outweighs the second reason. The proof of this is that Dijkstra1 barely outperforms FAT, and that FAT would perform as good as Dijkstra if the nodes made the best possible parent choice within the tiered architecture.

FAT's advantage over SPT in terms of T_{trav} for large α at very little cost. SPT does not require a tree construction phase because the tree is fixed, but it requires a notification phase with an overhead slightly smaller than that of FAT. When an event occurs, the new data sources need to request their ancestors to leave the quiet phase and enter the data transmission phase. The best way to propagate this request is using DMAC, which requires little more overhead than FAT.

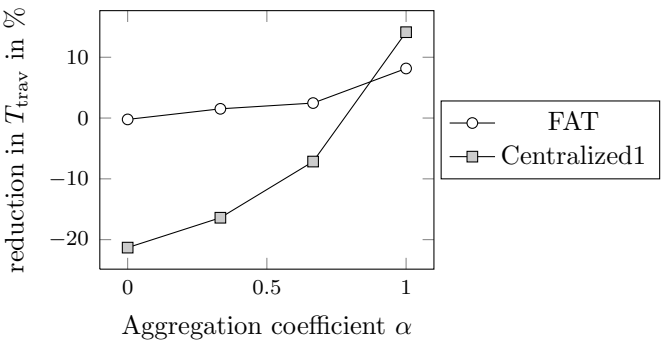


Figure 5.2: Reduction in T_{trav} relative to SPT.

Figure 5.3 compares DMAC, SPT, FAT and Centralized1 for $\alpha = 1$ as a function of the number of source nodes n_s. DMAC is the only protocol in the figure that does not perform data aggregation. If the number of sources n_s and the number of packets n are small, DMAC performs well, but as either of those number grows, the benefit of data aggregation increases. Figure 5.3 also shows that the difference between the three aggregation protocols is small and barely increases with the number of sources n_s.

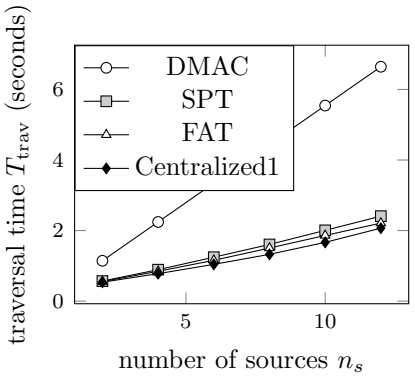


Figure 5.3: Tree traversal time T_{trav} as a function of the number of sources n_s.

5.7.2 Resilience to Node Failures

Consider a network in which every node fails with probability p_f. Node failures may make the network disconnected, or they may require to change the tiers of some

nodes. If the tiers are not updated, some nodes may not be able to find a parent node. Therefore, in order to maximize the probability that FAT connects a node with the data sink, the tree structure must be kept up to date.

However, keeping the tiered structure updated requires periodic maintenance operations that consume energy. Therefore, it is desirable to update the tiered structure as little as possible. In this section, we examine the tolerance of FAT to outdated tier information.

We define the *source isolation probability* p_I as the probability that a sensor node cannot find a path to the data sink. The following example illustrates the computation of p_I in a simple topology. If every node has n potential parents, the sources are h hops away from the sink, and the link failure probability is p_f, then p_I is $1-(1-p_f^n)^h$ in FAT. Therefore, FAT becomes more robust as the number of hops decreases and the node density increases.

5.7.2.1 Simulation setting

In order to evaluate the source isolation probability p_I in response to node failures in random deployments, we use simulations. In our simulations, all the nodes fail simultaneously. Our simulations compare three different techniques to construct the routing tree: SPT, FAT and optimal. The first technique, SPT, consists of using a fixed tree that is constructed before the link failures occur. The second technique, FAT, uses the improved parent selection algorithm from Section 5.6.1 but not the emergency construction from Section 5.6.2. The neighbor and tier information of FAT nodes is not updated after the link failures occur because we wish to study the tolerance of FAT to outdated information. The third technique, optimal, updates the neighbor information after the event occurs. Therefore, if after the link failures there remain sufficient functional links to make the network connected, optimal achieves

zero source isolation probability p_I.

The simulated area is a rectangle of width d and height $4r_t$. The sink and the event center are located at the middle of left and right borders of the rectangle. The data sources are the five closest nodes to the event center. Therefore, the distance from the sink to the data sources is approximately d. We simulate 3200 random deployments. We discard any deployment with over 20 % of disconnected nodes, which is rare for $\rho \geq 7$. After neighbor information has been collected, the node failures are simulated. The probability that a node is considered as failed is $p_f = 0.05$. After the node failures, the tree construction techniques are executed and the fraction of nodes that do not get connected (possibly across multiple hops) to the data sink is recorded in p_I.

The simulation parameters are the same as in Section 5.7.1, with the difference that here we set $T_{\text{tier}} = 3s$. Such a long T_{tier} makes it very likely that every sensor node obtains a parent node if any of its neighbors in the next tier is available. We use a long T_{tier} because our main purpose here is to assess the resilience of the tiered topology from Figure 5.1 to outdated topology information, not the speed of the improved parent selection algorithm from Section 5.6.1.

5.7.2.2 Simulation results

Figure 5.4 shows the source isolation probability p_I for different values of the node density ρ and the distance d from the event to the data sink. SPT is very vulnerable to link failures because a node cannot communicate with the data sink if any of its ancestors fails. FAT is more resilient to link failures than SPT because the tree is generated after an event. The process is successful as long as every node that needs a parent can communicate at least with one neighbor lying one hop closer to the data sink than itself. Finally, optimal refers to the execution of a full network discovery

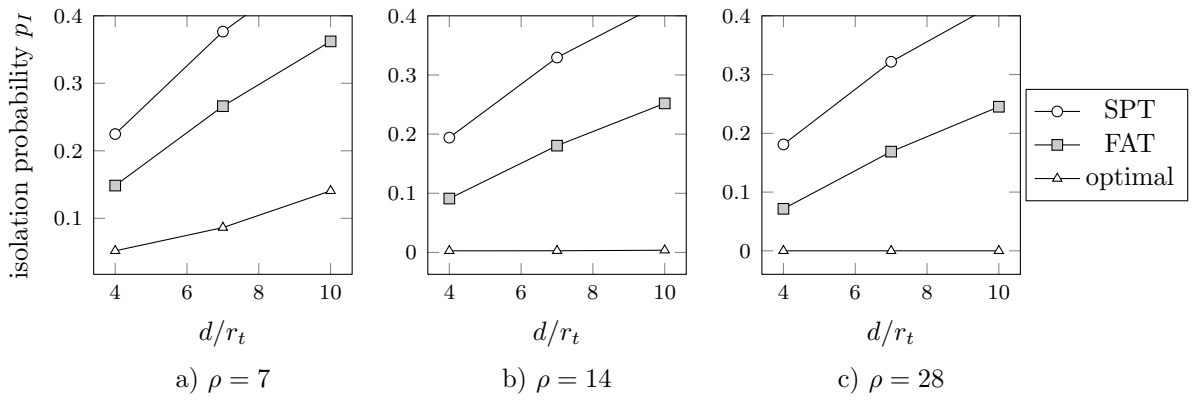


Figure 5.4: Source isolation probability p_I as a function of the normalized event distance to the data sink, d/r_t.

process and Dijkstra's algorithm.

Figure 5.4 shows that increasing the event distance d has a very detrimental effect on the source isolation probability p_I. This is because the event distance increases the number of points where the tree construction may fail. Figure 5.4 also shows the reduction of p_I with the node density ρ. This reduction is noticeable from $\rho = 7$ to $\rho = 14$, but insignificant from $\rho = 14$ to $\rho = 28$.

5.8 Conclusions

In this chapter we have proposed the FAT protocol, whose purpose is to construct a routing tree for repeatable aggregation quickly. FAT is executed during the quiet phase of the network and during the initial routing phase. It organizes the sensor nodes in tiers. The channel check instants of nodes of different tiers are staggered.

FAT's main advantage is that its tree construction time T_{constr} is approximately equal to the channel check period T_{CCA}. Such a fast tree construction enables the network to operate with a high T_{CCA} and still satisfy the delay constraint (5.2) with a high check period T_{CCA}. A high check period T_{CCA} yields important energy savings if the events are rare and short.

The routing tree constructed by FAT aggregates less data than the trees obtained by other heuristics. However, FAT is only outperformed slightly, particularly if the aggregation factor α is small. The heuristic Centralized1 outperforms FAT for $\alpha = 1$, but it performs very poorly for small values of α. By contrast, FAT performs well for all α.

Our simulations show that FAT is relatively vulnerable to outdated tier information. For example, if the node failure probability is 5 % and the data sources are four hops away from the data sink, the probability p_I that a data source cannot find a path to the data sink is as high as 15 %. The situation improves slightly with the node density and degrades severely with the number of hops. Therefore, tier-maintenance operations are essential in FAT. This makes FAT inappropriate in rapidly changing networks.

6 MAC Scheduling for Repeatable Aggregation

The previous chapter proposed a protocol to construct a routing tree for repeatable aggregation. Once the routing tree has been obtained, a TDMA transmission schedule is needed to transmit the data efficiently. EATP, the protocol that we presented in Chapter 3, can sometimes be used for this task. In this Chapter, we discuss how to extend EATP to one specific data aggregation situation. We call this extension Data Aggregation TDMA Protocol (DATP).

DATP can be executed right after FAT (Chapter 5.8) because the two protocols are designed for two consecutive steps. Although for optimal performance the two steps should be combined, few protocols attempt that [94]. Therefore, we evaluate each step separately and in this chapter we do not simulate FAT. Note that the execution time of DATP is generally much larger than that of FAT.

6.1 Problem Formulation

Consider a network where a routing tree has been constructed. Aggregation is repeatable and operates according to (5.1) with $\alpha = 1$, which implies the highest degree of compression. Every packet transmission occupies one DB, independently of the number of data sources whose information is contained in this packet.

The maximum and the histogram are two practical aggregation functions that verify $\alpha = 1$ [8]. This is because the maximum of any number of measurements is simply a number, and the histogram of any number of measurements can be stored in as many numbers as bins are used in the histogram. Therefore, the information that needs to be transmitted to the data sink is independent of the network size.[1]

Suppose that every node in the routing tree is a data source and generates one packet per TF. There are no packet losses due to random factors, only due to the interference from other concurrent transmitters. Then, every sensor node in the tree must be allocated one DB per TF.

The problem is to quickly obtain a TDMA schedule that assigns one DB to every sensor node. The schedule should have a low failure probability p_f, i.e. it should be unlikely to allocate DBs with excessive interference. Furthermore, the schedule should verify the precedence property for data aggregation, which states that if a sensor node is assigned DB_i and its parent is assigned DB_j, then i must be smaller than j. This precedence property differs from the one described in the non-aggregation case in Section 3.1, which assumed that every sensor node was assigned as many DBs as children it had plus one.

6.1.1 Motivation of the Precedence Property

The precedence property reduces the upstream data transmission latency because it allows the data from all the nodes to reach the data sink within a single TDMA frame. To illustrate this property, compare the schedule of Figure 6.1a, which verifies the precedence property, and the schedule from Figure 6.1b, which does not verify this property. With the first schedule, in the first DB, C transmits a packet to B. In

[1] In our third paper (cf. page 15), we evaluate DATP under other compression models. We omit these models to provide more unity to the thesis. The main result of the paper is that, if less aggregation is feasible, DATP still outperforms the BF protocol (cf. Section 6.2), but only slightly.

