San Jose State University
SJSU ScholarWorks

Master's Theses | Master's Theses and Graduate Research

2004

CMPE : cluster-management & power-efficient protocol for wireless sensor networks

Shen Ben Ho
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Ho, Shen Ben, "CMPE : cluster-management & power-efficient protocol for wireless sensor networks" (2004). *Master's Theses*. 2588.
DOI: https://doi.org/10.31979/etd.t4nf-9ht6
https://scholarworks.sjsu.edu/etd_theses/2588

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

CMPE:

CLUSTER-MANAGEMENT & POWER-EFFICIENT PROTOCOL

FOR WIRELESS SENSOR NETWORKS

A Thesis
Presented to
The Faculty of the Department of Computer Engineering
San Jose State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

By
Shen Ben Ho

May 2004

UMI Number: 1420470

Copyright 2004 by
Ho, Shen Ben

All rights reserved.

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

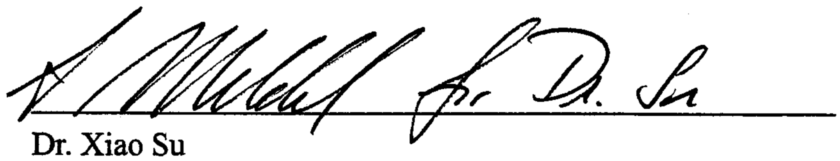
UMI®

UMI Microform 1420470
Copyright 2004 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

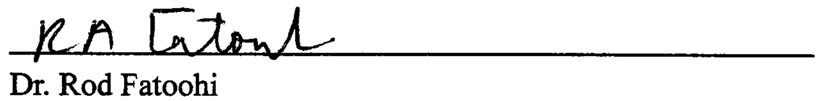
ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

© 2004
Shen Ben Ho
ALL RIGHTS RESERVED

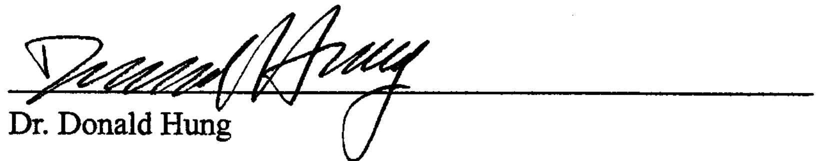
APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING



Dr. Xiao Su

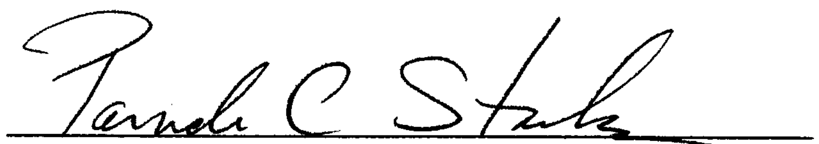


Dr. Rod Fatoohi



Dr. Donald Hung

APPROVED FOR THE UNIVERSITY



ABSTRACT

CMPE:
CLUSTER-MANAGEMENT & POWER-EFFICIENT PROTOCOL
FOR WIRELESS SENSOR NETWORKS

By Shen Ben Ho

Wireless sensor networks (WSN), a kind of distributed wireless ad-hoc network, are characterized by the large amount of sensors and the low power usage. In this paper, some design issues of wireless networks will be discussed as well as the current developments in hardware, software, routing and MAC protocols in WSN. As the current protocols are found to have some drawbacks in various aspects, a new protocol, CMPE – Cluster-Management and Power-Efficient protocol – is proposed and will be described in detail. The routing part of the protocol is based on a cluster and selective flooding algorithm to propagate the cost information: every sensor in the network will try to establish the minimum cost path to the cluster head, and the TDMA scheduling part of the protocol is to find and avoid extending the critical path. Lastly, simulation of the protocol and comparison with other protocols will be presented.

Table of Contents

1 Introduction 1
2 Design Issues 4
2.1 Free-space Propagation 4
2.2 The Hidden Station Problem and the Exposed Station Problem 6
2.2.1 The Hidden Station Problem 6
2.2.2 The Exposed Station Problem 7
2.3 Radio Model for Power Calculations 8
2.4 Direct and Multi-hops Connection 10
2.4.1 Direct Connection 11
2.4.2 Multi-hops Connection 12
2.4.3 Comparison Between Direct and Multi-hops Connection 16
2.5 Routing and MAC Protocols 18
3 Current Development 19
3.1 Mica2 and Mica2Dot – the Hardware 19
3.2 TinyOS – the Software 22
3.2.1 nesC 22
3.2.2 TOSSIM 24
3.2.3 Serial Forwarder 26
3.2.4 TinyViz 30
3.3 Routing Protocols 31
3.3.1 Direct Communication 31
3.3.2 Low-Energy Adaptive Clustering Hierarchy (LEACH) 32
3.3.3 Power-Efficient Gathering in Sensor Information Systems (PEGASIS) 35
3.3.4 Hybrid Indirect Transmissions (HIT) 37
3.4 Medium Access Control Protocols 41
3.4.1 Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) 41
3.4.2 Sensor-MAC (S-MAC) 43
3.4.3 Self-Organizing Medium Access Control for Sensor Networks (SMACS) 45
3.4.4 The TDMA Schedule of HIT (continued from Sect. 3.3.4) 47
4 Cluster-Management and Power-Efficient Protocol (CMPE) 50
4.1 Assumptions 50
4.2 Step One: Cluster Head Election 51
4.3 Step Two: Route Set-Up 53
4.3.1 Phase One: Cluster Head Advertisement 53
4.3.2 Phase Two: Path Setup 54
4.4 Step Three: TDMA scheduling 56
4.4.1 Phase One: Upstream Node Notice 59
4.4.2 Phase Two: Sending Downstream Node List and Blocking List 60
4.4.3 Phase Three: Schedule Setup 61
4.4.4 Phase Four: Schedule Broadcast 64

4.5 Step Four: Data Transmission 64
4.6 Examples 66
4.6.1 An Example for Routing 66
4.6.2 An Example for TDMA Schedule 73
4.7 The Routing Tree and TDMA Tree 76
5 Simulation and Results 79
5.1 The Simulator 79
5.2 Simulation Results 81
5.2.1 Parameter Tuning 81
5.2.2 Comparisons 83
6 Extension 87
6.1 Progressive Discovery 88
6.2 Three-Way-Handshake for Cost Discovery 89
7 Conclusions 91
References 93
Appendix A Source code of program used to test TOSSIM 95
Appendix B Source code of program used to test Serial Forwarder 99
Appendix C Stack Trace of Serial Forwarder when using with TOSSIM-radio 102
Appendix D Source Code of the Simulator 104

List of Figures

Figure 1.1 (a) UC Berkeley Cots Dust and (b) NASA JPL Sensor Webs Projects. 3
Figure 2.1 Radio Propagates as a Sphere. 4
Figure 2.2 The Hidden Station Problem. 6
Figure 2.3 The Exposed Station Problem. 7
Figure 2.4 The First Order Radio Model. 8
Figure 2.5 Energy Dissipation. 9
Figure 2.6 A Simple Liner Network. 10
Figure 2.7 Direct Connection: (a) Transmissions and (b) Timeline. 11
Figure 2.8 Multi-hops Connection: (a) Transmissions and (b) Timeline. 13
Figure 2.9 Optimization 1 of the Multi-hops Network. 14
Figure 2.10 Optimization 2 of the Multi-hops Network. 15
Figure 3.1 (a) Mica2, (b) Mica2Dot, (c) MIB510 (Base station), (d) Mica2 Block Diagram, and (e) Mica2Dot Block diagram. 22
Figure 3.2 nesC. 23
Figure 3.3 (a) TOSSIM (Running Application Surge) and (b) Program for Testing TOSSIM. 24
Figure 3.4 TOSSIM Packet Format. 26
Figure 3.5 Serial Forwarder: (a) Serial Forwarder Application, (b) Program for Testing Serial Forwarder, and (c) a System View. 27
Figure 3.6 Serial Forwarder Packet Format. 29
Figure 3.7 TinyViz Application. 30
Figure 3.8 An Example of Low-Energy Adaptive Clustering Hierarchy (LEACH). 32
Figure 3.9 An Example of Power-Efficient Gathering in Sensor Information Systems (PEGASIS). 35
Figure 3.10 Two Different Methods of Transmission in PEGASIS 36
Figure 3.11 An Example of Hybrid Indirect Transmissions (HIT). 37
Figure 3.12 Limited Search Area of Upstream Neighbors in HIT. 40
Figure 3.13 (a) Sender Sends a Request-To-Send, (b) Receiver Replies a Clear-To-Send, and (c) Timeline and the Network Allocation Vector in CSMA/CA. 42
Figure 3.14 Periodic Listen and Sleep in Sensor-MAC (S-MAC). 43
Figure 3.15 Self-Organizing Medium Access Control for Sensor Networks 45
Figure 4.1 (a) The range is too short and only a few nodes are connected. (b) The range is set to double the average distance between nodes. (c) The range is set to cover the whole network; the nodes are fully connected. 57
Figure 4.2 Network with Blocking Nodes. 57
Figure 4.3 Snapshot of the Schedule when Assigning Timeslot to Node G. 63
Figure 4.4 Pseudo-code for TDMA Scheduling. 64
Figure 4.5 Network for Routing Example. 66
Figure 4.6 Cluster Head and its Members. 67
Figure 4.7 Advertisement of Cluster Head and its Range. 68
Figure 4.8 Discovery of Node A. 69
Figure 4.9 Discovery of Node B. 70

Figure 4.10 Discovery of Node D. 71
Figure 4.11 Discovery of Node G. 72
Figure 4.12 Network for TDMA Scheduling Example. 73
Figure 4.13 Snapshot of the Schedule after Assigning Timeslot to Nodes C and B. 74
Figure 4.14 Snapshot of the Schedule after Assigning Timeslot to Nodes E and F. 75
Figure 4.15 Snapshot of the Schedule after Assigning Timeslot to Node D. 75
Figure 4.16 Snapshot of the Schedule after Assigning Timeslot to Nodes G and H. 76
Figure 4.17 A TDMA Tree Node (Reading from Top-down Fashion). 76
Figure 4.18 A Routing Tree for the Network Depicted in Figure 4.12. 77
Figure 4.19 A TDMA Tree for the Network Depicted in Figure 4.12. 78
Figure 5.1 Structure of the Simulator. 79
Figure 5.2 Energy Dissipation in One Round of Data Gathering. 84
Figure 5.3 Energy Dissipation in Route and MAC Setup. 85
Figure 5.4 Average Delay. 86
Figure 5.5 Network Utilization. 87
Figure 6.1 Timeline for Progressive Discovery. 88
Figure 6.2 Timeline for Cost Discovery. 89

List of Tables

Table 4.1 Information Gathered by Cluster Head for Scheduling. 74
Table 5.1 Results of Different Networks Using the CMPE. 82
Table 5.2 Energy Dissipation in One Round of Data Gathering...................................... 84
Table 5.3 Energy Dissipation in Route and MAC Setup. 85
Table 5.4 Average Delay. 86

Abbreviations

CDMA	Code Division Multiple Access
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear-To-Send
EEPROM	Electrically Erasable Programmable Read-Only Memory
FDMA	Frequency Division Multiple Access
HIT	Hybrid Indirect Transmissions
LAN	Local Area Network
LEACH	Low-Energy Adaptive Clustering Hierarchy
IEEE	Institute of Electrical and Electronic Engineers
LED	Light-Emitting Diode
ISM	Industrial, Scientific, and Medical Bands
MAC	Medium Access Control
MEMS	Micro-Electrical- Mechanical Device
NAV	Network Allocation Vector
NRZ	Non-Return to Zero
PC	Personal Computer
PEGASIS	Power-Efficient Gathering in Sensor Information Systems
RTS	Request-To-Send
S-MAC	Sensor-MAC
SMACS	Self-Organizing Medium Access Control for Sensor Networks
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TOSSIM	TinyOS Simulator
UART	Universal Asynchronous Receiver/Transmitter
WSN	Wireless Sensor Networks

1 Introduction

A *wireless sensor network*, or WSN for short, is a type of distributed wireless ad-hoc network. It may contain several hundreds or even thousands of tiny sensor nodes. The sensor nodes gather information from the surrounding environment and send the data back to the base station. The base station then processes the information and generates useful results. Reliable monitoring and high quality results can be achieved by using large number of sensors. As sensors can detect temperature, light, sound, gases, magnetic field, and any other such parameters, WSN are widely applicable in many areas, for civic or military purposes. It can be used for meteorological surveillance, environmental monitoring, health monitoring, security, intrusion detection, etc. It is believed that WSN will become more popular in the future.

A sensor node is a very small, light-weight, and self-organized *micro-electrical-mechanical* (MEMS) device. It may be as small as one centimeter in length and as light as 100 grams in weight. It has limited computational power and limited battery supply. Some examples of sensor nodes are shown in Figure 1.1. Besides sensing, a sensor node also acts as a router that helps transmitting data. A message may pass through several sensor nodes before reaching the base station.

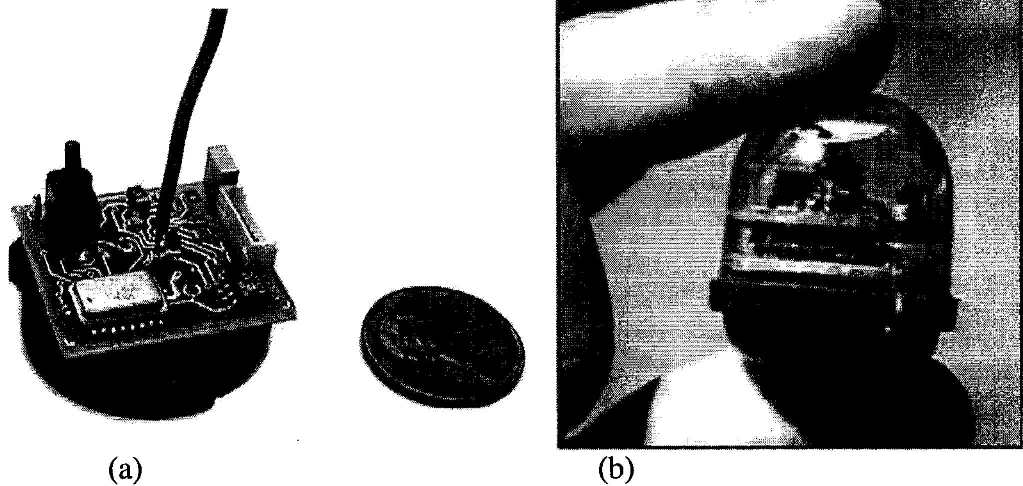
A typical sensor node may contain four subsystems (Raghunathan, Schurgers, and Srivastava (2002)): a *computing subsystem* which contains a microprocessor or microcontroller; a *communication subsystem* which consists of a short range radio for

wireless communication; a *sensing subsystem* that consists of a group of sensors and actuators and links the node to the physical world; and a *power supply subsystem*, which houses the battery and the DC-DC converter and powers the rest of the nodes.

Sensor nodes can be divided into three types according to their applications. The first type of sensor node is sensor node that detects the environment and reports to the base station periodically. This type of sensor node can be used in environmental monitoring and tracking. The second type is sensor node that detects the environment continuously and reports to the base station when the readings rise or drop to a pre-defined level. This type of sensor node can be used in instruction detection or environmental monitoring alarm. The third type is sensor node that reports the readings to the base station on user request. These three types of sensor nodes can also be combined to provide useful applications.

Ye, Heidemann, and Estrin (2002) have identified that the energy in WSN is mainly wasted in collision, overhearing, control packet overhead, and idle listening. *Collision* can happen when transmission is not organized and two packets collide at the receiver; the packet has to be discarded and energy is wasted. *Overhearing* occurs when one node hears packets that are destined to other nodes. If more energy is used in *control packet*, then less energy is used in transmitting useful data. Energy is also wasted in *idle listening* since the radio circuit takes up energy even though there is no traffic.

Because of its large-scale and limited power supply, it is not feasible and expensive to recharge the sensor nodes. A power-efficient protocol is necessary in WSN and there are many researches in different areas for power efficient protocols, e.g., localization, security, reliability, etc. It is a relatively new topic and most of the literature is after the year 2000.



(a) (b)

Figure 1.1 (a) UC Berkeley Cots Dust and (b) NASA JPL Sensor Webs Projects.

Source:
(a) Cots Dust: http://www-bsac.eecs.berkeley.edu/archive/users/hollar-seth/macro_motes/macromotes.html
("Copied with permission.")
(b) NASA JPL Sensor Webs Projects Briefing: http://sensorwebs.jpl.nasa.gov/resources/briefing1.shtml
("Copied with permission.")

2 **Design Issues**

In this section, some design issues about wireless networks will be discussed, as well as sensor networks. The following will be described: the propagation of radio wave in free-space, the two problems that result from this – the hidden station problem and the exposed station problem – then this is followed by some properties of direct and multi-hops connections and, finally, the relationship between routing and MAC protocols.

2.1 Free-space Propagation

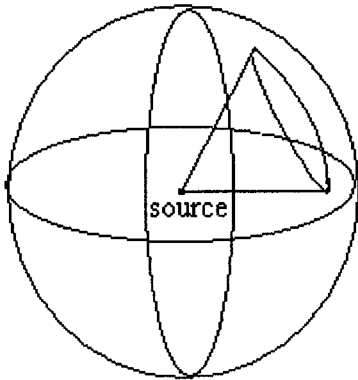


Figure 2.1 Radio Propagates as a Sphere.

Radio wave is an electro-magnetic wave. When a radio wave from an isotropic radiator transverses in a free-space, it propagates as a sphere centered at the source and the energy of the wave is distributed on the surface of the sphere (see Figure 2.1). Recalled that the surface area of a sphere is $4\pi r^2$, the power density of the wave is proportional to $1/r^2$ where r is the range. Because of this, wireless network has many properties different from a wired network. Some of them will be discussed here. Some equations on radio wave propagating in free space are reviewed in the followings:

The power density P of a free-space propagation of radio wave at a distance d is given by

$$P = \frac{P_T G_T}{4\pi d^2} \quad \text{............................... (1)}$$

where P_T is the transmitter power and G_T is the transmitter gain.

The power arriving at the receiver, P_R, is

$$P_R = P_T G_T G_R \left(\frac{\lambda}{4\pi d} \right)^2 \quad \text{............................... (2)}$$

where λ is the wavelength of the radio and G_R is the receiver gain.

The free-space path loss, L_P, the loss of power due to fading, is given by

$$L_P = \left(\frac{\lambda}{4\pi d} \right)^2 \quad \text{............................... (3)}$$

2.2 The Hidden Station Problem and the Exposed Station Problem

2.2.1 The Hidden Station Problem

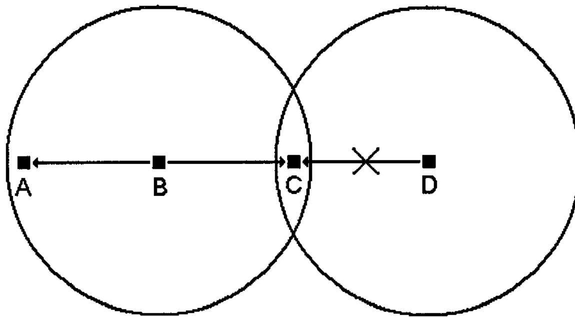


Figure 2.2 The Hidden Station Problem.

The most famous problem in designing MAC protocol for wireless network would be the *hidden station problem.* This happens because of the fading of signal: only the nodes close enough to the sender can hear the signal, but not nodes further away.

Consider a situation depicted in Figure 2.2, where the range of the radio of the nodes can only reach their immediate neighbors, i.e., node B can only talk to node A and node C but not node D; node C can only talk to node B and node D but not node A. Suppose that node D wants to talk to node C while node B is communicating with node C, then node D is not aware that node B is talking to node C since it is out of range of node B. It senses the channel and finds that it is clear to send to node C (which is not true). Node D sends out frames to node C and node C will receive both the frames from nodes B and D. All the frames will be corrupted. This is known as the hidden station problem and any MAC protocol designer should be aware of this.

