Rapid Data Transmission Techniques in Wireless Sensor Network

PREPARED BY :

Raxit Jani

11MICT06

GUIDED BY :

Prof.Gaurang Raval

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD

May-2013

Rapid Data Transmission Techniques in Wireless Sensor Network

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Information and Communication Technology

Prepared By

Raxit Jani

(11MICT06)

Guided By

Prof. Gaurang Raval

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May-2013

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in Information and Communication Technology at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgement has been made in the text to all other material used.

Raxit G. Jani

Certificate

This is to certify that the Major Project(Part-II) entitled “Rapid Data Transmission Techniques in Wireless Sensor Network” submitted by Raxit Jani (11MICT06), towards the partial fulfillment of the requirements for the degree of Master of Technology in Information and Communication Technology of Nirma University, Ahmedabad is the record of work carried out by his under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project part-II, to the best of my knowledge, haven’t been submitted to any other university or institution for award of any degree or diploma.

Prof. Gaurang Raval
Guide, Associate Professor,
PG-Coordinator(ICT),
Department of C.S.E.,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Sanjay Garg
Head of Department (C.S.E.),
Institute of Technology,
Nirma University, Ahmedabad.

Dr Ketan Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad.

Abstract

In this report we analyze and compare two algorithms for query based convergecast in WSNs and show that its possible to achieve better performances in terms of energy consumption and latency. Event aggregation in WSNs is a process of combining several low-level events into a high-level event to eliminate redundant information to be transmitted and thus save energy. Existing works on event aggregation consider either latency constraint or aggregation function, but not both. Moreover, existing works only consider optimal aggregation for single highlevel event, but many applications are composed of multiple high-level events. This studies the problem of aggregating multiple high-level events in WSNs with different latency constraints and aggregation functions. We propose relation matrix to define aggregation function, which describes the similarity among limited number of primitive events rather than the growing number of high-level events. Based on it, we propose an event aggregation algorithm jointly considering the two issues for single high-level event. This algorithm supports partial aggregation which is more general than fully aggregation and we focus on another important aspect of aggregation, i.e., delay performance. In conjunction with link scheduling, in-network aggregation can reduce the delay by lessening the demands for wireless resources and thus expediting data transmissions. We formulate the problem that minimizes the sum delay of sensed data, and analyze the performance of optimal scheduling with innetwork aggregation in tree networks under the node-exclusive interference model. We provide a system wide lower bound on the delay and use it as a benchmark for evaluating different scheduling policies. We numerically evaluate the performance of myopic and non-myopic scheduling policies, where myopic one considers only the current system state for a scheduling decision while non-myopic one simulates future system states. We show that the one-step non-myopic policies can substantially improve the delay performance. In particular, the proposed non-myopic greedy scheduling achieves a good tradeoff between performance and implementability. So we propose and efficinet distributed method that produces a collision freeschedule for data aggregation in WSNs. We theoritically prove that latancy of data aggregation schedule generated by our algorithm is at most 16R + Δ - 14 timeslots. Here R is network radius and Δ is the maximum node degree in the communication graph of the original network.

Acknowledgements

My deepest thanks to **Prof. Gaurang Raval**, Associate Professor, PG-Coordinator(ICT), Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad the Guide of the project that I undertook for giving his valuable inputs and correcting various documents of mine with attention and care. He has taken the pain to go through the project and make necessary amendments as and when needed.

I would like to thanks **Prof. Sanjay Garg,** Hon'ble Head of Department, Insti- tute of Technology, Nirma University, Ahmedabad for his unmentionable support, providing basic infrastructure and healthy research environment.

I would like to thanks **Dr.Ketan Kotecha,** Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for his unmentionable support, providing basic infrastructure and healthy research environment.

I would also thank my Institution, all my faculty members in Department of Computer Science. I would like to thank my colleagues for being with me in any condition. And I am thanking them to keep me calm and cool every time.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the first part of my dissertation work successfully.

-Raxit Jani
11MICT06

Contents

List of Figures

1

Introduction

Wireless Sensor Network (WSN) has developed in recent years. WSN because of its exibility in arrangement as well as the less effort demanded for maintenance, have exhibited promising applications in many fields like military, healthcare, en- vironmental applications, etc. WSN comprises of large number of tiny sensor nodes. Because of their small size and use of wireless medium for communication these nodes can be deployed in the phenomenon or close to it. These sensor nodes because of its size have some limitations. They have limited computation power, memory, communication capabilities and energy. WSNs have been extensively studied with the objective of energy efficiency whereas throughput, bandwidth utilization, fairness and latency were considered as the secondary objectives. One of the most prominent operations of WSN is convergecasting. Convergecast, namely the collection of data from a set of sensors toward a common sink over a tree-based routing topology, is a fundamental operation in wireless sensor networks (WSN). Two types of data collection: (i) aggregated convergecast where packets are aggregated at each hop (ii) raw-data convergecast where packets are individually relayed toward the sink. Aggregated convergecast is applicable when a strong spatial correlation exists in the data, or the goal is to collect summarized information such as the maximum sensor reading. Raw-data convergecast, on the other hand, is applicable within each sensor reading, is equally important, or the correlation is minimal. Aggregated convergecast also results in energy conservation and one of the most popular techniques used. In this it reduces the number of packets to be transmitted from source to sink which saves the energy of transmitting each packet individually. This process does increase the latency of communication.

The collection of data froma set of sensors toward a common sink over a treebasedrouting topology,called as convergecast, is a fundamental operation inwireless sensor networks (WSNs).it is essential to provide a guarantee on the delivery time as well as increase the rate of such data collection in many applications. For instance the applications in which the actuators and the controllers need to receive data from all sensors within a specific deadline, failure of which might lead to unpredictableand catastrophic events. This falls under the category ofone-shot data collection. On the other hand, applicationssuch as permafrost monitoring require periodic andfast data delivery over long periods of time, which fallsunder the category of continuous data collection.

In-network filtering and processing techniques like data aggregation can help to conserve the scarce energy resources.

In this report we study the energy savings and the traffic reduction at the expense of latency that can be obtained by introducing data aggregation in the described scenario. We also show how such performances are affected by factors such as the radio range of the nodes, the number of nodes, the maximum out-degree of the nodes in the broadcast tree.

Most previous works on data aggregation have mainly focused on energy saving issue. Building an energy-efficient topology (usually a tree construction) has been investigated. Tree topologies are often employed for wireless sensor networks not only because their structure is suitable for a network with one or a few sink nodes, but also because their simplicity is very attractive when network resources are limited. It has been shown that finding the optimal aggregation tree that minimizes the number of transmissions or maximizes the network lifetime is an NP-Hard problem and efficient approximation algorithms have been provided for constructing such trees. This is not the focus of our paper. Instead, we assume that an aggregation tree has been provided. Given a tree topology, we study the impact of in-network aggregation on the delay performance of wireless scheduling.

Another group of works have investigated the energylatency tradeoff. They have dealt with energy efficiency of wireless sensor networks constrained on the maximum delay of sensed data. Given a deadline, they minimize overall energy dissipation of sensor nodes, minimize the maximum energy consumption, or minimize the amount of missed data.

2

Literature Survery

2.1 Convergecast and Aggregation

Data aggregation in WSNs has been a largely studied topic. In the authors classify aggregation protocols in WSN as tree-based or cluster-based.

Tree-based protocols perform flat routing, i.e. all nodes in the network, except the sink(s) or base station(s), have the same capabilities and equal roles in performing the routing task. Every node is connected to the sink by a multi-hop path and the more a node is near the sink the more paths pass through it. So the routing structure is a tree with the sink as the root, source nodes as the leaves, and intermediate nodes which are allowed to perform data aggregation in order to reduce the number of transmission and save energy. SPIN and Directed Diffusion are the major protocols in this category.

In cluster-based protocols sensor nodes are grouped into cluster based on their proximity. Each cluster has a clusterhead which aggregates data from the members of the cluster and transmit data to the sink. The most famous of such protocol is LEACH.

Tree-based and cluster-based protocols are structure-based protocols: efficient aggregation is obtained through an explicit maintenance of a structure. The other extreme is to use opportunistic aggregation where packets are aggregated only if they happen to meet at a node at the same time. In such case there would not be overhead for structure construction.[2]

2.1.1 Algorithms

Algorithm 1:Simple Relaying Convergecast

In this algorithms which follow the same common elements: there is a single network flow that consists of a single data sink attempting to gather information from a number of data sources; such data sources are the nodes which sense the environment. The network flow consists of two phases: Phase 1 (Broadcast): the sink sends a query for data; the query diffusion produces a broadcast tree in the network. Phase 2 (Convergecast): the sensor nodes having the appropriate data (a random generated number) respond with a reply packet which follows a multi-hop path towards the sink. Note that only a subset of the nodes (the leaf nodes of the tree) are the sensor nodes.

In this algorithm intermediate nodes just relay packets from their children nodes to their parent node. It introduced a mechanism of jittering to implement a simple collision avoidance: leaf nodes wait a random time before sending the reply packet to avoid collisions at the receiver (their parent node). In fact, leaf nodes at the same depth in the tree receive the query and are ready to answer nearly at the same time. The random time is value between 0 and D msec.[2]

$$delay = random(0, D) \tag{2.1}$$

Algorithm 2: Data Aggregation Convergecast

The same considerations about collisions hold in this case. However in this algorithm also the intermediate nodes wait a random time before transmitting their packet.

The aggregate function is arbitrarily chosen because there are different ways for aggregating data. In some cases it means grouping together all the data received in a compressed message in order to avoid multiple transmissions. However, as we said before, the aggregation has been introduced for exploiting the redundancy of data so the most common form of aggregation allows intermediate nodes to combine different values in a single value. This kind of data aggregation is also called in network filtering or data reduction: the intermediate nodes process and filter the data received before retransmitting it in order to prevent the sink to process large amount of data and to reduce the number of packets transmitted. In this case the most common aggregation function are min, max, average.

Other forms of data aggregation are duplicates suppression and data fusion, in which row data coming from sources near each other is combined to produce more accurate data.

We chose the max function in our algorithm: each intermediate node in the tree, once received a packet from all its children nodes or once a timeout has expired, computes the maximum value among all the received values (if any) and send it to its father node with a jittering policy.

The timeout is set considering the depth of the node in the tree: the less the depth, the more is the time to wait because a node with low depth is located much more near the sink so it has a lot of successor nodes each taking its time to aggregate data. So each intermediate node, knowing its depth and given the parameters D and MAXDEPTH as the maxim depth of the tree, can compute the timeout in this way:

$$timeout = D * (MAXDEPTH - depth) \tag{2.2}$$

However this timeout introduces delay so it should be set accurately.

2.1.2 System Model

Tree Construction

In this its not investigated about the tree construction and maintenance because it falls outside the aim of our work which is focalized only on data aggregation. Moreover the creation of an optimal data aggregation tree is generally a NP-hard problem. So constructed the broadcast/convergecast tree in our network, i.e. we fed the simulator directly with a topology in which the broadcast tree is already constructed.[1]

The Python script genTopology.py does in some sense the dirty work: given the total number of nodes N, and a parameter β, it generates a random network topology and constructs the broadcast tree with the sink as the root, and each intermediate node with β as maximum out-degree. The leafs of the tree will be in the simulations the sensor nodes which have data to send to the sink. In addiction the script calculates the number of leaf nodes, the maximum depth of the tree (that will be balanced) and assigns IDs to nodes starting from ID(sink)= 1.

Latency model

According to our model the total latency of the algorithm is the total time taken for one round of data transmission which is composed of: 1) a broadcast phase and 2) a convergecast phase.

In the broadcast phase latency is the time taken for the query message transmitted by root node (sink) to reach all the nodes in the network. In this phase latency is determined by the latency of the critical path the path along which time to deliver packets is the longest, so generally the path from the root to the leaf with maximum depth in the tree.

In the convergecast phase latency is defined as time taken from start of transmission of reply messages from leaf nodes until all the data is received by sink. In this case latency depends on the number of parallel transmissions: higher the number, lower is the latency. In fact we will see that reducing the height of the tree by augmenting the maximum outdegree of the nodes causes a decrease of latency because: I) the critical path will be shorter (broadcast phase); II) there will be more parallel transmissions (convergecast phase).