Figure 6.1: Schedule a verifies the precedence property for data aggregation, but schedule b does not.

the second DB, D transmits its packet to B. Then, B aggregates the packets from C, D and B together. Finally in the third DB, B transmit the aggregated packet to A. Therefore, all the packets reach the data sink within one TDMA frame. By contrast, the second schedule requires two TDMA frames to transmit the data from node D to the data sink. This is because, although node B has been assigned the second DB, it cannot use it in the first TDMA frame because it has not received the data from D yet.

6.2 Related Work

The distributed TDMA protocols that use the k-hop interference model [39, 27, 31, 30] do not verify the precedence property for data aggregation. Recall that the precedence property for data aggregation (which we consider in this chapter) is different from the precedence from for convergecast (which we considered in Chapter 3). The FlexiTP protocol [27], which we used as a benchmark in Chapter 3, verifies the latter property, but not the former property, which is the one that we need. Therefore, we cannot use FlexiTP for comparison in this chapter. Also note that, to our knowledge, there are no adaptive scheduling algorithms that verify the precedence property for data aggregation. Therefore, our focus here is in the initial scheduling phase: its execution

time, the length of the schedule M, and the probability of assigning collision-free time slots.

The problem of obtaining the shortest possible schedule that verifies the precedence property for data aggregation under the 1-hop interference model is NP hard [85]. The algorithm in [85] computes a suboptimal schedule with schedule length M equal to $(\Delta - 1)R$, where Δ is the maximum number of neighbors per node and R is the network diameter, defined as the maximum hop distance the data sink. The algorithm in [85] obtains a schedule whose length M is $23R + \Delta - 18$, which is much smaller but still large. However, the main problem of these two algorithms is that their interference model, the 1-hop interference model, is very unrealistic. In addition, these algorithms do not provide time slots for transmitting acknowledgment packets.

When using the k-hop interference model with $k \geq 2$, the problem of obtaining the shortest possible schedule is also NP-hard [36, 95]. The algorithms in [35, 36] address this problem. Since the two algorithms are very similar, we treat both as the same algorithm and refer to it as BF_k. BF_k obtains an approximation of the optimal schedule in polynomial time. It assigns DBs by traversing the routing tree in Breadth-First order (hence the BF part of the name). It was proposed for the k-hop interference model with $k = 2$ (hence the subscript k in the name).

BF_k is slow because it is centralized. It starts its execution after an event is detected and consists of four phases. First, every sensor node discovers its neighbors if it has not kept track of them during the quiet phase in order to save energy. Second, every sensor node transmits to the data sink a list indicating its neighbors. Third, the data sink uses these lists to compute the schedule. Fourth, the data sink informs every sensor node of its DBs.

The EMAC protocol [96] is distributed and verifies the precedence property, but neglects interference generated more than 2 hops away in the routing tree. This

interference model is very unrealistic because it may assign the same time slot to two nodes within transmission range of each other.

6.3 DATP

Data Aggregation TDMA Protocol (DATP) is an extension of EATP (cf. Chapter 3) that is obtained by applying two changes to EATP. First, during the initial scheduling phase, in order to satisfy the precedence property for data aggregation, a node only contends to obtain a DB when all its children have obtained a DB or a certain interval has elapsed. Second, the ability to assign DBs during the data transmission phase is suppressed due to the difficulty in modifying an existing schedule while ensuring the precedence property for data aggregation.

To illustrate the difficulty of maintaining the precedence property, suppose that in Figure 6.1 node C cannot communicate with B in DB_1 and needs to be assigned a new DB. Then, it is not possible to assign a new DB to C without assigning a new DB to B because the DB assigned to B needs to have a bigger index than the DB assigned to C due to the precedence property.

Due to the suppression of adaptivity during the data transmission phase, no network changes can be handled during the data transmission phase. Additionally, no new nodes can be assigned to nodes that were unduly scheduled during the initial scheduling phase. Therefore, obtaining a collision free schedule is much more important here than it was in Chapter 3, because here the only way to obtain a new schedule is to execute the initial scheduling phase again.

parameter	value
◦ number H of slot pairs per CF in EATP	4
◦ node density ρ	9.6–24
◦ maximum fraction of disconnected nodes in the network	10 %
◦ normalized network size $\bar{x}$	3–8
◦ number of nodes in the network	28–484
◦ number of simulation runs for each node density and network size	100

Table 6.1: Simulation parameters in DATP.

6.4 Performance Evaluation

We use the same simulator as in Chapter 3. Most of the simulation parameters are the same as in Table 3.1. The new parameters are presented in Table 6.1. The maximum simulated normalized network side $\bar{x}$ is 8, which corresponds to a network of more than 8 hops and thus is considerably large. We set the maximum simulated node density ρ to 24, which results in a maximum number of simulated nodes of 484. We can afford to simulate a much larger number of nodes here than in Section 3.4.2 because the simulator is much faster here because far fewer DBs have to be assigned. The simulation results in this chapter are the average of 100 simulation runs.

6.4.1 Parameter Choice in DATP

Figure 6.2 shows the influence of H on the schedule length M. DATP needs a lower H than EATP (cf. Figure 3.3) because in DATP the number of contenders is smaller due to the precedence property for data aggregation. In the rest of our simulations, we use $H = 4$.

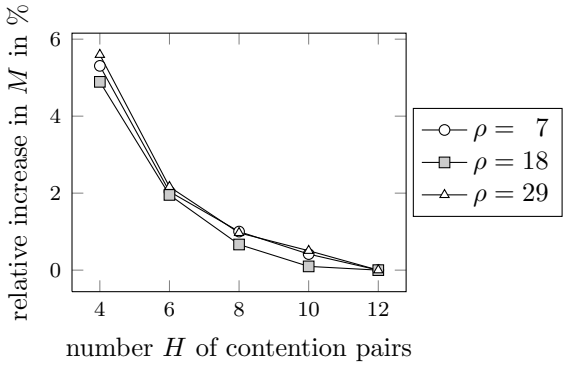


Figure 6.2: Length M of the computed schedule as a function of H and ρ. The y-axis shows $(M - M_0)/M_0$, where M_0 is the value of M obtained for $H = 12$.

6.4.2 Estimation of BF$_k$'s Execution Time

The implementation details of the first, second, and fourth phases of BF$_k$ are omitted in [35, 36], whose focus is the third phase. Designing an efficient implementation of the first phase is nontrivial because the back-off periods before every packet transmission must be chosen carefully. Making our own design would affect the validity of our conclusions because our design may be suboptimal. Instead, we derive an optimistic estimate of the execution time of the first phase and assume that this phase provides perfect information to the following phases. This section contains our derivation of the execution time of the first phase and describes our implementation of the second and fourth phases. We assume that the third phase is infinitely fast.

Here, but not in our simulations, we assume the existence of a constant transmission range r_t, which is equal to t, and a constant interference range r_i, which is equal to $2r_t$. BF$_k$'s first phase requires that every sensor node announces itself to all its one-hop neighbors by transmitting a packet that we call *identification packet*. For efficiency's sake, we assume that each node only transmits one identification packet. In order for this packet to be received successfully by all its neighbors, there must be no concurrent transmitters within their interference range. Therefore, the minimum distance between concurrent transmitters is $r_t + r_i = 3r_t$.

The maximum density of concurrent transmitters is achieved when the transmitters are deployed over a hexagonal grid as the one in Figure 3.4, with the difference that now the distance between transmitters is $3r_t$. This new grid has a transmitter density of one transmitter over a hexagon with side $1.5r_t$, which occupies an area $5.84r_t^2$ and contains 5.84ρ nodes. Since only one of these nodes can transmit at a time, the neighbor discovery phase lasts at least 5.84ρ time slots. This estimate is optimistic because in practice there are collisions and unused intervals during the first phase. In our simulations, the time slots used in both BF_k's neighbor discovery phase and DATP last for 20 ms.

BF_k's second phase serves to transmit neighbor information, and BF_k's fourth phase serves to transmit schedule information. The neighbor information consists of lists of node IDs, each of which occupies 16 bits. The schedule information contains time slot indices, each of which also occupies 16 bits. The packet size is 448 bits. Therefore, if every node has g neighbors, a node with d descendants in the routing tree has to transmit $\lceil 16(g+1)(d+1)/448 \rceil$ packets with neighbor information in the second phase. In addition, in the fourth phase, this node has to receive $\lceil 16 \cdot 2(d+1)/448 \rceil$ packets with schedule information if the compression coefficient γ is infinity.

The second and fourth phases are divided into pairs of time slots that are used to transmit neighbor and schedule information. The first slot in each slot pair has a duration of 30 ms, and is used to contend briefly and transmit a 448-bit data packet. This data packet can be used to transmit neighbor or schedule information. The second slot in each pair has a duration of 15 ms and is used to contend briefly and to transmit an acknowledgment packet in response to the preceding packet.

Every node X seeking to transmit a packet to a node Y keeps track of the number n_f of consecutive times that it failed to receive an acknowledgment in response to a

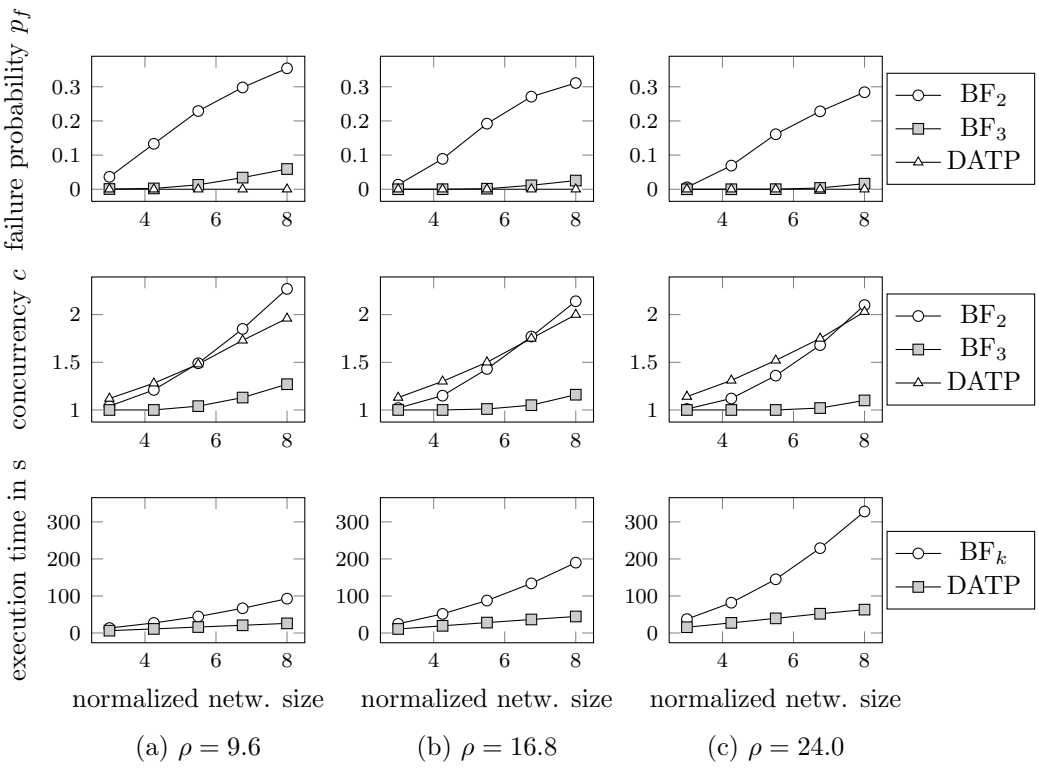
packet. During the first slot of a slot pair, X contends with probability 2^{-z}, where z is the minimum of n_f and a network-wide constant that we denote by n_m. We performed simulations to obtain the n_f that minimizes the duration of the initial scheduling phase; the optimum was $n_f = 4$. Contending consists in backing off for a random time and transmitting a packet if at the end of the back-off period the channel is idle. If Y receives X's packet, it backs off for random period, and only replies with an acknowledgment if at the end of the back-off period the channel is idle.