2.2.2 *The Exposed Station Problem*

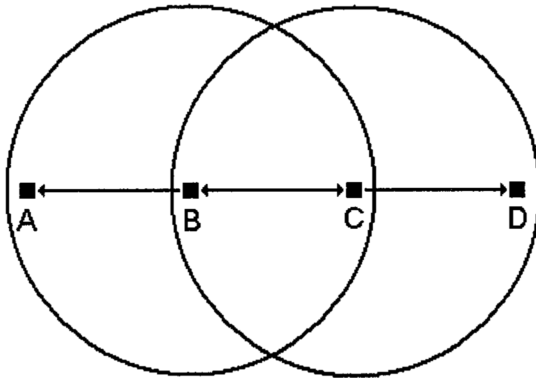


Figure 2.3 The Exposed Station Problem.

On the other hand, *exposed station problem* occurs when a node falsely concludes that it cannot send out data simply because it falls within the range of another transmitting node, as depicted in Figure 2.3. Suppose that node C wants to communicate with node D while node B is communicating with node A. Node C senses the channel and finds that the channel is busy and defers sending. In fact, frames from node C will not collide at node A since node A is out of the range of node C. It is safe for node C to send frames to D while node B is communicating with node A. Although the signal from node C will interfere at node B, as long as node B is not receiving, there is no problem. Note that this problem only introduces delays to the network but not impair the network.

The IEEE 802.11 wireless LAN uses CSMA/CA to overcome the two problems. This will be discussed in section 3.4.1.

2.3 Radio Model for Power Calculations

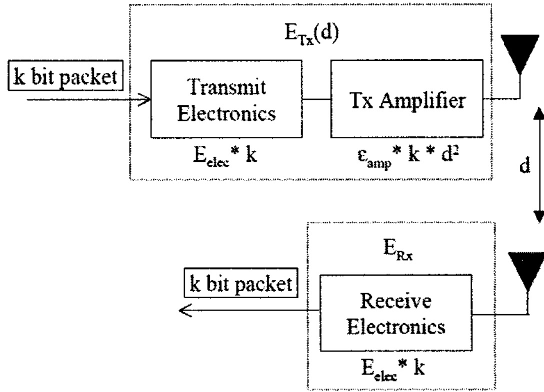


Figure 2.4 The First Order Radio Model.

Source:
Heninzelman, W. R., Chandrakasan, A. & Balakrishnan, H. (2000). Communication Protocol for Wireless Microsensor Networks. Proceedings of the Hawaii International Conference on System Sciences, 2000.
("Copied with permission.")

First order radio model presented by Heinzelman, Chandrakasan, and Balakrishnan (2000) will be used through-out this paper. The first order radio model models the transceiver of the radio as in Figure 2.4. It states that the energy used in transmission of a k-bit message of range d (in meter) is:

$$E_{tx}(k,d) = E_{elec}*k + E_{amp}*k*d^2 \quad (4)$$

and the energy used in receiving the k-bit message is

$$E_{rx}(k) = E_{elec}*k \quad (5)$$

As a result, the total energy used in one message transfer will be

$$E(k,d) = 2* E_{elec}*k + E_{amp}*k*d^2 \quad (6)$$

where E_{elec} is the transmitter and receiver circuitry and assumed to be 50 nJ/bits; and E_{amp} is the transmit amplifier and assumed to be 100 pJ/bit/m^2. These values are slightly better

than those of the state-of-the-art designs.

As seen from the equation, the energy usage depends on both the length and the range of the transmitted messages. The energy dissipation grows linearly with the length while grows quadratically with the range of the radio (see Figure 2.5).

Power can be saved if the number of bits being transferred or distance of transmission can be reduced. The number of bits can be reduced by compression or fusion; and the distance (in communication sense) can be reduced by breaking a direct link into several shorter links, i.e., $d \leq d_1 + d_2$ but $d^2 > d_1^2 + d_2^2$. Note that by breaking a direct communication, power is not always saved, as discussed below.

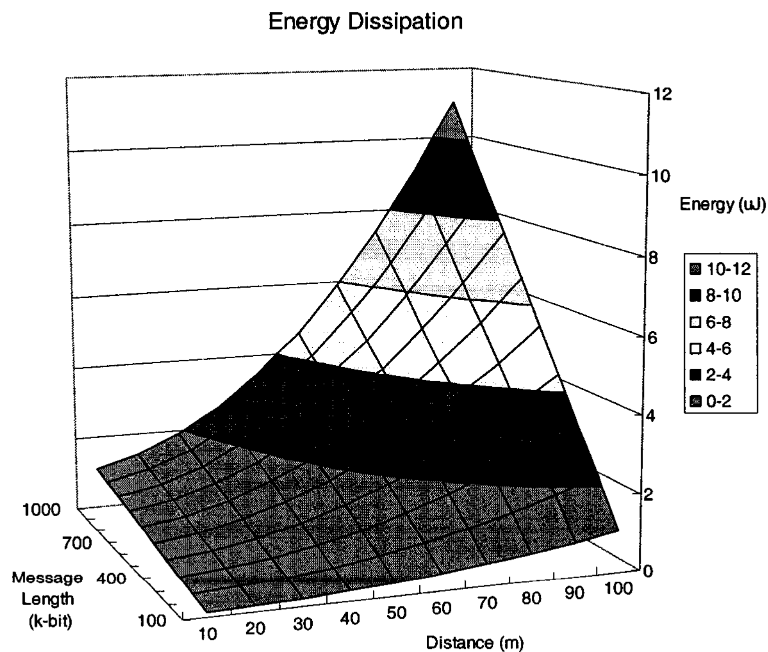


Figure 2.5 Energy Dissipation.

2.4 Direct and Multi-hops Connection

The routing methods can be roughly divided into two types: direct connection and multi-hops connection. In *direct connection*, each sensor node sends its message to the base station directly. This is the easiest and the simplest way to implement. In *multi-hops transmission*, each sensor node passes the message to a neighboring node towards the base station. Each message may pass through one or more intermediate nodes; each intermediate node acts as a router and passes the message to the base station. A routing algorithm should be used in this type of connection. The direct connection will have a "star" topology; the multi-hops connection will have a "chain" topology; and the types of connections in-between direct and multi-hops will have a "tree-like" or "mesh" topology.

This subsection will discuss the advantage and disadvantage (in terms of power efficiency and delay) of direct and multi-hops connections.

Firstly, consider the following topology (Figure 2.6):

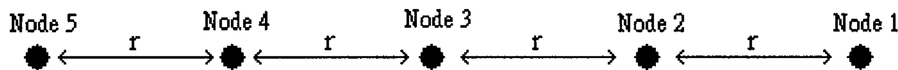


Figure 2.6 A Simple Liner Network.

It is a simple linear network with five nodes. Assume node 5 is the sink, the distance between each node is r, each has k-bit message to send, and there is only one channel. It is also assumed that the intermediate nodes will not fuse or compress the data. This may happen when raw data is needed; data is already compressed at source or the

hardware cannot support data fusion or compression.

2.4.1 Direct Connection

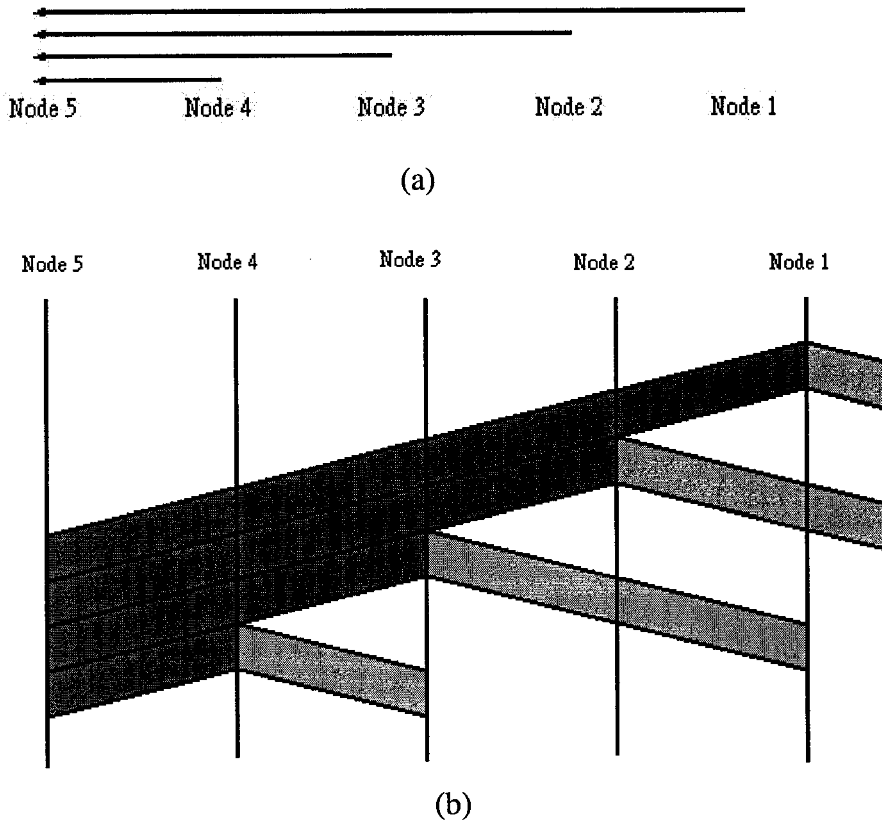


Figure 2.7 Direct Connection: (a) Transmissions and (b) Timeline.

In direct connection, every node sends its data to the sink directly (Figure 2.7(a)) and the time line is depicted in Figure 2.7(b). Since the radio is omni-directional, a node can hear the transmissions of adjacent nodes, as shown in the figure. The transmission for intended listener is marked in dark gray color while the "side effect" transmission is marked in light gray color.

Analysis

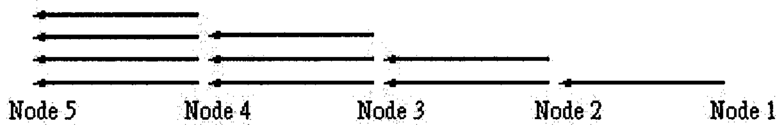
Number of messages exchange (transmitting) = n

Number of messages exchange (receiving) = 0

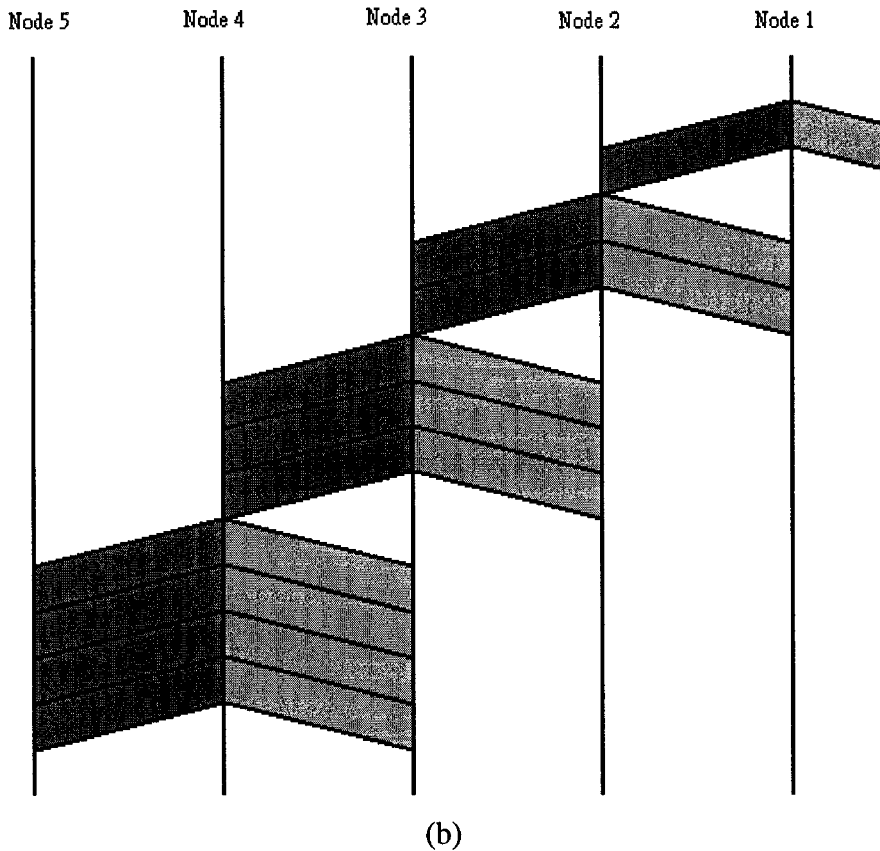
Time slots required = n

$$\text{Total energy consumed} = \sum_{i=0}^{n} \text{energy consumed by node i}$$
$$= \sum_{i=0}^{n} (E_{elec}*k + E_{amp}*k*d^2)$$
$$= \sum_{i=0}^{n} (E_{elec}*k + E_{amp}*k*(nr)^2)$$
$$= n*E_{elec}*k + E_{amp}*k*r^2*\sum_{i=0}^{n} n^2$$
$$= n*E_{elec}*k + [n*(n+1)*(2n+1)/6]*E_{amp}*k*r^2 \quad \text{............(7)}$$

2.4.2 Multi-hops Connection



(a)



(b)

Figure 2.8 Multi-hops Connection: (a) Transmissions and (b) Timeline.

In multi-hops connection, each node passes its data to its neighbors (Figure 2.8 (a)). After a node receives data from its neighbor, it will merge with its own data and send to the next hop, as shown in Figure 2.8 (b). As seen from the figure, there are more message exchanges than in direct connection. It is also found that the delay is much longer and the utilization of the channel is lower than that in direct communication.

The schedule can be slightly optimized by sending the packet earlier at nodes 4 or 5 (Figure 2.9 and Figure 2.10):

Optimization 1

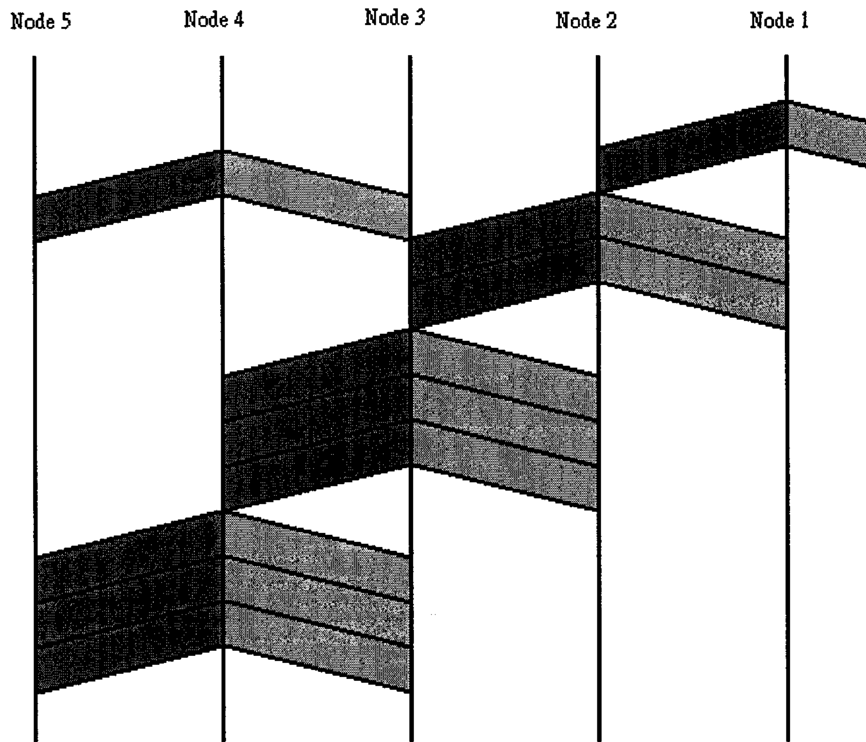


Figure 2.9 Optimization 1 of the Multi-hops Network.

Optimization 2

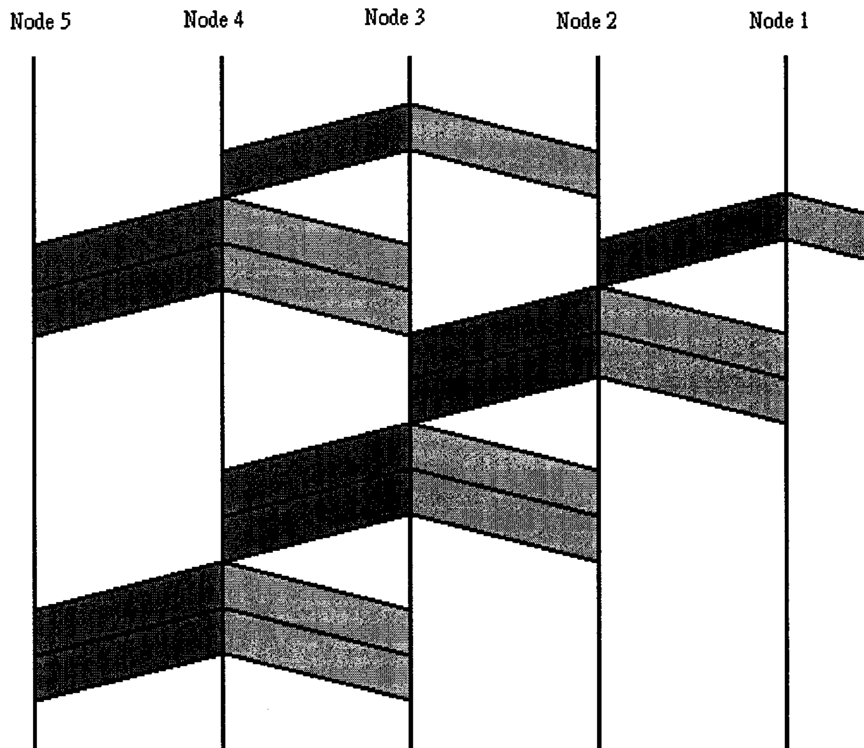


Figure 2.10 Optimization 2 of the Multi-hops Network.

Because of the properties of wireless network, a node cannot send out any packet when its neighbors are receiving. That means the two gray regions of the time line (no matter dark gray or light gray) cannot overlap at left and right at the same time for each node. The optimization in the network is limited. There are several researches studying how to utilize the free time slot in multi-hops connection.

Analysis:

Number of messages exchange (transmitting) = $\sum n$

Number of messages exchange (receiving) = $\sum (n-1)$

Time slots required = $\sum n = n(n+1)/2$

$$\text{Total energy consumed} = \sum_{j=0}^{n} \sum_{i=0}^{j} \text{energy consumed by node i in sending} + \sum_{j=0}^{n} \sum_{i=0}^{j-1} \text{energy consumed by node i in receiving}$$

$$= \sum_{j=0}^{n} \sum_{i=0}^{j} (E_{elec}*k + E_{amp}*k*d^2) + \sum_{j=0}^{n} \sum_{i=0}^{j-1} E_{elec}*k$$

$$= \sum_{j=0}^{n} \sum_{i=0}^{j} (E_{elec}*k + E_{amp}*k*r^2) + \sum_{j=0}^{n} \sum_{i=0}^{j-1} E_{elec}*k$$

$$= \sum_{j=0}^{n} j*(E_{elec}*k + E_{amp}*k*r^2) + \sum_{j=0}^{n} (j-1)\, E_{elec}*k$$

$$= \sum_{j=0}^{n} (2j-1)*\, E_{elec}*k + \sum_{j=0}^{n} E_{amp}*k*r^2$$

$$= n^2 *E_{elec}*k + n*(n+1)*E_{amp}*k*r^2/2 \quad \text{..........................(8)}$$

2.4.3 Comparison Between Direct and Multi-hops Connection

Heinzelman et al. (2000) states that the energy spent in multi-hops will be greater than the energy spent in direct connection when

$$\frac{E_{elec}}{E_{amp}} > \frac{r^2 n}{2} \quad \text{.............................. (9)}$$

The equation is slightly modified to

$$r^2 n < \frac{2*E_{elec}}{E_{amp}} \quad \text{...................................... (10)}$$

and recall that E_{elec} = 50 nJ/bits and E_{amp} =100 pJ/bit/m^2 ,

$$r^2 n < \frac{2 * E_{elec}}{E_{amp}} = \frac{2*50*10^{-9}}{100*10^{-12}} = 1000 \quad \text{.....................................} \quad (11)$$

For example, if the separation of the nodes is 10 m, a node will use more power in multi-hops connection when the number of hops is less than 10.

