IN-NETWORK VS CENTRALIZED PROCESSING
FOR
LIGHT DETECTION SYSTEM
USING
WIRELESS SENSOR NETWORKS

A PROJECT REPORT
SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science
in
Computer Science

Department of Computer Science
Lamar University
Beaumont, TX

Project Supervisor
Dr. Lawrence Osborne
Professor and Chair, Department of Computer Science

Nov 9, 2007

© Desai, Bhairav and Solanki, Arpan
2007
ROOM OCCUPANCY DETECTION USING WIRELESS SENSOR NETWORKS

BHAIRAV. DESAI.

ARPAN. SOLANKI.

Approved:

________________________________________________________________________
Dr. Lawrence Osborne
Supervising Professor

________________________________________________________________________
Dr. Dehu Qi
Committee Member

________________________________________________________________________
Dr. Jiangjiang Liu
Committee Member

Dr. Lawrence Osborne
Chair, Department of Computer Science
Acknowledgments

We are extremely thankful to Dr. Lawrence Osborne, Chair of the Department of Computer Science at Lamar University for his guidance and support throughout this project. We are also thankful to him for giving us this opportunity to work on this project under him.

We are also thankful to the Department of Computer Science at Lamar University, for providing us with all the facilities needed for this project, including the hardware equipments needed to set up the test bed. We offer a special thanks to the people of TinyOS mailing list, who provided feedbacks on our queries of TinyOS and Wireless Sensor Networks.

We are grateful to the lab administrators with the Department of Computer Science at Lamar University. Finally, we are thankful to Dr. Bo Sun, Mr. Frank Sun and Ms. Donna Blaisdell.
# Table of Content

## CHAPTER 1  INTRODUCTION ...........................................................................................1

1.1 OBJECTIVE...................................................................................................................... 1

1.2 MOTIVATION .................................................................................................................. 2

1.3 APPROACH...................................................................................................................... 3

## CHAPTER 2  BACKGROUND ..............................................................................................4

2.1 DATA CENTRIC NETWORKING ....................................................................................... 4

2.2 LEACH (LOW-ENERGY ADAPTIVE CLUSTERING HIERARCHY) ................................. 5

2.3 EXPANDING RING SEARCH.............................................................................................. 7

2.4 ADAPTIVE AND DECENTRALIZED OPERATOR PLACEMENT FOR IN-NETWORK QUERY PROCESSING ............................................................................................................................... 8

2.5 RUMOR ROUTING .......................................................................................................... 10

2.6 DATA PROCESSING TECHNIQUES ................................................................................. 11

   2.6.1 *In-network aggregation* ........................................................................................... 11

   2.6.2 *Centralized aggregation* .......................................................................................... 12

   2.6.3 *In-network vs. Centralized aggregation* .................................................................. 12

## CHAPTER 3  METHODOLOGY .............................................................................................15

3.1 FORMATION OF TOPOLOGY .......................................................................................... 15

3.2 IN-NETWORK AGGREGATION ...................................................................................... 18

   3.2.1 *Node discovery phase* ........................................................................................... 19

   3.2.2 *Aggregation phase* ................................................................................................... 22

   3.2.3 *Network response on topology change* ................................................................. 25

3.3 CENTRALIZED AGGREGATION ...................................................................................... 27

## CHAPTER 4  APPLICATION DESIGN, INSTALLATION, AND DEPLOYMENT ........30

4.1 APPLICATION DESIGN .................................................................................................. 30
4.2 TINYOS INTERFACES ................................................................. 31
4.3 PACKET STRUCTURE ............................................................... 33
4.4 INSTALLATION ................................................................. 34
  4.4.1 Application Files......................................................... 34
  4.4.2 Makefile ..................................................................... 36
  4.4.3 Makelocal File .............................................................. 37
  4.4.4 Message Interface Generator (MIG): ....................... 37
4.5 DEPLOYMENT: ................................................................. 38

CHAPTER 5  APPLICATION, RESULTS AND ANALYSIS....39
  5.1 APPLICATION ................................................................. 39
  5.2 TEST BED ................................................................. 39
  5.3 EXPERIMENTS, RESULTS AND ANALYSIS ................. 40
    5.3.1 Experiment 1: Effect of various epoch durations on packet loss ......... 40
    5.3.2 Experiment 4: Effect of various epoch duration on power consumption in in-
      network aggregation ................................................................. 43
    5.3.3 Experiment 3: Measuring power consumption in Centralized aggregation .... 46
    5.3.4 Experiment 4: Comparison of number of transmissions in the network .... 47
    5.3.5 Experiment 5: Comparison of number of bytes transmitted ............... 48
  5.4 APPLICATION RESULTS ANALYSIS ......................................... 49

CHAPTER 6  FUTURE WORK AND CONCLUSION ................. 50
  6.1 CONCLUSION ................................................................. 50
  6.2 FUTURE WORK ................................................................. 50

CHAPTER 7  BIBLIOGRAPHY................................................................. 53

APPENDIX A – TINYOS AND NESC ................................................. 56
**Table of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2-1</td>
<td>Cluster based routing technique LEACH</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2-2</td>
<td>Expanding ring search</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2-3</td>
<td>Operator placement</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2-4</td>
<td>Rumor routing</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2-5</td>
<td>Centralized Aggregation</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3-1</td>
<td>CTP Data Frame</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3-2</td>
<td>CTP Routing Frame</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3-3</td>
<td>Flow of network behavior</td>
<td>18</td>
</tr>
<tr>
<td>Figure 3-4</td>
<td>Processing plans</td>
<td>22</td>
</tr>
<tr>
<td>Figure 3-5</td>
<td>State transition diagrams (In-network aggregation)</td>
<td>24</td>
</tr>
<tr>
<td>Figure 3-6</td>
<td>Sensing nodes divided in groups</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3-7</td>
<td>Change in topology causing change in depth</td>
<td>26</td>
</tr>
<tr>
<td>Figure 3-8</td>
<td>State transition diagrams (Centralized aggregation)</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5-1</td>
<td>Hop count per epoch for t = 2000 ms</td>
<td>41</td>
</tr>
<tr>
<td>Figure 5-2</td>
<td>Hop count per epoch for t = 4000 ms</td>
<td>42</td>
</tr>
<tr>
<td>Figure 5-3</td>
<td>Hop count per epoch for t = 6000 ms</td>
<td>42</td>
</tr>
<tr>
<td>Figure 5-4</td>
<td>Voltage drop per epoch when t = 2000 ms</td>
<td>43</td>
</tr>
<tr>
<td>Figure 5-5</td>
<td>Voltage drop per epoch when t = 4000 ms</td>
<td>44</td>
</tr>
<tr>
<td>Figure 5-6</td>
<td>Voltage drop per epoch when t = 6000 ms</td>
<td>45</td>
</tr>
<tr>
<td>Figure 5-7</td>
<td>Power consumption in centralized aggregated network</td>
<td>46</td>
</tr>
<tr>
<td>Figure 5-8</td>
<td>Comparison of number of transmission per epoch</td>
<td>47</td>
</tr>
<tr>
<td>Figure 5-9</td>
<td>Comparison of total bytes transmitted</td>
<td>48</td>
</tr>
</tbody>
</table>
Chapter 1  Introduction

1.1  Objective

Wireless networking is an emerging field that provides the ease of communication between two nodes without the hassle of wired arrangements and installation costs. Such networks include cell phones, laptops, and PDAs. Motes are devices that transmit data over a radio frequency to communicate with each other. Motes can be equipped with sensors, which sense the data from the environment. Such data include light, temperature, and magnetic fields. These sensed data can be transmitted by the motes over the radio frequency to any desired location. A network that is consisting of motes, often equipped with sensors, is called a wireless sensor network.