6.4.3 The Three Performance Metrics

Figure 6.3 is a matrix of graphs where each matrix row shows one of the three performance metrics, and each matrix column refers to a different node density ρ.

The first row of Figure 6.3 shows that the failure probability p_f of BF_k increases with the normalized network side $\bar{x}$. This is because in small networks most nodes are near the border of the monitored area and thus are surrounded by few neighbors and few interferers. The first row of Figure 6.3 also shows that the failure probability p_f decreases with the node density ρ. This is because, as ρ increases, the number of hops between nodes becomes more correlated with their physical distance and because the number of conditions used by BF_k to determine whether two concurrent transmissions interfere with each other increases.

The first row of Figure 6.3 also highlights the high failure probability p_f suffered by BF_2 and BF_3, particularly in networks that are large and sparse. For $\rho = 9.6$ and $\bar{x} = 8$, p_f is 0.35 in BF_2, which is clearly intolerable. BF_3 performs much better, obtaining $p_f = 0.06$ for the same parameters. However, even $p_f = 0.06$ may be too high, particularly in networks with many hops. For example, if a node lies 10 hops away from the data sink, its packets reach the data sink with probability



(a) $\rho = 9.6$ (b) $\rho = 16.8$ (c) $\rho = 24.0$

Figure 6.3: Scalability with both the normalized network side $\bar{x}$ and the node density ρ.

$(1 - p_f)^{10} = 0.53$. By contrast, DATP achieves $p_f = 0$ because it only assigns the same DBs to nodes that have been proved empirically to tolerate each other's interference.

The second row of Figure 6.3 shows that DATP achieves as much concurrency as BF_2 and much more than BF_3. In small networks, the highest possible concurrency is 1 because no two nodes are far enough from each other to share a time slot. In larger networks, higher concurrency is possible and BF_3 obtains a much lower concurrency than BF_2 does because it imposes a larger distance between concurrent transmitters.

The third row of Figure 6.3 shows that DATP is much faster than BF_k, particularly for a large node density ρ and a large network side $\bar{x}$. This is because DATP is distributed and BF_k is centralized. The execution time of BF_k depends on the data volume to be transmitted in the second phase. This data volume increases with ρ and $\bar{x}$, but is independent of k. Therefore, the execution time is also independent of k.

6.5 Conclusion

This chapter proposes and evaluates DATP, which is an extension of EATP (cf. Chapter 3) for a network using repeatable data aggregation over a routing tree. The schedule obtained by DATP achieves low upstream latency because it verifies the precedence property for data aggregation.

DATP obtains zero failure probability p_f in static networks because only nodes that are empirically proved to tolerate each other's interference are assigned the same DB. The failure probability of BF_2 is intolerably high and that BF_2 is still significant. In terms of concurrency, DATP greatly outperforms BF_3 and performs similarly to BF_2. Furthermore, DATP is much faster than BF_k because it operates distributedly.

7 Algorithms for Repeatable Aggregation in Networks with Unreliable Links

In order to perform repeatable data aggregation in a network with *reliable* wireless links, our proposed network operation consists of obtaining the routing tree with FAT (cf. Chapter 5), obtaining a schedule using DATP (cf. Chapter 6), and executing the data transmission phase. During the data transmission phase, every node knows exactly when it will receive each packet because a schedule is used and links are reliable.

This chapter considers the use of a TDMA schedule in a network in which the wireless links are *unreliable*, and a tree has been obtained with FAT and a schedule has been obtained with EATP (cf. Chapter 3). DATP is unsuitable because the precedence property barely reduces the latency when the links are unreliable. In our considered scenario, the sensor nodes do not know when they will receive a packet if at all, and thus they need to decide for how long to wait for each packet. If they wait for a long time, they are very likely to receive all the packets and thus to be able to aggregate them, but the data sink receives the packets very late. This chapter proposes several timing and packet discard policies for data aggregation.

7.1 System Model

7.1.1 Link Model

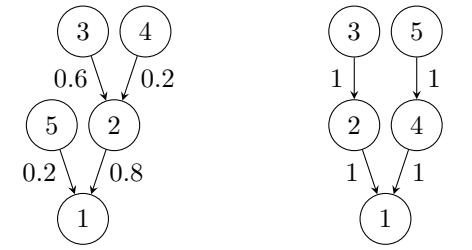
Every wireless link has a different transmission success probability p_s that remains constant over time. Figure 7.1 shows two simple networks whose links are characterized by p_s. An example of an application of this model is a WSN monitoring the vibrations of the lining of a tunnel. The sensor nodes are stationary, but their links are blocked intermittently by the vehicles traversing the tunnel. The success of any given transmission is unpredictable because it depends on the position of the vehicles much more than on the modulation scheme or transmit power. If the vehicular traffic is ergodic, the success probability is constant over time. Furthermore, if the vehicular traffic is fast, the success probability of two consecutive transmissions is uncorrelated.

7.1.2 Data Generation and Aggregation Model

We assume that time is divided in reporting intervals, which are labeled R_1, R_2 and so on. Every sensor node in the routing tree is a data source that generates one packet per reporting interval. The interval between the start of two consecutive reporting intervals is T_r. We use the same aggregation model as in Chapter 6: any number of packets from the same reporting interval can be compressed into a single DATA packet, and every DATA packet only provides information about only one reporting interval.

7.1.3 Data Requirements: the Node Count

Every packet contains the information from a certain number of nodes. This number is referred as the *node count* of the packet. Similarly, the number of nodes whose



(a) Network 1 (b) Network 2

Figure 7.1: Two sample networks. The number next to the links represents the link success probability p^s.

information about reporting interval R_i reaches the data sink is referred as the node count of R_i and is denoted by c_i. It is computed as the sum of the node counts of all the packets that the data sink receives about R_i, unless some packets are repeated or a sensor node contributes information about R_i in two different packets.

We impose the *node count requirement*, which states that the probability that any c_i is smaller than a certain threshold γ should be smaller than a certain probability p_0:

$$p(c_i < \gamma) < p_0 \tag{7.1}$$

When selecting γ and p_0, it has to be considered that increasing γ or reducing p_0 increases the amount of data received at the data sink, but increases the energy consumption.

The node count requirement is designed for networks in which the measurements from the sensor nodes are unreliable, and in which the information from many sensor nodes is needed to provide reliability. The node count requirement assumes that the measurements from all the nodes have equal value, and thus is inappropriate if some sensor nodes sense much stronger measurements than other nodes.

7.2 Problem Formulation

We assume that the transmit power P_t is much larger than the receive power P_r, which is typical if the sensor nodes are far away from each other. This assumption makes it easier to balance the energy consumption of different sensor nodes because the energy consumption of a node becomes independent of its number of children.

The problem is to find a scheme to decide which packets to transmit and when to do it, and which packets to discard and when to do it. The scheme must satisfy the node count requirement and minimize the maximum energy consumption in the network.

7.3 Existing Solutions

The multi-path aggregation approaches [56, 57, 58] transmit duplicates of the same information across multiple paths. Therefore, they are very robust to unreliable links, but consume excessive energy.

TAG [48] and Cascading Timeouts [49] use contention-based MAC and allocate a certain interval for the transmissions and retransmissions of packets about a reporting interval R_i. They assign transmission intervals to nodes based on their hop distance to the data sink. The intervals associated with different reporting intervals are non-overlapping. These protocols suffer frequent collisions because they use contention access. They also use bandwidth inefficiently because of their stringent restrictions on when each packet can be retransmitted. Furthermore, they fail to balance the energy consumption of different nodes because the nodes with the weakest links have to make many retransmissions.

In networks with unreliable links, TDMA protocols such as FlexiTP [27] and EATP (Chapter 6) avoid many packet collisions, but cannot prevent packet losses due to

quickly changing links. With these protocols, it is slow and inefficient to claim new DBs every time that a packet is lost and a retransmission is needed. Therefore, the sensor nodes should claim in advance the DBs they expect to need for retransmissions based on the quality of their links with their parents.

A simple strategy to select the number of DBs that are assigned to a node is to select as many DBs as it needs for all its packet transmissions and retransmissions. In this strategy, if a node generates q packets every T_r and its link with its parent has success probability p_s, the node is assigned at least q/p_s DBs every T_r. However, the nodes with poor links consume a lot of energy. Furthermore, energy is wasted in transmitting more data than necessary if the number of data sources is bigger than γ.

The *sensor-selection approach* consists in selecting γ data sources, which it calls *selected data sources*. The selected data sources transmit their data from every reporting interval to the data sink. The rest of the data sources do not need to transmit any data to the data sink, thereby saving energy.

Numerous sensor-selection protocols [97, 98, 99, 97, 100, 101] assume that each node provides information about a certain area, and aim to select the smallest number of data sources that cover all the areas of interest. This model assumes that the nodes provide reliable measurements, whereas our node-count requirement is designed to solve the problem of noisy or faulty measurements.

Most of the sensor-selection protocols neglect the energy consumed by the relays of the selected nodes. An exception is DPA [99], but DPA does not consider the link success probability p_s, and thus selects nodes with low p_s as often as nodes with high p_s. Therefore, the nodes with low p_s consume a lot of energy because they have to make many retransmissions.

MC-MIP [101] selects multiple sets of disjoint data sources. Each of these sets

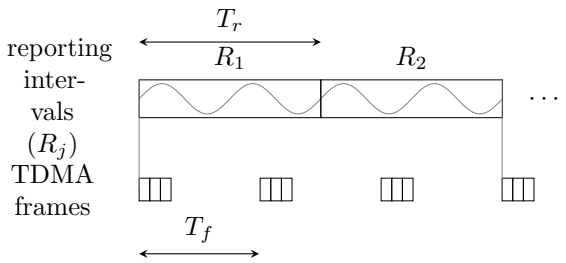


Figure 7.2: Each node generates one packet with period T_r and is allocated one transmission attempt every T_f. Here, $\Gamma = T_f/T_r = 2/3$.

provides information about the event at different times. MC-MIP multiplies the network lifetime by the number of disjoint sets when compared to the approach of making every node transmit data about every reporting interval. However, when different nodes have different p_s, the nodes with weak links suffer a high consumption.

7.4 ODF Approach

We propose the One-DB-per-Frame (ODF) approach, which consists in using TDMA and assigning one DB per node and per frame regardless of the link success probability p_s. The number of DBs assigned per unit time depends on the TDMA frame period T_f, which is different from the duration T_r of the reporting intervals. These two parameters are shown Figure 7.2, and we define the *normalized frame period* Γ as their ratio: $\Gamma = T_f/T_r$. In order to save energy, T_f should be large, and thus the largest Γ that ensures the satisfaction of the node count requirement should be used.

Every node listens in the DBs where it is scheduled to receive, which are as many as its number of children. It transmits in the DBs where it is scheduled to transmit, which is exactly one. It sleeps in any other DB. Since every node transmits the same number of times per frame, it consumes the same amount of energy under the assumption $P_t \gg P_r$. This is the way in which ODF balances the energy consumption.

Every sensor node has a packet buffer where it keeps the packets that it has to transmit. Each packet contains information about a certain reporting interval and has a certain node count. If a node with a packet from reporting interval R_i receives a packet about R_i, the node aggregates the two packets. The node count of the resulting packet is the sum of the node counts of the input packets. Every sensor node's packet buffer is finite. A node's packet buffer is very likely to overflow if the node's link success probability p_s is smaller than the normalized frame period Γ. If the buffer overflows, the choice of packets to be discarded is important.