The work can be extended to calculate the energy dissipation in the whole network:

$$E_{direct} < E_{multi\text{-}hop}$$

$$n*E_{elec}*k + [n*(n+1)*(2n+1)/6]*E_{amp}*k*r^2 < n^2 *E_{elec}*k + n*(n+1)*E_{amp}*k*r^2/2$$

$$6*E_{elec} + E_{amp}*r^2*(n+1)*(2n+1) < 6*n*E_{elec} + 3*(n+1)*E_{amp}*r^2$$

$$E_{elec} / E_{amp} > [3*(n+1)*r^2 - (n+1)*(2n+1)*r^2] / [6 - 6*n]$$

$$= [(3n + 3 - 2n^2 - 3n - 1) * r^2] / [-6(n-1)]$$

$$= [-2(n^2 - 1) * r^2] / [-6(1-n)]$$

$$= [(1+n) * r^2] / 3$$

$$(1+n) * r^2 = 3 * E_{elec} / E_{amp} \quad \text{.................} \quad (12)$$

By substituting the values of E_{elec} and E_{amp},

$$r^2 (1+n) = \frac{3 * E_{elec}}{E_{amp}} = \frac{3*50*10^{-9}}{100*10^{-12}} = 1500 \quad \text{.................} \quad (13)$$

As a result, the whole network of multi-hops connection will use more power when the separation of the nodes is 10 m and the number of hops is less than 15.

2.5 Routing and MAC Protocols

Since wireless sensor networks are relatively simple when comparing with other networks (e.g., LAN or internet), the network layer and link layer are closely related, i.e., the routing protocols and MAC protocols are dependent on each other, and they both depend on the applications of the sensor network.

The protocols (both routing and MAC protocols) can be roughly divided into two categories, namely, route-oriented and MAC-oriented. In *route-oriented* protocol, when a new neighbor is found, a route to the cluster head or base station is assigned, followed by computing how to access the channel. In *MAC-oriented* protocol, a time slot (or CDMA code or FDMA band) is assigned when new neighbor is found. How to route to base station or any other node is a later matter. The two of them are not compatible. Some of the MAC and routing protocols will be discussed in sections 3.3 and 3.4.

3 **Current Development**

3.1 Mica2 and Mica2Dot – the Hardware

Mica2 (Figure 3.1 (a),(d)) and *Mica2dot* (Figure 3.1 (b),(e)) are two of the most popular general purpose sensor nodes used in research field. Both of them are manufactured by Crossbow. Both of them have Amtel ATMega128 as the processor, Chipcon CC1000 as the RF transceiver and a 10-bit ADC. Both of them have flash memory and EEPROM to store coding and sensed data.

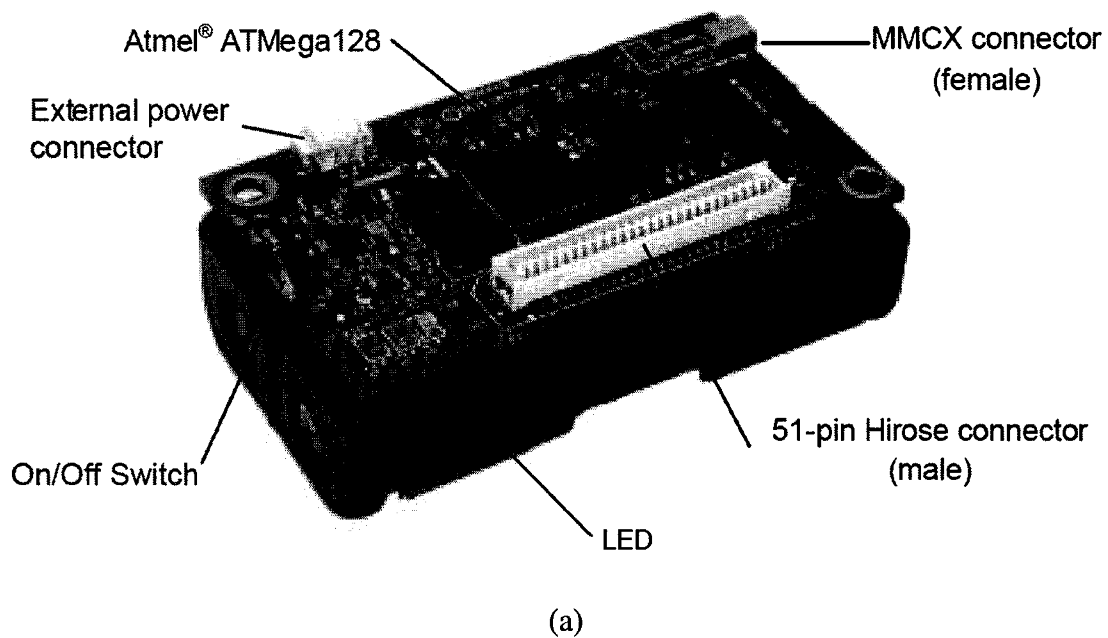
The Mica2 is box-shaped with dimension of 2.25"x1.25"x0.25"; and mica2dot is disc-shaped and the dimension is 1"x0.25". Mica2dot is much smaller and has only one LED while Mica2 is larger and has three LEDs and thus can provide more messages. Mica2 is powered by two AA batteries; and according to the datasheet, it can run for more than a year in sleep mode. Mica2dot is powered by a 3-volt coin cell.

Both Mica2 and Mica2dot can operate in the 914/433/315 ISM/SRD band and have a data rate of 38.4 kbaud. The Chipcon CC1000 can use Non-Return to Zero (NRZ) or Manchester coding. When NRZ is used, the data rate is 38.4 kbit/sec and halved when Manchester coding is used. The range of both the sensors can be up to 1000 ft. The radio power of both sensors is -20 - +10 dBm (0.01 to 10 mW) and the sensitivity is -101 dBm (8×10^{-11} W). The current draw with maximum transmission power is 25 mA and 8 mA for receiving.

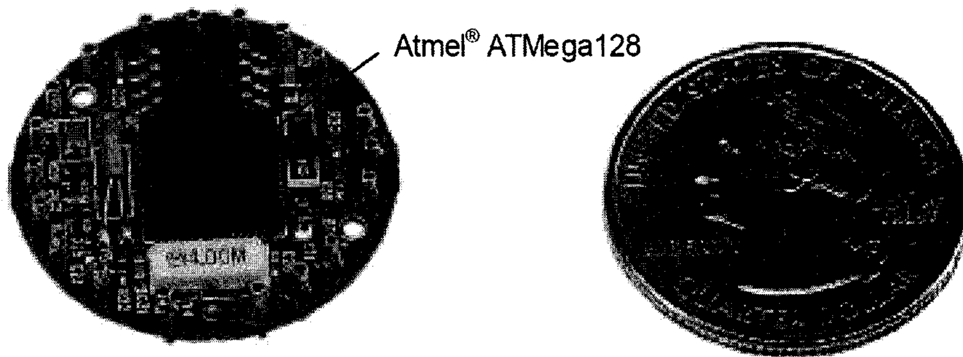
The two sensors have connectors which can connect to an external sensor board

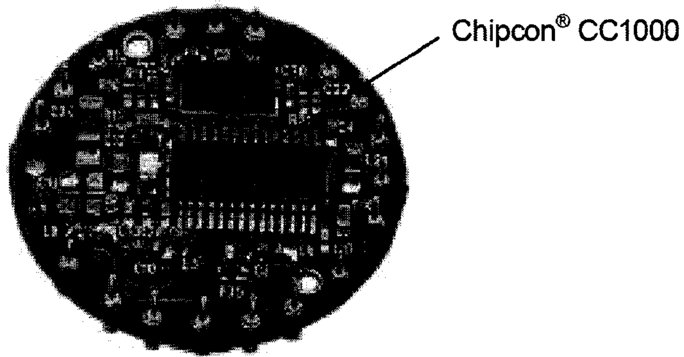
The two sensors have connectors which can connect to an external sensor board with sensors like temperature, humidity, barometric pressure, light, acoustic, sounder, magnetic, acceleration, etc.

Crossbow also manufactures an interface board (MIB510) (Figure 3.1(c)) which can act as a base station and connect the Mica and Mica2dot to a PC via a RS-232 port.

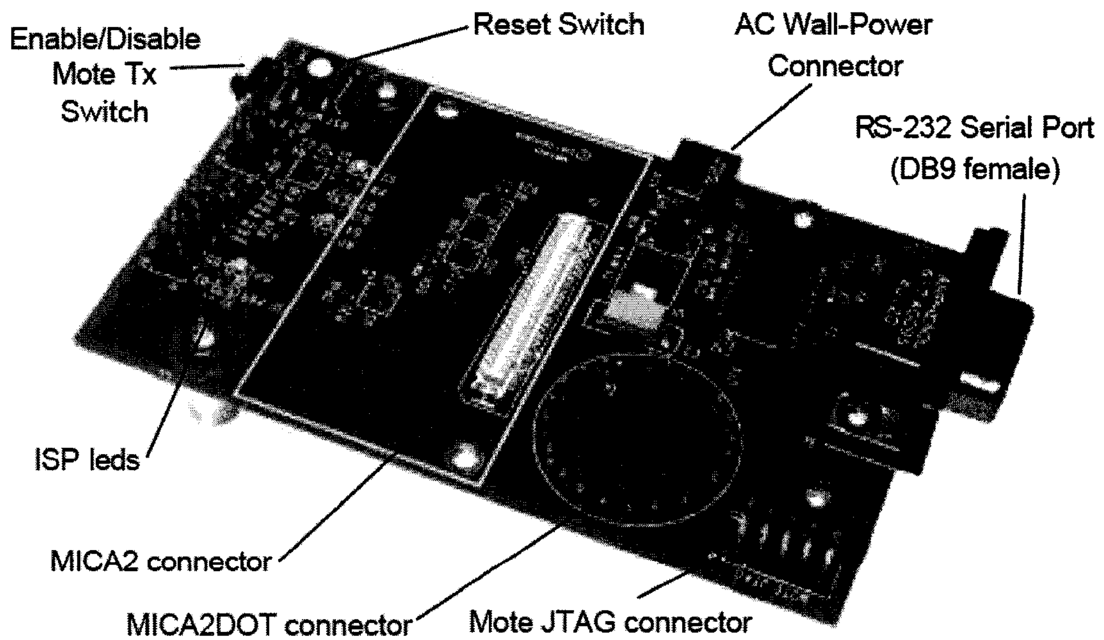


(a)





(b)



(c)

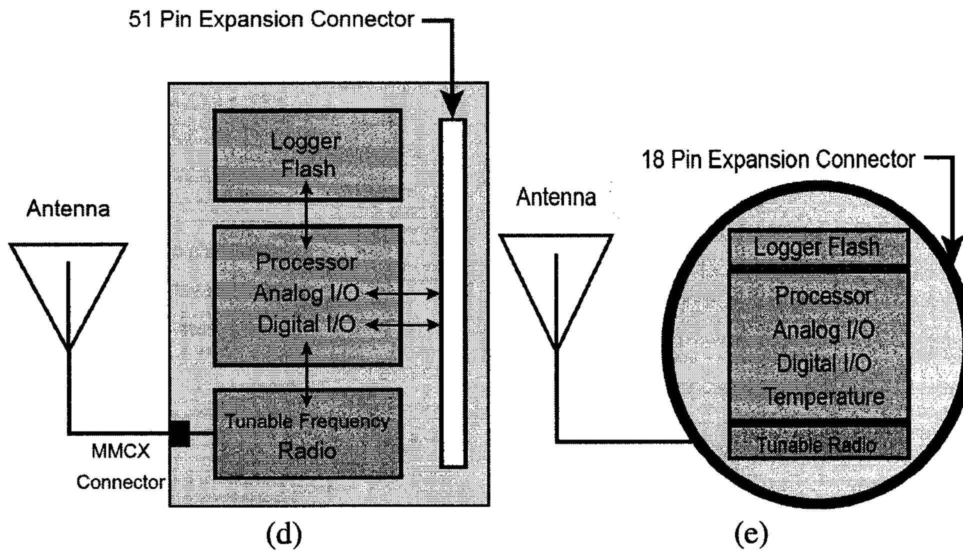


Figure 3.1 (a) Mica2, (b) Mica2Dot, (c) MIB510 (Base station), (d) Mica2 Block Diagram, and (e) Mica2Dot Block diagram.

Source:
(a), (b), (c) Mica2 and Mica2Dot Manual.
(d) Mica2 Datasheet.
(e) Mica2Dot Datasheet.
("All copied with permission.")

3.2 TinyOS – the Software

TinyOS is a set of tools that are used to deploy applications on sensor nodes. It is developed by the University of California, Berkeley. The reference platforms are mica, mica2, mica2dot, and mica128. The TinyOS package includes sample applications, documentations, tutorials, simulator, nesC compiler, libraries, and other tools.

3.2.1 nesC

nesC is a new language designed for use with TinyOS. It specifies the hardware that will be used by the application and their configuration, together with user defined

function to perform tasks. Each TinyOS application is formed by several components and they are connected by interfaces (Figure 3.2) and a component may provide or use more than one interfaces. The interface has commands and events (like C functions). When a component uses an interface, it can call the commands of the interface (like calling normal functions) and must implement all the events of the interface (like implementing call-back functions). When a component provides an interface, it can signal the events of the interface (like calling a call-back function from callee) and must implement the commands (like implementing a function for others to call).

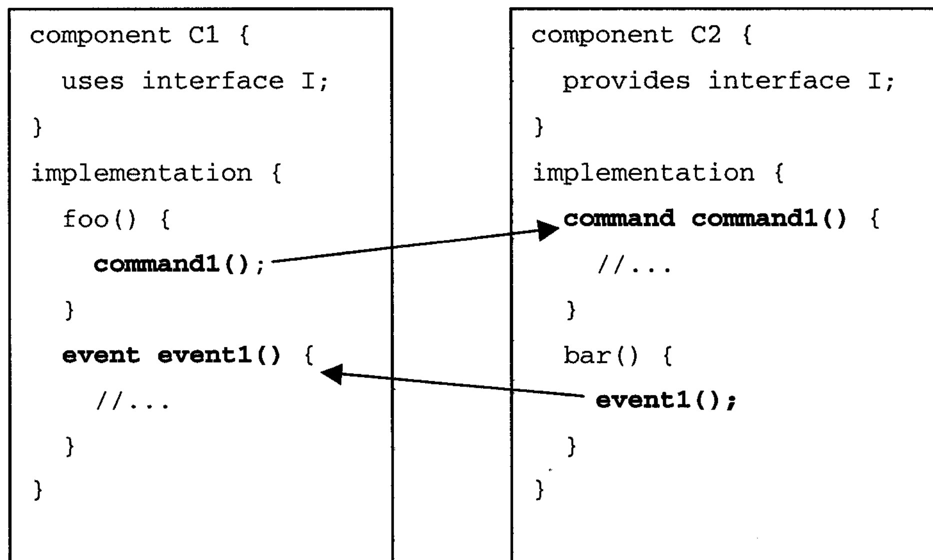


Figure 3.2 nesC.

3.2.2 TOSSIM

```
C:\WINDOWS\System32\cmd.exe
G:\test>main 3
0: MultiHopLEPSM timer task.
0:      addr    prnt    misd    rcvd    lstS    hop     rEst    sEst
0:      addr    prnt    misd    rcvd    lstS    hop     rEst    sEst
0: MultiHopLEPSM: Parent = 126
0: MultiHopLEPSM Sending route update msg.
0: QueuedSend: queue msg enq 0 deq 0
0: QueuedSend: Successfully queued msg to 0xffff, enq 1, deq 0
0: QueuedSend: sending msg (0x0)
0: QueuedSend: sending msg (0x0)
0: QueuedSend: sending msg (0x0)
0: QueuedSend: sending msg (0x0)
0: QueuedSend: sending msg (0x0)
0: QueuedSend: sending msg (0x0)
0: SurgeM: Timer fired
0: SurgeM: Got ADC reading: 0x104
0: SurgeM: Sending sensor reading
0: QueuedSend: queue msg enq 1 deq 1
0: QueuedSend: Successfully queued msg to 0x7e, enq 2, deq 1
0: QueuedSend: sending msg (0x1)
0: SurgeM: output complete 0x1
0: qent 1 dequeued.
1: MultiHopLEPSM timer task.
1:      addr    prnt    misd    rcvd    lstS    hop     rEst    sEst
1:      addr    prnt    misd    rcvd    lstS    hop     rEst    sEst
1: in chooseParent : TOS_LOCAL_ADDRESS=1, BASE_STATION_ADDRESS=0, ROUTE_TABLE_SIZE=16
```

(a)

```
C:\WINDOWS\System32\cmd.exe
G:\test>tossim_test -v
connected to TOSSIM
type=4, id=65535, time=0, size=4
03 00 00 00

type=0, id=0, time=18634304, size=512
data=MultiHopLEPSM timer task.

type=0, id=0, time=18634304, size=512
data=   addr    prnt    misd    rcvd    lstS    hop     rEst    sEst

type=0, id=0, time=18634304, size=512
data=   addr    prnt    misd    rcvd    lstS    hop     rEst    sEst

type=0, id=0, time=18634304, size=512
data=MultiHopLEPSM: Parent = 126

type=0, id=0, time=18634304, size=512
data=MultiHopLEPSM Sending route update msg.

type=0, id=0, time=18634304, size=512
data=QueuedSend: queue msg enq 0 deq 0

type=0, id=0, time=18634304, size=512
data=QueuedSend: Successfully queued msg to 0xffff, enq 1, deq 0
```

(b)

Figure 3.3 (a) TOSSIM (Running Application Surge) and (b) Program for Testing TOSSIM.

The applications of TinyOS are designed to run on sensor nodes. This makes it hard to observe the activities and the status of an application and to debug the

applications. *TOSSIM* can let the applications run on a PC with Linux or Windows with Cygwin and output the debug messages to user easily (see Figure 3.3).

TOSSIM is the TinyOS simulator. It allows the application to run on a Personal Computer (PC) instead of running it on a sensor node. By compiling the application with a "pc" flag, the nesC compiler can compile the nesC source code to an executable on PC. The nesC compiler first generates an intermediate C source code "app.c", then a generic C compiler is used to compile the intermediate file to the executable "main.exe".

TOSSIM has two helping threads other than the main threads; one of which accepts new listeners of the application, and the other accepts new commands.

TOSSIM has an event queue (which is a heap). The main thread will persistently pull out an event from the event queue and handle that event. The event may then trigger some other events and insert them into the queue to wait for being handled. These events may further trigger other events and insert into the queue and so on.

TOSSIM is run on Linux or Cygwin (or command prompt with appropriate configuration). Cygwin is a Linux-like environment on Windows machines on which one can run Linux command. TOSSIM can be run with a number of flags to specify the behavior of the simulation, e.g., boot-up time, time-scale relative to actual time, lossy ratio, etc. One has to specify the environment variable "DBG" to observe the desired

output from the TOSSIM. Types of output include AM, ROUTE, SIM, USR1, USR2, USR3, etc.

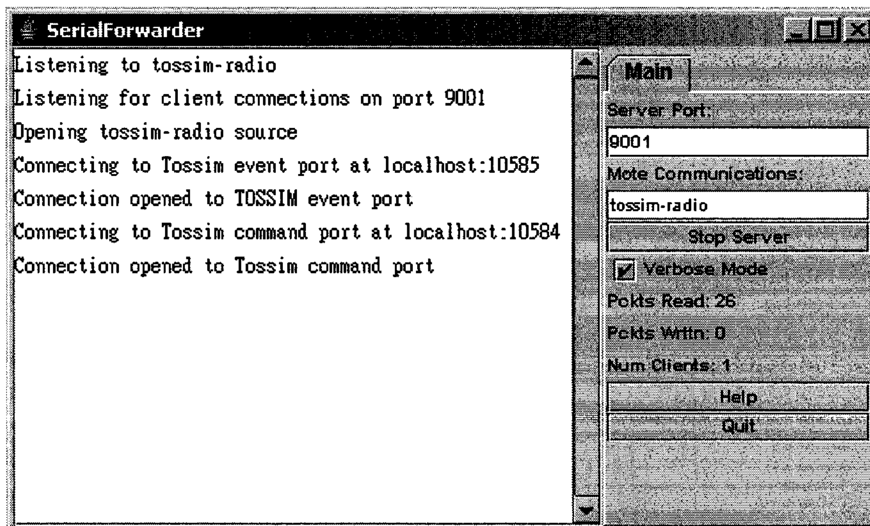
The user can also observe the output using TCP/IP protocol. The TOSSIM opens two TCP ports, one for sending event outputs to the listener (or client) and one for reading command from the client. The event port is 10585 and the command port is 10584.