Although wireless networks require no installation costs, motes require more power to transmit the data over the air, than through any other physical medium (Madden et al. 2002). The amount of power consumed by a mote depends highly on the amount of data transmitted over the radio frequency. Further, the transmission cost is more than the computation cost (Yao and Gehrke, 2002). A mote draws 12 mA of current while transmitting and 1.8 mA of current while receiving. This current is significant in comparison to the 5uA of the current that it draws when idle (Yao and Gehrke, 2002). Thus, most of the power is used in transmitting and receiving the data. Considering the limited energy resources of the motes, reducing energy consumption is a significant challenge for designers of wireless networks.

In wireless sensor networks, data is sensed frequently. In the most cases, this frequently sensed data needs to reach at a central location, so either the user of the network or some automated system or artificial intelligent system can analyze the data. In wireless sensor networks terminology, this central location is called the base station. It is possible that the sensing node and the base station are at such distant locations that the node cannot transmit the data directly to the
base station. In such cases, data needs to pass through a number of nodes to reach the base station after getting transmitted from the originating node. Most applications require some aggregation to be performed on the collected data.

Wireless motes lack energy resources. Developers always strive to build applications that consume low energy. The primary objective of this project is to develop a system that is not only energy efficient but is also resistant to the changing conditions of a real-time environment.

1.2 Motivation

Data transmission and reception are the two most energy consuming operations in wireless sensor networks. Considering the limited energy resources of wireless nodes, reducing the number of transmissions and receptions in the entire network is one of the key solutions to reducing the power consumption of the whole network. The main task of this project is to reduce the number of transmissions and receptions in the entire network by processing the data at network nodes instead of the base station.

According to TAG (Tiny Aggregation), aggregating the data in network effectively reduces the number of transmissions and receptions (Madden et al. 2002). Though the authors of TAG claim this approach works effectively on the simulators, the real world scenario has its own challenges. Such challenges include node failure, interruption in transmission, transmission of bad packets, and changes in topology. These authors have not mentioned how robust the system is against such challenges. Compensation in these changes in network requires some steps like reconfiguration of topology, to be taken. In certain environments, this compensation procedure requires more power. This power consumed by in-network processing approach may exceed the power consumed by the centralized aggregation approach.
1.3 Approach

The main purpose of this project is to check the “In-network aggregation” approach in a real world scenario and compare the number of bytes transmitted and amount of power consumed per epoch duration, in the entire network, with those of “centralized aggregation” approach. For in-network aggregation we are checking the impact of varying the epoch duration, during which the data is collected at the base station.

Also, we are measuring the accuracy of the system by measuring the number of nodes, which are able to transmit the data to the base station successfully (i.e. without losing the data in the network), out of a given number of nodes.

These tests are performed on a test bed of mica2 motes, with both the in-network and centralized aggregation approaches. The results show how effectively the in-network aggregation performs the operation in order to reduce the power consumption.

The remainder of the report is structured as follows. Chapter 2 describes the background required in the query processing of wireless sensor networks and also describes various approaches to data centric networking. Chapter 3 describes the methodology and the algorithms used to solve the problem. Chapter 4 describes how to deploy the program in the motes and how to set up the test bed required for the application. Chapter 5 describes analysis of the results obtained from various experiments. Chapter 6 describes the future work and conclusion.
Chapter 2  

Background

Various approaches exist to collect the data from the network. Some of them are explained below.

2.1 Data Centric Networking

Data Centric Networking works on the concept that query, storage and routing techniques in wireless sensor networks can be used efficiently, if we combine them according to the application rather than following the traditional IP based techniques that follow a common approach for all applications (Heidemann et al. 2001). The data centric approach is advantageous in wireless sensor networks because these networks are resource constrained and application specific. Further the data content that these networks working on are well defined before the start of the application. Data centric networking concentrates on the querying, storing and routing of data; however the way in which the topology of the network is formed plays an important role in how efficiently data can be gathered on the base station.

We use sensor networks to satisfy some of the physical answers that we need to gather from a site. In the field of computer science when we want to ask questions to a source that is generating data, querying techniques are one of the solutions. Depending on the application of a data centric networking, sensor networks can have the following query types (Madden et al, 2005).

*Monitoring Queries:* These queries produce output at a fixed interval of time. A query that reports the temperature of a room every ten seconds is an example of monitoring queries.

*Network Health Queries:* These queries ask the network questions regarding its health. Querying the network to ask the residual energy in the motes is an example of this kind of query.
Nested Queries: Implementing Nested Queries in a sensor network is a challenging task and is more an area of research at this moment because of the complex nature of its development process.

Actuation Queries: In actuation queries, the system takes some physical action when the query is fired. The triggering of the alarm when the temperature of chemical reactors goes beyond some threshold is an example of an actuation query.

Offline Delivery: In many cases, the user wants to log some rapidly occurring events for processing. The data of these rapidly occurring events cannot be processed and sent as they happen. So the data has to be logged on EEPROM before being processed. The logging of data that will be processed in short future is called Offline Delivery.

An application designer should select which of the above mentioned query type is best suited for the application. Now we will mention some of the routing techniques used in data centric networking, what kind of querying techniques are best suited for those particular routing techniques and how topologies are formed in those techniques.

2.2 LEACH (Low-Energy Adaptive Clustering Hierarchy)

LEACH (Heinzelman et al. 2000) can be used for continuous monitoring of a network and also for applications where in-network processing is needed. In LEACH, the nodes organize themselves in clusters with one node acting as a cluster head. All the nodes transmit data to the cluster head in their vicinity, and the cluster head forwards the data to the base station by transmitting through other cluster heads in the network.
Figure 2-1 (Krishnamachari 2001). Cluster based routing technique LEACH

Cluster heads receive data from other cluster heads, perform data manipulation, and send data to other cluster heads. Following this pattern, the data reaches the base station. The cluster heads are periodically changed, so their energy does not get drained. LEACH involves two phases:

1) **Set up phase:** Cluster heads are decided in this phase. Initially, a node decides itself to be a cluster head with some probability. The cluster heads are changed periodically in order to utilize the network energy efficiently because if data is going through a node periodically than its energy will get depleted. The position of the cluster head is self-elected after a periodic time interval. The decision of the cluster head appointment is also based on the suggested percentage of cluster heads in the network.

The cluster heads broadcast their status to other nodes in the network, sending a message using the CSMA MAC protocol. This message is called the advertising message (ADV).
The other nodes, after receiving the ADV, decide its cluster heads. The decision is based on communication energy criteria and the signal strength received. After a node decides its cluster head, it sends a reply message to its cluster head. This reply is sent using the CSMA MAC protocol. Once all the nodes have a cluster head, the cluster head creates a TDMA schedule. This schedule decides when the nodes should transmit the data, so the chances of collision are minimized. The nodes can remain in sleep mode when they are not transmitting.

2) **Steady Phase:** In this phase, nodes send data in their slot time assigned using TDMA. Once the cluster head receives the data from all its nodes, it aggregates the data and sends it towards the base station. Before sending the data, the cluster head senses the channel, and the data is sent only if the channel is free, i.e. no other node is transmitting.

2.3 **Expanding ring search**

When the query is a one shot query, than flooding is not a good option. In this case, the cost will be higher because we have to consider the cost of flooding the network, too. Consequently, alternatives to the flooding techniques are required.
One such alternative of query flooding is “expanding ring search” (Cheng and Heinzelman, 2003). In expanding ring search, the nodes interested in an event will use controlled flooding. In this, the interested node will flood only to its one hop neighbor. If the query is not resolved, then the flooding radius increases. This way there is some control over flooding. The results show no considerable improvement over flooding. If the control flooding routes are cached, then significant improvement can be achieved in the system because more chances that the query resolved earlier still exist, in certain cases, due to the cached information.