A small packet buffer is beneficial in two ways. First, if the oldest packets are discarded when the buffer is full, the newest packets are transmitted earlier and the latency is reduced. Second, it balances the energy consumption among different nodes because it prevents the buffers growing to a very long size. Such growth would cause the buffers' owners to transmit packets long after other nodes have finished reporting the event, and thus consume more energy than those nodes.

With ODF, most packets from nodes with a high p_s reach the data sink, whereas most packets from nodes with a low p_s end up being discarded due to packet overflows. Therefore, nodes with high p_s contribute more towards the node count than nodes with low p_s. This is desirable because, if all nodes contributed equally towards the node count requirement, the nodes with the weakest links would consume more energy than the rest.

The difference of ODF with the sensor selection protocols can be seen by considering the operation during a period of over a dozen reporting intervals. In ODF, within this period, the data sink is very likely to obtain information from every node at least once. By contrast, with sensor-selection approach, the data sink does not receive any data from the unselected data sources.

In the next two subsections, we present two protocols based on the ODF approach:

U-ODFP and P-ODFP. They differ in the packets that the nodes select to transmit and discard. They also differ in the way they choose the normalized frame period Γ.

7.4.1 U-ODFP

U-ODFP stands for Unplanned, One-DB-per-Frame Protocol. The protocol is said to be unplanned because it does not decide in advance the nodes that provide information about each reporting interval. U-ODFP is executed during the data transmission phase, after a routing tree has been obtained and each node has been assigned exactly one DB.

7.4.1.1 Discard and Select Policies

We distinguish two versions of U-ODFP. The first version is called *buffer-constrained* because it discards packets when the buffer is going to overflow. The second version is called *time-constrained* because it discards packets from reporting intervals older than a certain threshold. In both versions, the packet that a node selects to discard is the packet from the oldest reporting interval, and the packet that a node selects to transmit is the packet from the most recent reporting interval.

7.4.1.2 Selection of the Normalized Frame Period Γ

Increasing the normalized frame period Γ reduces the number of TDMA frames. Therefore, it reduces the energy consumed in listening for packets from their children. Furthermore, a protocol that can operate at a high Γ can increase the number of DBs per TDMA frame and accommodate more transmissions. However, Γ should be sufficiently small to provide sufficient transmission opportunities to satisfy the node count requirement. Therefore, it is desirable to choose the largest Γ that satisfies the node count requirement.

U-ODFP selects Γ iteratively as follows. Initially, the data sink commands the sensor nodes to use a certain Γ. If the information that it receives is insufficient to satisfy the node count requirement, the data sink commands all the nodes in the network to use a lower Γ. If, on the other hand, the data sink receives much more information than it needs, it commands the sensor nodes to use a higher Γ in order to save energy.

7.4.1.3 Limitations

U-ODFP does not aggregate as much data as possible. For example, a node X may transmit a packet about reporting interval R_i and later receive a packet about R_i. This is suboptimal because if X had waited for longer, it could have aggregated two packets and saved a transmission. Furthermore, U-ODFP suffers *wasted transmissions*, which are successful packet transmissions that provide no benefit to the data sink because the packet is discarded at a later time before it reaches the data sink.

7.4.2 P-ODFP

P-ODFP stands for Planned One-DB-per-Frame Protocol. The protocol is said to be *planned* because the data sink decides in advance which data sources provide information about which reporting intervals. This plan aims to reduce the number of wasted transmissions. The plan is represented through W lists referred to as *source lists* and denoted by $S_1, S_2, \ldots,$ and S_W. List S_i contains the nodes whose information about reporting interval R_i should reach the data sink. A sensor node X discards any packet about reporting interval R_i if $X \notin S_i$.

The sensor lists are periodical with period W, which means that S_i is equal to S_{i+W}. This implies that the data sources of reporting intervals R_i and R_{i+W} are the same. Therefore, it is sufficient to specify W lists to provide sufficient information

about the contributors of every reporting interval.

The protocol is designed to be used at specific normalized frame period: $\Gamma^* = W/Q =$, where Q is a global parameter that in our simulations takes the value of 10. This means that in every Q TDMA frames the network provides information about W reporting intervals. This is illustrated in Figure 7.3. For the data sink, the problem is, given the link success probabilities and Q, to find the largest possible set of source lists $\{S_1, S_2, \ldots, S_W\}$ that verifies the properties indicated in the following section.

7.4.3 Properties of the Source Lists

The source lists should verify the following properties. First, every source list should contain at least γ elements in order to satisfy the node count requirement. Second, a node i should not appear in more than q_i of the lists $S_1, \ldots, S_W$. Here, q_i is the largest integer smaller than $p_s^i Q$, and p_s^i is the probability that Node i's transmissions to its parent succeeds. This property reduces the probability of unplanned packet discards by ensuring that the nodes do not plan to transmit more packets than their links can support. Third, if a node is an element of S_j, so should all its ancestors, except the data sink. This property ensures that the packets generated by a node are relayed to the data sink. Figure 7.3 shows possible source lists for the network in Figure 7.1a.

7.4.4 Computation of the Source Lists

Figure 7.4 presents our algorithm to obtain the source lists $\{S_i\}$. It also lists the inputs of the algorithm. The algorithm is executed at the data sink and requires knowledge of the network. Each iteration of the loop starting in Line 1 fills one of the S_i lists with exactly γ elements. The loop concludes when it cannot fill any

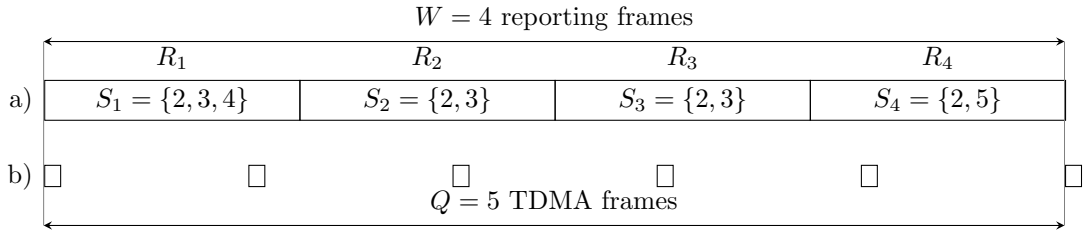


Figure 7.3: Operation of Network 1 of Figure 7.2 with P-ODFP at the optimal normalized frame period: $\Gamma^* = W/Q = 4/5 = 0.8$. *a)* Reporting intervals and source lists. *b)* TDMA frames.

more lists. In each iteration of the loop starting in Line 3, different lists that are candidates to become S_i are added to a list U. In Line 17, one of these lists is selected and assigned to S_i.

Each list represents a tree rooted at the data sink that is a subset of the routing tree and whose elements are the nodes in the list. The selected candidate is the list that represents the tree with the greatest depth, where the definition of *depth* depends on the kind of tree. For a tree with at least one node with multiple children, the depth is the minimum hop-distance to the data sink of the node with multiple children that lies the closest to the data sink. The depth of the tree is zero if the data sink has multiple children in this tree. For a tree without nodes with multiple children, the depth is the number of elements in the tree.

In order to illustrate why deep trees are preferable to shallow trees, consider the network in Figure 7.1b. Suppose $Q = 1$ and $\gamma = 2$, which yields $q_2 = q_3 = q_4 = q_5 = 1$. If S_1 is assigned the tree represented by $\{2,\ 4\}$, which has depth 0, it becomes impossible to find sufficient nodes for S_2 and thus the value of W is only 1. By contrast, if S_1 is assigned the list $\{2, 3\}$, which has depth 2, we can set $S_2 = \{4, 5\}$, and thus we obtain $W = 2$, which is the highest possible value for this network. This example is also interesting because, when selecting $S_1 = \{2, 3\}$ and $S_2 = \{4, 5\}$, the network in Figure 7.1b operates at a normalized frame period $\Gamma = W/Q = 2/1 = 2$.

Data: 1) A positive integer γ indicating the minimum number of data sources per reporting interval. 2) The number Q of TDMA frames within which the reporting needs to take place. 3) A list $\mathcal{L}$ with the indices of the nodes. 3) The children of every node. 4) The ancestors of every node. 5) The q of every node (for node i, $q_i = \lceil p_s^i Q \rceil$).

Result: The $S_1, \ldots, S_W$, where each list verifies the properties in Section 7.4.3 and W is desired to be large.

```
1  foreach W ∈ {1, 2, ...} do
2      U = new empty list of lists of integers;
3      foreach i ∈ L do
4          V = new list of integers containing i and its ancestors (in this order);
5          if q_j = 0 for any j ∈ V then
6              continue in Line 3;
7          Z = new stack containing the indices of the children of i;
8          while Z not empty and length(V) ≤ γ do
9              y = pop(Z);
10             if q_y > 0 then
11                 append y at the end of V;
12                 push(Z, x) for x ∈ children of y;
13         if cardinality(V) ≥ k_m then
14             append V at the end of U;
15     if U is empty then
16         continue in line 19;
17     S_W = list in U whose first element is more hops away from the sink;
18     decrement q_j for each j ∈ S_W;
19 Z = new stack containing the children of the data sink;
20 while Z not empty do
21     x = pop(Z);
22     push(Z, y) for y ∈ children of x;
23     while q_x > 0 and ∃S_j such that x ∉ S_j do
24         add x to S_j with smallest cardinality that verifies x ∉ S_j and
           parent(x) ∈ S_j;
25         decrement q_x;
```

Figure 7.4: Algorithm to obtain the source lists in P-ODFP. The operations defined on a stack S are `push(S, x)` (add x to the top of S) and `pop(S)` (remove and return element at the top).

Therefore, the normalized frame period Γ can be larger than one.

After adding a node i to one of the S_j lists, the algorithm decrements the nodes' capacity q_i in Line 18. This is to ensure that each node is not assigned to transmit more packets than its link success probability supports.

The part of the algorithm starting at Line 20 is optional. Its purpose is to increase the node count of some of the reporting intervals over γ. This is not necessary with our problem formulation, but may be valued in some applications. In this part of the algorithm, the spare capacity q of the sensor nodes is allocated to the sensing intervals with the smallest node count without increasing W.

7.5 Performance Evaluation

The simulations in this sections are obtained with a custom simulator written in C# that can be found in [78]. Section 7.5.1 evaluates the ODF protocols in a very simple network. Section 7.5.2 evaluates the ODF protocols in many random networks and compares their energy efficiency with that of a simple protocol.

7.5.1 Simulation in Simple Network

In this section we simulate a simple network, namely that of Figure 7.1a. The simulation parameters are contained in Table 7.1 and are described as follows.

The network is simulated 500 times, each time for 2000 TDMA frames. The sensor nodes have buffers with capacity of 30 packets. These numbers are chosen so that during the simulated TDMA frames, the buffers of the sensor nodes have had a chance to overflow, a stationary status in the buffers has been achieved, and the effects of the start of the packet transmissions have been reduced significantly.