The TOSSIM packet format is as follows (Figure 3.4):

2 bytes	2 bytes	8 bytes	2 bytes	max. 65536 bytes
message_type	mote_id	time-stamp	payload_size	payload

Figure 3.4 TOSSIM Packet Format.

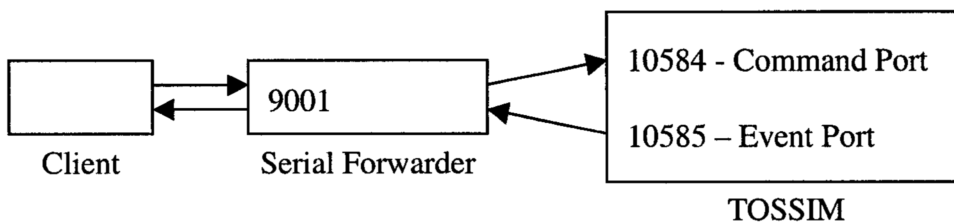
3.2.3 Serial Forwarder



(a)

```
C:\WINDOWS\System32\cmd.exe
G:\test>sf_test
connected to Serial Forwarder
addr=65535, handleID=250, groupID=125, payload_size=10
00 00 00 00 00 00 00 7e 00 00

addr=65535, handleID=250, groupID=125, payload_size=10
00 00 00 00 00 00 00 7e 00 00

addr=65535, handleID=250, groupID=125, payload_size=10
00 00 00 00 00 00 00 7e 00 00

addr=65535, handleID=250, groupID=125, payload_size=10
00 00 00 00 00 00 00 7e 00 00

addr=65535, handleID=250, groupID=125, payload_size=10
00 00 00 00 00 00 00 7e 00 00

addr=65535, handleID=250, groupID=125, payload_size=10
00 00 00 00 00 00 00 7e 00 00

addr=65535, handleID=250, groupID=125, payload_size=10
01 00 01 00 00 00 ff ff 00 00

addr=65535, handleID=250, groupID=125, payload_size=10
02 00 02 00 00 00 ff ff 00 00
```

(b)



(c)

Figure 3.5 Serial Forwarder: (a) Serial Forwarder Application, (b) Program for Testing Serial Forwarder, and (c) a System View.

In practice, the sensor network may deploy at a remote site from the back-end server. To receive data from the remote sensor network and send command to individual sensor node, the user can use *Serial Forwarder* (Figure 3.5 (a),(c)).

Serial Forwarder serves as a bridge between the sensor network and the computer network (perhaps the Internet). One of the sensor nodes will act as a base station and connect to a PC via the UART serial port (e.g., COM1). The Serial Forwarder will listen

to the TCP port 9001 and accept any client connect to that port. The Serial Forwarder then relays the message received from the serial port (i.e., from the base station and thus from the sensor network) to the TCP client or sends command from the client to the sensor network. Thus the user can monitor as well as control the sensor network at a remote site.

The Serial Forwarder can also work with TOSSIM. In this case, the Serial Forwarder forwards message to and from another TCP port instead of a real sensor node. The Serial Forwarder relays the message from the TOSSIM event port (10585) to the Serial Forwarder port (9001) and sends command received from the Serial Forwarder port (9001) to TOSSIM command port (10584) instead of using the serial port (e.g., COM1).

Two modes can be used with TOSSIM, namely, Tossim-radio and Tossim-serial. Tossim-radio sits on top of the network and replies any message sent or received in the simulated network and the Tossim-serial only relays the message received by the base station (i.e., the sensor node with id 0).

When connected to Serial Forwarder, the client first sends a block letter "T" and followed by a space, i.e., "T ", the Serial Forwarder then sends back a "T " to the client then the content of the message. The sensor packet is encapsulated into the Serial Forwarder packet, which just has one additional packet length field.

The Serial Forwarder packet format is shown in Figure 3.6.

1 byte	2 bytes	1 byte	1 byte	1 byte	
Packet len.	Dest. addr.	handlerID	groupID	Payload-len.	payload

Figure 3.6 Serial Forwarder Packet Format.

To analyze the TOSSIM and Serial Forwarder, two programs have been written and used to connect to TOSSIM and Serial Forwarder separately. Data collected from TOSSIM and Serial Forwarder are analyzed and output to user. The results of the two programs are depicted in Figure 3.3 (b) (for TOSSIM) and Figure 3.5 (b) (for Serial Forwarder). The source codes of the two programs are listed in Appendices A and B (for TOSSIM and Serial Forwarder respectively). A stack trace for the Serial Forwarder with TOSSIM has also been done and the result is listed in Appendix C.

3.2.4 TinyViz

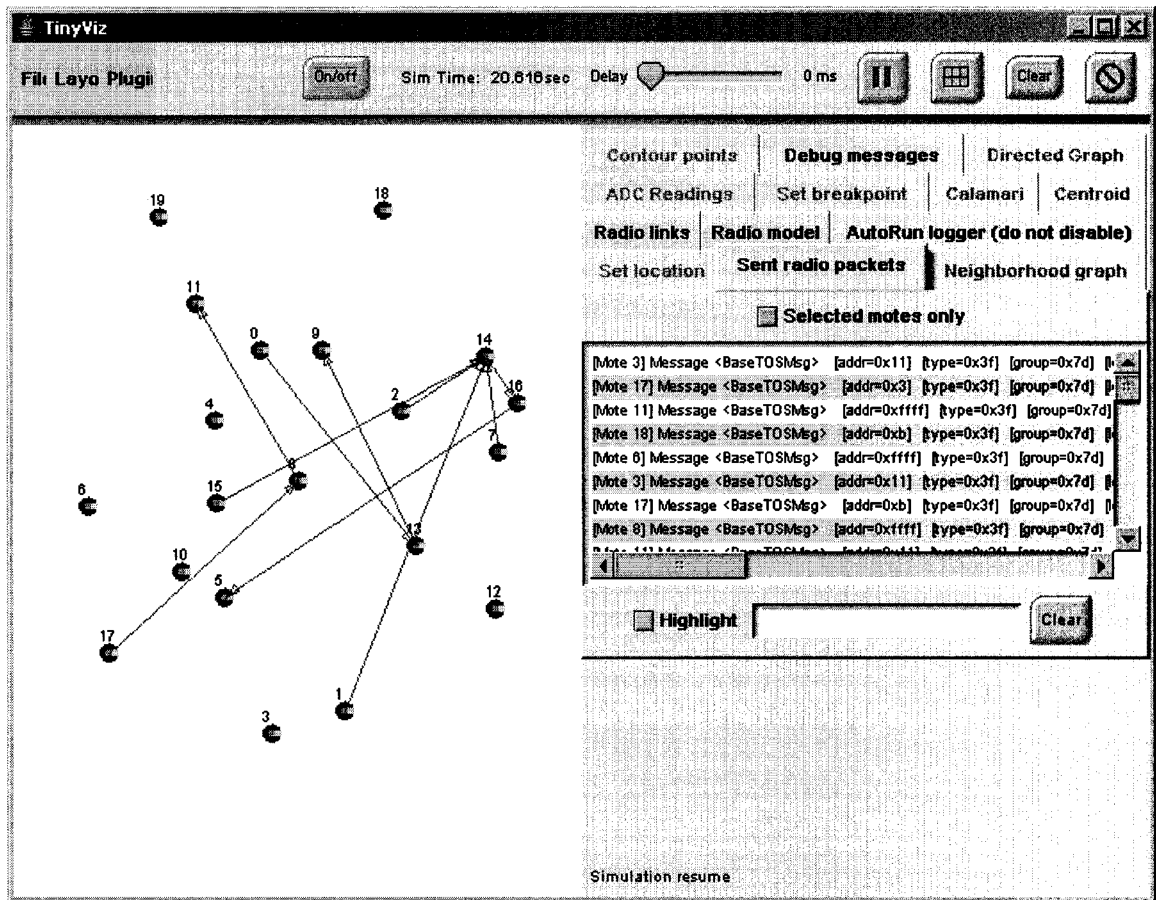


Figure 3.7 TinyViz Application.

TinyViz (Figure 3.7) is a Java graphical user interface program that visualizes TOSSIM. It can display the interaction between the sensor nodes on the screen.

TinyViz executes the TOSSIM by calling “Runtime.getRuntime().exec()” and creates a thread for Serial Forwarder. TinyViz then connects to the event port and the command port of the TOSSIM and displays the result received from the TOSSIM event port to the graphical interface.

Besides displaying the interaction, TinyViz also has a set of plug-in functions to control the sensor node or to specify simulation options. Examples of plug-in include debugging, setting break point, setting range of radio, specify radio link, etc. This can extend the functionality of TOSSIM and perform user defined job.

TinyViz can run with an "autorun" script that automatically configures the simulation, like loading plug-ins into the TInyViz, specifying parameters for TOSSIM, and setting environmental variables and log-file.

3.3 Routing Protocols

In the following two sections, some current routing and MAC protocols in the literature will be discussed. The routing protocols discussed include direct, LEACH, PEGASIS, and HIT; the MAC protocols discussed include S-MAC, SMACS, and the TDMA schedule of HIT.

3.3.1 Direct Communication

Direct communication is the simplest and the most intuitive way to send and collect sensor data. In direct connection, each sensor sends the data to the base station directly. The advantage of this protocol is its simplicity to implement. The main drawback of this protocol would be the large amount of energy used by nodes further away from the base station. In the view of equation (4), the energy drains more rapidly as the distance from the base station grows.

3.3.2 Low-Energy Adaptive Clustering Hierarchy (LEACH)

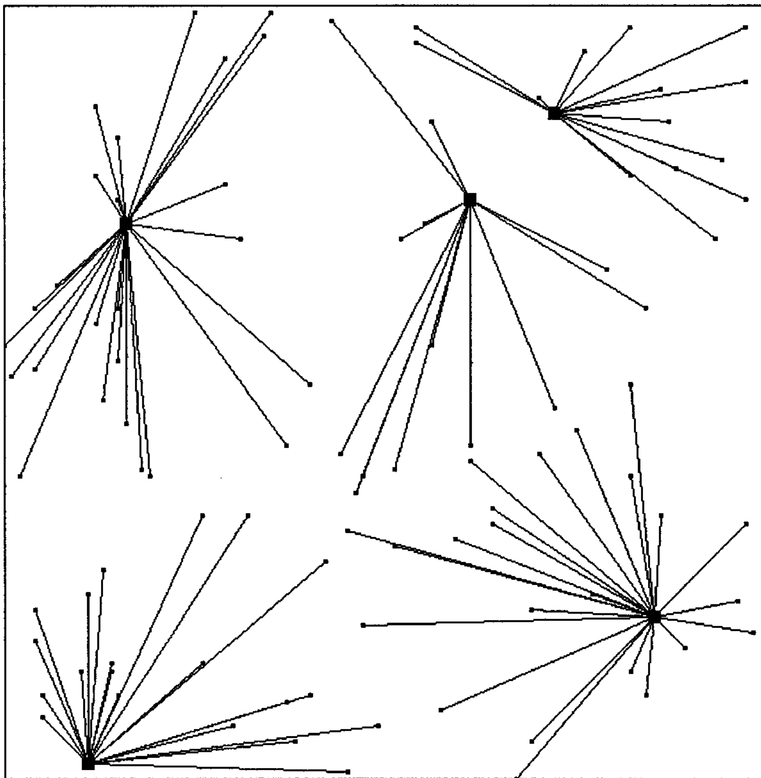


Figure 3.8 An Example of Low-Energy Adaptive Clustering Hierarchy (LEACH).

Low-Energy Adaptive Clustering Hierarchy (LEACH), Figure 3.8, is proposed by Heinzalman, Chandrakasan, and Balakrishnan (2000). The basic idea of this protocol is to divide the entire network into clusters. Each cluster has a cluster head and nodes in each cluster send their data to the cluster head; the cluster head then collects the data and relays to the base station. Each sensor will be the cluster head in turn and as the cluster head changes, the cluster will also change. Each cluster will use different CDMA codes. This protocol assumes symmetric links and the capability of nodes to communicate using different CDMA. Details of the protocol are as follows:

Phase One: Advertisement

In this phase, sensor node will self-elect to be a cluster-head. Each sensor in the network will choose a random number between 0 and 1 and compare with a threshold T(n), which is given by:

$$T(n) = \begin{cases} \dfrac{P}{1 - P * (r \bmod \frac{1}{P})} & \textit{if } n \in G \\ 0 & \textit{otherwise} \end{cases}$$

where n is the node ID, P is the desired percentage of cluster heads, e.g. 0.05, r is the current round and G is the set of nodes that have not been cluster heads in the last 1/P rounds. If the random number chosen is smaller than T(n), the node will become a cluster head. Therefore, when a node is elected to be a cluster head, it will not become a cluster head again in the next 1/P round (not in the set G).

The node that becomes a cluster head will broadcast advertisement message with fixed signal strength by using CSMA. The other nodes will listen to this advertisement, compare the strength from different cluster heads and join the cluster head with the strongest signal strength.

Phase Two: Cluster Set-Up

When a node joins a cluster, it will send a message to the cluster head to register by CSMA.

Phase Three: Schedule Creation

After received the registration message, the cluster head will create a TDMA schedule telling the members when to send data.

Phase Four: Data Transmission

This is the steady-state of LEACH, each node will send the data according to the TDMA schedule. Non-cluster-head-node can turn off the radio when not sending in order to save power. To avoid collision between different clusters, different CDMA codes are used in different clusters.

Conclusions

This is an example of branch-based protocol. This protocol is easy to implement and scalable. However, this protocol is not very energy efficient for nodes further from the cluster head since it uses a direct connection from non-cluster-head-node to cluster head which has a higher cost when compared to a multi-hops link. Also, the radio of cluster head must be turned on all the time which drains the power more rapidly. It is worth mentioning that this paper is one of the most frequently cited papers.

3.3.3 Power-Efficient Gathering in Sensor Information Systems (PEGASIS)

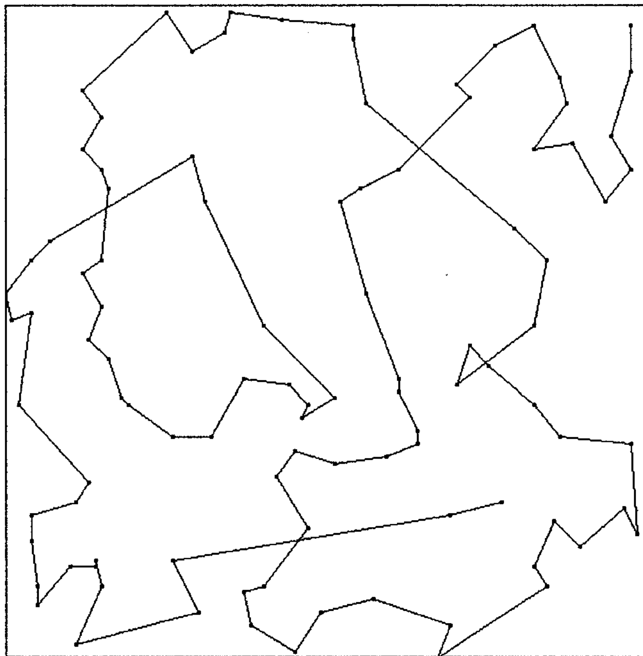


Figure 3.9 An Example of Power-Efficient Gathering in Sensor Information Systems (PEGASIS).

Power-Efficient Gathering in Sensor Information Systems (PEGASIS), Figure 3.9, is proposed by Lindsey and Raghavendra (2002). In contrast to LEACH, PEGASIS uses the chain-based approach. In fact, the whole network is connected by a single chain. To form the chain, PEGASIS starts from the node which is furthest away from the base station. It then uses a greedy algorithm to build the chain and the neighboring distances will increase gradually. When a node dies, the chain is reconstructed. It assumes every node has the global knowledge of the other nodes.

In each round of data transmission, one of the nodes in the chain will act as a head. The other nodes will send the data towards the head along the chain and the head will

relay the data to the base station. Two methods are mentioned in two different papers, the first one is passing the data hop-by-hop towards the head, then to the base station as depicted in Figure 3.10 (a). The second one is transmitting the data between pairs of nodes and eventually to the base station, as depicted in Figure 3.10 (b).

```
A → B → C ← D ← E          A→B  C→D  E→F  G→H
        ↓                       B→D  F→H
   Base Station                   D←F
                                   D
                                   ↓
                              Base Station
      (a)                         (b)
```

Figure 3.10 Two Different Methods of Transmission in PEGASIS

The main advantage of this protocol is the small total energy dissipation as each node only needs to communicate with it neighbors. The disadvantages are that it may not be practical to have a global knowledge of the network, and also it is not scalable. Also, when the chain grows longer, the delay will become undesirable. Furthermore, as mentioned in the last section, the amount of energy saved depends on the aggregation of the data. If the data cannot be fused or the compression ratio is low, more energy will be spent when the data passing through the nodes.

3.3.4 Hybrid Indirect Transmissions (HIT)

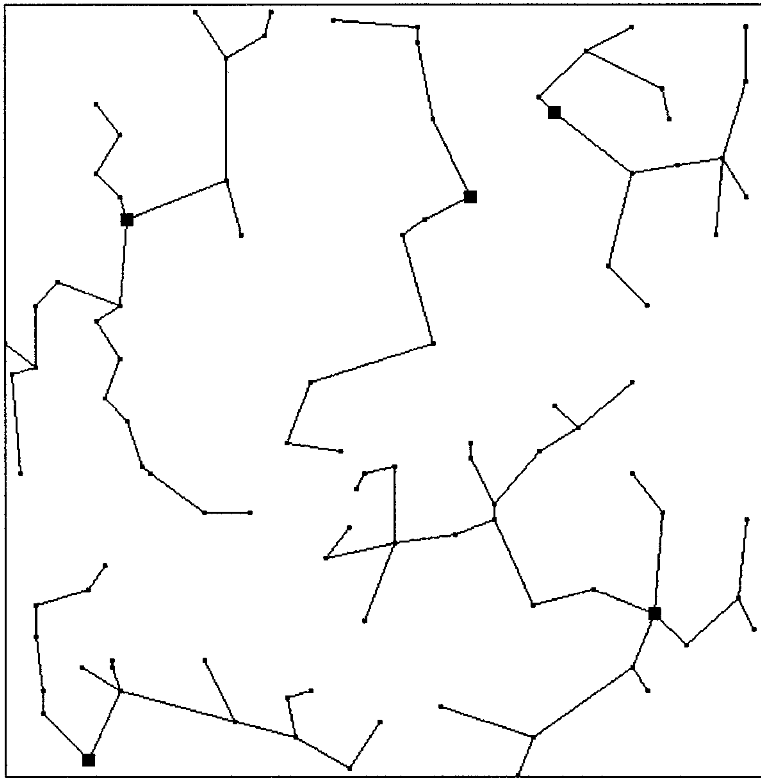


Figure 3.11 An Example of Hybrid Indirect Transmissions (HIT).

Hybrid Indirect Transmissions (HIT), Figure 3.11, is proposed by Culpepper, Dung, and Moh (2003) from NASA and SJSU. HIT contains a routing (or path setup) protocol and a TDMA schedule protocol. In this subsection, the routing protocol will be discussed and the TDMA schedule protocol is deferred to Section 3.4.4.

HIT is a clustered and branch-based protocol. Similar to LEACH, the network is divided into a number of clusters and each cluster has a cluster head to relay the messages; rather than sending the data to the cluster head directly, each node in HIT sends the data

hop-by-hop. The paths in each cluster can be viewed as a tree structure. Details of the protocol are as follows:

Phase One: Cluster Head Election

HIT states that it can be used with single cluster or multiple clusters depending on application. Node may become a cluster head depending on node ID, connectivity, energy or just randomly.

Phase Two: Cluster Head Advertisement

Similar to LEACH, a node will broadcast an advertisement with fixed signal strength after becoming a cluster head. The non-cluster-head-nodes will listen and compute the distance to the cluster head; save it as d(H, j) where H is the cluster head and j is the node ID.

Phase Three: Cluster Setup

In this phase, all non-cluster-head-node broadcasts a message containing d(H, j) with fixed signal strength to tell the others of its existence. Each node receiving this message knows its neighbors and the distance from the neighbors; and it will save the distance as d(i, j) where i is the node ID of the neighbor.

After receiving all the messages, each node has complete knowledge of its neighbors (distance to neighbors, distance from neighbors to cluster head). Then, the

node computes its upstream neighbor. Node u will become the upstream (of node i) if

$$\begin{cases} d(i,u) < d(i,H) \\ d(u,H) < d(i,H) \\ \min d(i,u) \end{cases}$$

Phase Four: Route Setup

In this phase, each node broadcasts a message containing the upstream neighbor and the distance to its upstream neighbor. Any node j hears node i claiming node j is the upstream neighbor will add node i to DOWN(j), which later is used for TDMA schedule.