2.4 Adaptive and Decentralized Operator Placement for In-Network Query Processing

Adaptive and Decentralized Operator Placement technique (Bonfils and Bonnet, 2004) makes the use of a correlation operator. The correlation operator is pushed into the network, which helps in reducing the message flow, toward the sinks, for in-network query processing. When an event occurs in two regions of the network, both regions report the event to the sink. Consequently the best place to move the correlation operator is toward the region where more data is generated. If
both regions produce the equivalent amount of data, then the operator should be placed in between the two regions.

![Operator Placement Diagram](image)

**Figure 2-3 (Krishnamachari 2001). Operator placement**

This is a dynamic operator placement strategy. There are two types of nodes involved in placing the operator: (1) The active nodes, which execute the operator; and (2) The tentative nodes, which calculate the cost to run that operator. The execution is transferred to the node that shows the minimum cost. For this processing, no global information is needed.

Initially, the operator is placed randomly. The operator placement is refined toward more and more optimal place time to time. So this strategy is decentralized, dynamic operator placement strategy. The disadvantage can be the excessive processing of messages. The calculations are needed at each step, which can be a better position to place the correlation operator.
2.5 Rumor routing

The query processing can be categorized in two approaches: (1) The push based approach, in which the query is pushed into the network; and (2) the pull based approach, in which the source notifies the sink about the occurring events. Rumor routing (Braginsky and Estrin, 2002) tries to combine both approaches and generate a protocol that reduces the number of messages that travel in the network.

The sink notifies every node in the network to look for a particular event through the push based approach. When an event occurs, at a particular node, the data packet starts a random walk toward the direction of the sink from the event source. When the query is pushed it leaves a sticky trail behind it to clarify from where the query came. During the event notification’s random walk, when this sticky trail is intersected, the packet is routed in the reverse direction of that sticky trail pointer, which is the direction toward the base station. Consequently the random flow of event notification messages do not flow in the network and the data reaches the sink.

Figure 2-4 (Krishnmachari 2001). Rumor routing
Figure 2.4 shows the source, the sink, and the point where during a random walk the packet from the source finds the path to the sink. Thus, rumor routing prevents considerable amounts of packet flooding in the network once an event is detected. Further, if extra information is to be gained after the detection of event, the sink can send the request packets in the same direction again toward the sink because the path is known.

2.6 Data Processing Techniques

The main aim of Data Centric Networking and Wireless Sensor Networks is to answer the physical queries and to process the data. Many data processing techniques are available in Computer Science. Few though are suitable for the resource and computation constrained environment of Wireless Sensor Networks. We will also discuss in-network data processing techniques and centralized data processing techniques in this section. In data processing techniques, data aggregation is widely used in our applications.

2.6.1 In-network aggregation

The “In-network aggregation” (Madden et al. 2002) is an approach toward the effort to reduce the energy consumption of the overall network. This approach is effectively used in Directed Diffusion (Intanagonwiwat et al. 2003), which is a data routing algorithm. Directed Diffusion uses caching and processing the data in network to save significant amounts of energy. Also, in-network aggregation has certain positive impacts on the energy constraint of wireless sensor networks (Krishnamachari et al 2002).

According to in-network aggregation, aggregation occurs at the intermediate nodes in the network rather than at the base station. In this way, only the aggregated data, which is the result of all the incoming data of a certain epoch interval, is required to forward towards the base station. This is in contrast to the centralized aggregation method, where all the data from the child nodes gets
forwarded toward the base station. As the amount of data forwarded in in-network aggregation is much less than the data forwarded in the centralized aggregation, the total number of transmissions in in-network aggregation is also much less than that of centralized aggregation. This will consume less energy in case of in-network aggregation because the energy consumption depends on the amount of data transmitted and received.

2.6.2 Centralized aggregation

When data is received from the sensing nodes at any node, if the data is not designated for that receiving node, it will just forward the data toward the designated node. In this way, the intermediate nodes will forward all the data, which they receive from the child nodes, resulting in large amounts of data transmitted over radio frequency and a lot of energy consumed. After collecting all the data at the base station, required aggregation can be performed on the collected data set. Performing aggregation at the centralized location is called centralized aggregation.

2.6.3 In-network vs. Centralized aggregation

In this section, we give an example to illustrate how in-network aggregation transmits fewer amounts of data compared to centralized aggregation. Suppose we want to count the number of nodes in the network. In order to perform this operation using centralized aggregation, every node in the network will send a one count to the base station, which will pass through a number of intermediate nodes.
It is clear from the figure 2-5 that, in order to perform the node count, the one count from all the nodes needs to reach to the base station before base station performs the addition to find out the total number of nodes in the network. But in case of the in-network aggregation, the intermediate nodes can perform the addition, so only the addition can be forwarded, in one transmission, toward the base station. As can be seen from the figure 2-6, the number of transmission in this case is significantly low.
Large amounts of transmissions in centralized aggregation cause congestion in the network. These transmissions further result in wasting the power resources. In-network aggregation uses time scheduling to deal with the congestion as we will see in section 3.2.1.
Chapter 3  Methodology

Formation of the in-network aggregation depends on a number of preliminary steps. In case of the centralized aggregation, these preliminary steps are comparatively few. Initially, topology is formed in both approaches. This is useful in deciding the route of the data that will travel from the sensor node to the root of the routing tree.

3.1  Formation of topology

Collection Tree Protocol (CTP) is a protocol used to generate the routing tree. When following this protocol in wireless sensor networks, some nodes act as root nodes depending on the application. Generally, we have the base station as a root node. But there can be applications where there can be more than one node as the root node, and not all are the base stations. The nodes of the network that follow the CTP have a task of routing the data to the root of the tree. CTP is address free in the sense that when a node has a packet to transmit it does not transmit to a destination address; but it transmits to its parent. The parent is its neighbor that has the best link quality to the root. The parent again sends the packet to its parent, so the data reaches the root of the tree. At each step, the duty of the intermediate nodes is to forward the data to the node that has the best link quality to the root.

Every node has a routing gradient called expected transmissions (EXT).

\[ \text{EXT}_{\text{node}} = \text{EXT}_{\text{parent}} + \text{EXT}_{\text{link to parent}} \quad \text{where } \text{EXT}_{\text{root}} = 0 \quad (3-1) \]

When transmitting data a node has to decide which neighbor has best EXT value. Lower EXT value is considered better than higher EXT value.
Routing loops are one of the factors that a node needs to eliminate. Routing loops arise because a node sends data to a node with a higher EXT value than that to the node, to which it was previously transmitting. Now in this process routing loops can occur, if a node receives a packet from another node which has EXT value lower than its own. If a node receives such a packet, it sends a beacon frame to the sender and asks it to adjust its routing parent accordingly.

Packet duplications can be another problem in CTP. These duplications happen when a node sends ACK for a received packet, but the sender does not receive that ACK back. To overcome this problem, the CTP protocol has a Time-has-lived (THL) field. The originator of a packet has a THL of 0, and when a receiver receives the packet, it increments the THL by 1. But link-level retransmission has the same source and sequence number and the THL field is not changed. Thus, duplicate packets can be detected using the THL field.

**CTP Data Frame**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>C</td>
<td>THL</td>
<td>EXT</td>
<td>Origin</td>
<td>Seqno</td>
<td>Collection_id</td>
</tr>
</tbody>
</table>

Figure 3-1 CTP Data Frame

*P*: P is called a routing pull. If a node sends a packet with a P bit set, the receiving node returns the routing information to the sender.