It compares the total energy consumption of the two algorithms running with the same parameters of the previous simulation (traffic). We can see that its possible to achieve huge energy savings with data aggregation thanks to the reduction in packet transmissions. As an example in a network of 200 nodes with 4 as maximum fan-out of the tree we can obtain a saving of 70% with algorithm 2.[2]

Finally we tested the latency. As we already mentioned data aggregation introduces delay due to the time that each node has to wait until it collects all (or almost all) the messages from its children, before it can transmit the aggregate data to its parent node. The main factors that introduce delay are: 1) the maximum depth of the tree: higher the number of parallel transmissions, lower is the latency; 2) the maximum delay for jittering.

The first implements a relaying convergecast and we have shown that there is high energy dissipation due to many transmissions. Moreover the high traffic in the network leads to an increase in packet collisions which degrades the reliability of the algorithm.

The second algorithm implements a convergecast with data aggregation. Simulating such algorithm we obtained interesting results: the fewer total transmissions lower the energy consumptions because in sensor networks the tx/rx operations dominates all the other operations in terms of energy dissipation.

However, the energy saving trades-off with higher delay due to data aggregation: in fact every intermediate node must wait for all (or almost) the packets from its child to be received before aggregating the data and send it to its father.[1]

In sensor networks latency is an issue, and in some scenarios it may be as relevant as energy saving. Therefore a possible future work would address the problem of the high latency enhancing our second algorithm.

The second major drawback of our work is that both the algorithms need an already constructed broadcast tree in the network to work. So we fed the simulator with a priori constructed broadcast tree topologies.

2.2 Event Aggregation

Early works in WSNs are mainly about data processing and the latest ones begin to discuss event processing. Event processing is a natural extension of data processing in WSNs, which encapsulates collected data in events. Correspondingly, while data aggregation generates the summary of raw data, event aggregation deduces high-level events from low-level events. Event aggregation is an effective approach to reduce data amount in WSN communication.

In existing works, some analyze the impact of aggregation functions on aggregation structure, where aggregation function describes the correlations among different events. Others consider latency constraints to meet time sensitive requirements. An aggregation with both of the factors is more general but has not been considered yet. The difficulty partly arises from the aggregation function which is usually assumed as fully aggregation function which means two lowlevel events with one unit data amount can merge into a composite event with one unit data amount, or more general function which takes low-level events as input and composite event as output. It is not clear how to combine aggregation function with latency constraint. More importantly, hardly any existing works have been conducted for multiple high-level events. Considering a WSN based intelligent traffic system, many events are required by different users for different purposes. Some users may have interests in the average speed of vehicles, while some others may want to know the speed of individual vehicles. The events have different correlations among low-level events hence have different aggregation functions. Moreover, the latency requirements also may be different. Existing approaches have not investigated the diversity of requirements in event aggregation.[3]

In this report, we investigate the aggregation problem of multiple high-level events with different latency constraints and aggregation functions in WSNs. We propose relation matrix as a simple approach to define aggregation function, which use the similarity among limited number of primitive events rather than growing number of high-level events. After that we design a algorithm named DBEA to build the aggregation tree for single high-level event considering both the two issues. This algorithm supports partial aggregation which is more general than fully aggregation. The conflicting optimal parent candidates in building the tree is also considered. Finally, we use dynamic programming to select some of the high-level events as base events subject to a reliable constraint, and then build the aggregation tree using DBEA. firstly, proposed a simple approach to describe aggregation function and enable them to combine with latency constraint. Secondly, considered both latency constraint and aggregation function in individual high-level event aggregation. Thirdly, extended the approach for multiple high-level events aggregation and consider the practical reliable constraint.

Two issues have influence on event aggregation: latency constraint and aggregation function. In this report, latency is defined as the time duration from the time point when the first event is sent at a source node to the time point when the last event is received by the sink node. In a usual WSN application, the latency is mainly determined by communication distance. When latency constraint is tight, the aggregation tree trends to be the shortest path tree and little opportunity to undertake the aggregation. By contrast, if latency constraint is loose (eg. no latency constraint), optimal aggregation can be achieved. Aggregation function also affects event aggregation, which describes correlations among the events and hence the data amount reduction. A larger data amount reduction means the event is more sensitive to event aggregation.

Existing works encounter two problems when building the aggregation tree. Firstly, they considered only aggregation function but no latency constraint. Secondly, they cannot be directly adopted for multiple high-level events with different latency constraints and aggregation functions. For the first problem, in later section we will introduce an solution for single event. For the second problem, several straightforward approaches can be proposed but all have deficiencies. The first one is to build the aggregation tree every time an event comes which results in unacceptable overhead and latency. The alternative approaches include building the aggregation tree with the minimum latency and a special aggregation function, or building all aggregation trees, which may lead to a suboptimal solution or excessive large storage not supported by current sensor nodes Mica2 and Micaz respectively.

The reasonable approach is to build a number of m aggregation trees in advance according to a number of m latency constraint-aggregation function pairs, where m is determined by available data memory in a sensor node. For simplicity, m events are selected as base events to acquire the m latency constraint-aggregation function pairs. We call the latency constraints of base events as base latency constraints, and the aggregation functions of base events as base aggregation functions. In reality, all the events with base latency constraints are required to meet an reliable requirement. For example, if l_i is a base latency constraint, all the events with l_i need bound their performance degradation to some degree.[3]

2.2.1 Problem Formulation

The Event Aggregation with Different Latency Constraints and Aggregation Functions (EALA) problem is formulated: Given:

a. A wireless sensor network G(V,E), a set of source nodes S and a sink node r.

b. A high-level event set CE involving n composite events ce_i(i = 1...n) which need detected. Each composite event has latency constraint l_i and aggregation function ag_i.

c. At most m composite events $ce_{v1}...ce_{vm}$ can be selected as base events to construct the aggregation tree. find a aggregation tree which aggregates all composite events to the sink node with minimal message traffic cost. subject to

a. Each composite event meets latency constraint

b. All the events with base latency constraints meet the reliable requirement.

2.2.2 Delay Bounded Event Aggregation Algorithm

We propose Delay Bounded Event Aggregation Algorithm (DBEA) to build the aggregation tree considering both latency constraint and aggregation function. Different from fully aggregation, DBEA supports partial aggregation. During the tree building, if a source node joins in the tree, the distance increased is not the distance between this node and the tree, but the distance between this node and the sink node.[4]

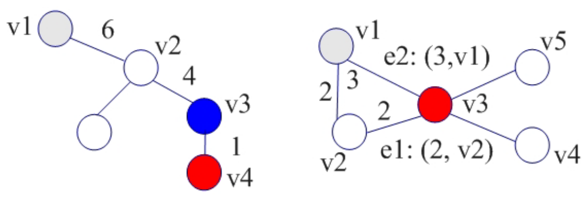

Figure 2.1: Design rationale of delay bounded event aggregation algorithm

For example, in Figure 3.1, aggregation tree T already includes v1, v2 and v3; the distance between v4 and T is 1, which is exactly the increased distance in fully aggregation since when a packet traverses from v4 to v3, it merges into the packet of v3 hence not increase the cost from v3. This is not the case in partial aggregation, the packet traversing to v3 may aggregate into a packet with data amount more than 1(eg. 1.5), then it has additional cost from v3 to the sink node (eg. $(1.5 \ \ 1)(6 + 4) = 5)$). We also introduce the concept of possible distance, because different events may have conflicting optimal parent candidates. As in Figure 3.1, v1, v2 and v3 are already in T; When event e1 transmits to v3, the optimal distance is 2 subject to the parent of v2, while for event e2, the optimal distance is 3 subject to the parent of v2. In the aggregation tree, v3 have only one parent. This is so called possible and need adjustment when a source node joins in the tree. The details are shown in Algorithm 1.

We randomly create networks with minimum latency 4 and aggregation function N0(norm of A_2 is 0, norm=0 for short), then issue a set of events with increasing latency constraints from 0 to 25. We use DBEA to build aggregation tree and calculate the maximum latency. The experiment is repeated with different aggregation function N1(norm = 1), N2(norm = 2) and N3(norm = 3). The result and each point represents 50 times executions.All the results are between the minimal latency and latency constraint, which shows all latency constraints are strictly meet, at the same time the approach gets benefit compared with the approach simply selecting the minimal latency. Growing benefit goes with increasing aggregation function. The aggregation tree with N3 is longer than all others. For event 12, the latency of aggregation tree with N3 is 13 while that with N0 is only 8. The aggregation characteristic is recognized in the algorithm and generate a longer aggregation tree hence more opportunities for performance improvement.

The simulation results show that significant energy (up to 35% in our system) can be saved by using our algorithm.

2.3 Delay Performance of Scheduling with Data Aggregation

Most previous works on data aggregation have mainly focused on energy saving issue. Building an energy-efficient topology (usually a tree construction) has been investigated. Tree topologies are often employed for wireless sensor networks not only because their structure is suitable for a network with one or a few sink nodes, but also because their simplicity is very attractive when network resources are limited. It has been shown that finding the optimal aggregation tree that minimizes the number of transmissions or maximizes the network lifetime is an NP-Hard problem and efficient approximation algorithms have been provided for constructing such trees. This is not the focus of our paper. Instead, we assume that an aggregation tree has been provided. Given a tree topology, we study the impact of in-network aggregation on the delay performance of wireless scheduling.

Another group of works have investigated the energylatency tradeoff. They have dealt with energy efficiency of wireless sensor networks constrained on the maximum delay of sensed data. Given a deadline, they minimize overall energy dissipation of sensor nodes, minimize the maximum energy consumption, or minimize the amount of missed data.

we formulate the problem by minimizing the total delay of sensed data, and thus reduce the average delay. Associated with innetwork aggregation, scheduling also needs to be considered since one has to determine which node forwards data, and which node holds data for aggregation, which might improve the performance in the future by reducing the amount of bits in-flight. The problem is further exacerbated by interference in the wireless environment. Our main contributions are as follows:

- We formulate the scheduling problem that minimizes the delay sum of data in wireless sensor networks, and qualify the gain of in-network aggregation techniques for tree networks.

- We study the characteristics of optimal scheduling with aggregation, and analyze the optimal performance by providing a lower bound.

- We evaluate the performance of scheduling policies against the delay bound, and show that the performance can substantially improve by considering the network state one-step ahead, i.e., by estimating the network state of the next time slot.

2.3.1 Network Model

We consider a tree network T(V,E), where V denotes the set of n wireless nodes and E denotes the set of wireless links. One sink node denoted by v_s is located at the root of the tree, and collects data from (n - 1) sensor nodes. The task of the network is to compute a function from the sensed data, and to deliver the function value through the sink node. Data is forwarded via hop-by-hop communications to save energy, which costs a unit of transmission time per forwarding. The function should allow in-network aggregation from a partial collection of data without changing the final results or increasing transmission cost (time) for forwarding aggregated data. Common examples of such functions include min, max, average, histogram, etc.

We assume a time-slotted system with a single frequency channel for the entire system. Channels are assumed error free. However, there is wireless interference between channels, which is modeled by the well-known node-exclusive interference model1. Under this model, two links sharing a node cannot transmit at the same time. The model is suitable for Bluetooth and FH-CDMA networks. At each time slot, a link can transmit a single packet, and multiple links can transmit simultaneously only when they are greater than one hop away from each other. It is also denoted by the 1-hop interference model or the primary interference model.[5]

The overall system operations are described as follows. Assume that a measurement event occurs at time t = 0. Each node senses its environment and generates data, which is denoted by a message. All messages from sensor nodes have to be delivered to the sink. During the first time slot, each message m generated at node v 1) is contained in a packet for transmission and forwarded to the parent node of v, 2) stays at node v due to transmissions from an interfering neighbor, or 3) waits at node v for further aggregation. The parent node that receives the packet may forward the message to its own parent node in the next time slot. After the first time slot, some of the nodes may have multiple messages. Under certain sensor network applications, nodes with multiple messages can compress the data and put them into a single packet without affecting the final results. This procedure is called as in-network aggregation. For example, suppose that the task of the network is to compute an average of the messages obtained from sensor nodes. If we format the content of a packet in a tuple of (the average of messages, the number of averaged messages), an intermediate node can compute an average of all the messages that it has received or generated, and package the computation result into a single tuple without losing any information. We assume that aggregation is permanent and messages that are aggregated will not split later. At each time slot, the process of transmission and aggregation is repeated until the sink collects all the messages. Clearly, aggregation will improve scheduling efficiency by reducing the number of packet transmissions. We assume that if in-network aggregation is allowed, it can be done with a negligible cost of energy or computing power. If aggregation is not allowed, we assume that one packet can contain only one message.