After this phase, the HIT starts TDMA scheduling and data transmission.

Conclusions

HIT combines the advantages of branch-based and chain-based network: low delay and low energy cost path. HIT is also relatively simple to implement. Like LEACH, it uses the cluster method to make the network scalable.

The main drawback of HIT would be the limit in the search of the upstream neighbor u. Since the equation states that d(i, u) < d(i, H) and d(u, H) < d(i, H), the solutions of the two equations would be the overlap region of two circles with radii d(i, H) and center at i and H (the shadowed region in Figure 3.12):

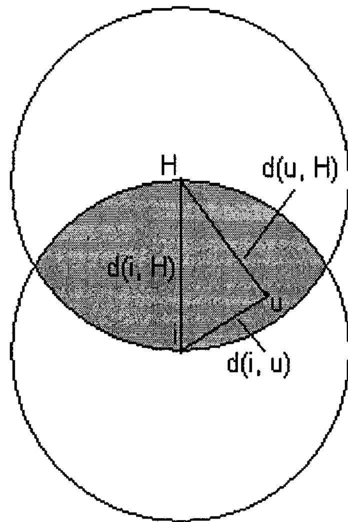


Figure 3.12 Limited Search Area of Upstream Neighbors in HIT.

In fact, an upstream neighbor may be found with a lower cost path (in communication sense) outside this region. This is the main motivation of the effort in deriving a new protocol: to find an upstream neighbor with lower cost wherever the upstream neighbor is located.

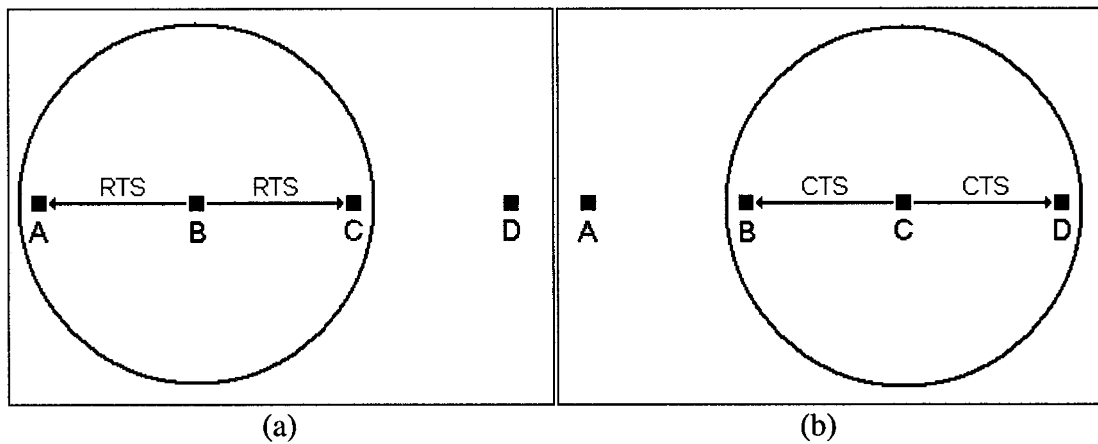
The second drawback would be the total coverage of the advertisement message. In order to find an upstream neighbor, a node must know the distance to the cluster head. This means a node must hear at least one cluster head and the advertisement message must cover the whole network. This introduces a great overhead to HIT since such a transmission is expensive. Therefore the second motivation for a new protocol is to propagate the advertisement hop-by-hop instead of one long range transmission.

It is clear that the newly proposed protocol should be clustered to minimize delay

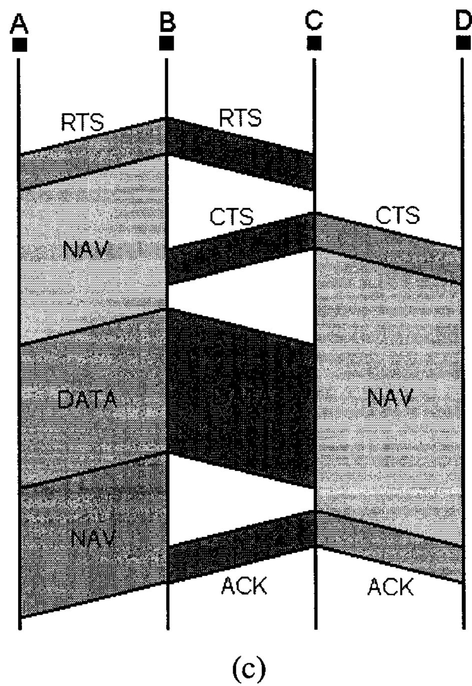
and the transmission should be hop-by-hop to reduce transmission power. It will be seen that these two issues are both addressed in the proposed protocol, to be discussed to in section 4.

3.4 Medium Access Control Protocols

3.4.1 Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)



(a) (b)



(c)

Figure 3.13 (a) Sender Sends a Request-To-Send, (b) Receiver Replies a Clear-To-Send, and (c) Timeline and the Network Allocation Vector in CSMA/CA.

Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) is the MAC protocol used in 802.11 wireless LAN. The details are as follows:

When a node wants to communicate with another node, it first senses the channel to see if the channel is busy or not. If the channel is busy, it defers sending until it is clear. If the channel is clear, it sends a Request-To-Send (RTS), see Figure 3.13 (a), packet to the receiver. After receiving the RTS, the receiver will send a Clear-To-Send (CTS), see Figure 3.13 (b), packet back to the sender. Then the sender starts sending data and the receiver sends back an acknowledgment for each frame.

By using RTS/CTS, the neighbors of both of the sender and receiver know there is a transmission and will not start a new transmission. In addition, the RTS/CTS packets contain information about the length of transmission and the neighbors can estimate how long they should wait. The length of time is called the Network Allocation Vector (NAV), Figure 3.12 (c). It decreases as time goes on. If NAV is non-zero, it means the channel is busy and the node will not start sending.

The RTS/CTS protocol is good for the wireless LAN where the devices in the network have more power supply and the RTS/CTS frames are relatively small when

compared with the data frames. However, in the sensor network, the length of data is usually short and the overhead of RTS/CTS becomes more prominent. Also, sensors are likely to sense and send the data at about the same time, which causes the RTS/CTS packets to be corrupted and retransmission of the RTS/CTS are required. As a result, RTS/CTS are not suitable for wireless sensor networks.

3.4.2 Sensor-MAC (S-MAC)

Sensor-MAC (S-MAC) is proposed by Ye, Heidemann, and Estrin (2002). It is a flat protocol (i.e., does not use a hierarchy approach). The basic idea of S-MAC is to switch the radio of a node on and off alternately to minimize idle listening and hence power is saved (see Figure 3.14). To reduce overhead, each node tries to synchronize the schedule with its neighbors. The contention method used in the listening period is RTS/CTS.

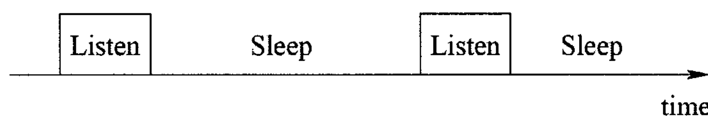


Figure 3.14 Periodic Listen and Sleep in Sensor-MAC (S-MAC).

Source:
Ye, W., Heidemann, J., & Estrin D. (2003). Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks. Technical Report ISI-TR-567, USC/Information Sciences Institute, January, 2003.
("Copied with permission.")

Schedule Broadcast

When the protocol starts, every node in the network listens for a period of time. If a node does not receive any message, it will choose its own schedule; if the node receives a schedule message, it follows that schedule. It is possible for a node to receive a schedule different from its own, e.g., node at boundary of two schedules. The node may adopt the two schedules or adopt one and save the other one.

Synchronization

The synchronization in S-MAC is not as strict as that in TDMA scheme. It is done in two ways. The first way is that all the timestamp is relative rather than absolute. The second way is to adopt a listening period long enough to tolerate clock drift. To synchronize the listen and sleep periods between nodes, each node sends a SYNC packet to tell the other nodes when it will go to sleep periodically.

Adaptive Listening

S-MAC also introduces adaptive listening to minimize delay. When a node overhears its neighbor sending or receiving message in the listening period, it wakes up at the end of that transmission to see if it is the next hop node, even it is in the sleep period. It starts receiving if it is the next hop node; otherwise, it turns back to sleep.

Conclusions

S-MAC can significantly reduce the time and energy used in idle listening.

However, as mentioned above, the over head of RTS/CTS is large and they cause collisions. It is not a very desirable solution to sensor network. Also, the flat topology makes it only applicable to peer-to-peer routing algorithm.

3.4.3 Self-Organizing Medium Access Control for Sensor Networks (SMACS)

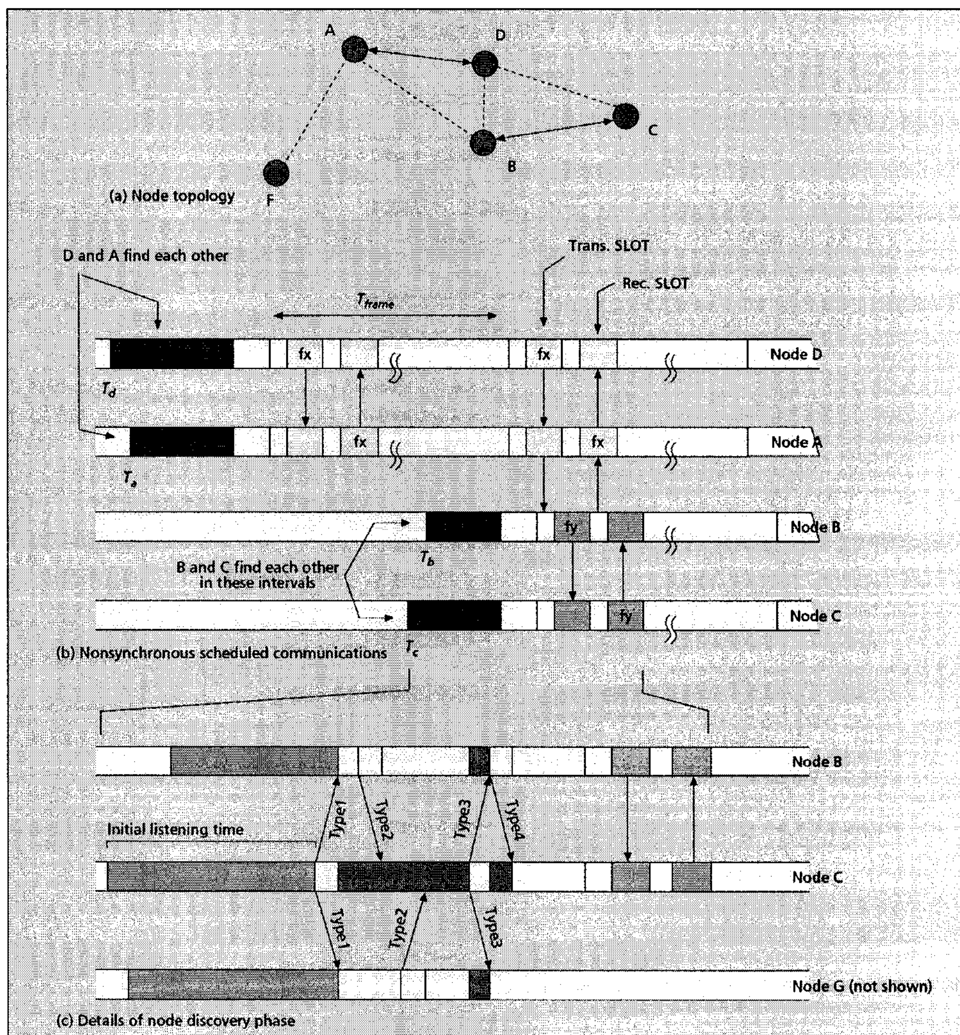


Figure 3.15 Self-Organizing Medium Access Control for Sensor Networks

Source:
Sohrabi, K., Gao, J., Ailawadhi, V., & Pottie G. J. (2000). Protocols for Self-Organization of a Wireless Sensor Network. IEEE Personal Communications, vol. 7, no. 5, pp. 16-27, October 2000.
("Copied with permission.")

Self-Organizing Medium Access Control for Sensor Networks (SMACS), Figure 3.15, is proposed by Sohrabi, Gao, Ailawadhi, and Pottie (2000). Similar to S-MAC, SMACS is a flat protocol and nodes in the network only communicate with their neighbors. The major difference between SMACS and S-MAC is that SMACS assigns a specific time-slot and frequency band to different neighbors (or links) and none of the neighboring nodes will content for same channel.

As an example, when node A and node D in Figure 3.15 wake up and find each other, a frequency band fx is assigned to that link; later, when node B and node C wake up, another frequency band fy is assigned, so that node B and node D will not interfere with each other even the time slots collide.

SMACS assumes that there is relatively large number of frequency bands available in the network (There are 2600 - 10 kb/s frequency bands in 902-928 MHz ISM.) and the nodes are able to communicate in different frequency bands (i.e., all the nodes support the use of FDMA).

When a node wakes up, it will listen to a channel of particular frequency band for some random time period. If it does not receive invitation message from other nodes, it

will broadcast the invitation message (TYPE 1 message), as node C in Figure 3.15. The node is called inviter. Any node receiving the TYPE 1 message will reply a respond message (TYPE 2 message) with random delay to avoid collision, as node B and node G in the figure. These nodes are called invitees. After receiving the TYPE 2 messages, the inviter will only choose one of the invitees (node B in this case) and assign a link with a frequency band to it. The inviter then sends a TYPE 3 message to the chosen node advising the schedule information. The chosen invitee will then find a free time slot and assign a frequency channel and send this information to the inviter in a TYPE 4 message. Other nodes that are not chosen will turn to sleep and start this process again. After the first message exchange between the two nodes, the link is fixed and becomes permanent. As the number of links grows, the number of pairs of nodes will also grow and the whole network may be fully connected.

SMACS solves the idle listening and collision fairly well as nodes only turn on their radio when needed. The use of different bands or frequency hopping can resist to vulnerability and jamming. However, since the time slot used in a pair of nodes is independent from the other pairs, the length between time slots may vary. New time slot may not fit into the existing timeframe if “free space” between allocated time slots is not long enough. This is similar to the problem of assigning memory to process in operating system. The scattering of time slots also cost inefficiency to the usage of channel. In addition, if a node has more neighbors, a node may need to switch the radio on and off more frequently, which increases the overhead.

3.4.4 The TDMA Schedule of HIT (continued from Sect. 3.3.4)

Different from the two protocols discussed above, the Hybrid Indirect Transmissions (HIT) uses TDMA as the MAC protocol. Since every node in the network only need to communicate with its neighbors, the range of the radio is short and parallel communication is possible. HIT creates the TDMA schedule such that the packets from downstream nodes will not collide at upstream nodes. It defines node i blocks node j if and only if $d(i, u_i) > d(i, u_j)$

Phase Five: Blocking Set Computation

In the previous phase, a node broadcast information of its upstream node. Neighbors of the node then know their downstream neighbors as well as the nodes that block their downstream neighbors. Each node will send a message with fixed signal strength about its blocking list.

Phase Six: MAC Schedule Setup

In this phase, the TDMA schedule will be set up using information collected in previous phases. The algorithm has three loops, one outer loop and two inner loops. The first inner loop finds the nodes that have no unscheduled downstream nodes and gives the list of these nodes to the second inner loop. The second inner loop then removes one node in the list, assigns a time slot to the node and removes all the nodes that block or blocked by the assigned node. The second loop will continue until there are no more nodes in the list. Note that all the time slots assigned are the same. After the second loop

is finished, the time slot will be advanced by one and this process (the two inner loops) will go on until all the nodes are scheduled. Under this protocol, each node computes its own schedule independently and in parallel.

Conclusions

The TDMA schedule of HIT is relatively simple and it makes use of the parallel transmission to reduce the delay. The main drawback of this protocol is that it does not support variable time slot. Since the algorithm iterates using time slot, the time slot must be fixed. When the size of the packet varies, it may cause inefficiency as the amount of time used to send a small packet is the same as that for large packet.

4 Cluster-Management and Power-Efficient Protocol (CMPE)

In this section, a new proposed protocol – *Cluster-Management and Power-Efficient Protocol* (CMPE) – will be presented and discussed. The proposed protocol is based on a clustered-network and that each node will try to find the path with minimum cost to the cluster head with minimum delay. The protocol is divided into four steps:

I. *Cluster head election*: sensor node will self-elect as a cluster head in this step;

II. *Route Set-Up*: in this step, a selective flooding algorithm is used to find the minimum cost path to the cluster head; path to cluster head and eventually to base station will be set up at the end of this step;

III. *TDMA scheduling*: the critical path in the route created in step two will be identified in this step; the scheduling algorithm will assign the time slot to each node and avoid extending the critical path at the same time; and

IV. *Data transfer*: this is the main part and sensor node will start collecting and transmitting data according to the time slot and path assigned.

After a specific number of rounds of data collections and transmissions are completed (step four), the whole process (steps one to four) will be repeated.

4.1 Assumptions

It is assumed that the followings will hold in the protocol:

1. The link is symmetric so that the cost of the link from node A to node B is the same as

the cost from node B to node A.

2. All the nodes in the network are stationary.
3. The sensor nodes can adjust the out-going transmission power and can measure the in-coming signal strength.
4. Every node in the network has the same initial energy.
5. There is no obstacle in the network, i.e., every node that falls in the range of the radio of the sender can hear the message.
6. The base station can talk to every node in the network.
7. There is no specific node-to-node communication.
8. Radio can be switched on and off.

The details of the protocol are discussed in the following sections.

4.2 Step One: Cluster Head Election

The first step is cluster head election. During this phase, sensor node will self-elect as a cluster head which collects sensor data and relays message to the base station. Different applications may require different ways to elect a cluster head; four of them will be discussed here:

1. Random

Each sensor node chooses a random number and compares with a constant or an equation, as specified in LEACH. This method requires hardware or software to generate

a random number, which may not be applicable to hardware or software that do not have the capability.

2. Node ID

Each node takes turn to become a cluster head based on node ID.

3. Residential energy

Sensor node with more residential energy will become a cluster head. This method is good in terms of load balancing. Since cluster head will use up more energy in transmitting data to base station (which is a long range transmission) and has more data to receive, sensor nodes with more energy can take this job. In order to extend the lifetime, sensor nodes with less energy just need to send and relay messages from its neighbors. This method may require a global knowledge of residential energy, and this is not feasible. Thus, this method may need to work with random number instead.

4. Connectivity or geography location

Sensor node with high connectivity to other nodes will have more chance to become a cluster head than sensor node with low connectivity, in such case the sensor node is likely to be in the boundary area of the whole network. The main drawback of this method is that some sensor nodes may die more rapidly than the others while the whole network will have a gain as link to high connectivity node is usually a low cost link. In the sense of the whole network, more energy can be saved by making use of low

cost links.

4.3 Step Two: Route Set-Up

In this step, the protocol will try to find a path to the cluster head with the minimum cost. The protocol uses an algorithm similar to flooding to propagate the cost information. Instead of flooding the same information through-out the network, the protocol only propagates new information when a node finds a lower cost path.

4.3.1 Phase One: Cluster Head Advertisement

After a sensor node is elected to be a cluster head, it sends out packets with fixed signal strength with cost equals to zero. (The cost from cluster head to itself is surely zero.) It is to announce which node has become the cluster head. Every node should turn on the radio in this phase.

The message will have the following fields:

```
Type: DISCOVERY
Cost: 0
```

Any node receives this packet will know

i) the ID of cluster head (from the source field of the packet), and

ii) the signal strength of the packet and thus the cost to the cluster head can be calculated.

Note that the radio is omni-directional and the sensor node *only* knows the cost to the cluster head but not the geography location of the cluster head. Also note that this is not a true broadcast message and not all the nodes in the network receive this message. The node then calculates the cost of the link and sends a message to tell the other nodes about this cost information.

The message will have the following fields:

```
Type: DISCOVERY
Cost: (cost to cluster head)
```

4.3.2 Phase Two: Path Setup

In this phase, the cost to upstream node and eventually to the cluster head will be discovered and the path to the cluster head will be set up. All the packets sent in this phase are of fixed signal strength so the nodes in the network can measure the cost to the senders. The packet format is the same as previous phase: the cost information of the node will be in the payload of the packet. Again, every node should switch on the radio in this phase.