*C*: If a node detects congestion, it sets C bit on all the outgoing packets.

*THL*: The time-has-lived field is set to 0 by the originator and incremented by every node that receives that packet. The THL field has a maximum value of 255, and after that, it is reset to 0 again.

*EXT*: Expected Transmissions

*Origin*: The original sender of the packet.

*Sequence no*: The packet sequence number set by the sender of the packet.
Collection_id: There can be many roots in CTP. Every root has a collection id. Due to these roots, there will be many corresponding trees. Each tree is associated with a collection id. A node can be in more than one collection trees.

Data: The data that a node is programmed to send.

CTP Routing Frame

<table>
<thead>
<tr>
<th>P</th>
<th>C</th>
<th>Parent</th>
<th>EXT</th>
</tr>
</thead>
</table>

Figure 3-2 CTP Routing Frame

P, C and EXT are the same as that of the Data Frame.

The Parent is the estimated parent that has decided on the path quality determined by the EXT gradient.

Collection tree protocol has three major components as follows:

1) Link Estimator: This component is responsible for calculating a single hop EXT for transmission to its neighbors. This component sends periodic beacon packets to its neighbors, with the sending time rate incremented exponentially. A node will reset the timer when its EXT value reaches a certain threshold, or it receives a packet with P bit set. The sender who wants to determine its EXT value sends a beacon signal and receives acknowledgement. For every expected acknowledgement not received, it increments its EXT value for that particular neighbor. The lower the EXT value, the better the path.

2) Routing Engine: This component decides which path to choose next using the EXT gradient. The path that has the least number of expected retransmissions is chosen to be the forwarding path.

3) Forwarding Engine: This component actually transmits the data to the next node. This component also decides when to transmit the packet to the next node. All the route inconsistencies are determined by this component. After detecting the inconsistencies, it informs
the routing engine regarding the inconsistencies. This component also detects loops and queues the messages before transmitting than by maintaining the transmission queue. This component maintains a cache to decide if it receives a duplicate packet.

3.2 In-network aggregation

As soon as the routing tree is formed the network is ready to transmit the data from source to the base station. Algorithm for in-network aggregation is divided in two phases as shown in figure 3-3.

![Figure 3-3 Flow of network behavior](image-url)
3.2.1 Node discovery phase

The premise behind saving the power resources is to perform aggregation on intermediate nodes instead of at the base station. To perform the aggregation, knowing the topology is important. Nodes are divided into sensing nodes and non-sensing nodes. Non-sensing nodes are used to forward the data toward the base station in cases where the sensing nodes cannot transmit the data directly to the base station.

Once the topology is known, every sensing node can start sensing the data and forwarding it toward the base station. But in this case, the aggregating node may receive the data, after it has forwarded the aggregated data to its parent node, for that epoch duration. In this way, many chances of missing the data to be aggregated exist. To avoid this, the time interval must be assigned to each node, so the nodes can transmit the data. In this way, the nodes at each level will have their own time interval through which they can transmit. It is also necessary that nodes at the same level transmit during the same time interval.

The total epoch duration is divided in total number of levels i.e. the depth of the tree and the time interval is assigned to each level. All the nodes of the same level will transmit in their assigned time period as can be seen in the figure 3-4.
Assigning the time interval depends on the depth of the routing tree and the level of the node in that tree. For example, if $T$ is the total epoch duration at which we want the data, $D$ is the depth of the tree, and $L_{\text{node}}$ is the level of node in the tree, then the duration at each node $T_{\text{node}}$ can be given by following equation:

$$T_{\text{node}} = (D - L_{\text{node}}) \left( \frac{T}{D} \right) \quad (3-2)$$

The level calculations and depth calculations will be performed at the base station. The base station will assign each node in the network their respective level and depth of the tree. After receiving this information, all the nodes in the network will calculate their duration with equation 3-2.

a) Routing topology

When data is sensed at any node, it needs to pass through a number of nodes to the base station. Initially, this chain of nodes through which the data will travel is unknown. So the first step is to decide the data routing path. Section 3.1 explains how CTP is used to decide the routing path in the network.

b) Assigning the time intervals

Immediately after booting, the nodes find their parents and children and form the topology as discussed in previous sub-section. Initially, the topology is not visible to the programmers. To perform in-network aggregation, it is utmost important to find the total depth of the tree and the levels of each node in the tree, as already discussed.

There are two approaches to retrieve the depth of the tree and levels of each node in the tree. Again, these approaches are (1) in-network and (2) at the base station.
i) **In-network calculations**

In this approach, the base station will transmit a packet with a count zero. Every node which receives the packet will increment the count by one, store the count, and forward the packet toward the leaf nodes in the network. Logically, the count stored by the node is its level in the tree. The maximum level of the leaf node is the depth of the tree.

According to equation 3-2, we need the depth of the tree at each level to calculate the time duration at the nodes. To do this, each leaf nodes need to inform the other nodes in the network about its level in the tree. The nodes in the network will decide what the maximum level is among all the leaf nodes. Since initially the topology is not visible to the user, the leaf nodes are unknown. Therefore, before performing this task, the leaf nodes must be discovered.

To discover the leaf nodes, each node will start sending a packet to its parent at a regular interval. If any node receives no such packets, it considers itself as a leaf node because it has no children. After knowing the leaf nodes, the leaf nodes can send their depth to all other nodes and calculate the time duration.

ii) **Calculating at the base station**

To calculate the time intervals at the base station, the base station needs to know the topology. According to CTP, each node knows its parent. On boot up, each node can start sending its own id and its parent id toward the base station. Once information from each node reaches the base station, the base station can easily calculate the levels of each node and depth of the tree and again send to each node its respective levels.
Clearly, the second approach, i.e. the calculations of the levels at the base station, involves comparatively fewer numbers of transmissions, for it consumes less energy. So in our implementation, we have calculated the levels of each node at the base station.

### 3.2.2 Aggregation phase

Once all the nodes know their respective levels, they can calculate the time duration in which they can transmit the data. Each sensing node will start sensing the data and send it toward its parent in its assigned time interval. The parent node of the sensing node can be either a sensing node or a non-sensing node. In either case it should perform the aggregation.

![Processing plans](image)

Figure 3-4 Processing plans (a) Sensing leaf node, (b) Non-sensing intermediate node and (c) Sensing intermediate node
Figure 3-4 shows three different processing plans adapted by the nodes in the network according to their tasks. Figure 3-4 (a) shows the processing plan for a sensing node which is a leaf node. This node does not have any incoming data. All it has to do is sense the data and transmit it toward its parent. The figure 3-4 (b) is interesting because it shows a processing plan for an intermediate node (Yao and Gehrke, 2002). According to the figure 3-4 (b), an intermediate node, which is also a sensing node, needs to aggregate own data together with the data coming from its children nodes. If the intermediate node is a non-sensing node, it does not have to worry about aggregating its own data; it will just aggregate the incoming data and forward the partially aggregated data towards its parent as shown in figure 3-4 (c). The base station falls in the category of a non-sensing node, which can perform the aggregation. The base station will receive all incoming data and after performing aggregation, the aggregated data will be sent to be displayed on the PC.

The complete operations of all the three types of nodes - sensing leaf nodes, sensing intermediate nodes and non-sensing intermediate nodes - can be described by state transition diagrams in figure 3-5.
Figure 3-5 State transition diagrams (a) Sensing leaf nodes, (b) Sensing intermediate nodes, (c) Non-sensing intermediate nodes.

There are certain applications where it is required to collect the data of particular regions. Once we aggregate all the data, the origin of the data is lost. Considering this fact, we need to perform the aggregation according to the data group, as can be seen in figure 3-6.
At any given node, data will be aggregated only if it belongs to the same group. Consequently, we will have the aggregated data for all the groups at the base station.