Our objective is to minimize the total delay in collecting all the messages related to a particular measurement event. Let V denote the set of all messages of the event. Each message is generated at a sensor node during t = 0 and no message is generated for t > 0. Messages do not need to measure the same physical environment, and may have a different meaning with a different importance, in which case, we quantify the importance of each message m using a weight $w_m > 0$. Let f_m(t) denote the node where message m is located at^2 time t. We denote the system state at time t by its vector f(t) := f_m(t). Also let S(t) denote the schedule during time slot t, which is the set of nodes that transmit during time slot t. Note that each node v . S(t) will transmit a packet to its parent node p(v), and those set of active links (v, p(v)) will correspond to a matching under the node-exclusive interference model. Let S denote the set of all possible schedules. Given a system state, let H (v) denote the set of messages that can be transmitted when node v is scheduled. Without aggregation, the set H (v) will have a single message, which might be the one with the largest weight among the messages located at node v. (See Fig. .) With aggregation, we assume that H(v) includes all messages at node v.

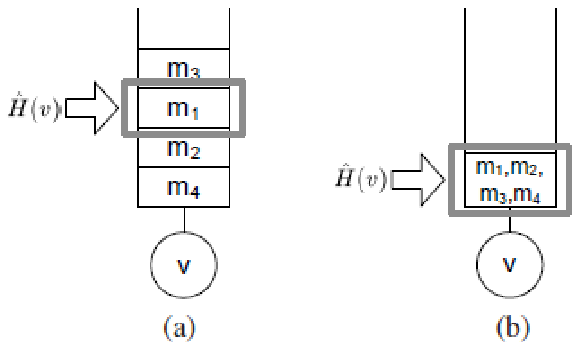

Figure 2.2: Messages in the queue of node v with and without aggregation. Without aggregation, when node v is scheduled, the message m1 will be forwarded. With aggregation, all messages are aggregated into a single packet and will be forwarded simultaneously. (a) Without in-network aggregation. (b) With in-network aggregation.

After time slot t, the system state will change as follows:

$$f_m(t+1) = \begin{cases} p(f_m(t)) & \text{if } f_m(t) \in S(t) \\ & \text{and } m \in \hat{H}(f_m(t)), \\ f_m(t) & \text{otherwise.} \end{cases}$$

Let tm denote the time when message m arrives at the sink v_s, i.e., t_m := min{t — $f_m(t+1) = v_s\}$. Multiple messages, if aggregated, can arrive at the sink simultaneously and have the same arrival time. Our objective is to solve the following problem:

$$\underset{\{S(t) \in \mathcal{S}\}}{\text{minimize}} \sum_{m \in \hat{V}} w_m t_m$$

We rewrite the above problem using the delays that each message experiences. Let S(t) denote the set of messages that are scheduled during time slot t, i.e., S(t) := m — f_m(t) S(t),m H(f_m(t)). Further, let h(v, v') denote the number of hops between two nodes v and v', and let hm denote the number of hops between the initial location fm(1) of message m and the sink v_s, i.e., h_m := h(f_m(1), v_s). Note that hm is the least number of time slots for m to arrive at the sink when there is no other message. When multiple messages coexist in the network, two packets that contain a different message can compete with each other for scheduling resources. While a node is transmitting a packet, its neighboring nodes cannot transmit simultaneously due to wireless interference and the messages in those nodes have to be delayed staying at the current location. Let

D_m denote the delays experienced by each message m in the network until it arrives at the sink, i.e., $D_m := t_m h_m$. Since is a constant, (2) is equivalent to

$$\underset{\{S(t)\in\mathcal{S}\}}{\text{minimize}} \quad \sum_{m\in\hat{V}} w_m D_m$$

At each time slot, we have to determine which message m has to be forwarded, and which should stay for aggregation. Hence, in-network aggregation results in a tradeoff between scheduling efficiency and delays. Remark: The problem (3) is similar to the scheduling for minimum delay, where the delay optimal scheduler for linear networks has been provided. Although there is a similarity in the objective, the problem becomes much more complex here because the topology is richer (a tree topology), and we have to deal with aggregation. Hence, the solution of is not applicable.

2.3.2 Impact of Aggregation

In this section, we study the effectiveness of in-network aggregation in terms of the total delay. Given a tree network of n nodes including the sink, we consider a scenario in which each sensor node generates a message, and all (n - 1) messages are equally important. Specifically, all weights wm are set to 1. We show that without aggregation, the optimal delay is lower bounded by $O(n^2)$, while it is upper bounded by O(n log n) with aggregation. Therefore, the gain is quite substantial O(n/ log n).

Performance without aggregation

For a given tree network, let d and W denote the maximum depth and the maximum width of the tree, respectively. At time t, a message m is said to be located at depth d if h($f_m(t), v_s$) = d. The following proposition provides a bound on the optimal delay performance.

Proposition 1: Without in-network aggregation, the delay is lower bounded by

$$\sum_{m\in\hat{V}} D_m \geq O(n^2)$$

when d = O(log n).

Proof: Since messages cannot be aggregated, each packet has to include only one message. Then, at the root, the sink can receive at most one packet during each time slot under the node-exclusive interference model. Since it takes at least (n - 1) time slots to receive all (n - 1) messages.

$$\sum_{m\in\hat{V}} D_m \geq \sum_{m\in\hat{V}} t_m - \sum_{m\in\hat{V}} h_m = O(n^2)$$

Performance with aggregation

In-network aggregation can significantly reduce the delays by integrating multiple messages into a single packet. Since the reduction in the number of transmissions implies less interference, messages can be forwarded faster. We assume that messages can be aggregated into a single packet with no cost if they are located in the same node.

Proposition 2: With in-network aggregation, the delay is upper bounded by

$$\sum_{m\in\hat{V}} D_m \leq O(n \log n)$$

Proof: We provide a scheduling policy based on link coloring and show its delay bound, which will serve as an upper bound on the delay of the optimal scheduler.

Since each node has at most (W + 1) links, we can color links using (W + 2) different colors such that two links with the same color are not adjacent to each other. Then we can schedule links with the same color simultaneously. We choose a color in a time slot in a round robin manner. Note that under this policy, each node transmits a packet to its parent every (W +2) time slots. We can estimate the maximum delay that a message experiences before it arrives at the sink.[5]

Optimal Scheduling

Let A(t) and T(t) denote the set of messages that have arrived at the sink, and are transient in the network, respectively, at time slot t. That is, A(t) = {m — f_m(t) = v_s}, and T(t) = {m — f_m(t) —= v_s}. Clearly, we have A(1) = 0 and T(1) = V . Let Dm(t) denote the delays experienced by m during [0, t]. Hence

$$D_m(t) = \begin{cases} D_m(t-1) & \text{if } m \in \hat{S}(t) \cup \hat{A}(t), \\ D_m(t-1)+1 & \text{otherwise,} \end{cases}$$

Numerical Results

We now investigate the performance of several scheduling algorithms versus our delay bound as a performance benchmark. We assume that the objective function, e.g., average of sensed data, conforms to our aggregation constraints, i.e., in-network computations with a partial collection of data do not change the final results and do not increase transmission cost of aggregated data. We classify scheduling policies into myopic and non-myopic ones: Myopic policies make a scheduling decision based only on the current system state. Most conventional scheduling algorithms are myopic. Nonmyopic policies simulate the network with a sequence of schedules, and make a scheduling decision for time slot t based on simulation results. For example, a non-myopic policy that can simulate the network up to tmax steps can solve (3) by simulating S(1), . . . , S(tmax).

In our experiments, we focus on binary tree networks. We first examine the optimal performance for full binary trees and move to binary trees that are randomly generated. We also compare the analytical results with the performance of scheduling policies when weights are assigned at random.

Myopic scheduling policies

For a feasible schedule S S, let D(f(t), S) denote the total delay increment if S is applied to the system state f(t).

Since myopic scheduling policies make scheduling decisions based on the current state of the network, we consider the following conventional scheduling algorithms:

- Maximum Weight Matching (MWM): During each time slot t, it schedules the set of nodes that minimize the delay increment of messages for the time slot.

- Maximum Size Matching (MSM): During each time slot t, it schedules the set of nodes that maximize the total number of scheduled nodes with a message.

- Greedy Maximal Matching (GMM): At each time slot t, it sorts nodes based on the sum of message weights in each node, and includes nodes in the schedule S(t) in decreasing order conforming to the inference constraints.

One may expect that MWM will achieve the best performance since it serves the largest weights during the time slot.

Non-myopic scheduling policies

In this section, we propose three non-myopic scheduling policies and evaluate their performance. Under non-myopic scheduling policies, a scheduling decision is based on the current and the future system states, which can be obtained by simulating the network. A higher performance is expected as further future states are taken into account. However, due to computational complexity in implementation, we consider one-step non-myopic policies only.

Specifically, we propose the following algorithms:

- Non-Myopic MWM (NM-MWM): During each time slot t, it schedules the set of nodes that will minimize the delay increments during [t, t + 1]
- Non-Myopic MWM with Analysis (NM-MWM+A): It schedules the set of nodes that minimize the future delay bound, which can be obtained using our analysis.
- Non-Myopic GMM (NM-GMM): It is similar with NMMWM except that for each feasible S, it chooses the second schedule S' using the GMM algorithm.

This analysis is accurate, in particular when the network size is small. We also evaluate the performance of several myopic and non-myopic scheduling policies, and show that the one-step non-myopic scheduling policies achieve substantially lower delays than the conventional myopic ones. In particular, the non-myopic GMM seems to achieve good performance with moderate complexity making it attractive as a viable alternative over the optimal solution.[6]

There are many interesting directions for future work. One can obtain analytic results when the underlying tree networks have a larger width, and develop a simple distributed algorithm with high performance. Performance of scheduling policies will depend on different interference models, e.g., matrixbased interference model or K-hop interference models, and needs to be quantified in those settings. It could also be interesting to study the scheduling performance when the energy cost of in-network aggregation itself is not negligible.

3

Proposed Technique

We propose efficient algorithms that will construct a data aggregation tree and a TDMA schedule for all links in the tree such that the latency of aggregating all data to the sink node is approximately minimized. For simplicity of analysis, we use the following interference model: when a node v is receiving data from a sender u, v is not within the interference range r_I of any other active sender x. As an illustration, we first present an efficient centralized algorithm that will build a TDMA schedule of links based on the aggregation tree which is build distributively. Our schedule uses a bottom-up approach: schedule nodes level by level starting from the lowest level. For simplicity of analysis we assume that the interference range r_I = r, and then we theoretically prove that the latency of the aggregation schedule generated by our algorithm is at most 16R + Δ - 14 time-slots. Notice that, for general r_I , our algorithm will produce a collision-free schedule for aggregation whose latency is at most $\Theta((r_I/r)^2R + \delta)$ timeslots. We then present an efficient distributed algorithm that builds an aggregation tree and gives a schedule for each link. For simplicity, our distributed method assumes that the clocks of all nodes are synchronized. Unlike our centralized algorithm, our distributed algorithm will not explicitly produce a schedule for links in the aggregation tree. The link schedule is implicitly generated in the process of data aggregation. Our distributed scheduling algorithm thus works well in dynamic networks, as long as the constructed backbone of the network by our algorithm remains unchanged. Obviously, when r_I = r, for a network G with radius R and the maximum node degree Δ, the latency by any data aggregation algorithm is at least R. This implies that our algorithm is within a small constant factor of the optimum.