Similar to phase one, any node receives the cost information packet will know

i) the cost from the sender to the cluster head, and

ii) cost to the sender (probably upstream node of the receiver).

It can then update and calculate the new cost to the cluster head by adding the two costs; the node then compares the current cost to the cluster head. (The cost is set to maximum value before each round begins.) If the new cost is lower than the current cost, the sender will become the new upstream node; and the node will send out the new information (with fixed signal strength). If the new cost is higher than the current cost, the node simply ignores it.

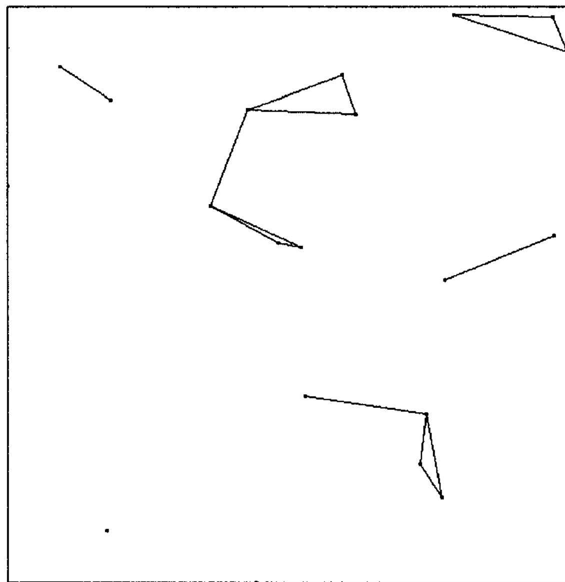
The message will have the following fields:

```
Type: DISCOVERY
Cost: (cost of upstream neighbor + cost to upstream
        neighbor)
```

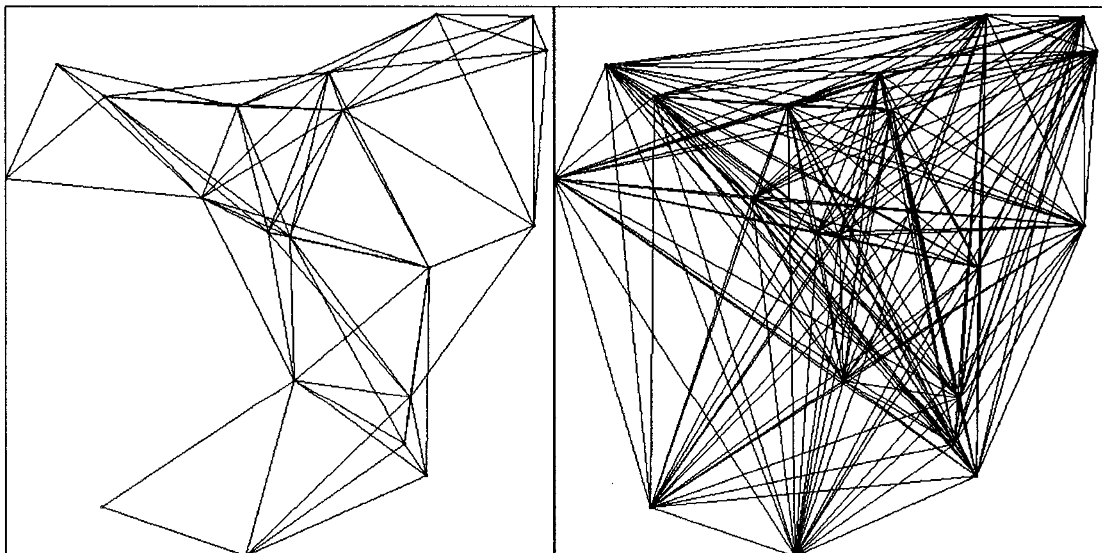
The range of the radio is important in this phase. Since the protocol is flood-based, if the range is too short, some nodes in the network may not be reached (see Figure 4.1(a)). If the range is too long, then more nodes will hear the cost information message, causing more updates and transmissions (see Figure 4.1(c)). In fact, these updates and transmissions are unnecessary, since there might be nodes lying in between the farthest nodes and the sender and these nodes in between may eventually act as upstream nodes. What is worse is that these unnecessary transmissions will cause more redundant transmissions which will propagate through-out the network.

Timeout can be added to solve this problem. Firstly, if a node has not heard from

any node, it can send a message with stronger signal and find an upstream node actively. Secondly, timeout can be set so that a node only listens to the message from the other nodes and updates its upstream node without sending any message before the timeout; and it only sends the message when the timer fires.



(a)



(b) (c)

Figure 4.1 (a) The range is too short and only a few nodes are connected. (b) The range is set to double the average distance between nodes. (c) The range is set to cover the whole network; the nodes are fully connected.

4.4 Step Three: TDMA scheduling

After the cluster and the path are set up, each of the sensor nodes in the cluster knows its direct downstream neighbors as well as the sensor nodes that block it from receiving. The definition of blocking presented by Culpepper et al. (2003) is used: node A blocks node B if and only if $d(A, u_A) > d(A, u_B)$ where u_N is the upstream neighbor of node N and d(n, m) is the distance between node n and m. It means that the upstream neighbor of node B falls in the range of node A. When node A sends data to its upstream neighbor, the upstream neighbors of B (u_B) also hears the data. Hence, node A and node B cannot send their data to their upstream neighbors simultaneously as the packet will corrupt at u_B.

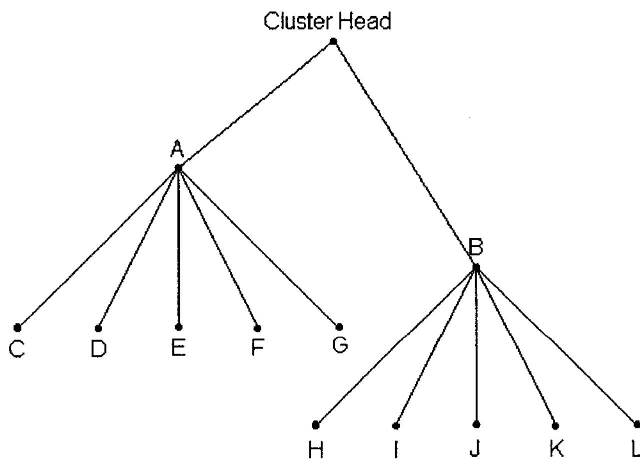


Figure 4.2 Network with Blocking Nodes.

Consider the network depicted in Figure 4.2, which is drawn to scale. Node A is the upstream neighbor of nodes C, D, E, F, and G while node B is the upstream neighbor of nodes H, I, J, K, and L. When node G sends data to node A, since node B falls into the range of the transmission, node B can hear the message. So, nodes H to L cannot send their data at this time since the signal is interfered. It is said that node G blocks nodes H to L. However, when node H sends its data to its upstream neighbors B, the upstream neighbor of G (which is node A) does not hear it as it is out of range. Hence, node H does not block node G.

To maximize power efficiency, the goals are to send as few data (whether in bits or in bytes) as possible and fuse or compress as much data as possible. To achieve this goal, a sensor node at upper level should wait and collect all the data from downstream nodes before sending any data to its upstream node. The node can fuse or compress the data when receiving or it can wait until all the data are collected. As a result, the data will flow from the leaf nodes to cluster head and base station and be fused or compressed on the way.

Besides fusing data, the delay should be minimized as well. Since the node just needs to talk to its neighbors, the range of the radio used is short; and the transmission only affects a small number of sensor nodes when compared with the whole network, the sensor nodes that are apart far enough can be arranged to send data simultaneously. This can save a significant amount of time.

It is obvious that the critical path is the path from the root to the leaf node with the greatest height and its sibling nodes. All the leaf nodes should be counted in the critical path since all the leaf nodes need to take turn to send the data (the upstream node should wait for all its downstream nodes since they use the same channel). The protocol can make use of the critical path as a guideline to set up a schedule and prevent extending that critical path.

In this model, fixed time slot is used for simplicity and the protocol schedules the node that should send last first. To build the sending schedule, each sensor node will send schedule information to the upstream neighbors; the base station (or cluster head) then creates the schedule based on the information collected and then it will broadcast the schedule to the sensor nodes (or send back to the nodes hop-by-hop).

There are four phases in this step:

1. Upstream node notice
2. Sending downstream node list and blocking list
3. Schedule setup
4. Schedule broadcast

4.4.1 Phase One: Upstream Node Notice

During this phase, each sensor node tells its upstream neighbor that it is the upstream node of the sender with signal strength just strong enough to be heard by the

upstream neighbor. Any node that hears this message targeted to it will put the sender node to the downstream list; if the message is targeted to another node, then it should put the node to the blocking list as the node blocks its downstream nodes from sending.

The message will have the following field:

```
Type: UpStream
Upstream: ID of the upstream neighbor
```

4.4.2 Phase Two: Sending Downstream Node List and Blocking List

After phase one, every node knows its own downstream neighbors as well as nodes that block its downstream neighbors from sending. Leaf nodes in the routing tree will then initiate and start sending three information to the upstream node:

i) the total number of downstream nodes plus the blocking nodes,

ii) the list of downstream nodes, and

iii) the list of blocking nodes.

Any non-leaf node in the routing tree will buffer the message, recalculate the total number of downstream, and blocking nodes; merge its downstream node list and blocking node list with the lists from its downstream nodes. Only the downstream node list of downstream node is needed as only the downstream nodes are involved in scheduling; the blocking nodes will not be propagated to the upstream node as they do not belong to that subtree.

To help cluster head or base station doing less computation and lessen the complexity, each node can sort out the downstream node list, with descending order of their total number of downstream and blocking nodes, before sending it to the upstream node.

Similar to phase one, each node can send the message with signal strength just strong enough to be heard by the upstream node.

The message will have the following fields:

```
Type: Downstream / Blocking List
Total number of downstream and blocking node
Downstream list: the IDs of downstream neighbors
Blocking list: the IDs of blocking nodes
Downstream list of downstream neighbors: the IDs contained
     in the downstream list of the downstream neighbors
```

4.4.3 Phase Three: Schedule Setup

When the base station or cluster head receives all the information, it starts scheduling. The protocol assigns the time slot "reversely". In other words, node assigned first will send message last and the node assigned last will send message first during the data transmission step. Each node will send its data right after it has collected all the data in the schedule.

Assume that all the downstream nodes (including the downstream node of the cluster head) of a node is sorted in descending order of the total number of downstream and blocking nodes. If the list is not sorted, sort it before scheduling. Let the node with largest number of downstream and blocking nodes send last as there is higher probability that the critical path will pass through that node. As the node is assigned to send last, it and its downstream nodes will have more time to collect data and there is no need to extend the critical path.

The protocol creates a queue that holds the nodes that are just scheduled. The protocol first puts the cluster head to the queue, assigns it with the time slot 0 and starts a loop. In each iteration of the loop, the protocol removes the first node in the queue, fetches the first downstream node and saves its time slot. The protocol then advances the time slot by one (the time slot that is right after that of the upstream node) and checks if any blocking node is assigned to that time slot. If this is the case, the protocol has to advance the time slot since signal from the assigned node and the blocking node will be interfered. After the protocol has found a free time slot, the protocol assigns that time slot to the node.

Then, the protocol checks if the node blocks any other nodes. If this is the case, the protocol shifts the time slot of the blocked node by one. The blocked node is chosen to shift because the protocol does not want to further defer the time slot of the node just assigned. Moreover, since the sibling nodes of the blocked node are also blocked nodes

(as they are from the same upstream node), the blocked nodes are likely to be assigned consecutive time slots. If the protocol chooses to shift the node just assigned, the protocol needs to shift several time slots to find the first available time slot and this is not desirable. As a result, the protocol decides to move the blocked node. Notice that it will increase the complexity (difficulty) to implement, but the performance is better.

Consider the network in Figure 4.2. Suppose the protocol has the following schedule (Figure 4.3) when node G is scheduled:

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7
B	A					
	H	I	J	K	L	
		G?				

Figure 4.3 Snapshot of the Schedule when Assigning Timeslot to Node G.

After checking blocking node, node G is assigned to slot 3 and the protocol finds that node G blocks node I. Since node G blocks nodes H to L, if the protocol chooses to shift G, the protocol has to move 4 time slots, which causes a long delay. On the other hand, if the protocol chooses to move nodes I to L, the protocol has only one delay.

Next, the protocol adds the assigned node to the queue, advances the time slot by one, fetches the next sibling node (next downstream node from the same upstream node), and starts the process again. This process will continue until there are no more nodes found in the queue.

Figure 4.4 lists the pseudo-code for the TDMA scheduling (assume the downstream list is sorted according to their respective number of downstream and blocking node):

```
// Pseudo code for the TDMA scheduling
1.    add the cluster head to QUEUE;
2.    while (QUEUE is not empty) {
3.        UPSTREAM_NODE = first node in QUEUE;
4.        TIME_SLOT = the assigned time slot of UPSTREAM_NODE + 1;
5.        for (each DOWNSTREAM_NODE in UPSTREAM_NODE) {
6.            while (TIME_SLOT is blocked by another node)
7.                TIME_SLOT++;
8.            assign TIME_SLOT to DOWNSTREAM_NODE;
9.            if (DOWNSTREAM_NODE blocks the other node BLOCK_NODE)
                  // i.e., BLOCK_NODE is blocked by DOWNSTREAM_NODE
10.               shift BLOCK_NODE;
11.           add DOWNSTREAM_NODE to QUEUE;
12.           TIME_SLOT++;
          }
      }
```

Figure 4.4 Pseudo-code for TDMA Scheduling.

4.4.4 Phase Four: Schedule Broadcast

After the TDMA schedule is completed. The cluster head or base station will send the TDMA schedule information to the node.

The message will have the following fields:

```
Type: Timeslot
Time slot: the time slot assigned.
```

4.5 Step Four: Data Transmission

After the path is set up and the TDMA schedule is created, all the sensor nodes start sensing and send the data to their upstream neighbors according to the TDMA schedule. This phase should be long enough to minimize the overhead in setting up the path and the schedule. Note that the data will be flowing from the leaf node hop-by-hop to the cluster head then to the base station. The data may be fused or compressed on the way to the base station. Since the TDMA schedule is used to send data, there is no collision in data transmission.

4.6 Examples

The following two examples (one for route setup and one for TDMA schedule) illustrate how the protocol works.

4.6.1 An Example for Routing

Firstly, consider the following network for route setup (Figure 4.5):

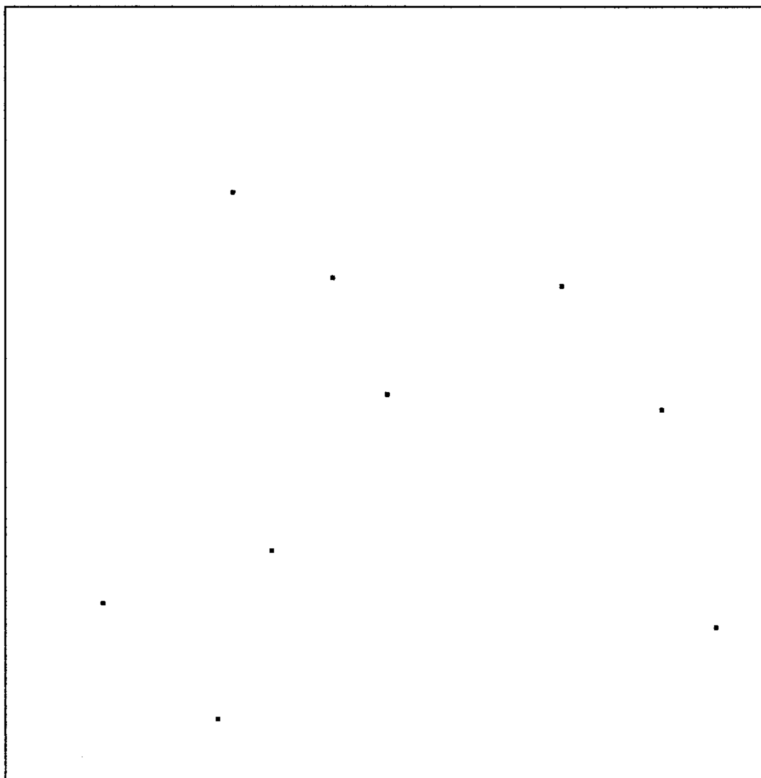


Figure 4.5 Network for Routing Example.

There are nine nodes in the network and the diameter of the network is 100 m x 100 m. The range of the radio selected for setup phase in this example is 1/3 of the network diameter, i.e., 33 m.

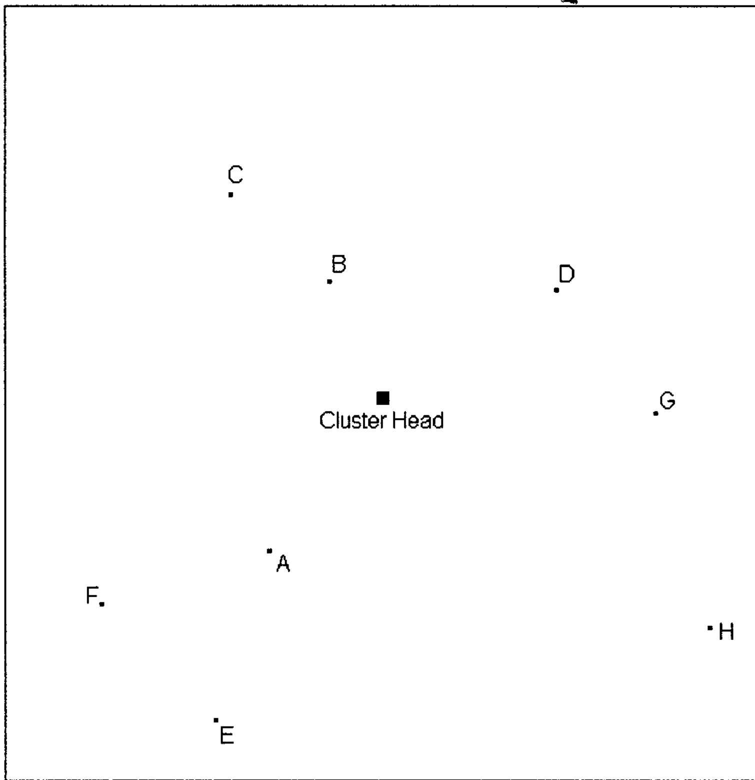


Figure 4.6 Cluster Head and its Members.

In the example, the node at the center of the network is selected as the cluster head (Figure 4.6). Note that every node can be a cluster head. The central one is selected here just for illustration purpose only.

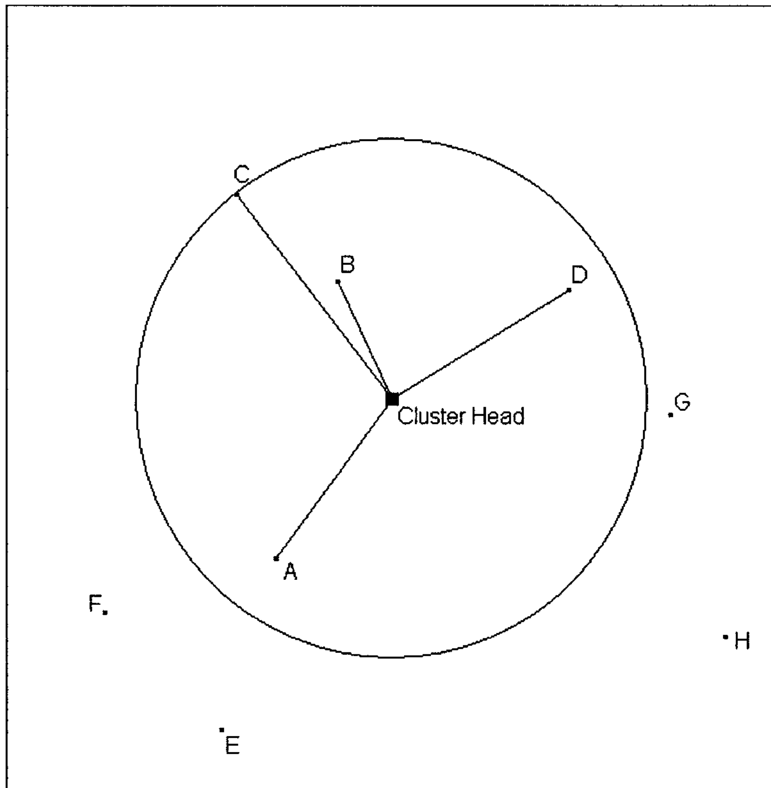


Figure 4.7 Advertisement of Cluster Head and its Range.