The minimum node count γ has to be at least 1, and it cannot be larger than the

parameter	value
∘ number of times that the network is simulated	500
∘ number of TDMA frames simulated per iteration	2000
∘ buffer capacity of the sensor nodes	30
∘ minimum number γ of nodes whose information must reach the data sink in every reporting interval	3
∘ number Q of TDMA frames within which the algorithm in Figure 7.4 manages to transmit information about W reporting intervals	10

Table 7.1: Parameters for simulation of small network in Section 7.5.1.

number of sensor nodes in the network excluding the data sink, which is four. In our simulations, we set $\gamma = 3$. In the computation of P-ODFP's source lists, we use $Q = 10$. This value of Q is sufficiently large so as not to give a much shorter Γ that can be achieved for larger Q, and is sufficiently small so as keep the execution time of the algorithm small. not to make The simulated protocols are P-ODFP and the two versions of U-ODFP. The time-constrained version of U-ODFP discards a packet about reporting interval R_i if the current reporting interval is R_j and $j > 15 + i$.

Figure 7.5 evaluates the three protocols as a function of the normalized frame Γ. The y-axis shows the probability that the node count is smaller than γ, which is desired to be small. Recall that from an energy perspective a large Γ is preferable, but it cannot be too large because (7.1) must be satisfied. Therefore, each protocol should operate at the highest Γ that satisfies (7.1). From Figure 7.5 it can be derived that, if $p_0 = 0.15$, buffer-constrained U-ODFP should operate at $\Gamma = 0.3$, time-constrained U-ODFP should operate at $\Gamma = 0.25$, and P-ODFP should operate at $\Gamma = 0.46$. Therefore, P-ODFP is the most energy efficient of the simulated protocols.

U-ODFP can be easily modified by changing its policies for selecting which packets to transmit and which packets to discard. For example, preference can be given to packets with high node count. Some of these policies sometimes significantly outperform the others, but their performance greatly depends on the specific network.

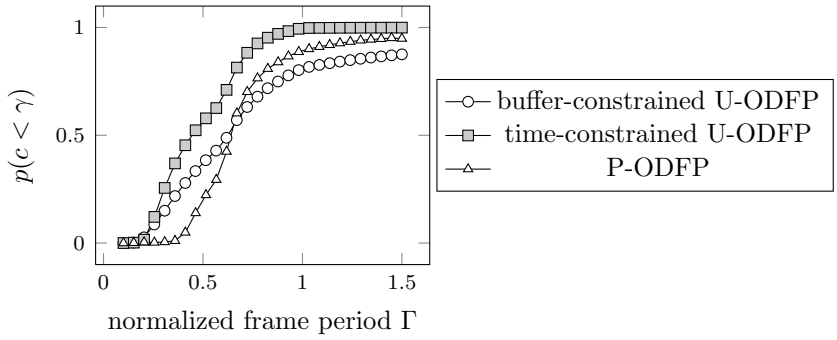


Figure 7.5: Probability that the node count of a reporting interval, denoted by c, is smaller than $\gamma = 2$ in the network of Figure 7.1b.

We performed experiments in a variety of simple networks. The packet discard policy that performed the best in most of these networks was to discard the oldest packet, which is the buffer-constrained policy of U-ODFP. In the next section, we use this discard policy in U-ODFP.

7.5.2 Benchmark: FS

As seen in Section 7.3, many of the existing protocols are not appropriate for our problem formulation. For example, the sensor selection algorithms take different inputs. In order to have a benchmark to evaluate the performance of our protocols, we propose a simple protocol called Full Schedule (FS). FS is meant to represent the trivial solution to the problem we have formulated.

FS uses a TDMA schedule for data aggregation. FS does not report a fraction p_0 of the packets because it is unnecessary according to (7.1). The nodes decide in advance which reporting intervals they will ignore. If a node's link success probability with its parent is p_s^i, the node is assigned $\lceil N_p/p_s^i \rceil$ DBs per TDMA frame, where N_p is a global parameter that is set to 8 in the simulations. This number of DBs allows every sensor node to transmit N_p packets per TDMA frame. The TDMA

frame period in FS is

$$T_f = N_p T_r (1 - p_0). \tag{7.2}$$

FS allows sufficient DBs for every sensor node to transmit all its packets, which avoids wasting energy in transmitting packets that end up being discarded. However, the data sink receives data about a fraction $1 - p_0$ of the reporting intervals from all the data sources, whereas according to (7.1) the data from only γ sources is sufficient. The ODF protocols save energy by transmitting data from fewer nodes while satisfying (7.1).

7.5.3 Simulation Parameters

The simulation parameters are presented in Table 7.2 and Table 7.3 and we discuss them as follows. All the results are the average of 100 random deployments. The simulated area is a square with side $3t$, where t is the transmission range of the sensor nodes. The node density ρ is the average number of neighbors per node. We set $\rho = 9$, which yields a total of 25 nodes. We choose a node density ρ larger than 6 because a smaller value is likely to lead to a disconnected network. The values of the network size and ρ are chosen to give a network that is sufficiently large to study the performance in large networks, and small enough to keep a short execution time of the simulator. In another set of simulations, we set the node density to values between 8 and 24 in order to explore the effect of high node densities. For the highest node density, the number of nodes is 69 nodes.

A random deployment is discarded if more than 10 % of the nodes have no neighbors within their transmission range.

The link success probabilities p_s of different nodes are independent random variables uniformly distributed between 0.5 and 1. We set values of p_s larger than 0.5 because if p_s is smaller than 0.5 the use of the link consumes too much energy and bandwidth.

parameter	value
◦ default node density ρ	9
◦ range of values of the node density ρ	8–24
◦ transmission range (in this chapter, it exists and is shared by all nodes)	40 m
◦ range of values of the link success probability p_s (values taken from a uniform distribution with this range)	0.5–1
◦ length of each side of the monitored area (the monitored area is a square)	120 m
◦ fraction of disconnected nodes that, if exceeded, leads to discarding a deployment of sensor nodes and generating a new deployment	0.1
◦ default number L of nodes in the network	25
◦ range of values of the number L of nodes	22–69
◦ number of different networks simulated	100
◦ number of simulation periods per network	5
◦ duration of all data slots	20 ms
◦ duration of all ACK slots	4 ms
◦ power consumption in sleep mode	60 μW
◦ power consumption in receive mode	63 mW
◦ power consumption in idle mode	63 mW
◦ default value of the power consumption in transmit mode	630 mW
◦ range of values of the power consumption in transmit power	63–2508 mW

Table 7.2: Some of the parameters used in the simulation of random networks.

parameter	value
∘ number N_p of packets from every node that FS transmits per frame	8
∘ number of TDMA frames per simulation period in FS	1600
∘ number of TDMA frames per simulation period in ODF	5000
∘ default value of the maximum number of slots per TDMA frame in FS	200
∘ default value of the maximum number of slots per TDMA frame in ODF	25
∘ range of values of the maximum number of slots per TDMA frame in FS	184–552
∘ range of values of the maximum number of slots per TDMA frame in ODF	23–69
∘ maximum fraction p_0 of the reporting intervals that can be insufficiently reported to the data sink (c.f. Eq. 7.1)	0.15
∘ default value of the fraction ψ of nodes whose information must reach the data sink (c.f. 7.3)	0.4
∘ range of values of ψ	0.13–0.83
∘ number of packets that a node can store before having to discard packets	30
∘ number Q of TDMA frames within which the algorithm in Figure 7.4 manages to transmit information about W reporting intervals	20
∘ initial value of the normalized frame period Γ	0.1
∘ factor by which the normalized frame period Γ is multiplied in each iteration of U-ODFP	1.05

Table 7.3: Other parameters used in the simulation of random networks.

Each wireless links is characterized completely by p_s, which means that there are no other causes of packet losses. We use a default transmit power that is 10 times larger than the receive power. Such a large quotient occurs when the distance between transmitter and receiver is much larger than 100 m. In a set of simulations, we change the value of P_t/P_r between 1 and 63. We do not set a value smaller than 1 because the transmit power and the receive power are typically of the same order, although values of P_t/P_r as small as 0.7 might still be sensible [7]. We do not set values of P_t/P_r larger than 63 because that would require very distant nodes which would separate us even further from the properties of most real wireless sensor networks.

transmit power between

Every sensor node has a buffer with a capacity of 50 packets. Smaller buffers yield a higher risk that the nodes do not have any data to transmit in their allocated slot. Larger buffers would increase the upstream latency and require the simulation of more TDMA frames in order to reach a stationary status.

Each network is simulated in 5 simulation periods. We simulate multiple short simulation periods instead of a single long simulation period in order to reduce the effect of the starting point of the simulation. In each simulation period, 5000 TDMA frames are simulated in the case of our two protocols, and 1600 TDMA frames in the case of FS. The reason for this difference is that TDMA frames are longer in FS than they are in the ODF protocols, which only assign one DB per TDMA frame. This does not affect the validity of this section because all the energy consumption is normalized by the number of reporting intervals. After each simulation period, a new routing tree is constructed. In the construction of the new tree, each node sets its neighbor in the next tier (closer to the data sink) with the lowest energy consumption as its parent node. By periodically changing the tree, the energy consumption of the sensor nodes is more evenly distributed. If these periodic changes are not made, the

advantage of ODF over FS in terms of having a lower maximum energy consumption is even larger.

The normalized frame period Γ in U-ODFP is chosen as follows. The initial value of Γ is 0.1. Then, the protocol is executed and the data sink computes $p(c_i < \gamma)$. If this probability is smaller than p_0, the data sink multiplies Γ by 1.05 and the network is simulated again. This process is repeated until $p(c_i < \gamma)$ exceeds p_0. When this happens, the data sink sets Γ to the last value of γ that satisfied (7.1). In P-ODFP, Γ is set to $\Gamma^* = W/Q$, which our simulations proved to verify (7.1). The parameters of the fidelity requirement in (7.1) are p_0 and γ. We define ψ by

$$\gamma = \psi L, \tag{7.3}$$

where L is the number of nodes in the network. Our default parameters are $p_0 = 0.15$ and $\psi = 0.4$. In a set of simulations, we explore the influence of ψ by variating its value between 0.13 and 0.83. We do not choose values of ψ smaller than 0.13 because then the number of sources would be very small and these sources would be concentrated near the data sink, which would not allow us to see the operation of the protocol in large networks. We do not choose values of ψ larger than 0.83 because then the best policy would be to simply transmit packets from every sensor node.

7.5.4 Simulation Results

Figure 7.6 presents in a matrix the reduction in the energy consumption of the ODF protocols with respect to FS. The first row Figure 7.6 shows the reduction in the mean energy consumption, and the second row shows the reduction of the maximum energy consumption. The maximum is more important than the mean because typically the lifetime xoof the network is determined by the most energy consuming nodes in the network. Each column of the matrix studies the influence of

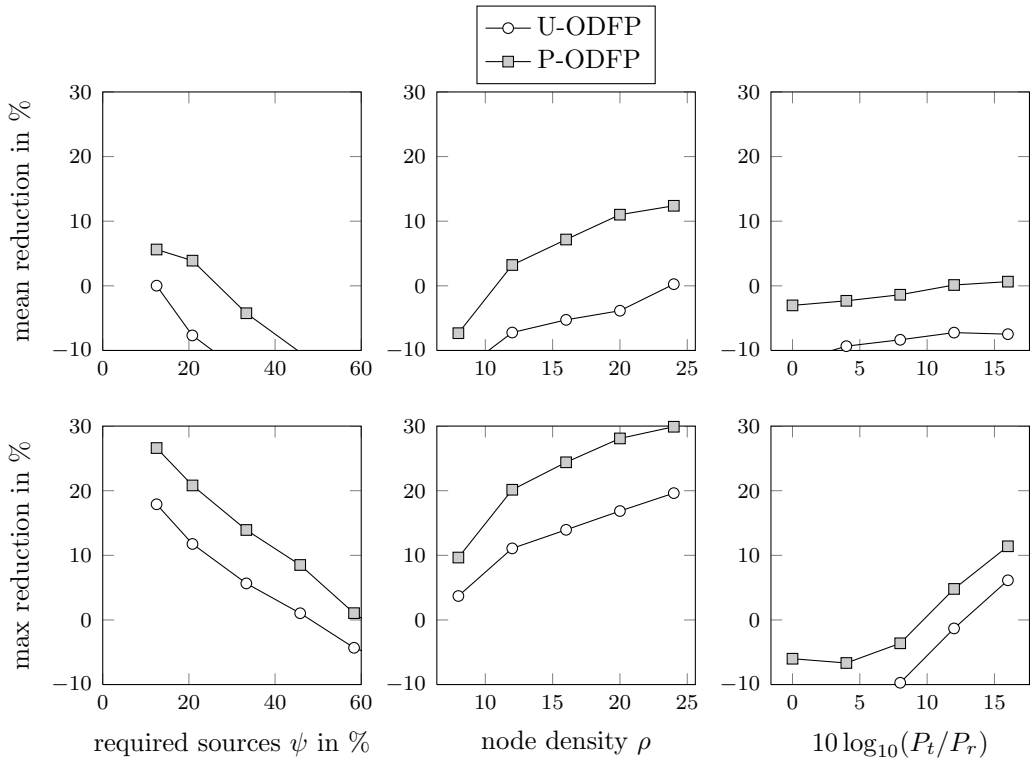


Figure 7.6: Improvement of the ODF protocols over FS.

a different parameter.