3.1 System Models

3.1.1 Aggregation Functions

The database community classifies aggregation functions into three categories: distributive (e.g., max, min, sum, count), algebraic (e.g., plus, minus, average, variance) and holistic (e.g., median, kth smallest or largest).

A function f is said to be distributive if for every pair of disjoint data sets X_1, X_2, we have $f(X_1UX_2) = h(f(X_1), f(X_2))$ for some function h. For example, when f is sum, then h can be set as sum; when f is count, h is sum. Assume that we are given an aggregation function F that can be expressed as the combination of k distributive functions for some integer constant k, i.e.,

f(X) =$h(g_1(X), g_2(X), ..., g_k(X))$.

For example, when f is average, then k = 2, g_1 can be set as sum, g_2 can be set as count (obviously both g_1andg_2 are distributive) and h can be set as $h(y_1, y_2) = y_1/y_2$. Thus, instead of computing f, we will just compute $y_i = g_i(X)$ distributively for i [1, k] and $h(y_1, y_2, ..., y_k)$ at the sink node.

3.1.2 Network Model

We consider a wireless sensor network consisting of n nodes V where $v_s \epsilon V$ is the sink node. Each node can send (receive) data to (from) all directions. For simplicity, we assume that all nodes have the same transmission range r such that two nodes u and v form a communication link whenever their Euclidean distance $\|u - v\|$ ≤r. In the rest of the paper we will assume that r = 1, i.e., normalized to one unit. Then the underneath communication graph is essentially a unit disk graph (UDG). Let A,B$\subset VandA \cap B = \phi$. We say data are aggregated from A to B in one time-slot if all the nodes in A transmit data simultaneously in one time-slot and all data are received by some nodes in B without interference. We will define interference at the end of this section. Then a data aggregation schedule with latency l can be defined as a sequence of sender sets $S_1, S_2, ..., S_l$ satisfying the following conditions:

- $S_i \cap S_j = \phi, All_i \neq j$;

- $\cup_{i=1}^{l} S_i = V \backslash v_s$;
- Data are aggregated from $S_k to V \backslash \cup_{i=1}^{l} S_i$ at time-slot k, for all k = 1, 2, , l and all the data are aggregated to the sink node v_s in l time-slots.

Notice that here $U_{i=1}^{l} S_i = V \backslash v_s$ is to ensure that every data will be aggregated; $S_i \cap S_j = \phi., i \neq j$ is to ensure that every data is used at most once. To simplify our analysis, we will relax the requirement that $S_i \cap S_j = ., i \neq j$. When the sets $S_i, 1 \leq i \leq l$ are not disjoint, in the actual data aggregation, a node v, that appears multiple times in $S_i, 1 \leq i \leq l$, will participate in the data aggregation only once (say the smallest i when it appears in Si), and then it will only serve as a relay node in the following appearances.

The distributed aggregation scheduling problem is to find a schedule $S_1, S_2, ..., S_l$ in a distributed way such that l is minimized. proposes an approximate distributed algorithm with latency 16R+Δ -14 time-slots, where R is the network radius and R is the maximum node degree.

We assume that a node cannot send and receive data simultaneously. In protocol interference model, we assume that each node has a transmission range r and an interference range $r_I \geq r$. A receiver v of a link uv is interfered by another sender p of a link pq if $\|u - v\| \leq r_I$. we first assume that r_I = r, which is normalized to 1 unit in this paper. We will later study a more general case $r_I \geq r$.

3.1.3 Related Terminologies

For simplicity, we present our distributed algorithms in a synchronous message passing model in which time is divided into slots. In each time-slot, a node is able to send a message to one of its neighbors for unicast communication. Note that, at the cost of higher communication, our algorithms can also be implemented in asynchronous communication settings using the notions of synchronizer.

In a graph G = (V,E), a subset S of V is a dominating set (DS) if for each node u in V , it is either in S or is adjacent to some node v in S. Nodes from S are called dominators, whereas nodes not in S are called dominatees. A subset S of nodes is independent set (IS), if for any pair of nodes in S, there is no edge between them. An IS S is a maximal IS if no another IS that is a superset of S. Clearly, a maximal IS is always a DS. A subset C of V is a connected dominating set (CDS) if C is a dominating set and C induces a connected subgraph. Consequently, the nodes in C can communicate with each other without using nodes in V $\backslash C$. A CDS is also called a backbone here.

3.2 Distributed Aggregation Scheduling

Our data aggregation scheduling (DAS) algorithm consists of two phases: 1) aggregation tree construction and 2) aggregation scheduling. As an illustration of our methods, we first present a centralized version of our data aggregation scheduling. We adopt an existing method for the first phase and the second phase is the core of our algorithm. We will present these two phases in the following two sections. At the end of the section, we present a distributed implementation based on our centralized aggregation scheduling algorithm.

3.2.1 Aggregation Tree Construction

In the first phase we construct an aggregation tree in a distributed way using an existing approach. We employ a connected dominating set (CDS) in this phase since it can behave as the virtual backbone of a sensor network. A distributed approach of constructing a CDS has been proposed by Wan et al.. In their algorithm, a special dominating set using a MIS of the network is constructed first and then a CDS is constructed to connect dominators and the other nodes. All nodes in the MIS is colored black and all other nodes are colored grey. This CDS tree can be used as the aggregation tree in our scheduling algorithm with a small modification as follows.
1) We choose the topology center of the UDG as the root of our BFS tree. Notice that, previous methods will use the sink node as the root. Our choice of the topology center enables us to reduce the latency to a function of the network radius R, instead of the network diameter D proved by previous methods. Here a node v0 is called the topology center in a graph G if $v_0 = argmin_v max_u d_G(u, v), whered_G(u, v)$ is the hop distance between nodes u and v in graph G. R = $max_u d_G(u, v_0)$ is called the radius of the network G. Notice that for most networks, the topology center is different from the sink node.
2) After the topology center gathered the aggregated data from all nodes, it will then send the aggregation result to the sink node via the shortest path from the topology center v_0 to the sink node vs. This will incur an additional latency $d_G(v_0, v_s)$ of at most R. The rank of a node u is (level, ID(u)), where level is the hop-distance of u to the root

in the BFS. The ranks of nodes are compared using lexicographic order.

3.2.2 Centralized Approach

The second phase is aggregation scheduling which is the core of the whole algorithm. It is based on the aggregation tree constructed in the first phase. As an illustration, we first present an efficient centralized algorithm.

Step1 shows how the data from the dominatees are aggregated to the dominators. Our method is a greedy approach, in which at every time-slot, the set of dominators will gather data from as many dominatees (whose data has not been gathered to a dominator yet) as possible. Notice that since the maximum degree of nodes in the communication graph is Δ , our method guarantees that after at most Δ time-slots, all the dominatees data will be gathered to their corresponding dominators when considering the interference, which will be proved in Lemma 2. The basic idea is as follows: each dominator will randomly pick a dominatee whose data is not reported to any dominator yet. Clearly, these selected dominatees may not be able to send their data to corresponding dominators in one time-slot due to potential interferences. We then reconnect these dominatees to the dominators (and may not schedule some of the selected dominatees in the current time-slot), using Step2, such that these new links can communicate concurrently.

Step1 Aggregate Data to Dominators[11]

```
1:for i = 1, 2,...,$\Delta$ do
2: Each dominator randomly chooses 1 neighboring dominatee, whose data is not gathered yet, as transmitter. The set of such chosen links form a link set L.
3: If there are conflicts among chosen links, resolve interference using Algorithm 2;
4: All the remaining links in L now transmit simultaneously;
5: i = i + 1;
```

Suppose that two directed links $u_i z_i$ and $u_j z_j$ interfere with each other(see Fig.), where the dominatees $u_i and u_j$ are transmitters in these two links respectively and $z_i and z_j$ are dominators. For each dominatee v, let D(v) be the set of neighboring dominators. Obviously, $\|D(v)\| \leq 5$ for any node v. Let $D(u_i) = D(u_i)\backslash z_i, D(u_j) = D(u_j)\backslash z_j$. Notice that here $D(u_i) and D(u_j)$ may be empty, and $D(u_i) \cap D(u_j)$ may also not be empty.

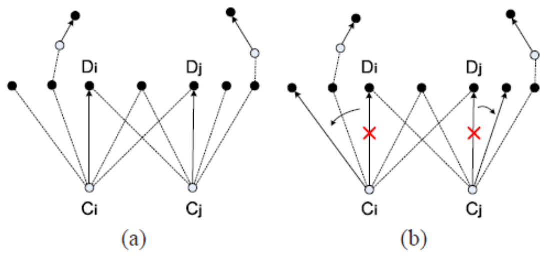

Figure 3.1: (a) An interference between 2 links. (b) A state after rescheduling the two links.[11]

For each active transmitter v, v $\neq u_i$ and $v \neq u_j$, we delete all dominators from D(u_i) (and also from D(u_j)) that are within the transmission range of v. Notice that we can discard these dominators since their degrees are already decreased by at least 1 because of the existence of some active transmitter v. We also delete the dominators that are within transmission range of both ui and uj from $D(u_i) and D(u_j)$. Notice that we can do this because these dominators degree will be decreased by 1 since our re-scheduling can guarantee at least one transmitter of ui and uj will remain as an active transmitter.

Let $D_i (resp. D_j)$ be the set of remaining dominators in $D(u_i)(resp. D(u_j))$.

Fig illustrates one possible state after the preceding two deletions of dominators from $D(u_i) and D(u_j)$. Notice that

1) The distance between u_i and any member of D_j is greater than 1. The distance between u_j and any member of D_i is greater than 1.

2) Both $D_i and D_j$ may be empty and may not be empty. Algorithm 2 shows how to re-connect dominatees to dominators to avoid the interference.

After all the data in the dominatees have been aggregated to dominators, our next step is to aggregate all the intermediate results in the dominators to the root.

We can see that in each layer of BFS tree, there are some dominator(s) and some dominatee(s). For every dominatee, it has at least one dominator neighbor in the same or upper level. Thus, every dominator (except the root) has at least one dominator in the upper level within two-hops. Using this property, we can ensure that all the data in the dominators can reach the root finally if every dominator transmits its data to some dominator in upper level within two-hops. From another point of view, considering dominators in the decreasing order of their levels, a dominator u in level L aggregates data from all dominators in level L+1 or L+2 that are within two-hops of u. This will ensure that all the data will be aggregated to the root. Step 3 presents our method in detail.

In Step 3 we only concentrate on communications between dominators. Since dominators cannot communicate directly. we have to rely on some dominatees, each of which acts as a bridge between two dominators. Hereafter we rename these dominatees as connectors. The algorithm runs from lower level to upper level in aggregation tree, every dominator will remain silent until the level where it locates begins running. When it is its turn, the dominator will try to gather all the data from other dominators in lower levels that have not been aggregated. If a dominators data has been collected before, then it is unnecessary to be collected again. Actually we have to guarantee this for every data should be and only be used once. Our algorithm implements this by discarding the dominators after their data have been gathered to upper levels.

Notice that in our algorithm after we process dominators B_i (all dominators in level i), there may still have some dominators in B_{i+1} whose data are not aggregated. This could happen because a dominator in B_{i+1} could be within 2-hops of some dominator in B_{i-1}, but not within 2-hops of any dominator from B_i. We conclude that after the execution of all the dominators in B_i , all dominators in B_{i+2} have already been aggregated their data.

3.2.3 Distributed Implementation

Now we present a distributed implementation for our data aggregation scheduling. The distributed implementation consists of three stages:

- Every dominatee transmits its data to the neighboring dominator with the lowest level,

- Data are aggregated from dominators from lower levels to dominators in upper levels and finally to the root of the aggregation tree which is the topology center of the network,

- Topology center then transmits the aggregated data to the original sink via the shortest path.