When the network starts, the cluster head in the example will send out an advertisement with fixed transmission power (remember that the range of the radio is 33m), which is represented by the circle depicted in Figure 4.7. All the nodes that fall within the circle, i.e., nodes A, B, C, and D, can hear the advertisement. They will update their upstream neighbors to be the cluster head as indicated by the link. They will send out the updated information after receiving the advertisement.

Firstly, consider what will happen at node A (Figure 4.8):

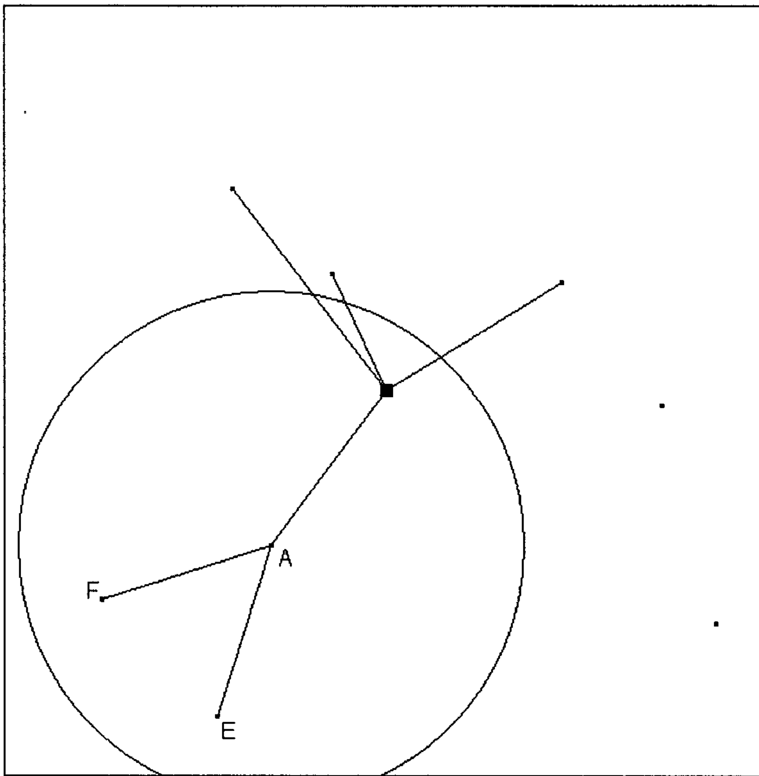


Figure 4.8 Discovery of Node A.

After node A sends out its packet, nodes E and F will hear it. As both of them are not attached to the network, they will update their upstream neighbors respectively. (In fact, both the nodes will send out a packet of this update, which is not shown in the figure).

Then, consider nodes B and C (Figure 4.9):

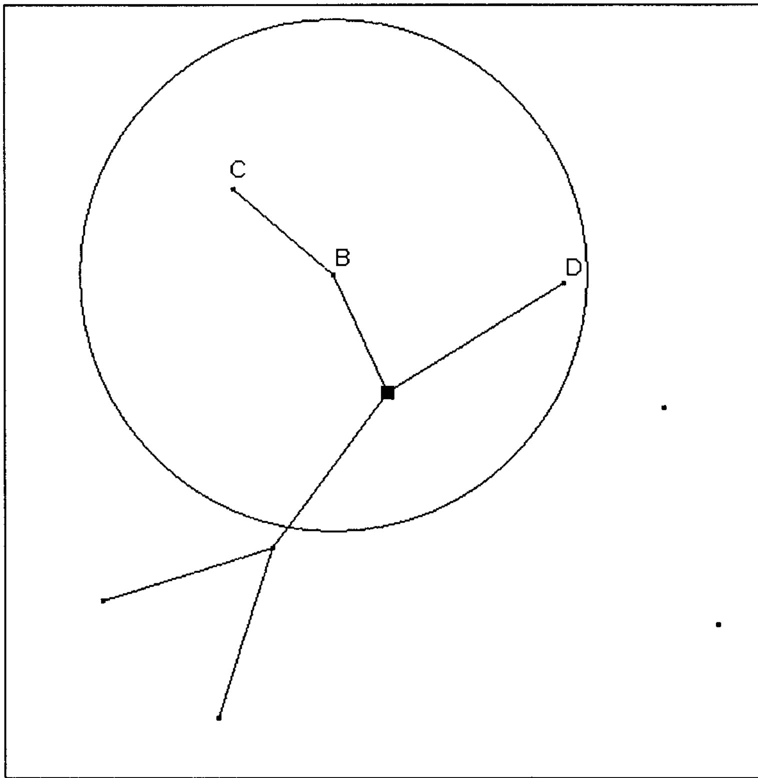


Figure 4.9 Discovery of Node B.

When node B sends out its packet, both nodes C and D can hear it. Node C finds that it can lower the cost by sending data through node B (i.e., the cost sending from node C to B plus the cost sending from node B to cluster head is lower than the cost sending from Node C to cluster head directly). Node C will then change its upstream neighbor to node B and send out a message of this update. However, node D does not have the same finding. So it will not update its state.

After node D receives the advertisement from cluster head (and before the updated message from node B), it sends out its packet (Figure 4.10). Node G hears the packet and it will update its upstream neighbor and send out an update packet which is heard by node H (Figure 4.11). Similarly, node H will update its upstream neighbor and send out an update packet (not shown).

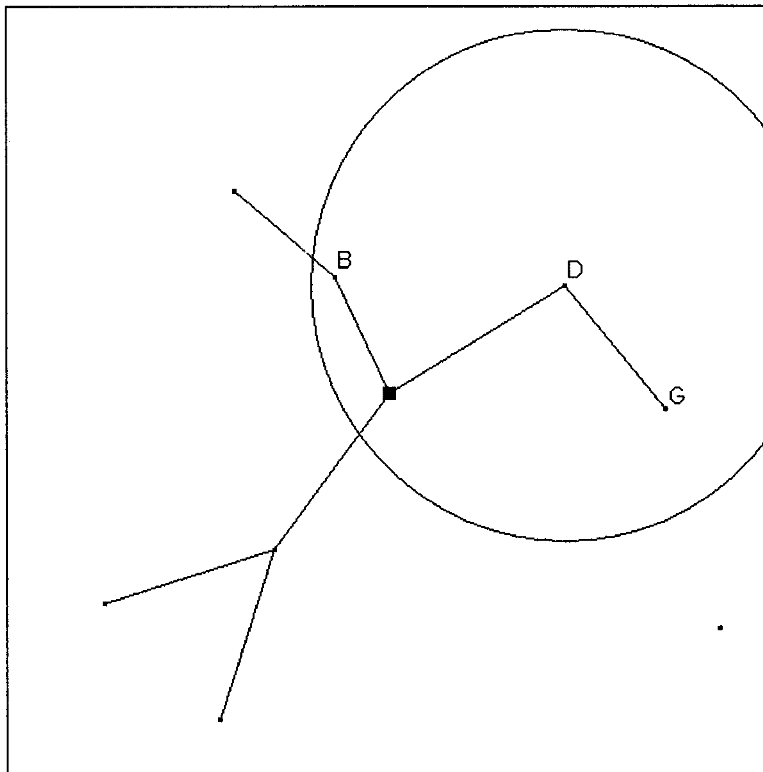


Figure 4.10 Discovery of Node D.

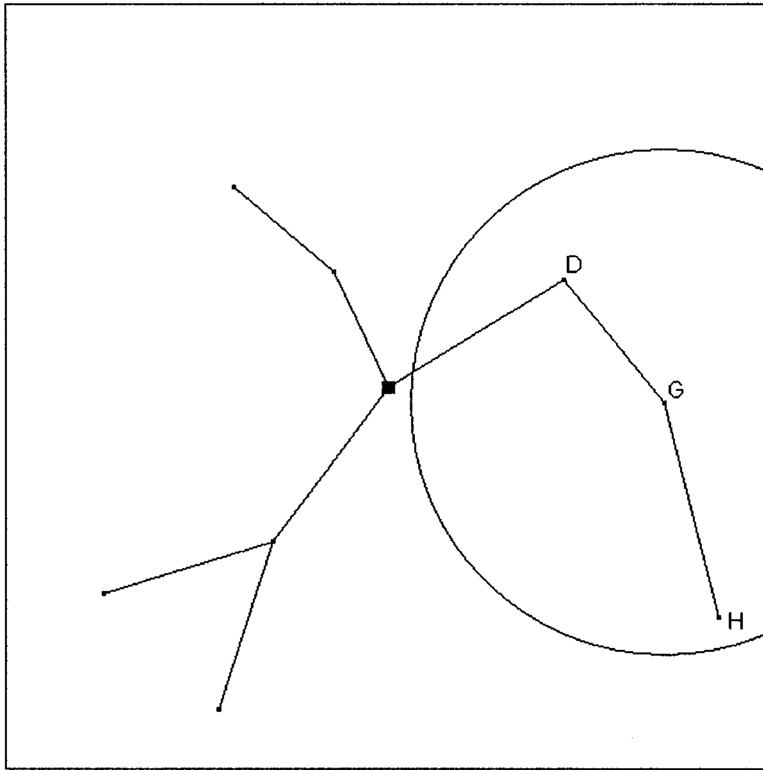


Figure 4.11 Discovery of Node G.

From now on, all the nodes are connected and routing is finished.

4.6.2 An Example for TDMA Schedule

Now consider the following network for TDMA schedule (Figure 4.12):

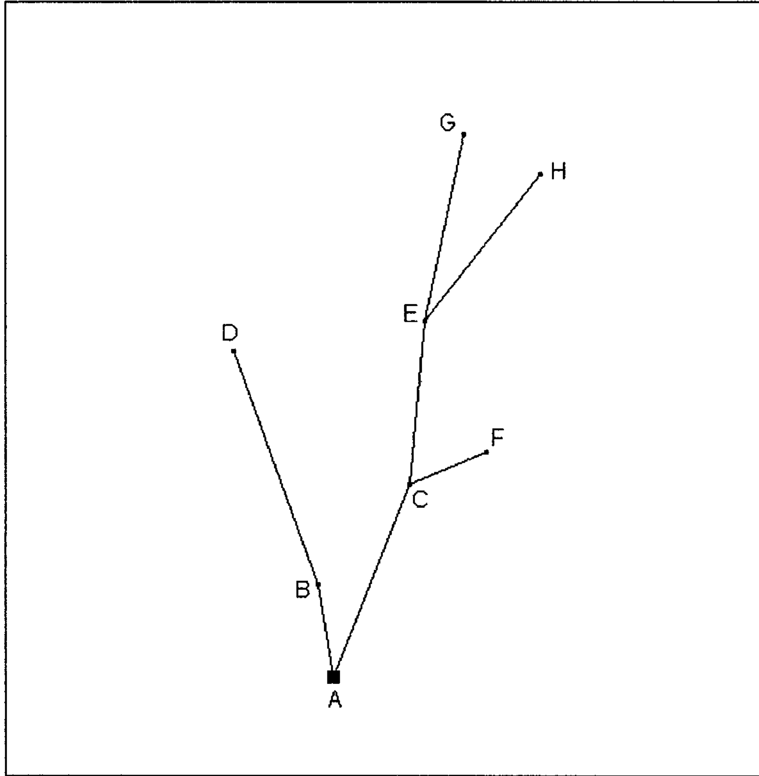


Figure 4.12 Network for TDMA Scheduling Example.

Again this is a 100 m x 100 m network but with eight nodes only.

After exchanging packets, the cluster head has the following information (table 4.1):

Node	Total downstream and blocking nodes:	Downstream nodes:	Blocking nodes:
A	9	B, C	-
B	1	D	-
C	6	E, F	D
D	0	-	-
E	3	G	D
F	0	H	-
G	0	-	-
H	0	-	-

Table 4.1 Information Gathered by Cluster Head for Scheduling.

The protocol first performs the TDMA scheduling at node A which is the cluster head. (Line 1 in pseudo-code). Node A has two downstream neighbors: B and C. Since node C has more downstream and blocking nodes, the protocol assigns a time slot to node C first. A snapshot of the schedule is depicted in Figure 4.13.

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
C	B			

Figure 4.13 Snapshot of the Schedule after Assigning Timeslot to Nodes C and B.

Since nodes C and B do not block nor blocked by other nodes, the protocol simply assigns the time slot and put them into the queue. (Line 11 in pseudo-code)

The protocol removes the first node in the queue, which is node C. Node C has downstream nodes E and F. Since node E has more downstream nodes, node E is assigned first (right after the slot of node C). The snapshot of the schedule is depicted in Figure 4.14. Node with an asterisk on right is in the queue.

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
C	B*			
	E*	F*		

Figure 4.14 Snapshot of the Schedule after Assigning Timeslot to Nodes E and F.

Then the protocol assigns the downstream node of node B – node D. Since none of the nodes blocks node D, the protocol assigns node D to slot 3. However, node D blocks the downstream nodes of C and E (i.e., nodes E, F, G, and H), the protocol has to shift node F to the next slot. (Lines 9 and 10 in pseudo-code). The snapshot of the schedule is depicted in Figure 4.15.

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
C	B			
	E*	(dummy)	F*	
		D*		

Figure 4.15 Snapshot of the Schedule after Assigning Timeslot to Node D.

Next, the protocol schedules nodes G and H (the downstream nodes of node E).

Again, since node D blocks nodes E, F, G, and H; node G cannot be assigned to slot 3, the protocol has to assign slot 4 to node G instead (Figure 4.16).

Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
C	B			
	E	(dummy)	F*	
		D*		
		(dummy)	G*	H*

Figure 4.16 Snapshot of the Schedule after Assigning Timeslot to Nodes G and H.

Up to now, as all the nodes in the queue are leaf nodes and have no downstream nodes, the protocol has finished the TDMA scheduling.

4.7 The Routing Tree and TDMA Tree

To facilitate the TDMA scheduling, a *TDMA tree* structure is introduced. A node in the tree has the following structure (see Figure 4.17, note the position of the nodes in the figure):

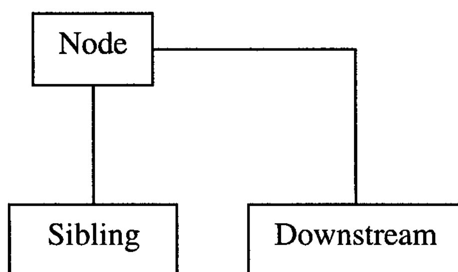


Figure 4.17 A TDMA Tree Node (Reading from Top-down Fashion).

The TDMA tree has the following properties:

1. Nodes in vertical are downstream nodes of the same node, i.e., they are siblings in the route tree.
2. Nodes in horizontal have the same time slot.

For example, the protocol has a correspondent routing tree as depicted in Figure 4.18 for the network depicted in Figure 4.12.

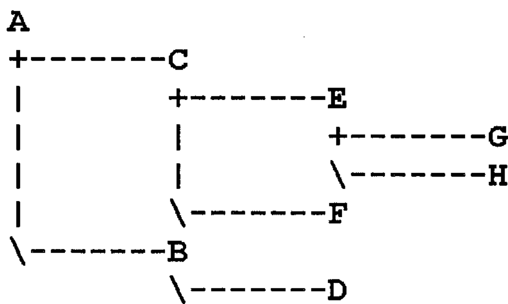


Figure 4.18 A Routing Tree for the Network Depicted in Figure 4.12.

The figure is read from left to right, nodes on the right are downstream nodes of node at left, e.g., nodes B and C are downstream nodes of A and nodes E and F are downstream nodes of C. When scheduling, the protocol will follow the order from left to right and then from top to bottom, i.e., the scheduling order for the routing tree in Figure 4.18 will be nodes C, B, E, F, D, G, and H. (Note that this routing tree is a sorted tree, node with more downstream and blocking nodes appears first.)

Given the routing tree in Figure 4.18, the following TDMA tree can be deduced (Figure 4.19):

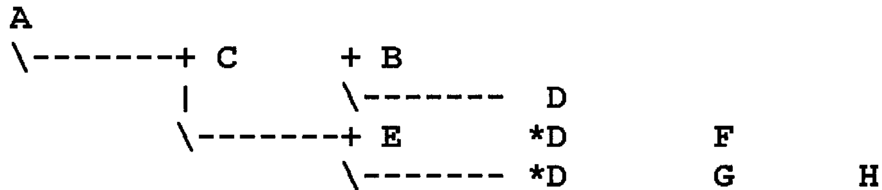


Figure 4.19 A TDMA Tree for the Network Depicted in Figure 4.12.

Note that this figure is read from left to right: nodes in horizontal are siblings while nodes having the same time slot are in vertical. Therefore, node A has children (downstream nodes) nodes B and C; and node C has nodes E and F as children. Node with asterisk is dummy. As seen from the figure, nodes B and E; and nodes F and G are in vertical, which means they have the same time slot and they can send their data at the same time (they do not block each other).

As mentioned earlier, when the protocol assigns node D, the protocol has to shift node F. By using this TDMA tree, the protocol can easily find the affected nodes by finding the nodes in the subtree and shift the nodes affected.

5 Simulation and Results

This section will discuss about the simulator and the simulation results.

To evaluate the proposed protocol, CMPE, and compare the protocol with the other current protocols, a simulator was created and some simulation tests were worked out. The simulation results are presented and discussed in this section.

5.1 The Simulator

In order to examine the different protocols and understand how each protocol works, a simulator is created. The simulator is written in Java language and run on a PC in Windows environment. The source code of the simulator is in Appendix D. The simulator has the following structure (Figure 5.1):

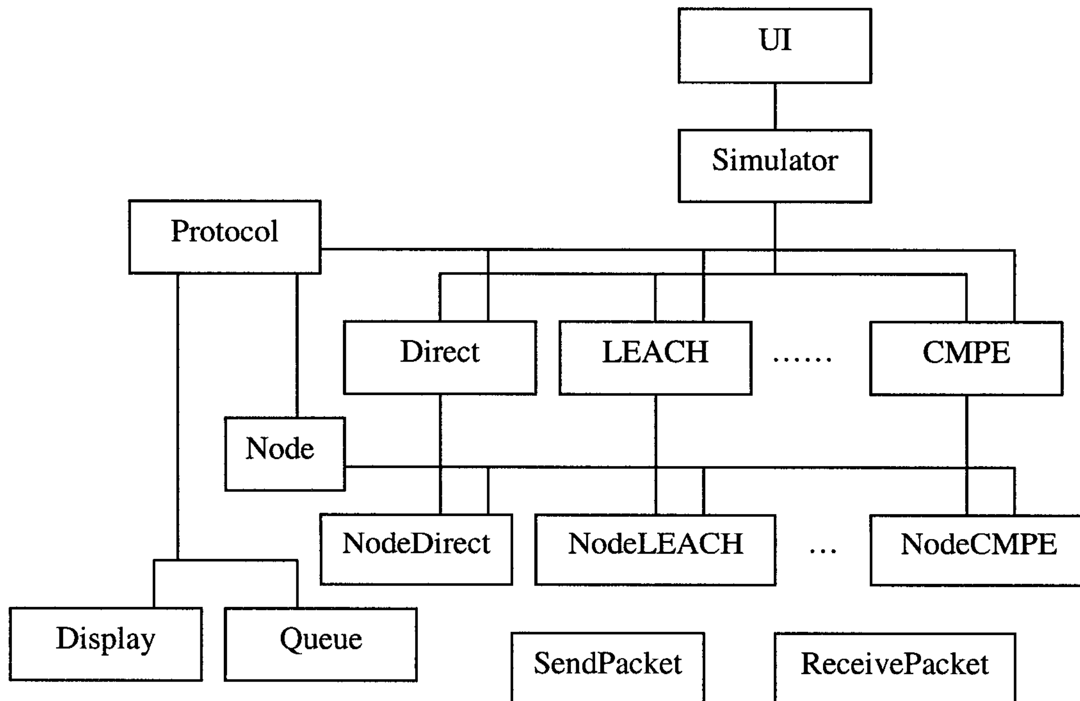


Figure 5.1 Structure of the Simulator.