The first column of Figure 7.6 analyzes the variation of the energy gain with the fraction of required data sources ψ. For big values of ψ, FS outperforms the ODF protocols because it aggregates many packets, discards no packets, and transmits little unnecessary information. However, for small ψ, FS wastes energy by obtaining unnecessarily high node counts. In this case, P-ODFP's maximum energy consumption is a 27 % lower than FS's.

The second column of Figure 7.6 shows that energy gain over FS grows with the node density ρ, which can be explained as follows. All the data that reaches the data sink goes through the children of the data sink. In sparse networks, these children are few, and thus the probability that all those children have poor links is high. In order to satisfy (7.1), the whole network needs to operate at a low frame period Γ,

which consumes a lot of energy. By contrast, in dense networks, the data sink has a large number of children, and thus it is more likely that at least some of them have good links. If these nodes and their descendants have sufficient information to satisfy (7.1), the whole network can operate at a high Γ and thus the energy consumption is low.

The third column of Figure 7.6 shows that the energy gain grows with $Q = 10\log(P_t/P_r)$. A value of Q of 0 indicates that $P_t = P_r$, and a large Q indicates $P_t \gg P_r$. It can be seen that P-ODFP only outperform FS if $P_t > 10P_r$, and that U-ODFP needs even a greater Q to outperform FS. This dependence on Q is natural because the ODF protocols are designed to balance the energy consumption by balancing the transmission energy, but they neglect the reception energy because they assume $P_t \gg P_r$.

7.6 Conclusions

In a network with unreliable links, the ODF protocols select which packets to discard and which packets to transmit in order to balance the energy consumption of the sensor nodes while providing sufficient information to the data sink. They prolong the network operational lifetime by up to 26 % when compared to FS, which is essentially a TDMA protocol that decides to transmit the packets from every sensor node. The exact gain depends on several parameters, and for some sets of parameters the gain becomes a loss. The gain of the ODF is the biggest if the minimum node count γ is small, the node density is high, and the transmit power is much larger than the receive power. The gain of the two ODF protocols over FS is bigger for the maximum energy consumption than for the mean energy consumption, which indicates that the ODF approach is successful at balancing the energy consumption.

8 Conclusions and Discussion

8.1 Conclusions

This thesis proposes protocols for periodic data gathering in wireless sensor networks using TDMA and data aggregation. TDMA is efficient for periodic transmissions, and data aggregation can reduce the amount of data to be transmitted if neighboring sensor nodes generate correlated information.

Chapters are organized according to the level of data aggregation that they assume, from no aggregation to intense data aggregation. Chapter 3 assumes no data aggregation, Chapter 4 assumes unrepeatable aggregation, and chapters 5 to 7 assume repeatable aggregation. This order is typically used when developing monitoring solutions iteratively. Initially, exhaustive measurements are collected to discover the properties of the monitored signals and their significance. Then, the system is refined by developing increasingly stronger compression functions.

Chapter 3 presents a TDMA protocol called EATP that is designed and evaluated for networks without data aggregation, but can be adapted for data aggregation over different topologies. For example, it can be used in a clustered topology such as the one in Chapter 4, or in tree-based topologies as discussed in Chapter 6. EATP provides two scheduling methods: one for the initial scheduling phase, and another one for the data transmission phase. During the initial scheduling phase, EATP is faster and likelier to obtain a feasible schedule than FlexiTP is. During the data

transmission phase, EATP is more energy efficient than FlexiTP.

We show that EATP outperforms FlexiTP simultaneously in terms of energy consumption and scheduling delay. It is necessary to compare these variables at the same time because it is possible to save energy at the price of increasing the delay. EATP occasionally needs up to 15 TDMA frames to assign a DB and requires that the schedule does not change very quickly. The main contribution of EATP is to show a way in which the sensor nodes can obtain transmission slots efficiently and distributedly without knowledge of their neighbors' schedule. EATP is the first protocol to operate in this way to the best of our knowledge.

Chapter 4 considers the first level of data aggregation of this thesis: unrepeatable aggregation. It assumes a network divided in clusters where data aggregation occurs in the cluster heads and only once. It formulates the problem of dividing large networks in clusters of sizes whose sizes vary with their distances to the data sink. It proposes a scalable algorithm to approximate the solution to this problem. The algorithm yields up to a 12 % network lifetime improvement over other the solution in which all the clusters have equal size. We also identify the circumstances where this improvement is the highest.

Chapters 5, 6 and 7 study the different phases of aggregating data over routing trees. First, Chapter 5 constructs the routing tree quickly. Second, Chapter 6 obtains a transmission schedule quickly. Third, Chapter 7 decides which packets to discard when following the TDMA schedule in a network with unreliable links.

Chapter 5 proposes the FAT protocol, which is executed during the quiet phase and the initial routing phase. FAT is faster and less energy-consuming than comparable protocols if the network changes slowly, consists of many hops, and has to report infrequent and short events. The disadvantage of FAT is that it obtains a more inefficient tree than some centralized protocols, particularly if the data aggregation

function is very strong.

Chapter 6 proposes DATP, a modification of EATP for networks with stringent delay constraints and high compression capability. These networks require a special ordering of the DBs that makes modification of the schedule difficult and thus we suppress the modification ability. Therefore, obtaining low failure probability in the initial schedule quickly becomes paramount. We show DATP outperforms a centralized protocol in terms of failure probability, concurrency, and execution time.

Chapter 7 proposes U-ODFP and P-ODFP, which are two protocols for handling packet losses in unreliable networks that use TDMA. Their goal is to select which packets to transmit, which packets to discard, and the timing of these actions. They differ from the existing protocols in that they consider different links have different but constant success probability. Our protocols obtain up to a 26 % lifetime increase when compared to a naive solution.

8.2 Main Contributions

Our main contributions are the following:

- We propose EATP, the first TDMA scheduling protocol for WSNs in which the sensor nodes take scheduling decisions distributedly without knowledge of the obtained schedule. We show that this approach yields up to a 25 % reduction in the energy consumption when compared with a protocol operating at a higher latency. We also identify the circumstances when EATP is suitable, namely networks with infrequent schedule changes that can tolerate a low scheduling speed. Such networks are common in WSNs for periodic data gathering.

- We formulate the optimal cluster-size distribution problem and propose an algorithm to approximate its solution. Our formulation is the first to assume

multi-hop clusters of unequal sizes. Multihop clusters are useful to aggregate the data from many sensor nodes if the events are detected in large area. Unequal cluster sizes distribute the energy cost uniformly, and thus extend the network operational lifetime. Our approximation algorithm scales well with the network size, thereby allowing to quickly determine an upper bound on the benefit that be achieved by using clusters of unequal sizes. The highest benefit of unequal clusters is shown to be obtained when the nodes dedicate over 30 % of their energy in their internal traffic.

- We show through FAT the importance of jointly designing the quiet phase and the initial routing phase in order to minimize the energy consumption. FAT obtains the routing tree quickly by grouping the nodes in tiers and arranging their channel-check times. The fast tree construction enables the nodes to increase the interval between their clear channel assessment operations, thereby saving much energy. FAT constrains a node's potential parents to its set of neighbors lying one hop closer to the data sink. Our simulations show that this constraint hardly reduces the aggregation ability of the tree. We also highlight the importance of keeping the tiered structure updated in large networks.

- We propose DATP, an energy efficient protocol for low-latency repeatable aggregation over routing trees. DATP outperforms the existing protocols in terms of failure probability, concurrency and execution time. Like EATP, it enjoys low failure probability because it tests empirically the compatibility of concurrent transmitters.

- We propose U-ODFP and P-ODFP, two protocols that show a way to balance the energy consumption of different nodes in unreliable networks. They achieve it by assigning the same number of DBs to every node, which means that the

nodes with stronger nodes contribute more information than the nodes with weaker links. We show the importance of the choice of packets to transmit and to discard. In U-ODFP the nodes make this choice just before they need it, whereas in P-ODFP the nodes follow a plan that they receive from the data sink in advance. As a result, P-ODFP is more efficient in stationary networks than U-ODFP. U-ODFP extends the network lifetime by up to a 19 % when compared to the naive solution, and P-ODFP extends it by up to 27 %. However, U-ODFP is faster and distributed, and thus more suitable if the link success probabilities change frequently over time.

Theses contributions enhance our understanding on protocol design for periodic data gathering and collection. Our approach is cross-layer and systematically covers different aggregation models. It achieves important benefits in terms of delay, energy, concurrency, and satisfaction of data requirements.

8.3 Discussion and Limitations

One reason why we have confidence in our simulation results is that the simulator that we developed and used in Chapters 3 and 6 [78] is fairly realistic because it uses the signal to noise plus interference ratio to determine the success of a transmission. Furthermore, it assumes exponential attenuation and log-normal shadow fading. This realism is necessary in order to estimate the failure probability accurately.

We also trust our results because, having built it ourselves, we know exactly the assumptions they make. Because the simulator is written in a high level programming language and only considers the most relevant factors, the simulator code is short and easy to understand, which reduces the probability of bugs.

In order to validate our results, we have executed the simulators step by step in small networks and made sure it operated as required. Then, we gradually increased

the network size. We have also made sure the results make sense and that the main factors are modelled.

Our simulators take great care not to benefit our protocol when compared to the existing protocols. For example, where a part of the implementation of an existing protocol was missing in Chapter 3 or Chapter 6, we either considered an optimistic bound of its performance or benefited the existing protocol in other way. Futhermore, because the existing protocols and our proposed protocols share same code, if there are any failures in our model they are likely to affect all the protocols in the same way, so no advantage is given.

Some other factors that can be considered in the future for more detailed simulation include faulty nodes, asymmetric links, nodes with transceivers with different properties, synchronization errors, hacker attacks, or fast changing-networks (although our protocols are not designed for them). Our simulations operate at a packet level as opposed to at a more detailed bit level. Although not clear to us, perhaps some insights might be derived from a more detailed bit-level simulator.

The clustering algorithm in Chapter 4 can be executed for extremely large networks because its model is simple. However, this simplicity suggests that the energy gains presented in the chapter are optimistic. Therefore, they should be regarded as an upper bound of the gains that can be obtained.