The distributed implementation differs from the centralized one in that the distributed one seeks to transmit greedily: we will try to allocate a node v a time-slot to transmit whenever v has collected the aggregated data from all its children nodes in the data aggregation tree T . Thus the first two phases may interleave in our distributed implementation. The interleaving will reduce the latency greatly since it increases the number of simultaneous transmissions. Later, we will provide the simulation result of our distributed method, which shows that our distributed implementation is quite close to (1 + $\varepsilon)R + \Delta + \Theta(1)$, where " is a small positive constant. Therefore we conjecture that the data aggregation latency by our distributed implementation indeed has a theoretical performance guarantee of $(1 + \varepsilon)R + \Delta + \Theta(1)$. It will be interesting if we can prove or disprove this conjecture, which is left as a future work.

To run our Step, every node vi should maintain some local variables, which are

- Leaf indicator: Leaf[i] 0, 1, to indicate whether the node v_i is a leaf node in the data aggregation tree.

- Competitor Set: CS[i], the set of nodes such that for each $j\epsilon CS[i]$, nodes $v_i and v_j$ cannot transmit simultaneously to their parents due to interference. In other words, if $j\epsilon CS[i]$, we have either the parent pT (i) of node $v_i$in the data aggregation tree T is within the interference range of node v_j ; or the parent pT (j) of node v_j in the data aggregation tree T is within the interference range of node vi; or both. Notice that under the interference model studied in this paper, each node in CS[i] is within a small constant number of hops of i.

- Ready Competitor Set: RdyCS[i], which is the set of nodes that collides with i and it is ready to send data to its parent, i.e., it has received the data from all its children nodes.

- Time Slot to Transmit: TST[i], which is the assigned time-slot that node vi indeed sends its data to its parent.

- Number of Children: NoC[i], which is the number of children nodes of vi in the data aggregation tree T .

Observe that here, at sometime, if we let Rdy be the set of nodes which are ready to transmit (i.e., v Rdy iff v has collected the aggregated data from all its children nodes in the data aggregation tree T), and let F denote all the nodes which have finished their transmission, then RdyCS[i] = CS[i] ∩ Rdy . F. The TST of all nodes are initialized to 0. The details of our distributed algorithm are shown in step 4.

When a node vi finishes its scheduling, it sends a message FINISH to all nodes in its competitor set CS[i]. When a node i received a message FINISH, it sets its TST[i] to the larger one of its original TST[i] and TST[j] + 1. When all the children of node vi finished their transmission, the node vi is ready to compete for the transmission time slot and it will send a message READY(i, r_i) to all nodes in its competitor set. When a node vi received a message READY from another node v_j , it will add the sender j to its ready competitor set RdyCS[i] if j is in CS[i]. When the scheduling ends, all nodes will transmit their data based on TST[i]. In the end, the topology center aggregates all the data and sends the result to the sink node via the shortest path.

3.3 Performance Analysis

In this section we first theoretically prove that the latency of the data aggregation based on our scheduling is at most 16R + $\Delta - 14$, where R is the radius of the network and Δ is the maximum node degree in the original communication graph. We conjecture that the theoretical performance of our centralized and distributed algorithms could actually be much better than 16R+Δ-14, which is supported by our extensive simulations. On the other hand, we also present a network example to show that our centralized algorithm cannot achieve a latency lower than 4R + Δ- 3. It remains a future work to find bad network examples to show that our distributed methods could perform worse than (1 + ε)R for a sufficient small constant ε >0. At last, we present a general lowerbound on the latency of data aggregation for any Step.

3.3.1 Performances of Our Step

First we show that, by using Step 2, for each dominator, the number of neighboring dominatees whose data is not collected is reduced by at least 1. [11]

Claim 1: For Step 2, our schedule will ensure that after every time-slot, for each dominator, the number of neighboring dominatees whose data is not collected is reduced by at least 1.

Proof: First, according to Algorithm 1 each dominator u chooses a dominatee randomly from its neighbors and let the chosen dominatee transmits to u. The selected dominatees are called active transmitters for the set of selected links. Therefore, there are n links transmitting at the same timeslot. If these n links do not conflict with each other, our claim holds. Here two directed links uv and xy conflict with each other if either receiver y is in the interference range of sender u or receiver v is in the interference range of sender x.

If there exists interference among chosen links, there are only 3 possible cases. For each case, it is easy to show that Step 2 re-schedules these two links to avoid the interference while the number of neighboring dominatees whose data is not collected is reduced by at least 1.

Lemma 2: Given a communication graph G of network, under the assumption that the interference range r_I is the same as the transmission range r, Step 1 (aggregating data from dominatees to dominators) costs at most Δ time-slots where Δ is the maximum node degree in G.[12]

Proof: Assume that there are n dominators z_i, (i = 1, 2,..., n) in the network. Each dominator has at most Δ neighboring dominatees. We define a dominators degree as the number of the neighboring dominatees whose data has not been aggregated to dominators yet. At first, each dominators degree is bounded by Δ. Our adjustment (Step 2) repetitively adjusts the set of links whenever there is a pair of conflicting links. Observe that since the recursive nature of our adjustment algorithm, we must prove that our Step will terminate in a limited number of rounds. Clearly, when it terminates, there is no pair of conflicting links, i.e., the remaining links from Step 2 are a valid schedule. In addition, we also have to show that when the adjustment terminates, our schedule can ensure that each dominator's degree indeed will be reduced by at least 1.

We define **Loop Invariant** as: every dominator's degree will be decrease at least 1. We can see that loop invariant is always kept before and after one round of execution (which is exactly Step 2) in the loop, To show that Step 2 terminates, we define a **Potential Function** for a schedule as the cardinality of the set C = $(x_1, x_2) \mid x_1, x_2$are active transmitters and their corresponding links x_1y_1, x_2y_2 are conflicting links. We call the pair $(x_1, x_2)\epsilon C$ a pair of conflicting transmitters. Clearly, the initial cardinality of the set C is at most n(n-1)/2. After one round of re-scheduling, the interference between at least one pair of conflicting transmitters is resolved. Observe that, our adjustment will not introduce new pairs of conflicting transmitters. Thus the potential function will be decreased by at least 1 after 1 round, which means that Step 2 will terminate after at most n(n-1)/2 rounds of execution of the while loop in Step 2.

We now bound the number of connectors that a dominator u will use to connect to all dominators within 2-hops. Our proof is based on a technique lemma.

Lemma 3: Suppose that dominator v and w are within 2-hops of dominator u, v' and w' are the corresponding connectors for v and w respectively. Then either $\mid vw' \mid \leq 1$ or $\mid vw' \mid \leq 1 if \angle vuw \leq 2$ arcsin 1/4.[12]

Lemma 4: In Step 3, a dominator requires at most 12 connectors to connect to all dominators within 2-hops.

Proof: Consider any dominator u, let I_2(u) be the set of dominators within 2-hops of u in the original communication network G. Assume that we have already deleted all the redundant connectors for node u. Let C be the set of connectors left for a dominator u. Then for each remaining connector x C, there is at least one dominator (called a non-sharing dominator) that can only use this connector to connect to u (otherwise, connector x is redundant and thus will be removed). Assume there are 13 connectors in C. Then there are at least 13 nonsharing dominators in I_2(u). From pigeonhole principle, we know that there must have 2 dominators $v_1 and v_2$ such that $\angle v_1uv_2 \leq 2\pi/13 < 2$ arcsin(1/4). Thus, using Lemma 3, v_1 and v_2 will share a common connector in C, which contradicts to the selection of $v_1 and v_2$.

In the rest of the proof, for a dominator u, we use C(u) to denote the set of connectors used to connect all dominators in D_2(u).

Lemma 5: In Step 3, a dominator u in level i can receive the data from all neighboring dominators D_2(u) in at most 16 time-slots.[12]

Proof: Each dominator u will collect the aggregated data from all dominators within 2-hops in lower level. Any connector in C(u) has at most 4 other neighboring dominators, besides u. Similar to the proof of Lemma 2, we can show that it takes at most 4 time-slots for each connector to collect data from those neighboring dominators other than

u. Recall that at most 12 connectors are needed for u to reach all dominators in D_2(u). Thus, it will take at most 12 time-slots for the dominator u to collect data from all these connectors. Consequently, within at most 12 + 4 = 16 time-slots, every dominator u can collect the aggregated data from all the dominators in D_2(u).

Theorem 6: By using Step 3, the sink can receive all the aggregated data in at most 17R + Δ - 16 time-slots.[13]

Proof: Every dominatee's data can be aggregated to a dominator within Δ time-slots from Lemma 2. Observe that every dominator, except the root of the data aggregation tree T , connects to at least one dominator in the upper level within 2-hops. Then Step 3 ensures that every dominator's data can be aggregated upwards the root finally. For each level of BFS, every dominator u including the root of data aggregation tree T , can collect aggregated data from all dominators in D_2(u) within at most 16 time-slots by Lemma 5. Since there is no dominator in level 1, after at most 16(R-1) time-slots, every dominator's data can be aggregated to the root. The root then uses at most R time-slots to transmit data to the original sink node via the shortest path. Therefore within 17R+Δ-16 time-slots, all the data can be be aggregated to the sink node.

Next, we provide a revised schedule that only need 15 timeslots for dominators in level i (i $\geq$ 2) to aggregate data from some dominators within 2-hops, which can also ensure that data will be aggregated to the root finally. This means that we can reduce our latency by R - 1 time-slots totally.

For a dominator u other than the root, we denote all dominators within 2-hops of u as B_2(u). Notice that B_2(u) includes at least one dominator v located in upper level of u. By Lemma 4, u needs at most 12 connectors to connect to B_2(u), we denote the set of at most 12 connectors as C(u). There must exists a connector w ϵC(u) which connects u to v. Then all dominators in B_2(u) that are connected to w are also 2-hop neighbors of the dominator v, we denote the set of these dominators as B'_2(u), thus $B'_2(u) \subset B_2(v)$. Clearly all data in B'_2(u) can be collected by v, it is not necessary for them to be collected by u. So we let u only collect the data in B_2(u) $B'_2(u)$. It requires at most 11 connectors (all the connectors in C(u) w) to connect to the dominators in B_2(u) B'_2(u). So at most 15 (= 4 + 11) time-slots is required for u to aggregate the data from B_2(u) B'_2(u). If every dominator u other than the root aggregate the data from B_2(u) B'_2(u), all the data can be aggregated to the root.

Theorem 7: By using Step 3, the sink can receive all the aggregated data in at most 16R + Δ- 14 time-slots.[14]

Proof: Similar to the proof of Theorem 6, we need Δ time-slots for dominators to aggregate data from dominatees. After that, for each level of BFS, every dominator u, other than the root of the data aggregation tree T , can collect aggregated data from all dominators in $B_2(u)\backslash B'_2(u)$ in at most 15 timeslots as stated above. Thus it costs at most 15(R-2) for data to be aggregated to the dominators in level 2. The root r_T can collect the aggregated data from dominators in level 2 within 16 time-slots. Therefore within 15(R-2) + 16 timeslots, every dominatorfs data can be aggregated to the root. The root then transmit the result to the original sink node in R time-slots. In all, within 16R + Δ - 14 time-slots, all the data can be aggregated to the sink node.

Observe that, although our analysis is based on the centralized method, it is easy to show that all results carry to the distributed implementation (Step 4). Consequently, we have

Theorem 8: By using Step 4, the sink can receive all the aggregated data in at most 16R + Δ - 14 time-slots.[14]

In-network aggregation means computing and transmitting partially aggregated data rather than transmitting raw data in networks, thus reducing the energy consumption.

There are a lot of existing researches on in-network aggregation. Suppression scheme and model-driven approach were proposed towards reducing communication cost. The tradeoff between energy consumption and time latency was considered. A heuristic algorithm for both broadcast and data aggregation was designed. Another heuristic algorithm for data aggregation was proposed, aiming at reducing time latency and energy consumption. Kesselman et al. proposed a randomized and distributed algorithm for aggregation in wireless sensor networks with an expected latency of O(log n). Their method are based on two assumptions: One is that sensor nodes can adjust their transmission range without any limitation. The other is that each sensor node has the capability of detecting whether a collision occurs after transmitting data. Both assumptions pose some challenges for hardware design and is impractical when the network scales. A collision-free scheduling method for data collection is proposed, aiming at optimizing energy consumption and reliability. All these work did not discuss the minimal-latency aggregation scheduling problem.