### 3.2.3 Network response on topology change

Real time environments are full of uncertainties. Even the positions of the nodes are not static. Certain movements of other entities can interfere with the network behavior. Such interference can result in a change in topology. For example, certain movements in the network shown in figure 3-6 above make any node change its parent as shown in figure 3-7.
As can be seen from figure 3-7, a change in topology causes a change in the level of the nodes in the network. For example, node 20 in figure 3-6 is at level three, but a change in topology caused it to change to level two. To make it worse, node 32 changes its parent from 31 (figure 3-6) to 33 (figure 3-7), changing the depth of the tree from 3 to 4. As the time duration to send the data at each node highly depends on its level in the tree and the total depth of the tree (see equation 3-2), the levels of each node and total depth of the tree must be recalculated.

Every node must check whether its parent in the current epoch is the same as the parent in the last epoch. If the parent of current epoch differs from the parent in last epoch, then it is obvious that there are certain changes in the topology. As can be seen from the figures 3-6 and 3-7, it is possible that the node has changed its level or the depth of the entire tree is affected. So, to calculate the depth and time duration of only the affected node, (the node whose parent is...
changed) is insufficient. To acquire consistency, all the nodes in the network must recalculate their time duration.

Informing all the nodes in the network about the change in topology is the responsibility of the base station. As soon as a node finds out a change in its parent, it will inform the base station about the change in topology by sending the topology change signal. After receiving the topology change signal, the base station will ask all the nodes to again send their information (i.e. the node id and the parent id). All the nodes, after receiving the signal from the base station, will start sending the information immediately. Thus, we can say that the network, once again, will enter in the node discovery phase.

### 3.3 Centralized aggregation

The centralized aggregation approach is simpler to in-network aggregation. Again, the nodes can be divided into three different categories: (1) Sensing leaf nodes; (2) Sensing intermediate nodes; and (3) Non-sensing intermediate nodes. In this case, no aggregation is performed at the nodes. Each sensing node, after sensing the data, forward it to the base station. Any intermediate node which is also a sensing node will sense the data and send it toward the base station. Also it will intercept any incoming data from its children and simply forward it toward the base station without performing any operation. All the non-sensing intermediate nodes do the job of forwarding the data coming from the children towards the base station.

The operation of these different nodes can be explained by the following state transition diagrams in figure 3-8.
Figure 3-8 State transition diagrams (a) Sensing leaf nodes, (b) Sensing intermediate nodes, (c) Non-sensing intermediate nodes
The base station has a major task to perform in the case of centralized aggregation. As the name suggests, the aggregation is performed at the centralized location, i.e. at the base station. All the data from the network nodes are collected at the base station. It is important that base station collects data from all the nodes in the network before performing aggregation. As the number of transmissions increase in the network, there are higher chances of data loss. So the base station may wait for a long time to receive the data from all the network nodes. Once all the data from the network reaches the base station, it can perform the aggregation and send the aggregated data to the PC screen to display.

In case of the centralized aggregation, if the intermediate node has three children, it has to forward the data three times; but in the case of in-network aggregation, no matter how many children a node has it has to forward the aggregated data just once per epoch. As a result, the number of transmissions is significantly low.
Chapter 4    Application Design, Installation, and Deployment

4.1 Application Design

We have used nesC programming language to program the application. NesC is programming language for network embedded systems. Sensor networks have memory and power constraints. To support this constrained domain of wireless sensor networks nesC provides various functionalities like event based execution mode, compile time code analysis, component based application design, and a concurrency model that suits this resource constraints.

NesC programming model consist of components that are linked together. Interfaces that are used by, or are part of components, form an important role in how the application provides its functionalities. Interfaces provide commands or events. Commands are called by the user to instruct the TinyOS to perform some operation. Events are fired when TinyOS performs some functionality like the sensing or reading of data.

For TinyOS application programming, there are two kinds of files: Module and Configuration. The Configuration file contains how the components are wired together. The module file contains the actual implementation. The extension used by nesC for all its files is “.nc”.

The concurrency model of nesC is different from other programming languages. Concurrency in nesC is provided by tasks and hardware event handlers. The execution of tasks and even handlers differ from each other and do not preempt each other. Hardware event handlers are interrupts generated by the hardware and may be preempted by tasks. Because of this preemption the code that is susceptible to race conditions should be put under an atomic block.
4.2 TinyOS Interfaces

Figure 4-1 Component Diagram
The Interfaces and components shown in the figure 4.1 provide the major functionality for our project. Some of the major interfaces are discussed below.

Send: This interface provides functionalities for an address free send and also canceling a pending send. This interface also provides an event that is signaled when a send is successful or not. It also provides functionality to access the payload of the message.

Receive: This interface provides functionalities for receiving a message and pointers to the payload of the message and the payload length.

Intercept: In multihop routing, the intercept event is called if the intermediate nodes (the nodes that are not the destination of the packet) receive the packet. So this interface is useful for in-network processing.

RootControl: This interface has functions RootControl.setRoot( ). When this function is called on a node it is set as a root of the collection tree. RootControl.getRoot( ) returns the root of the collection tree on which the node recites. RootControl.unsetRoot( ) makes the node no longer the root of the collection tree.

Boot: The application starts from the Boot.booted( ) event. Boot.booted( ) event signifies that the boot process has completed and the application is ready to run.

Timer: The Timer Interface has the following commands

Timer.startPeriodic( ): It starts the timer which fires periodically after the specified duration and signals the fired event.

Timer.fired ( ): This event is called when the timer is fired, so the code that is to be executed when the timer is fired should be mentioned in this event.

Timer.startOneShot( ): If we want the timer to be fired just once, then this command should be called.

Timer.stop( ): This command will stop the timer on which it is called.
**Read:** The Read interface has the functions Read.read() and Read.readDone. Read.read( ) reads the value from the component on which it is wired to. Read.readDone signals that the reading has been completed.

We perform two read operations. We read the light and battery voltage of a mote. So the photo sensor and the voltage reader components are wired to this read interface.

**Ctpinfo:** It has functionalities, when used, can help in quality routing.

getParent( ) functionality of Ctpinfo returns the parent of the node on which it is called.

isNeighbourCongested( ) returns true if the node’s neighbor is congested.

### 4.3 Packet Structure

Data is send from and received by every node with the help of the packet structure Aggregate.

typedef nx_struct aggregate {
    nx_uint8_t version;
    nx_uint8_t id;
    nx_uint8_t hopcount;
    nx_uint8_t parent;
    nx_uint8_t depth;
    nx_uint16_t room[NUM_ROOMS];
    nx_uint16_t light
    nx_uint16_t volt;
} aggregate_t;

<table>
<thead>
<tr>
<th>Version</th>
<th>Id</th>
<th>Hopcount</th>
<th>Parent</th>
<th>Depth</th>
<th>Room</th>
<th>Light</th>
<th>volt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>2 bytes/room</td>
<td>2 bytes</td>
</tr>
</tbody>
</table>

Figure 4-2 Packet structure
Version: The version field contains the version of the packet. It suggests what phase the sending node is in. The sending node sets its version of all its outgoing packets to 1 to indicate that it’s in a level discovery phase. If the sending node sets its version field to 2, it indicates that it is sending the sensed light and voltage readings. If the node sends version 3, it indicates that it has noticed a change in its parent, and the level discovery phase has to be re run by all the nodes.

id: It is TOS_NODE_ID macro of the node. The TOS_NODE_ID contains the id of the node assigned to the node when the nesC firmware was loaded on the mote.