Chapter 5 showed FAT's intolerance to outdated tier information, which implies that this information must be updated periodically. The cost of these updates needs to be quantified in order to determine the suitability of FAT in slowly-changing networks.

Chapter 7 is designed for the case where transmit power is much larger than the receive power. This assumption holds if the spacing between nodes is large, but not if the sensor nodes are close to each other. The speed and the adaptation cost of the

ODF protocols it is not evaluated.

8.4 Future Work

This thesis makes several assumptions about the data aggregation functions and the data requirements at the data sink. These assumptions are reasonable but need to be justified with examples of real applications. Currently, there is little detailed information of real applications, particularly applications with high data rate that use data aggregation [17]. Many more real applications need to be implemented and described. Some obstacles are the cost of large numbers of sensors, access to the target environment, and the long development and debugging time.

More real deployments are needed because they provide information such as which measurements are relevant, their meaning, and how much they can be compressed. The first step to gain this knowledge is to collect exhaustive measurements, for example using EATP (cf. Ch. 3). Then the aggregation functions need to be gradually refined. Due to the cost of this process, it is important to select applications with important economical value.

The importance of deploying real networks does not imply the unimportance of more theoretical studies. Theoretical studies are important because they identify the tradeoffs with different assumptions and because many applications have flexible performance requirements. For example, a latency requirement may be loosened if doing so saves much energy. Similarly, if it is proved that aggregation functions with a certain property yield great energy savings, functions with that property may be developed and used even if they discard some information.

The protocols in this thesis can be further validated with more detailed simulations. Possible elements to be incorporated include heterogeneous, faulty, or poorly synchronized nodes. Other interesting aspect is the robustness to hacker attacks.

Finally, implementation on actual hardware may raise new research problems.

The nodes in EATP (cf. Ch. 3) back off for a random time before transmitting their testing packets. The backoff algorithm can be improved to assign DBs faster and at a lower energy cost. In this improved algorithm, a node can consider the number of testing packets it transmitted, the number of testing packets it overheard, and its estimated number of neighbors. A node can obtain this estimate by overhearing its neighbors' transmissions during the TF.

When modeling the clustering problem (cf. Ch. 4), we made several approximations. The accuracy of these approximations can be quantified with simulations as a function of the node density and the inaccuracy of the location information. Additionally, it is desirable to propose algorithms to change the cluster heads periodically in order to balance the energy consumption. It is also interesting to propose new unequal clustering algorithms that can operate in more irregular areas and that do not require location information. Such algorithms should not assume that every node can communicate directly with data sink as UCS [89] and EEUC [90] do.

The work on FAT (cf. Ch. 5) can be extended by developing efficient ways to maintain the tiered structure. The analysis of the cost of this operation can be used to determine the maximum level of channel variability under which FAT outperforms other protocols. A related problem is to propose a model of channel variability that is useful for many WSNs.

The scheduling algorithm for data aggregation (cf. Ch. 6) can be extended by providing a way to adapt the schedule during the data transmission phase. This can be facilitated by reserving some DBs in the schedule computed during the initial scheduling phase. Increasing the number of reserved slots improves the adaptivity of the schedule but worsens its spectral efficiency. This tradeoff can be explored.

Chapters 6 and 7 assume that every number of packets from the same reporting

interval can be compressed into one packet with equal size to that of the input packets. It is interesting to study the case where the output has a different and must be split into multiple packets. If any of these packets is lost, the rest of the packets may become useless or unsuitable for aggregation.

The work in Chapter 7 can be extended in several ways. First, the selection of the constants used in the node count requirement can be investigated. Second, the influence of the buffer size in the ODF protocols can be studied. Third, the latency of different packet select and discard policies can be compared. Fourth, a scheme to balance the energy consumption when the receive and the transmit power are similar. This scheme would probably change the routing tree periodically and assign different number of DBs to different nodes.

References

[1] T. Arampatzis, J. Lygeros, and S. Manesis, "A survey of applications of wireless sensors and wireless sensor networks," in *Proc. IEEE Mediterranean Conf. Control and Automation*, Limassol, Cyprus, Jun. 2005, pp. 719–724.

[2] WINES Consortium, "Wired and Wireless Intelligent Networked Systems (WINES) – Smart Infrastructure Project," *[Online]. Available: www.winesinfrastructure.org*, 2008.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–114, 2002.

[4] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Elsevier J. Computer Networks*, vol. 51, no. 4, pp. 921–960, Mar. 2007.

[5] B. Sadler, "Fundamentals of energy-constrained sensor network systems," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 20, no. 8, pp. 17–35, 2005.

[6] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Elsevier J. Computer Networks*, vol. 52, no. 12, pp. 2292 – 2330, 2008.

[7] K. G. Langendoen, "Medium access control in wireless sensor networks," in *Medium Access Control in Wireless Networks, Volume II: Practice and Standards*, H. Wu and Y. Pan, Eds. Hauppage, New York: Nova Science Publishers, May 2008, pp. 535–560.

[8] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: new aggregation techniques for sensor networks," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys)*, Nov. 2004, pp. 239–249.

[9] T. He, B. M. Blum, J. A. Stankovic, and T. Abdelzaher, "AIDA: Adaptive application-independent data aggregation in wireless sensor networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 2, pp. 426–457, 2004.

[10] G. Manson, K. Worden, K. Holford, and R. Pullin, "Visualisation and dimension reduction of acoustic emission data for damage detection," *J. Intelligent Material Systems and Structures*, vol. 12, no. 8, pp. 529–, 2001.

[11] E. Cayirci, "Data aggregation and dilution by modulus addressing in wireless sensor networks," *IEEE Commun. Lett.*, vol. 7, no. 8, pp. 355–357, 2003.

[12] E. Cayirci and T. Coplu, "Distributed spatial data aggregation and dilution based on hashing and relational algebra in wireless sensor networks," in *Proc. IEEE Int'l Conf. Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Melbourne, Australia, Dec. 2004, pp. 373–379.

[13] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. D. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proc. ACM/IEEE Conf. Information Processing in Sensor Networks (IPSN)*, 2007, pp. 254–263.

[14] C. U. Grosse and F. Finck, "Quantitative evaluation of fracture processes in concrete using signal-based acoustic emission techniques," *Elsevier Journal on Cement and Concrete Composites*, vol. 28, no. 4, pp. 330–336, Apr. 2006.

[15] C. U. Grosse, M. Krüger, and C. Panagiotis, "Acoustic emission techniques using wireless sensor networks," in *Int'l Conf. Sustainable Bridges—Assessment for Future Traffic Demands and Longer Lives*, Wroclaw, Poland, Oct. 2007, pp. 191–200.

[16] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "Robust multi-hop time synchronization in wireless sensor networks," in *Proc. Int'l Conf. Wireless Networks (ICWN)*, Jun. 2004, pp. 454–460.

[17] B. Raman and K. Chebrolu, "Censor networks: a critique of "sensor networks" from a systems perspective," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 75–78, Jul. 2008.

[18] u. Crossbow Technology Inc., "Xmesh user's manual," *Crossbow, San Jose, CA*, 2007.

[19] H. Zhang, A. Arora, Y.-r. Choi, and M. G. Gouda, "Reliable bursty convergecast in wireless sensor networks," in *Proc. ACM Int'l Symp. Mobile Ad Hoc Networking and Computing*. ACM Press New York, NY, USA, 2005, pp. 266–276.

[20] J. L. Hill and D. Culler, "Mica: A wireless platform for deeply embedded networks," *IEEE Micro*, vol. 22, no. 6, pp. 12–24, 2002.

[21] T. van Dam and K. G. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys)*, Los Angeles, CA, USA, Nov. 2003, pp. 171–180.

[22] A. El-Hoiydi and J.-D. Decotignie, "WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks," in *Proc. IEEE Symp. Computers and Comm. (ISCC)*, vol. 1, 2004, pp. 244–251.

[23] J. Polastre, J. L. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys)*, 2004, pp. 95–107.

[24] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *Proc. IEEE Conf. Computer Comm. (INFOCOM)*, vol. 3, 2002, pp. 1567–1576.

[25] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An adaptive energy-efficient and low-latency MAC for tree-based data gathering in wireless sensor networks," *Wiley J. Wireless Comm. and Mobile Computing*, vol. 7, no. 7, pp. 863–875, 2007.

[26] S. S. Kulkarni and M. U. Arumugan, "TDMA service for sensor networks," in *Proc. IEEE Int'l Conf. Distributed Computing Systems*, vol. 1, Mar. 2004, pp. 244–251.

[27] W. L. Lee, A. Datta, and R. Cardell-Oliver, "FlexiTP: A flexible-schedule-based TDMA protocol for fault-tolerant and energy-efficient wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 6, pp. 851–864, Jun. 2008.

[28] I. Rhee, A. Warrier, M. Jeongki, and L. Xu, "DRAND: distributed randomized TDMA scheduling for wireless ad-hoc networks," in *Proc. ACM Int'l Symp. Mobile Ad Hoc Networking and Computing.* New York, NY, USA: ACM, 2006, pp. 190–201.

[29] G. Pei and C. Chien, "Low power TDMA in large wireless sensor networks," in *Proc. IEEE Military Comm. Conf.* , vol. 1, 2001, pp. 347–351.

[30] V. Rajendran, K. Obraczka, and J. García-Luna-Aceves, "Energy-efficient, collision-free medium access control for wireless sensor networks," *Springer J. Wireless Networks*, vol. 12, no. 1, pp. 63–78, Feb. 2006.

[31] V. Rajendran, J. García-Luna-Aceves, and K. Obraczka, "Energy-efficient, application-aware medium access for sensor networks," in *Proc. IEEE Int'l Conf. Mobile Ad-Hoc and Sensor Systems Conf. (MASS)*, Nov. 2005, pp. 630–637.

[32] S. Borbash and A. Ephremides, "Wireless link scheduling with power control and SINR constraints," *IEEE Trans. Inf. Theory*, vol. 52, no. 11, pp. 5106–5111, 2006.

[33] G. Brar, D. M. Blough, and P. Santi, "Computationally efficient scheduling with the physical interference model for throughput improvement in wireless mesh networks," in *Proc. ACM/IEEE Int'l Conf. Mobile Computing and Networking (Mobicom)*, Los Angeles, CA, USA, Sep. 2006, pp. 2–13.

[34] S. Kompella, H. Sherali, and A. Ephremides, "Optimal scheduling in interference limited fading wireless networks," in *Proc. IEEE Global Telecommunications Conf. (GLOBECOM)*, Honolulu, HI, USA, dec 2009, pp. 1–6.

[35] V. Annamalai and S. K. S. Gupta, "On tree-based convergecasting in wireless sensor networks," in *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC)*, S. Gupta, Ed., vol. 3, 2003, pp. 1942–1947.

[36] R. Mangharam, A. Rowe, and R. Rajkumar, "FireFly: a cross-layer platform for real-time embedded wireless networks," *Springer J. Real-Time Systems*, vol. 37, no. 3, pp. 183–231, Dec. 2007.

[37] S. Gandham, Z. Ying, and H. Qingfeng, "Distributed minimal time convergecast scheduling in wireless sensor networks," in *Proc. IEEE Int'l Conf. Distributed Computing Systems*, 2006, pp. 50–50.

[38] A. F. Harris III, R. Kravets, and I. Gupta, "Building trees based on aggregation efficiency in sensor networks," *Elsevier J. Ad Hoc Networks*, vol. 5, no. 8, pp. 1317–1328, Nov. 2007.

[39] L. van Hoesel and P. Havinga, "A lightweight medium access control (LMAC) for wireless sensor networks," in *Proc. Int'l Workshop Networked Sensing Systems (INSS)*, Jun. 2004, pp. 1–4.