In addition, the minimum latency of data aggregation problem was proved NP-hard and a (Δ- 1)-approximation algorithm was proposed, where Δ is the maximum degree of the network graph. Another aggregation scheduling algorithm was proposed, which has a latency bound of 23R + Δ + 18, where R is the network radius and Δ is the maximum degree. All the algorithms mentioned above are centralized. In many cases centralized algorithms are not practical, especially when the network topology changes often in a large sensor network.

In data collection, since data cannot be merged, the sink must receive N packets from all the nodes, where N is the number of sensor nodes in the network. Thus the lower-bound of latency is N. The upper bound of the time latency of this algorithm is max(3_{n_k} - 1,N), where nk is the number of nodes in the largest one-hop-subtree. A distributed scheduling algorithm generating collision-free schedules that has a latency bound of 24D+ 6Δ+16, where D is the network diameter which is the best result for minimum latency of data aggregation so far.

4

Simulators Study

Applying wireless sensor network (WSN) represents a relatively new approach in the field of wireless communication. The network consists of many low-cost sensor nodes and one or more base stations (BS) which gather the sensed data for further processing. WSN is a highly distributed network and the number of communicating sensor nodes may vary from units to thousands according to the area of interest and size of the environment. The sensors may monitor many attributes like temperature, humidity, light, leakage of chemical compounds or anything that can be sensed. WSN may be utilized in various fields like environmental protection, guarding of buildings, military purposes, factories.

4.1 Simulator and Emulator

Simulator is universally used to develop and test protocols of WSNs, especially in the beginning stage of these designs. The cost of simulating thousands of nodes networks is very low, and the simulation can be finished within very short execution time. Both general and specialized simulators are available for uses to simulate WSNs. The tool, which is using firmware as well as hardware to perform the simulation, is called emulator. Emulation can combine both software and hardware implementation. Emulator implements in real nodes, thus it may provide more precision performance. Usually emulator has highly scalability, which can emulate numerous sensor nodes at the same time. In this survey, seven simulation tools are also categorize into this two types.

4.1.1 NETWORK SIMULATOR

NETWORK SIMULATOR is the abbreviation of Network simulator version two, which first been developed by 1989 using as the REAL network simulator. Now, NETWORK SIMULATOR is supported by Defense Advanced Research Projects Agency and National Science Foundation. NETWORK SIMULATOR is a discrete event network simulator built in Object-Oriented extension of Tool Command Language and C++. People can run NETWORK SIMULATOR simulator on Linux Operating Systems or on Cygwin, shown in Figure 3, which is a Unix-like environment and command-line interface running on Windows. NETWORK SIMULATOR is a popular non-specific network simulator can used in both wire and wireless area. This simulator is open source and provides online document.

NETWORK SIMULATOR contains both merits and limitations when people use it to simulate WSNs. To the merits, firstly as a non-specific network simulator, NETWORK SIMULATOR can support a considerable range of protocols in all layers. For example, the ad-hoc and WSN specific protocols are provided by NETWORK SIMULATOR. Secondly, the open source model saves the cost of simulation, and online documents allow the users easily to modify and improve the codes.

However, this simulator has some limitations. Firstly, people who want to use this simulator need to familiar with writing scripting language and modeling technique; the Tool Command Language is somewhat difficulty to understand and write. Secondly, sometimes using NETWORK SIMULATOR is more complex and time-consuming than other simulators to model a desired job. Thirdly, NETWORK SIMULATOR provides a poor graphical support, no Graphical User Interface (GUI) ; the users have to directly face to text commands of the electronic devices. Fourthly, due to the continuing changing the code base, the result may not be consistent, or contains bugs.

In addition, since NETWORK SIMULATOR is originally targeted to IP networks but WSNs, there are some limitations when apply it to simulate WSNs. Firstly, NETWORK SIMULATOR can simulate the layered protocols but application behaviors. However, the layered protocols and applications interact and can not be strictly separated in WSNs. So, in this situation, using NETWORK SIMULATOR is inappropriate, and it can hardly to acquire correct results. Secondly, because NETWORK SIMULATOR is designed as a general network simulator, it does not consider some unique characteristics of WSN. For example, NETWORK SIMULATOR can not simulate problems of the bandwidth, power consumption or energy saving in WSN. Thirdly, NETWORK SIMULATOR has a scalability problem in WSN, it has trouble to simulate more than 100 nodes. As the increasing of the number of nodes, the tracing files will be too large to management. Finally, it is difficult to add new protocols or node components due to the inherently design of NETWORK SIMULATOR. In sum, NETWORK SIMULATOR as a simulator of WSN contains both advantages and disadvantages.[16]

4.1.2 TOSSIM

TOSSIM is an emulator specifically designed for WSN running on TinyOS, which is an open source operating system targeting embedded operating system. In 2003, TOSSIM was first developed by UC Berkeleys TinyOS project team. TOSSIM is a bit-level discrete event network emulator built in Python[Python], a high-level programming language emphasizing code readability, and C++. People can run TOSSIM on Linux Operating Systems or on Cygwin on Windows. TOSSIM also provides open sources and online documents.

TOSSIM contains both merits and limitations when people use it to emulate WSNs. To the merits, the open source model free online document save the emulation cost. Also, TOSSIM has a GUI, TinyViz, which is very convenience for the user to interact with electronic devices because it provides images instead of text commands.

In addition, TOSSIM is a very simple but powerful emulator for WSN. Each node can be evaluated under perfect transmission conditions, and using this emulator can capture the hidden terminal problems. As a specific network emulator, TOSSIM can support thousands of nodes simulation. This is a very good feature, because it can more accurately simulate the real world situation. Besides network, TOSSIM can emulate radio models and code executions. This emulator may be provided more precise simulation result at component levels because of compiling directly to native codes.

However, this emulator still has some limitations. Firstly, TOSSIM is designed to simulate behaviors and applications of TinyOS, and it is not designed to simulate the performance metrics of other new protocols. Therefore, TOSSIM can not correctly simulate issues of the energy consumption in WSN; people can use PowerTOSSIM, another TinyOS simulator extending the power model to TOSSIM, to estimate the power consumption of each node. Secondly, every node has to run on NesC code, a programming language that is event-driven, component-based and implemented on TinyOS, thus TOSSIM can only emulate the type of homogeneous applications. Thirdly, because TOSSIM is specifically designed for WSN simulation, motes-like nodes are the only thing that TOSSIM can simulate. In sum, TOSSIM as an emulator of WSN contains both advantages and disadvantages.[16]

4.1.3 SensorSim

SensorSim is a discrete event network simulator built in Java. This simulator provides GUI library, which facilities users to model or compile the Mathematical Modeling Language, a text-based language written to Sensorsim models. Sensorsim provides open source models and online documents. This simulator is commonly used in physiology and biomedicine areas, but it also can be used in WSN simulation. In addition, Sensorsim can simulate real-time processes. It contains both merits and limitations when people use it to simulate WSNs. To the merits, firstly, models in Sensorsim have good reusability and interchangeability, which facilities easily simulation. Secondly, Sensorsim contains large number of protocols; this simulator can also support data diffusions, routings and localization simulations in WSNs by detail models in the protocols of SensorSim. SensorSim can simulate radio channels and power consumptions in WSNs. Thirdly, SensorSim provides a GUI library, which can help users to trace and debug programs. The independent platform is easy for users to choose specific components to solve the individual problem. Fourth, comparing with NETWORK SIMULATOR, SensorSim can simulate larger number of sensor nodes, around 500, and SensorSim can save lots of memory sizes. However, this simulator has some limitations. The execution time is much longer than that of NETWORK SIMULATOR. Because SensorSim was not originally designed to simulate WSNs, the inherently design of SensorSim makes users hardly add new protocols or node components.[16]

4.1.4 ATEMU

ATEMU is an emulator of an AVR processor for WSN built in C; AVR is a single chip microcontroller commonly used in the MICA platform. ATEMU provides GUI, Xatdb; people can use this GUI to run codes on sensor nodes, debug codes and monitor program executions. People can run ATEMU on Solaris and Linux operating system. ATEMU is a specific emulator for WSNs; it can support users to run TinyOS on MICA2 hardware. ATEMU can emulate not only the communication among the sensors, but also every instruction implemented in each sensor. This emulator provides open sources and online documents.

ATEMU contains both merits and limitations when people use it to simulate wireless sensor network. To the merits, firstly, ATEMU can simulate multiple sensor nodes at the same time, and each sensor node can run different programs. Secondly, ATEMU has a large library of a wide rage of hard devices. Thirdly, ATEMU can provide a very high level of detail emulation in WSNs. For example, it can emulate different sensor nodes in homogeneous networks or heterogeneous networks. ATEMU can emulate different application run on MICA. Also users can emulate power consumptions or radio channels by ATEMU. Fourthly, the GUI can help users debug programs, and monitor program executions. The open source saves the cost of simulation. ATEMU can provide an accurate model, which helps users to give unbiased comparisons and get more realistic results. The ATEMU components architecture is shown in Figure 6. However, this emulator also has some limitations. For instance, although ATEMU can give a highly accuracy results, the simulation time is much longer than other simulation tools. In addition, ATEMU has fewer functions to simulate routing and clustering problems. Therefore, both merits and limitation contains in ATEMU.[16]

4.1.5 Avrora

Avrora is a simulator specifically designed for WSNs built in Java. Similar to ATEMU, Avrora can also simulate AVR-based microcontroller MICA2 sensor nodes. This simulator was developed by University of California, Los Angeles Compilers Group. Avrora provides a wide range of tools that can be used in simulating WSNs. This simulator combines the merits of TOSSIM and ATEMU, and limits their drawbacks. Avrora does not provide

GUI. Avrora also supports energy consumption simulation. This simulator provides open sources and online documents. However, this simulator has some drawbacks. It does not have GUI. In addition, Avrora can not simulate network management algorithms because it does not provide network communication tools.

Avrora contains both merits and limitations when people use it to simulate WSNs. To the merits, firstly, Avrora is an instruction-level simulator, which removes the gap between TOSSIM and ATEMU. The codes in Avrora run instruction by instruction, which provides faster speed and better scalability. Avrora can support thousands of nodes simulation, and can save much more execution time with similar accuracy. Avrora provides larger scalability than ATEMU does with equivalent accuracy; Avrora provides more accuracy than TOSSIM does with equivalent scales of sensor nodes. Unlike TOSSIM and ATEMU, Avrora is built in Java language, which provides much flexibility. Avrora can simulate different programming code projects, but TOSSIM can only support TinyOS simulation. [16]

4.1.6 Castalia

This section deals with the simulator OMNeT++ and especially with its extension Castalia, version 3.0. Castalia is based on OMNeT++ platform and is developed for networks of lowpower embedded devices such as wireless sensor nodes. The main declared advantages are advanced channel model based on empirically measured data and radio model based on real radios for low-power communication and monitoring of the power consumption.

The authors of Castalia claim that their simulator is the most realistic, among the others, concerning the wireless channel. It takes into account various important features which are discussed bellow. Castalia is a tunable simulator where many parameters may be adjusted using input files to simulate real environment. Several examples of simulation applications are provided.

Three different interference models can be used. In the first one there, is no collision happening at all. The second model always produces a collision at the receiver if that receiving node receives two or more signals at the same time (even though some of them have minimal strength). The last option is to use additive interference model where the strongest transmission is chosen to receive. In this case, the success depends on the SINR and parameters of the simulated radio.

Castalia is a simulator for Wireless Sensor Networks (WSN), Body Area Networks (BAN) and generally networks of low-power embedded devices. It is based on the OMNET++ platform.Castalia can also be used to evaluate different platform characteristics for specific applications, since it is highly parametric, and can simulate a wide range of platforms. The main features of Castalia are [8]: Advanced channel model based on empirically measured data.

- Model defines a map of path loss, not simply connections between nodes.
- Complex model for temporal variation of path loss.
- Probability of reception based on SINR, packet size, modulation type. PSK FSK supported, custom modulation allowed by defining SNR-BER curve.
- Highly exible physical process model.
- Sensing device noise, bias, and power consumption.
- MAC and routing protocols available.
- Designed for adaptation and expansion.
- Node clock drift.
- Extended sensing modelling provisions.
- Multiple TX power levels with individual node variations allowed.
- States with different power consumption and delays switching between them.