Hopcount: The hopcount is incremented every time a packet is intercepted by any other node in the network. The originator of the message will send hopcount to 0 and every node that intercepts the packet will increment the hopcount.

Parent: This field is used in the level discovery phase. Every node will send its parent to the base station. This field contains the parent of the node.

Depth: This field contains the depth of the node that will be send by the base station.

Room[NUM_ROOMS]: This field will hold the average light readings of every room, where NUM_ROOMS defines the total number of rooms for which aggregation has to be performed.

Light: Light will be the sampled light of the node.

Volt: It is the sampled voltage by the node.

4.4 Installation

The system has been tested on test bed of 15 mica2 motes with mts300 sensors and mib510 programming board.

4.4.1 Application Files

The files which make up the source code for the In-Network aggregation are

1) AggregateC.nc

2) AggregateAppC.nc
3) Aggregate.h

The file `Aggregate.h` contains the packet structure mentioned in figure 4-2.

`AggregateAppC.nc` is the configuration file that contains how each component is wired together. According to the standards mentioned by nesC every configuration file should be ending with `AppC`. This file contains the information about how the various components of our application are wired together.

`MainC` is the component from where the boot process starts. So the Boot component is wired to `MainC`. Our application component called the `AggregateC` uses various components. `AggregateC` is wired to `DemoSensorC` component, which is the default sensor on the mote. Since we are using sensor mts300, our `DemoSensorC` has a default sensor as the light sensor. So the default readings returned from the Read interface is the light reading. `AggregateC` is also wired to `CollectionC`, which plays an important role in the collection process in which the data reaches the base station from the motes.

The important interfaces that are part of `CollectionC` are the Receive interface, Intercept interface, `RootControl`, and the `CtpInfo` interface. The `Intercept` interface provides the function called `forward` which is used to the manipulate data if the node which receives the data is not the intended receiver of the data. In our case, only the BaseStation is the intended receiver of the data, so all other nodes that receive a packet will call the `Intercept.forward` for processing the packet and not the Receive interface.

In our application, only the base station will call the `Receive.Receive` functionality of the receive interface, as the Receive interface in a MultiHop Routing Engine is used by only the intended
receiver of the packet. In our application, only the base station is the intended receiver of the data.

*CtpInfo* provides a lot of functionalities that can be used for Routing and Congestion control. The Send interface is part of the *CollectionSenderC* component which is wired to the *AggegateC* component. The *Send.send* function is invoked whenever a node has a packet to send.

*TimerMilliC* component is the timer, which is fired at a regular interval in our application, and we use the interface Timer for that. The *PoolC* and *QueueC* are the components that are used to buffer the packets as they arrive at a node so that a receiving node is not overwhelmed by the sending node and the receiver can buffer packets if it is not in a position to process them directly as it receives the packets.

*AggregateC.nc* contains the actual implementations of the algorithms mentioned in chapter 3 of this paper.

### 4.4.2 Makefile

The Makefile should contain the line `COMPONENT=AggregationAppC`. The component is the main component to which the boot component is wired. The make file also contains the libraries that have to be included for a particular application. In our case, we need to include the libraries of Collection tree protocol. Also a crucial part of Makefile is the `MAKERULES` variable. It contains the path to all the rules needed to compile the application and build an executable binary image out of it.
4.4.3 Makelocal File

The TinyOS 2.x tree contains a file called Makelocal. The Makelocal file contains the instructions that have to be executed every time a program is to be loaded on the mote.

We have added the operating frequency 916700000 in the Makelocal file using the command `PFLAGS += -DCC1K_DEF_FREQ=916700000`. We have also mentioned the default group of the motes as 0x33 using the command `DEFAULT_LOCAL_GROUP = 0x33`, so our application motes receive no interference from motes of other group ids. The default sensor is set as mts300 using the command `SENSORBOARD=mts300` in the Makelocal file. The Makelocal file is available in the directory `\opt\tinyos-2.x\support\make`.

4.4.4 Message Interface Generator (MIG):

When the packet comes to the UART from the base station mote, it is in raw format; we have to parse and interpret the bytes in the packet. TinyOS provides the MIG tool that helps the application developers to visualize data in a better manner, by automatically generating message objects. These message objects are generated from the packet structure provided in the .h file of the application. In our case, the Aggregate.h file has a packet structure called aggregate_t. The MIG tool parses the packet structure and provides java accessor, mutator functionalities for it, so we can directly print the packet on the terminal and further process the data as desired by the application.

The MIG tool takes 3 arguments; the language in which the assessors and mutators are to be generated, the file in which the packet structure resides, and the name of the structure.

In our application, once we get the packet on the serial port and then to the pc, we interpret the data and then decide based on the received data of the room numbers, whether the rooms are
vacant. If we receive the average lux value of light of a room to be more than 350 we conclude that the room is occupied. Any value less than 350 lux leads to the conclusion that the lights in the room are off, which means the room is empty.

4.5 Deployment:

Here, we will discuss how we will load the firmware on the motes. For the In-Network Aggregation go to the directory named In-Network Aggregation located in opt/tinyos2.x/apps/ and give the command

$> make mica2

Then to load the firmware on the mote, give the command

$> make mica2 install, <mote id> mib510,com1

This command will install the firmware on the mote with the mote id provided in the command.

We have tested the code by using 15 motes for our application. The mote with id 0 will be plugged into the programming board and will act as a base station. The above mentioned procedure is repeated for the centralized approach using the same commands except that the working directory will be opt/tinyos2.x/apps/Centralized
Chapter 5 Application, Results and Analysis

5.1 Application

The light detection system can be used to determine occupancy of a room by adding more features which are discussed in future work section. This system can be used in many places. If used in the libraries, we can detect vacant study rooms from the ground floor instead of going to each floor and checking for vacant study rooms. Similarly, this system can be used to detect vacant conference rooms.

The motes that are in the room will sense the light. Every mote that receives a packet will perform aggregation on the data and group the light data according to room number. Thus, at the end of every epoch, the average light readings of rooms will be on the base station. Then on base station, we can decide whether the room is vacant on the basis of readings we obtained. Any reading of a room above 350 Lux leads to conclusion that the room’s lights are ON and any readings less than 350 Lux tells us that lights in that particular room are OFF and thus the room is vacant.

5.2 Test Bed

We have used 15 mica2 motes to perform the experiments and to test our application. The programming board used is mib510, and the motes have sensors mts300 attached to them to take the light readings.

Once all the 15 motes have the firmware installed, they are placed in rooms. We try to keep 2 motes per room. The other motes will be placed at appropriate distances from the motes in the room to make sure that the data is able to reach the base station. The motes not in the room are to
be placed at least within the transmission range of the motes in the room, so the network has no hole.

The transmission range of mica2 motes is around 30 feet when there is no interference in the network. Since our application is an indoor application it will have interferences from walls and other solid objects available in the site. So we try not to spread the motes more than 10 feet from each other while keeping at least 2 motes per room to produce accurate results by gathering data from entire room.

5.3 Experiments, Results and Analysis

Our main aim in the chapter is to compare the performance of in-network aggregation to centralized aggregation. We compare both of these approaches in terms of the total effective bytes transmitted and the hop count per epoch.

5.3.1 Experiment 1: Effect of various epoch durations on packet loss

In this experiment, we query the nodes for their light with epoch duration for the query being $t = 2000$ ms, $4000$ ms and $6000$ ms. So we obtain hop counts on the base station every 2 sec, 4 sec and 6 sec respectively. The hop count that is obtained on the base station is sum of all hop counts for all packets for that particular epoch.
Figure 5-1 is the graph of the epoch against the hop count obtained from the experiments performed on our system. The epoch duration is 2000 milliseconds in this case. So the ideal scenario in this case should have been 15 every time, since we have used 15 motes in our test bed.