[40] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Commun. Mag.*, vol. 14, no. 2, pp. 70–87, 2007.

[41] S. Arora and B. Barak, *Complexity Theory: A Modern Approach.* Cambridge University Press, 2009.

[42] M. Garey and J. D., *Computers and intractability.* San Francisco, CA: Freeman, 1979.

[43] B. Krishnamachari, D. Estrin, and S. Wicker, "The impact of data aggregation in wireless sensor networks," in *Proc. IEEE Int'l Conf. Distributed Computing Systems Workshops*, Nov. 2002, pp. 575–578.

[44] S. Pattem, B. Krishnamachari, and R. Govindan, "The impact of spatial correlation on routing with compression in wireless sensor networks," *Proc. ACM/IEEE Conf. Information Processing in Sensor Networks (IPSN)*, pp. 28–35, 2004.

[45] P. v. Rickenbach and R. Wattenhofer, "Gathering correlated data in sensor networks," in *Proc. ACM Joint Workshop on Foundations of Mobile Computing*, 2004, pp. 60–66.

[46] Y. Zhu, K. Sundaresan, and R. Sivakumar, "Practical limits on achievable energy improvements and useable delay tolerance in correlation aware data gathering in wireless sensor networks," in *Proc. IEEE Comm. Society Conf. on Sensor and Ad Hoc Comm. and Networks*, 2005, pp. 328–339.

[47] W. Ye, J. Heidemann, and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 12, no. 3, pp. 493–506, 2004.

[48] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: A Tiny AGgregation service for ad-hoc sensor networks," in *Proc. ACM SIGOPS Symp. Operating System Design and Implementation (OSDI)*. Boston, MA, USA: ACM Press, Dec. 2002, pp. 131–146.

[49] I. Solis and K. Obraczka, "The impact of timing in data aggregation for sensor networks," in *Proc. IEEE Int'l Conf. Comm. (ICC)*, vol. 6, 2004, pp. 3640–3645.

[50] P. Popovski, F. Fitzek, H. Yomo, T. Madsen, R. Prasad, and N. Vej, "MAC-layer approach for cluster-based aggregation in sensor networks," in *Proc. IEEE Int'l Workshop on Wireless Ad-Hoc Networks (IWWAN)*, Oulu, Finland, May 2004, pp. 89–93.

[51] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Trans. Wireless Commun.*, vol. 1, no. 4, pp. 660–670, 2002.

[52] Y. Yao and J. Gehrke, "The Cougar approach to in-network query processing in sensor networks," *ACM SIGMOD Record*, vol. 31, no. 3, pp. 9–18, Sep. 2002.

[53] ——, "Query processing for sensor networks," in *Conference on Innovative Data Systems Research*, 2003.

[54] O. Younis and S. Fahmy, "HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," *IEEE Trans. Mobile Comput.*, vol. 3, no. 4, pp. 366–379, Oct.-Dec. 2004.

[55] Y. Zhu, R. Vendantham, S.-J. Park, and R. Sivakumar, "A scalable correlation aware aggregation strategy for wireless sensor networks," in *Proceedings of the First Int'l Conf. Wireless Internet*, 2005, pp. 122–129.

[56] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," *ACM Transactions on Sensor Networks*, vol. 4, no. 2, pp. 1–40, Mar. 2008.

[57] A. Manjhi, S. Nath, and P. Gibbons, "Tributaries and deltas: efficient and robust aggregation in sensor network streams," in *Proc. ACM Int'l Conf. Management of Data (SIGMOD)*. Baltimore, MD, USA, Jun. 2005, pp. 287–298.

[58] S. Chen and Z. Zhang, "Localized algorithm for aggregate fairness in wireless sensor networks," in *Proc. ACM/IEEE Int'l Conf. Mobile Computing and Networking (Mobicom)*. Los Angeles, CA, USA, 2006, pp. 274–285.

[59] G. di Bacco, T. Melodia, and F. Cuomo, "A MAC protocol for delay-bounded applications in wireless sensor networks," in *Proc. Mediterranean Ad Hoc Networking Conference*, Jun. 2004, pp. 208–220.

[60] K. W. Fan, S. Li, and P. Sinha, "Structure-free data aggregation in sensor networks," *IEEE Trans. Mobile Comput.*, vol. 6, no. 8, pp. 929–942, 2007.

[61] S. Pradhan, J. Kusuma, and K. Ramchandran, "Distributed compression in a dense microsensor network," *IEEE Signal Process. Mag.*, vol. 19, no. 2, pp. –, 2002.

[62] D. Slepian and J. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. Inf. Theory*, vol. 19, no. 4, pp. 471–480, Jun. 1973.

[63] "Python programming language – official website," *www.python.org*, 2011.

[64] SimPy authors, "SimPy," *[Online]. Available: http://simpy.sourceforge.net/*, 2008.

[65] "Matplotlib, a Python 2d plotting library for producing publication quality figures," *http://matplotlib.sourceforge.net/*, 2011.

[66] C. Feuersänger, "Pgfplots, a package for drawing high-quality function plots in LaTeX," *http://sourceforge.net/projects/pgfplots/*, 2011.

[67] K. Radeck, "C# and Java: comparing programming languages," *http://msdn.microsoft.com/en-us/library/ms836794.aspx*, 2003.

[68] "Mono, cross platform, open source .NET development framework," *http://www.mono-project.com/Main_Page*, 2011.

[69] "The Network Simulator—ns-2," *http://www.isi.edu/nsnam/ns/*, 2011.

[70] W. authors, "ns (simulator)," *http://en.wikipedia.org/wiki/Ns_(simulator)*, 2011.

[71] "The ns-3 network simulator," *http://www.nsnam.org/*, 2011.

[72] "OMNeT++, the extensive, modular, component-based C++ simulation library, primarily for building network simulators," *http://www.omnetpp.org/*, 2011.

[73] "Castalia, a simulator for WSNs," *http://castalia.npc.nicta.com.au/naq.php*, 2011.

[74] "TOSSIM, the simulator of entire applications written for the the TinyOS operating system," *http://docs.tinyos.net/index.php/TOSSIM*, 2011.

[75] "TinyOS, the open-source operating system designed for embedded sensor networks," *http://docs.tinyos.net/index.php/Main_Page*, 2011.

[76] B. L. Titzer, D. K. Lee, and J. Palsverg, "Avrora: scalable sensor network simulation with precise timing," *Proc. ACM/IEEE Conf. Information Processing in Sensor Networks (IPSN)*, 2005.

[77] O. Labs, "Sun SPOT world," *http://sunspotworld.com/*, 2011.

[78] M. O. Díaz-Anadón, "Implementation of TPDA," *https://github.com/ornediaz/wsnpy/blob/master/y3/ne3.py*, 2010.

[79] I. Rhee, A. Warrier, M. Aia, J. Min, and M. L. Sichitiu, "Z-MAC: A hybrid MAC for wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 511–524, Jun. 2008.

[80] ——, "Z-MAC: a hybrid MAC for wireless sensor networks," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys)*. ACM Press New York, NY, USA, 2005, pp. 90–101.

[81] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys)*, 2003, pp. 14–27.

[82] B. Hohlt, L. Doherty, and E. Brewer, "Flexible power scheduling for sensor networks," in *Proc. ACM/IEEE Conf. Information Processing in Sensor Networks (IPSN)*, 2004, pp. 205–214.

[83] N. Ramanathan, M. Yarvis, J. Chhabra, N. Kushalnagar, L. Krishnamurthy, and D. Estrin, "A stream-oriented power management protocol for low duty cycle sensor network applications," in *IEEE Workshop on Embedded Networked Systems (EmNets)*, May 2005, pp. 53–62.

[84] G. P. Halkes and K. G. Langendoen, "Crankshaft: An energy-efficient MAC-protocol for dense wireless sensor networks," in *Proc. European Conf. Wireless Sensor Networks (EWSN)*. Springer, 2007, pp. 228–244.

[85] X. Chen, X. Hu, and J. Zhu, "Minimum data aggregation time problem in wireless sensor networks," in *Mobile Ad-hoc and Sensor Networks*, ser. Lecture Notes in Computer Science, X. Jia, J. Wu, and Y. He, Eds. Springer Berlin / Heidelberg, 2005, vol. 3794, pp. 133–142.

[86] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic, "Impact of radio irregularity on wireless sensor networks," in *Proc. ACM Int'l Conf. Mobile*

Systems, Applications, and Services (MobiSys), Boston, MA, USA, Jun. 2004, pp. 125–138.

[87] M. Clark, K. Leung, B. McNair, and Z. Kostic, "Outdoor IEEE 802.11 cellular networks: radio link performance," in *Proc. IEEE Int'l Conf. Comm. (ICC)*, vol. 1, 2002, pp. 512–516.

[88] V. Mhatre and C. Rosenberg, "Design guidelines for wireless networks: communication, clustering and aggregation," *Elsevier J. Ad Hoc Networks*, vol. 2, no. 1, pp. 45–63, Jan. 2004.

[89] S. Soro and W. Heinzelman, "Prolonging the lifetime of wireless sensor networks via unequal clustering," in *Proceedings of the 19th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS)*, 2005.

[90] G. Chen, C. Li, M. Ye, and J. Wu, "An unequal cluster-based routing protocol in wireless sensor networks," *Springer J. Wireless Networks*, vol. 15, no. 2, pp. 193–207, Feb. 2009.

[91] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[92] K. W. Fan, S. Li, and P. Sinha, "Scalable data aggregation for dynamic events in sensor networks," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys)*, 2006, pp. 181–194.

[93] Chipcon Products from Texas Instruments, "Cc2420 chip: 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver," *http://focus.ti.com/docs/prod/folders/print/cc2420.html*, 2010.

[94] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: a mathematical theory of network architectures," *Proc. IEEE*, vol. 95, no. 1, pp. 255–312, 2007.

[95] G. Sharma, R. Mazumdar, and N. Shroff, "On the complexity of scheduling in wireless networks," in *Proc. ACM/IEEE Int'l Conf. Mobile Computing and Networking (Mobicom)*. ACM, 2006, p. 238.

[96] X. Liang, W. Li, and T. A. Gulliver, "An energy-efficient MAC protocol exploiting the tree structure in wireless sensor networks," in *Proc. IEEE Military Comm. Conf.* , W. Li, Ed., 2007, pp. 1–7.

[97] K.-P. Shih, Y.-D. Chen, C.-W. Chiang, and B.-J. Liu, "A distributed active sensor selection scheme for wireless networks," in *Proc. IEEE Symp. Computers and Comm. (ISCC)*, Pula-Cagliari, Sardinia, Italy, Jun. 2006, pp. 923–928.

[98] B. Cărbunar, A. Grama, J. Vitek, and O. Cărbunar, "Coverage-preserving redundancy elimination in sensor networks," *Proc. IEEE Comm. Society Conf. on Sensor and Ad Hoc Comm. and Networks*, pp. 377–386, 2004.

[99] H. Gupta, Z. Zhou, S. R. Das, and Q. Gu, "Connected sensor cover: self-organization of sensor networks for efficient query execution," *IEEE/ACM Trans. Netw.*, vol. 14, no. 1, pp. 56–67, Feb. 2006.

[100] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated coverage and connectivity configuration in wireless sensor networks," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys)*, Los Angeles, California, USA, Nov. 2003, pp. 28–39.

[101] M. Cardei and D. Du, "Improving wireless sensor network lifetime through power aware organization," *Springer J. Wireless Networks*, vol. 11, no. 3, pp. 333–340, 2005.