OMNET's basic concepts are modules and messages. A simple module is the basic unit of execution. It accepts messages from other modules or itself, and according to the message, it executes a piece of code. The code can keep state that is altered when messages are received and can send new messages. There are also composite modules. A composite module is just a construction of simple and/or other composite modules. Figure 4.1 The modules and their connections in Castalia from reference [8].

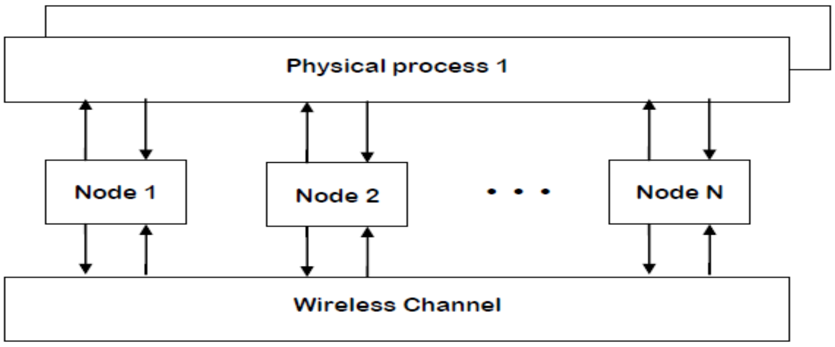

Figure 4.1: The modules and their connections in Castalia from reference [15]

In this figure nodes do not connect to each other directly but through the wireless channel modules. The arrows signify message passing from one module to another. When a node has a packet to send this goes to the wireless channel which then decides which nodes should receive the packet. The nodes are also linked through the physical processes that they monitor. The nodes sample the physical process in space and time by sending a message to the corresponding module, to get their sensor readings. There can be multiple physical processes, representing the multiple sensing devices that a node has. The node module is a composite one. Implementation in Castalia is done with the use of the OMNET++ NED language. With this language we can easily define modules, i.e., define a module name, module parameters, and module interface (gates in and gates out) and possible submodule structure.Files with the suffx ".ned" contain NED language code. The Castalia structure is also reflected in the hierarchy of directories in the source code. Every module corresponds to a directory which always contains a .ned file that defines the module. If the module is composite then there are subdirectories to define the submodules. If it is a simple module then there is C++ code (.cc, .h files) to define its behavior. This complete hierarchy of .ned files defines the overall structure of the Castalia simulator.

Castalia has a modular structure with many interconnecting modules. Each one of the modules has one or more parameters that affect its behavior. An NED file (file with extension .ned in the Castalia source code) defines the basic structure of a module by defining its input/output gates and its parameters. In the NED file we can also define default values for the parameters. A configuration file (usually named omnetpp.ini and residing in the Simulations dir tree) assigns values to parameters, or just reassigns them to a different value from their default one. This way we can build a great variety of simulation scenarios. Open the file Simulations/radioTest/omnetpp.ini.In this file parameters such as the simulation time,number of nodes and field size values must be defined in this section.

[General] include ../Parameters/Castalia.ini
sim-time-limit = 100s
SN.field$\_x = 200\#$ **meters**
SN.field$\_y = 200\#$ **meters**
SN.numNodes = 3

SN is the topmost composite module. The name SN stands for Sensor Network.The parameter SN.deployment is a string that describes where the nodes are placed on the field. When defined, SN.deployment can be one of the following types:

- **uniform:** nodes are placed in the field using a random uniform distribution.
- **NxM:** N and M are integer numbers. Nodes placed in a grid of N nodes by M nodes.
- **NxMxK:** same as above but for 3 dimensions.

Below two lines set 2 parameters of the Radio module. The RadioParametersFile is a specially formatted file defining the basic operational properties of a radio. The second parameter, TxOutputPower sets the power that the radio transmits its packets. We can access all the radio modules in all the nodes with the use of [*].

- SN:node[*]:Communication:Radio:RadioParametersFile = "Radio=CC2420:txt"
- SN:node[*]:Communication:Radio:TxOutputPower = " - 5dBm"