During some epochs, we received hop counts less than 15. This result is due to packet loss. Though we have tried to deal with congestion, a certain amount of congestion is inevitable. If we lose a packet at a node, which is near the base station, then the final total received will be considerably less. This is due to the fact that, the dropped packet near the base station will be having hop count of all the nodes from which it has traveled. So in many cases, we have received hop counts less than 10.

Wireless sensor networks have unpredictable nature. There are packet losses due to the interference from the environment in which experiments are performed. The nodes at each level transmit data in their assigned time interval. This means all the nodes, at the same level should transmit in the same time duration. To achieve accuracy for transmitting time at each level, time
synchronization is required. Absence of time synchronization in our system can also be one of the factors causing the packet loss.

Figure 5-2 Hop count per epoch for $t = 4000$ ms

Figure 5-2 shows the hop count obtained per epoch when the epoch duration is set to 4000 milliseconds. This case is also similar to one shown in figure 5-1. Here also we can see a considerable amount of packet loss.

Figure 5-3 Hop count per epoch for $t = 6000$ ms
Figure 5-3 describes the case where hop count is obtained for the epoch duration being $t = 6000$ milliseconds. The hop count obtained on the base station was ideal in some cases, but some packet dropped due to interference and collusion. We expected good results from this experiment, but this was not the case. We concluded that all children nodes transmit to their parent at the same time, so packets can always be lost. Further, interference is one of the reasons packets drop.

5.3.2 Experiment 4: Effect of various epoch duration on power consumption in in-network aggregation

In this experiment, we queried the network for epoch durations of 2000 milliseconds, 4000 milliseconds and 6000 milliseconds, to find out the total amount of power consumed in the network for their respective epoch durations. The total voltage was the sum of the voltage of each mote in the network.

![Voltage drop per epoch, t = 2000](image)

Figure 5-4 Voltage drop per epoch when $t = 2000$ ms
From figure 5-4, we can see that the drop in voltage is not consistent; sometimes the total voltage obtained for an epoch is more than the one obtained in the previous epoch. This inconsistency is due to the loss in packets. If a packet from a mote that has higher voltage is lost, it will show a drop in voltage.

Figure 5-4 shows a drop in voltage when considering the start and end points. Since we are aggregating the data, we are finding the total voltage of the network, not the individual node voltage, so the packet loss causes the unpredictable voltage. Although the wavy nature of graph is the same as the hop count, the voltage graph drops as the motes transmit and receive the packets.
Figure 5-5 shows the voltage obtained when epoch duration is set to 4000 milliseconds. Again, the graph has this wavy nature which shows a loss in packets, but the graph also has this dip which shows a total voltage drop in the network. When $t = 4000$ ms, the nodes will transmit at the half rate, in comparison with $t = 2000$ ms. Because of this, we expect a considerably low amount of voltage drop.

![Power drop per epoch, t = 6000](image)

Figure 5-6 Voltage drop per epoch when $t = 6000$ ms

Figure 5-6 shows a similar dip as the above two experiments. Here we are not able to compare all these voltage drops of different epoch durations. This is due to the packet loss factor mentioned above. So we cannot compare the voltage of all these graphs, since to know the ideal or expected voltage is impossible.
5.3.3 Experiment 3: Measuring power consumption in Centralized aggregation

In this experiment, we query the sensor network using the centralized approach for the total network voltage. Here, the total voltage, which is the sum of all voltages, is calculated at the base station rather than using in-network processing. The epoch duration for the motes is 4000 milliseconds.

The voltage graph of the Centralized approach is shown in figure 5-7. A considerable dip in comparison to the voltage graph of the in-network processing is expected. This should result because the centralized approach uses more packet communication in comparison to in-network processing. But again, actual voltage of entire network is hard to obtain because of packet loss. Since these packets contain the power of the motes, power of the mote, whose packet is dropped, never reach to the base station.
5.3.4 Experiment 4: Comparison of number of transmissions in the network

In this experiment, we compare the hop count obtained in the in-network processing with the hop count obtained using the Centralized approach. The epoch duration is 4000 milliseconds. Hop count represents the number of transmissions in the network.

![In-network vs Centralized (Hop/Epoch)](image)

Figure 5-8 Comparison of number of transmission per epoch

We can see from figure 5-8 that the centralized approach uses many transmissions in comparison with the in-network processing approach. This large number of transmissions results because the in-network processing merges the packets at each node whereas a centralized approach does not. Each packet in the centralized approach then, has to travel the network up to the base station without merging with other packets.

From figure 5-8, we can conclude that in-network processing is better than a centralized approach as in-network processing has a fewer number of transmissions per epoch as compared to the centralized approach. Due to this low transmission, the energy consumed by the in-network
approach will be less than the centralized approach. So in-network processing is better in comparison to the centralized approach in a power-constrained environment of wireless sensor networks.

### 5.3.5 Experiment 5: Comparison of number of bytes transmitted

The main aim of this experiment is to compare the total effective bytes transmitted in the centralized approach with that in the in-network processing approach. In this experiment, we add the total bytes transmitted each time a packet is received at the base station to obtain to total bytes transmitted in certain period of time.

![In-network vs Centralized (Bytes transmitted)](image)

**Figure 5-9**

Figure 5-10 Comparison of total bytes transmitted after 260 epochs

From figure 5-9, we see a big gap between the two curves. The in-network processing curve is much lower than that of the centralized approach, in terms of total effective bytes transmitted. The effective bytes transmitted are the bytes that we used for the aggregation of the total hop count for a particular approach. The graph in figure 5-9 clearly shows that in-network processing transmits fewer bytes than centralized approach.
5.4 Application Results Analysis

In this section, we try to measure the accuracy of our application in both approaches. For the In-network aggregation approach, the accuracy of our application was an average of 75%. This result is due to the packet loss that is discussed in the above experiments. Due to the packet loss, data from some of the parts of the network will be lost, and the light readings of room are affected. Thus, we are not able to get complete accuracy in many cases. A 75% accuracy means that, if we want to detect the light readings of 4 rooms, we sometimes will get the readings from 3 rooms only, and thus giving a 75% accuracy of the system.

In case of centralized approach we got 100% accuracy most of the times because in this approach we wait for all the data to come to the base station and then only send the data to the pc, but we keep the track of the epoch duration. It is also noted that, it takes multiple epochs to reach the data from all the motes to the base station. The number of epochs varies from 2 to 10. As the base station waits for the data from all the motes, even if data of a particular mote is lost the base station will wait for that mote to retransmit the data.
Chapter 6  Future Work and Conclusion

6.1 Conclusion

Lossy nature of the environment has prevented us to compare the accuracy with the measurement of the power of entire network. Though it is not clear from the figures 5-4, 5-5, 5-6 and 5-7, that in which scenario the power consumption of the network is high, it is proven that the total number of bytes transmitted with the in-network aggregation approach is much lesser than that with the centralized aggregation approach. This can be seen in figure 5-9. Also it is proven from figure 5-8, that the total number of transmissions per epoch in the network is high in case of the centralized aggregation approach than that in case of the in-network aggregation approach.

As we have already mentioned that the total amount of power consumed in the network is directly proportional to the number of bytes transmitted in the network, the in-network aggregation approach leads to a considerable amount of power saving in comparison with the centralized aggregation approach.

6.2 Future Work

One of the reasons of packet loss can be lack of time synchronization. Nodes that are transmitting on the edges of assigned time interval, has higher chances of dropping the packets. A better accuracy can be achieved by synchronizing the nodes to send the packets in accurate timely manner.

Timing-sync protocol (TPSN) (Ganeriwal et al. 2003) is one of the options that can be used for time synchronization. It provides sender-receiver synchronization. Suppose there are two nodes A and B that have to be synchronized. Node A will send a message to node B. Node A will time-stamp the message locally as T1. On receiving, node B will time-stamp the message as T2. Again
B will send the message to node A and time stamp it at $T_3$. Node A, on receiving the message, will time stamp it as $T_4$.

Let the clock drift between the two nodes be $\delta$ and propagation delay be $D$ then we have:

$$T_2 = T_1 + \delta + D$$
$$T_4 = T_3 - \delta + D$$

therefore,

$$\delta = \frac{(T_2 - T_4) - (T_1 - T_3)}{2}$$

$$d = \frac{(T_2 + T_4) - (T_1 + T_3)}{2}$$

Thus we can know the propagation delay and the clock drift between the two nodes. Consequently the nodes can be synchronized.

TPSN uses tree hierarchy for time-synchronization. Nodes at level 1 will be synchronized with nodes at the root. Nodes at level 2 will be synchronized with nodes at level 1. Thus this process continues down the level and we can achieve network wide synchronization.

Since the aggregation is done per group and the groups are decided from the location of the nodes, it is necessary for us to know the locations of all the nodes in advance. We need some means by which we can know the locations of the nodes. Localization is highly desirable to obtain the position of each node, if the number of nodes is large and the locations of the nodes are not known in advance.
To decide whether the room is empty or not it is not enough to consider only light readings. To detect a vacant room we can further add the magnetic field for detecting whether a chair is moving or not. This is based on the assumption that the motes will be cheap in the future. We can attach a mote to each chair and when a person is sitting on the chair it will move and we can detect that room is occupied. Also we can detect sound from a room, so if someone is speaking, the mote will detect that, and we can conclude that the room is not vacant. In this case, we will be querying the sensor network in which the query has multiple entities.
Chapter 7 Bibliography

(Intanagonwiwat et al. 2000)

(Gay et al. 2003)

(Heidemann et al. 2001)

(Heinzelman et al. 2000)
(Cheng and Heinzelman 2003)


(Braginsky and Estrin 2002)


(Bonfils and Bonnet 2004)

J. Bonfils and P. Bonnet, Adaptive and Decentralized Operator Placement for In-Network Query Processing, Telecommunication Systems - Special Issue on Wireless Sensor Networks, January 2004

(Madden et al. 2002)


(Yao and Gehrke 2002)

Y. Yao and J. Gehrke, The cougar Approach to In-Network Query Processing in Sensor Networks, SIGMOD, March 2002

(Madden et al. 2002)

(Ganeriwal et al. 2003)
S. Ganeriwal, R. Kumar, and M. B. Srivastava, Timing-Sync Protocol for Sensor Networks,
Proceedings of ACM SenSys’03, November 2003

(TinyOS Forum)
TinyOS Mailing list, http://www.tinyos.net/

(TinyOS Naming Conventions 2002)
TinyOS Naming Conventions, http://www.tinyos.net/tinyos-1.x/doc/tutorial/naming.html

(TinyOS Introduction 2003)

(Dissemination Protocol 2004)

(Collection Protocol 2004)

(The Collection Tree Protocol 2004)


Appendix A – TinyOS and nesC

Hardware Platform

The motes used were mica2 motes from CrossBow Technologies.

The sensor that is used is mts300. Mts300, when plugged on to the mica2 mote, has the capability of sensing light, temperature and battery voltage of the mica2 mote.

The base station interface used is the mib510 board, which is RS232 based. It helps in reprogramming a mote, when the mote is plugged onto this board. In addition to that, it acts as a base station in any application when a mote is mounted on it. Many applications are possible that have more than one base station. In that case also we can use multiple mib510 boards for the application. Mib510 must be connected to the pc via RS232 interface.

Programming Platform

NesC is the programming language used to program the motes, which operates on tiny operating system called TinyOS.

NesC: NesC is an extension to the C language. NesC is designed to support the design of TinyOs. There is a vast difference between nesC and some of the other modern day programming languages like Java, C++, or C.

Unlike other languages, the program analysis takes place at compile time. So the races in the code can be detected at compile time. NesC uses a static approach instead of a dynamic one in a sense that the compiler knows the entire call graph at the time of compilation. This is different from C++, where run time execution is an important concept. The static approach is useful in the
resource constrained environment of motes since the resource allocation will not exceed in the future when application is running. Interfaces of nesC can be parameterized, so they eliminate the needs for dynamic memory allocation and dynamic dispatch.

Unlike other programming languages, nesC interfaces can also have commands, which the users can use directly. For instance, the timer has command \texttt{setTimer}, which will set the periodic timer firing time.

Unlike other programming languages, nesC interfaces are split phase. For example, when we call \texttt{Send.send} to send a packet, once the sending process on the mote completes sending a packet, a \texttt{Send.SendDone} is signaled. This is useful as the control does not block until the completion of functionality.

NesC programming language is completely event driven unlike java, which is hybrid in nature and not solely event driven. Concurrency in nesC is completely event and task based rather than the thread based concurrency found in Java, C++, or C.

Since races are handled at compile time, concurrency is a safe operation in nesC. Linking model is completely different for nesC in comparison to other programming languages, as there is no dynamic linking. Further, nesC does not possesses global name space, so components can reference only the variables in their own local name space, however a component can declare that it uses a function defined by another component.

In java, classes can be instantiated many times, but nesC modules can only be instantiated once. NesC introduces the use of some new functionalities. One of them is rCombine. We can use
rCombine to call two functions together. rCombine returns true only if both the functions called through rCombine returns true.

*tinyos:* TinyOS is used as an operating system for motes that form a part of the wireless sensor networks. TinyOS is developed taking into consideration that motes have limited power and resources. To take this feature into account, TinyOS does not have any file system; it follows a simple path of execution and has a simple task model. Also language that is used in TinyOS, mostly nesC, tries to make the execution binaries as small as possible by targeting only those parts of TinyOS that are necessary for the application.

TinyOS has component based architecture. TinyOS provides a library of components that can be used to perform tasks. Components are wired together. All components, along with their software functionalities, build the execution model.

TinyOS provides a split phase interface. In a split phase interface, the method that is called is different from the method from which the functionality completes. The method from which the functionality completes, signals an end of the process of the method that called the first function. For example, if we want to take a light reading with the help of a sensor. We call the light reading functions and the control returns back immediately because the light reading from a sensor may take time, and there is no point in waiting for the readings to come. Instead, the operating system is notified only when the light reading has been completed. Split phase interfaces also return the status whether the operation was completed or not.

In the case of the TinyOS packet communication model, the application layer prepares the packet with the destination address in the packet and passes the packet to the operating system to handle it, using the *Send* command. The operating system then receives the packet that has to be sent.
The operating system then signals *sendDone* when it has sent the entire packet. At the receiving end, the operating system receives the packet and calls the *Receive* function at the application layer. In this way, Send and Receive occur.

Concurrency in TinyOS is provided by tasks and hardware event handlers. Their execution differs and usually do not interrupt each other. Although event handlers have a high priority than the tasks and the event handlers can preempt the tasks. If we need that some portion of the code should not be preempted, we have to put that in an atomic block. The code under the atomic block has to be executed without preemption, so the application designer has to take care that the code under atomic block should be as small as possible. Tasks are mostly user written while events are due to hardware interrupts.

TinyOS does not support global name space. All variables that a component uses are within that component itself. TinyOS does not support a dynamic memory allocation to reduce the overheads incurred from dynamic memory allocation. All these functionalities have been incorporated into TinyOS to make the application as lightweight as possible. TinyOS removes all data races at compile time. A complete call graph is prepared at the run time and it is not delayed until execution time.