5

Simulation and Result Analysis

5.1 Simlution Results of Simple Aggregation

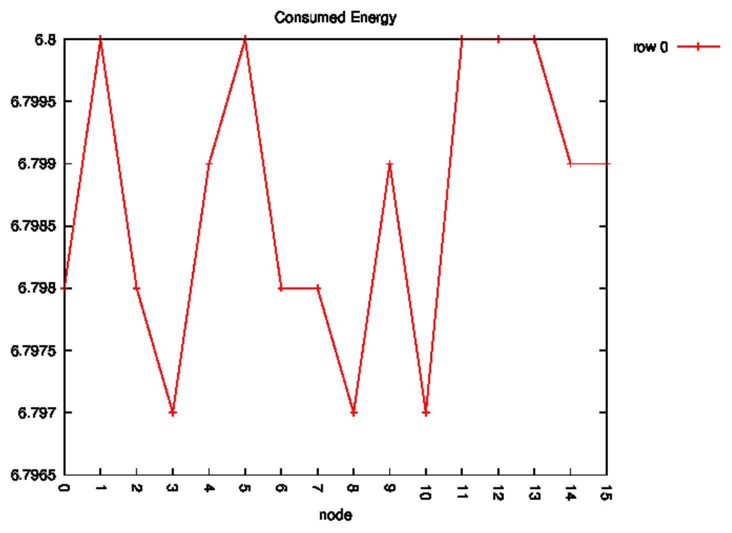

Figure 5.1: Simulation Result of Energy on 16 nodes in 60x60 meters

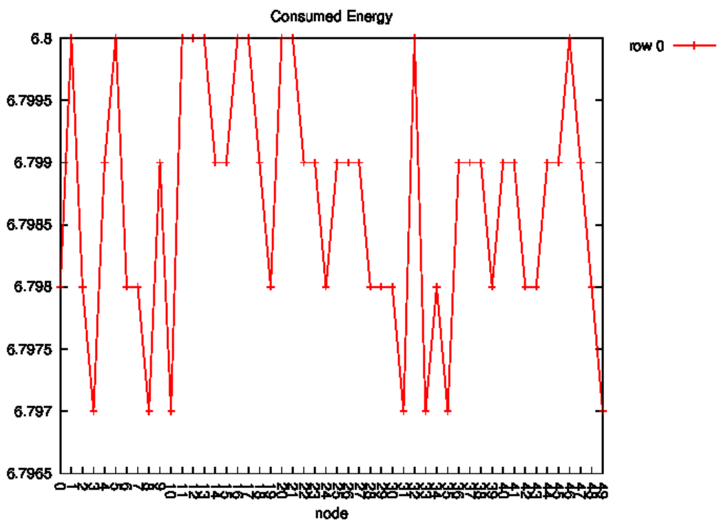

Figure 5.2: Simulation Result of Energy on 50 nodes in 100x100 meters

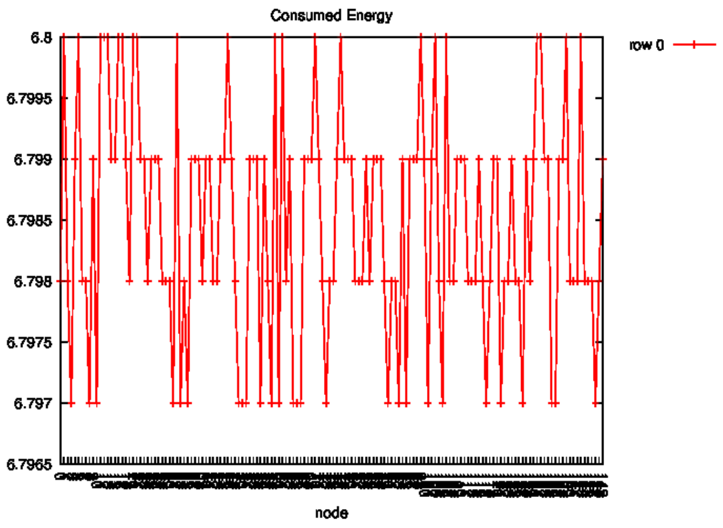

Figure 5.3: Simulation Result of Energy on 100 nodes in 150x150 meters

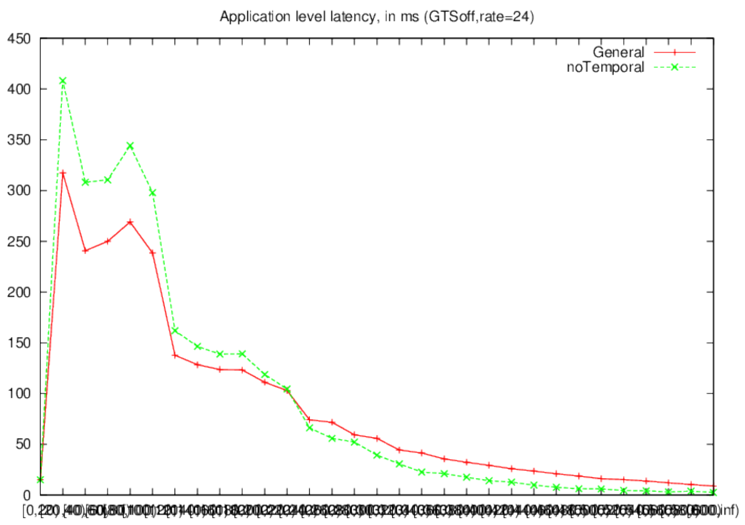

Figure 5.4: Simulation Result of Latency on 16 nodes in 60x60 meters

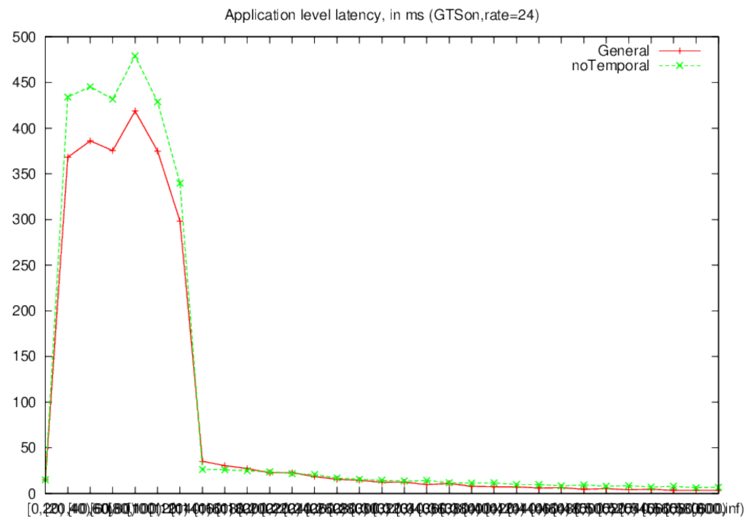

Figure 5.5: Simulation Result of Latency on 50 nodes in 100x100 meters

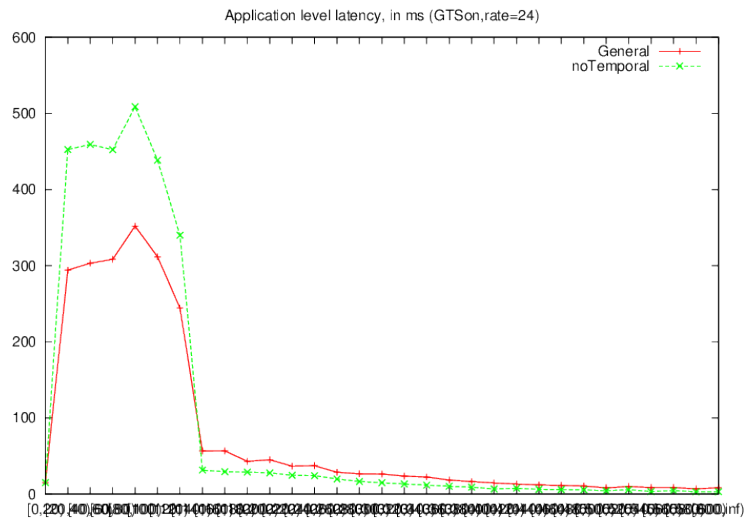

Figure 5.6: Simulation Result of Latency on 00 nodes in 150x150 meters

5.2 Simlution Results With Aggregation

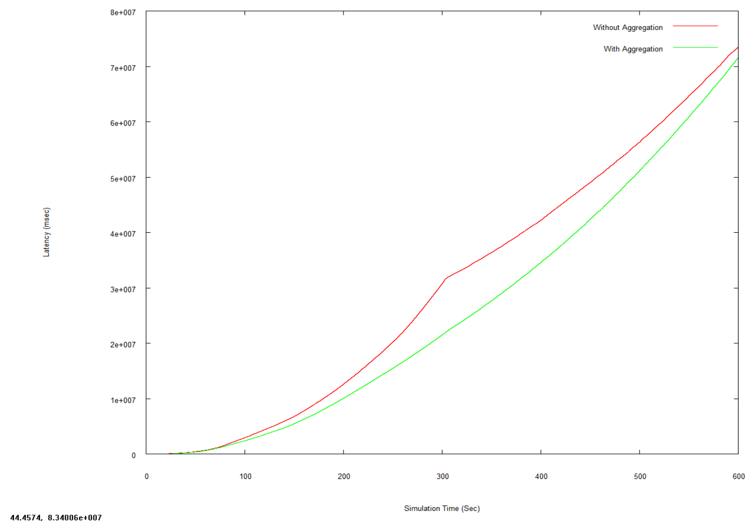

Figure 5.7: Simulation Result of Latency on 10 nodes in 500x500 meters

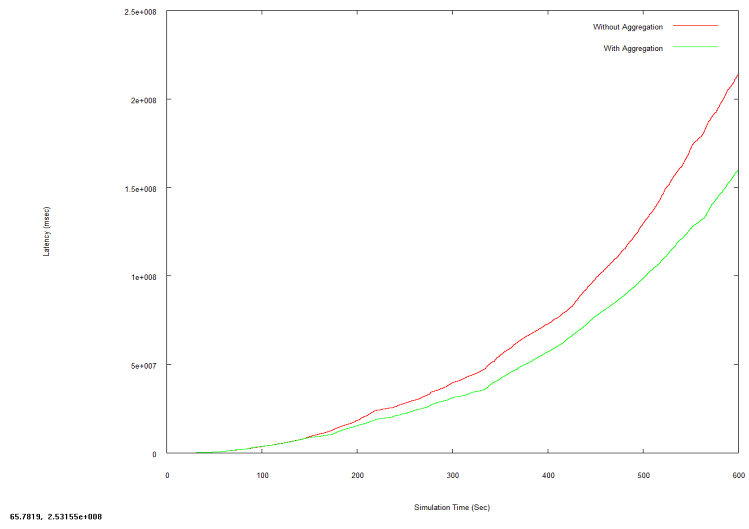

Figure 5.8: Simulation Result ofLatency on 25 nodes in 1000x1000 meters

5.3 Simlution Results With Event Aggregation

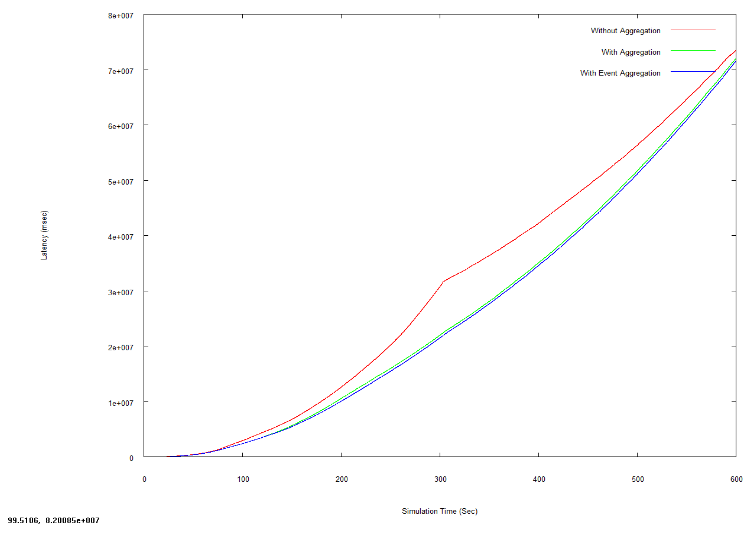

Figure 5.9: Simulation Result of Latency on 10 nodes in 500x500 meters

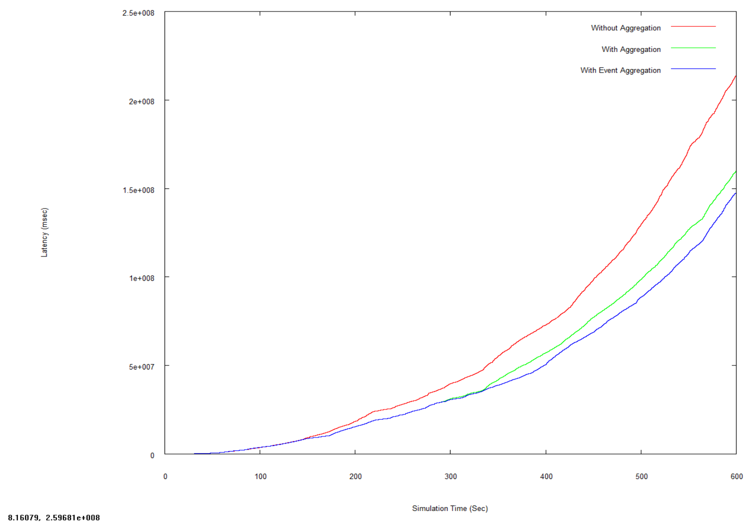

Figure 5.10: Simulation Result of Latency on 25 nodes in 1000x1000 meters

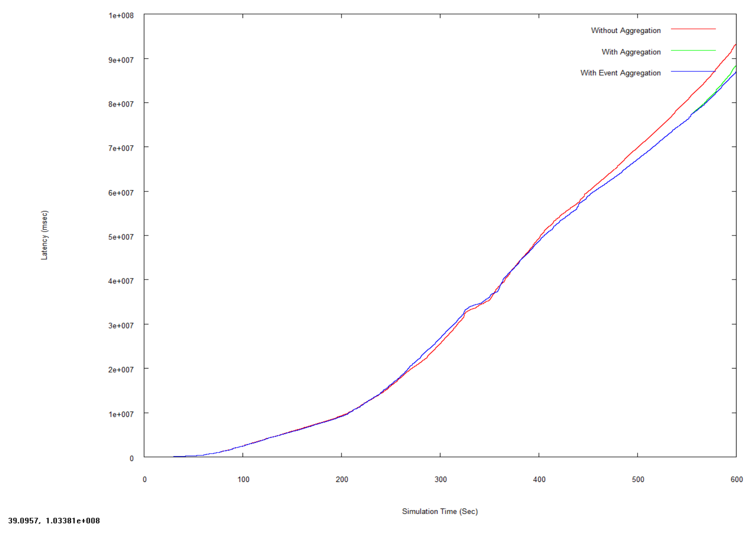

Figure 5.11: Simulation Result of Latency on 50 nodes in 1500x1500 meters

6

Conclusion and Future work

In this report, We mainly focused on data aggregation first. the first two algorithms gave good results on energy but have tradeoff with heigherdelay. We proposed solutions to this problem including a simple approach to describe aggregation function, the DBEA algorithm considering both latency constraint and aggregation function for single high-level event, and the optimal base events selection algorithm for aggregating multiple high-level events given a reliable constraint. Also tried to ckeck delay performance of aggregation and scheduling techniques with numerical techniques. The drawback of this work is that first two algorithms need an already constructed broadcast tree in the network to work. So we fed the simulator with a priori constructed broadcast tree topologies.We evaluate our analysis using binary trees with different weight assignments. This analysis is accurate, in particular when the network size is small. We also evaluate the performance of several myopic and non-myopic scheduling policies, and show that the one-step non-myopic scheduling policies achieve substantially lower delays than the conventional myopic ones. In particular, the non-myopic GMM seems to achieve good performance with moderate complexity making it attractive as a viable alternative over the optimal solution.One can obtain analytic results when the underlying tree networks have a larger width, and develop a simple distributed algorithm with high performance.

So we propose and efficinet distributed method that produces a collision freeschedule for data aggregation in WSNs. We theoritically prove that latancy of data aggregation schedule generated by our algorithm is at most 16R + Δ - 14 timeslots. Here R is network radius and Δ is the maximum node degree in the communication graph of the original network.

Bibliography

[1] Mr. BHABANI PRASADV, Mrs. P.CHITRAKALA,*"A Fast Convergecast Method for Tree-Based Wireless Sensor Networks,"* Electronics Research Laboratory Memorandum Number M98/2, 2009.

[2]]Claudio Barone, Fabrizio Ciarlo, Sebastiano Testa*" Energy Efficiency and Traffic Optimization through Data Aggregation in Wireless Sensor Networks,"* in Proc. EEE MEMS Workshop, Nagoya Jan. 2009, pp. 350355.

[3] K.-W. Fan, S. Liu,*"Structure-free Data Aggregation in Sensor Networks,"* journal IEEE Transactions on Mobile Computing, vol. 6 Issue 8. August 2007.

[4] Weiping Zhu, Jiannong Cao, Yi Xu," Event Aggregation with Different Latency Constraints and Aggregation Functions in Wireless Sensor Networks" IEEE ICC 2011.

[5] H. Luo, Y. Liu and S. K. Das, Routing Correlated Data with Fusion Cost in Wireless Sensor Networks, IEEE Trans. on Mobile Computing, vol. 5, no.11, pp. 1620-1632, 2006.

[6] Changhee Joo, Jin-Ghoo Choi, and Ness B. Shroff, "Delay Performance of Scheduling with Data Aggregation in Wireless Sensor Networks" IEEE INFOCOM 2010.

[7] L. Becchetti, P. Korteweg, A. Marchetti-Spaccamela, M. Skutella, L. Stougie, and A. Vitaletti, Latency Constrained Aggregation in Sensor Networks, in the 14th conference on Annual European Symposium, 2006.

[8] Athanassios Boulis, "Castalia A simulator for Wireless Sensor Networks and Body Area Networks"Ver 3.1,3.2 NICTA December 2010

[9] Amitabha Ghosh, Ozlem Durmaz Incel, V.S. Anil Kumar, and Bhaskar Krishnamachari , "Algorithms for Fast Aggregated Convergecast in Sensor Networks" University of Southern California 2007.

[10] A. Boukerche, Chatzigiannakis and S. Nikoletseas, A New Energy Efficient and Fault-tolerant Protocol for Data Propagation in Smart DustNetworks using Varying Transmission Range, in the Computer Communications Journal, Elsevier, 2005.

[11] XiaoHua Xu, ShiGuang Wang, XuFei Mao, ShaoJie Tang, Ping Xu and XiangYang Li, "Efficient Data Aggregation in Multi-hop WSNs", 2008.

[12] A.Sivagami, K. Pavai and D. Sridharan, "Latency Optimized Data Aggregation Timing Model for WirelessSensor Networks" IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 6, May 2010.

[13] Bo Yu, Jianzhong Li, Yingshu Li, "Distributed Data Aggregation Scheduling in Wireless Sensor Networks" IEEE INFOCOM 2009.

[14] Shashidhar Gandham, Ying Zhang and Qingfeng Huang, "Distributed Minimal Time Convergecast Scheduling in Wireless Sensor Networks" , IEEE International Conference on Distributed Computing Systems 2008.

[15] http://castalia.npc.nicta.com.au/pdfs/castalia%20-%20user%20MAnual.pdf

[16] E. Egea-Lpez, J. Vales-Alonso, A. S. Martnez-Sala, P. Pavn-Mario, J. Garca-Haro "Simulation Tools for Wireless Sensor Networks" Summer Simulation Multiconference - SPECTS 2005