The simulator program has a UI which is the graphical user interface of the simulator. After the user starts the simulation, a Simulator object will be created. The Simulator contains and controls the instances of the protocols under testing (i.e., Direct, LEACH, PEGASIS, HIT, and CMPE). Each protocol has its own corresponding nodes. All of these protocols and nodes are extended from two abstract classes called Protocol and Node respectively. The Protocol and Node contain common settings, fields, and methods to their subclasses. Protocol also contains a Display and a Queue. The Display shows the locations of the nodes and the routing results in graphical means and the Queue is a message queue which functions as described below.

When each round of the simulation starts, the Protocol instance (e.g., instance of CMPE) will elect the cluster head and initiate all the required transmissions. All transmissions are put into a message queue. The Protocol will then process the message queue by removing messages from the queue one by one and sending the messages to the nodes that are covered in the range. This process will continue until the queue is empty. How to process the message depends on the protocol. The node may just update its state or it may send out another message (like CMPE). The Protocol may need to process the message queue for several rounds for different types of messages.

5.2 Simulation Results

5.2.1 Parameter Tuning

The protocol is first analyzed with different parameters. The reference model is a network with diameter of 100 m x 100 m and has 100 nodes in the network; five percents of the nodes will be cluster heads and every node will send exactly one packet of 100 bits. Every node has initial energy of 1 J. The cluster heads and the paths to cluster head are recomputed every 1000 rounds of data gathering. The range of the simulation is set to be

$$\sqrt{\frac{Wdith \times Height}{No.of\ nodes}} \times 2 \qquad \text{.................................... (14)}$$

The energy used in path setup and TDMA scheduling (only transmitting and receiving) is also considered.

This section will present the results of the reference model here; followed by network models of different diameters; then the protocol will be tested with different percentages of cluster heads; and finally, networks with different packet sizes will be simulated. The ten test cases have been run five times each and the median results are shown in table 5.1.

		Round Number		
Test ID	Network	First node dies at	Half of the nodes die at	Last node dies at
1.	100 nodes, 100mx100m, 100bits, 5%CH	2966	12582	19149

2.	100 nodes, 10mx10m, 100bits, 5%CH	11426	16795	31840
3.	100 nodes, 10mx10m, 100bits, 1%CH	12775	22000	33000
4.	10 nodes, 10mx10m, 100bits, 10%CH	10000	10000	10000
5.	100 nodes, 1000mx1000m, 100bits, 5%CH	0	4000	8002
6.	100 nodes, 100mx100m, 100bits, 1%CH	2936	24582	30236
7.	100 nodes, 100mx100m, 100bits, 10%CH	2262	7761	12654
8.	100 nodes, 100mx100m, 50bits, 5%CH	7866	23000	32149
9.	100 nodes, 100mx100m, 200bits, 5%CH	679	8878	15121
10.	100 nodes, 100mx100m, 1000bits, 5%CH	132	3000	5024

Table 5.1 Results of Different Networks Using the CMPE.

In the reference (test case 1), the first node dies near 3000th round and the last node dies near 19000th round.

In test case 2, the diameter of the network is reduced to 10 m while the other parameters remain unchanged. The results show improvement: the first node dies near 11000th round and the last node dies near 30000th round. Furthermore, if the percentage of cluster heads is reduced to 1% (test case 3), the results can be slightly improved further.

Referring to test case 4, which is a small scale network (10 nodes in 10 m x 10 m), it is found that all the nodes tend to die at the same round. The other four sets of data of this test case also agree with these results. For a large scale network (100 nodes in 1000 m x 1000 m, see test case 5), the performance decreases dramatically. The first node dies when the network starts and the last node dies near round 8000th. This is expected as the

energy dissipation grows quadratically with distance (equation 4).

Test cases 6 and 7 are for comparing varying percentage of cluster heads. According to test case 6, if the percentage of cluster heads is reduced to 1% in the network, the performance improves (half of the nodes still are alive at round 24000th and the last node dies near 30000th). In test case 7, the percentage of cluster heads is increased to 10%, the performance decreases. These results also show the same trend as those in test cases 2 and 3.

Referring to test cases 8 to 10, packet sizes of 50-bit, 200-bit, and 1000-bit are tested. As can be expected from equation 4, the performance decreases when the length of the packet size increases. This is verified by the test results of test cases 8 to 10.

5.2.2 *Comparisons*

In this subsection the CMPE protocol will be compared with other current protocols, namely, Direct, LEACH, PEGASIS, and HIT. The reference used here is a 100-node-network with diameter of 500 m x 500 m. The packet size is 100-bit with no data fusion or compression, i.e., the upstream node must relay all the messages from downstream. Five percents of the nodes will be cluster heads.

The energy used in one data gathering will be compared. The energy dissipated in route and MAC setup will also be measured. Lastly, the delay (number of time slot in

order to send data) and the network utilization of each protocol will be measured. Each test has been run five times and the average is shown in the tables 5.2 to 5.4 and plotted in figures 5.2 to 5.5.

5.2.2.1 Energy dissipation in one round of data gathering

Protocol	Direct	LEACH	PEGASIS	HIT	CMPE
Energy (J)	0.618545	0.632123	0.830132	0.635644	0.596171

Table 5.2 Energy Dissipation in One Round of Data Gathering.

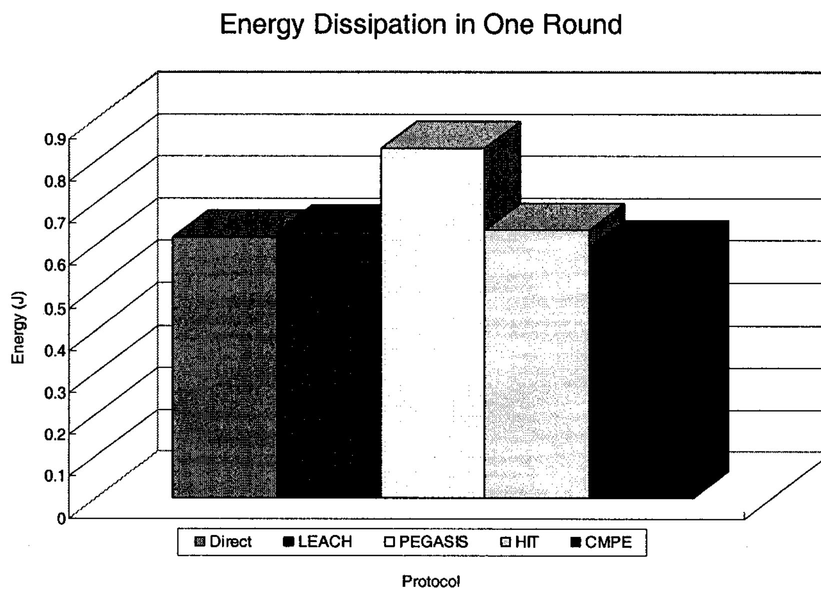


Figure 5.2 Energy Dissipation in One Round of Data Gathering.

From the simulation results, it is found that PEGASIS is the protocol that consumes the most amount of energy while the proposed protocol, CMPE, is the protocol consumes the least amount of energy.

5.2.2.2 Energy dissipation in route and MAC setup

Protocol	Direct	LEACH	PEGASIS	HIT	CMPE
Energy (J)	0.0	0.005282	0.0	0.8678	0.5792

Table 5.3 Energy Dissipation in Route and MAC Setup.

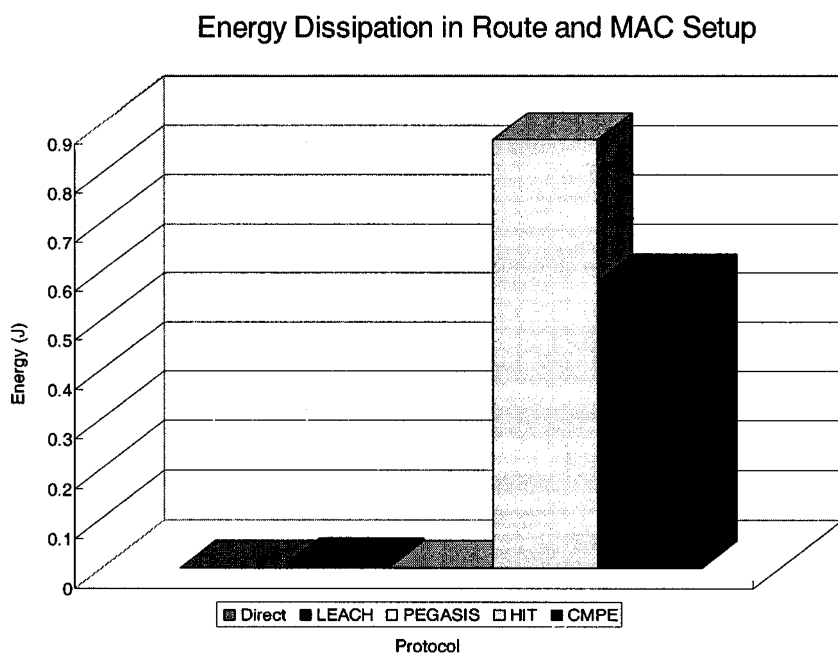


Figure 5.3 Energy Dissipation in Route and MAC Setup.

Obviously, the Direct protocol does not need route and MAC setup, so it does not use any energy in this part. For LEACH, only the cluster head takes part in the route setup, the overhead is limited. Since PEGASIS assumes the global knowledge of the network, it does not consume any energy in route setup. When comparing HIT and CMPE, HIT has more broadcast messages while CMPE has more local messages, so HIT consumes more energy.

5.2.2.3 Delay

Protocol	Direct	LEACH	PEGASIS	HIT	CMPE
Delay	100	41.2	100	16.2	16.2

Table 5.4 Average Delay.

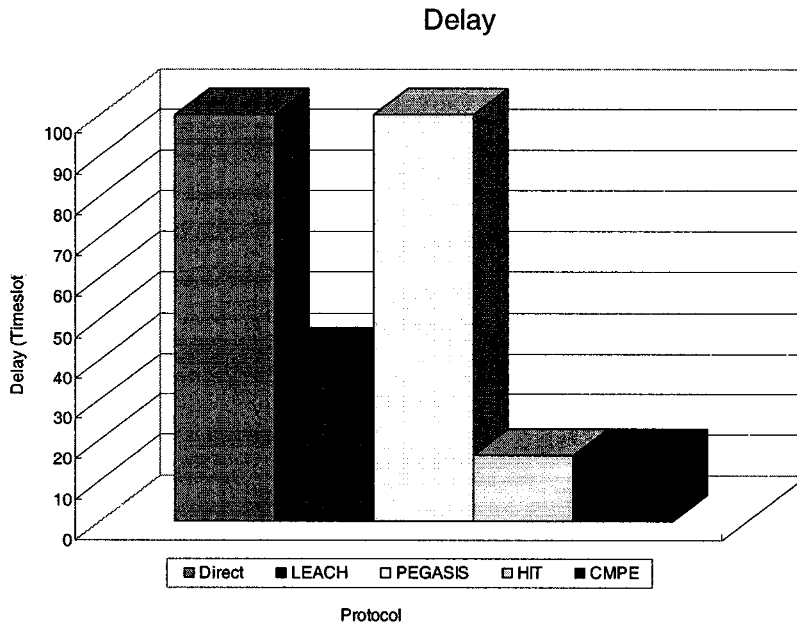


Figure 5.4 Average Delay.

For the delay, both Direct and PEGASIS have a delay of 100 (see Figure 5.4) since they do not support parallel transmissions. In LEACH, parallel transmission only occurs in different clusters and the amount of time saving is limited. In HIT and CMPE, parallel transmission takes place through-out the network and their delay is the smallest.

5.2.2.4 Network Utilization

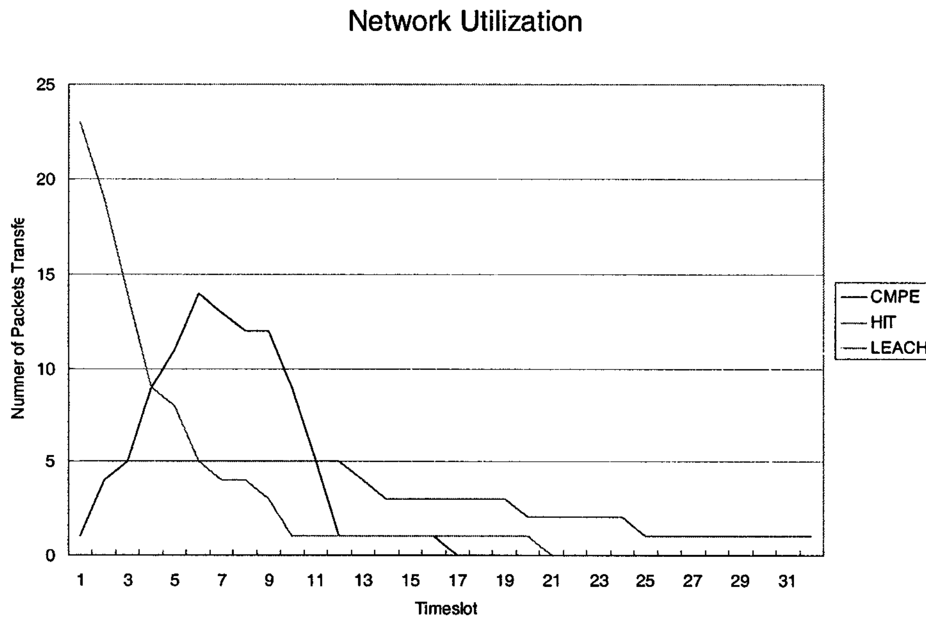


Figure 5.5 Network Utilization.

Figure 5.5 shows the network utilization: the number of transmissions at certain time slot. It can show the level of parallel transmissions in different protocols. As Direct and PEGASIS do not support parallel transmissions, only LEACH, HIT, and CMPE are considered.

As seen from the figure, LEACH is fixed at five transmissions for the first 12 time slots. It is because the level of parallel transmission in LEACH is equal to the number of clusters. For HIT, since the leaf nodes start sending data as early as possible, the level of parallel transmission is the highest at the beginning and then drops rapidly. In contrast, as the proposed protocol tends to delay sending, the peak is at the middle and the graph is dome-shaped.

This section will discuss about two extensions, namely, *progressive discovery* and *three-way-handshake for cost discovery*. These extensions are not specific to the protocol but also can apply to any other protocol as well.

6.1 Progressive Discovery

Since not all sensor nodes are able to find the signal strength, protocols depend on signal strength may be useless on this kind of sensor nodes. However, if the node can send a discovery packet (e.g., a HELLO packet) with increasing transmission power, the neighboring nodes may finally hear it and give a reply. The node then can know the cost to its neighbors.

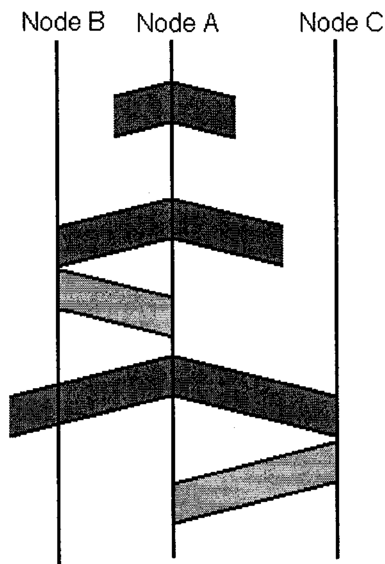


Figure 6.1 Timeline for Progressive Discovery.

For example, consider the time line above (Figure 4.20), node A sends out three

different packets with different signal strengths. The first packet gets no reply. For the second packet, node A gets a reply from node B, so it will know its distance to node B. Similarly, node A will know the distance after getting the reply from node C.

6.2 Three-Way-Handshake for Cost Discovery

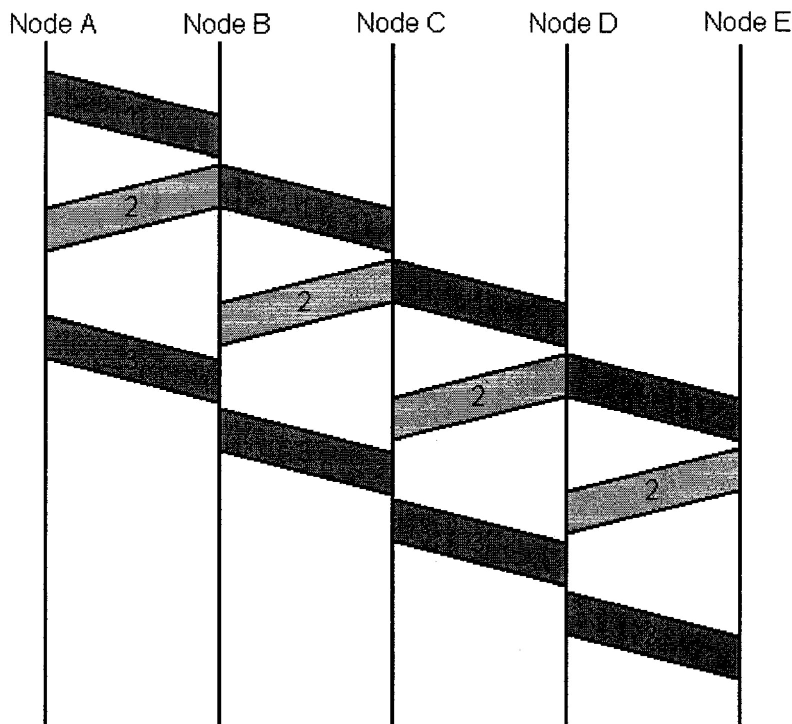


Figure 6.2 Timeline for Cost Discovery.

The previous section discussed how to find out the signal strength by increasing the radio power progressively. However, if the link is not symmetric, i.e., the costs of the two directions of a link are different, even the hardware is able to find the signal strength, it is not possible to find out the "out-going" signal strength by just looking at the "in-coming" packet. To find out the "out-going" cost, the node needs feedback from the

it is not possible to find out the "out-going" signal strength by just looking at the "in-coming" packet. To find out the "out-going" cost, the node needs feedback from the other nodes.

To illustrate how the three-way-handshake works, consider an example as depicted in Figure 4.21. Node A in Figure 4.21 first sends out a message with fixed signal strength (message 1). Node B hears the message and finds out the cost from node A to node B by measuring the signal strength. Node B then sends this cost information to node A (message 2). When node A receives the message, Node A is told the cost from node A to node B. Also, node A can find out the cost from node B to node A by measuring the signal strength. Node A then sends this cost information to node B (message 3). When node B receives the message, node B is told the cost from node B to node A. This process involves three message exchanges and they should be sent at fixed signal strength so that the results are comparable.

Since the radio used is omni-directional, when node B sends the cost information to node A, sensor nodes adjacent to node B can also hear the message. This can act as the first message and initiate another three-way-handshake, as depicted in Figure 4.21. The first message is called a discovery message. This three-way-handshake will propagate through-out the network.

7 Conclusions

In this paper, some design issues have been reviewed: the free-space propagation of a radio wave, hidden station problem, exposed station problem, comparison of direct and multi-hops communication, and the relationship between routing and medium access control. Also, current hardware such as Mica2 and Mica2Dot, software tools such as TinyOS, and some routing and MAC protocols such as LEACH, PEGASIS, HIT, S-MAC, SMACS, etc, have been described.

In the view of various weaknesses of current protocols, a new protocol is proposed and discussed, Cluster-Management and Power-Efficient Protocol (CMPE), which consists of a routing and a TDMA scheduling algorithm. The routing part is cluster-based and adopts a selective flooding algorithm to find the minimum cost path to cluster head, and the TDMA scheduling part is to find and avoid extending the critical path.

To simulate and compare the results of the proposed protocol with other current protocols, a simulator is created to evaluate the performance in various aspects, such as energy efficiency, delay, and network utilization. The simulation results show that CMPE saves nearly 30% of energy in one round of data gathering when compared to PEGASIS, 7% of energy when compared to LEACH and HIT and save more than 33% of energy in MAC and TDMA setup when compared to HIT. CMPE also reduces the delay by more than 80% when compared with Direct and PEGASIS and 60% when compared with

LEACH. Also, the CMPE protocol shows a more even network utilization when compared with HIT.

Finally, two extensions are introduced to overcome the limitation of sensor: how to find out the cost of the path when the sensor cannot measure in-coming signal strength and how to find out the cost of the path for non-symmetrical link.

References

Chipcon. *Chipcon CC1000 Datasheet.* Retrieved February 29, 2004, from http://www. chipcon.com/files/ CC1000_Data_Sheet _2_1.pdf

Crossbow. *Mica2 Datasheet.* Retrieved February 29, 2004, from http://www.xbow.com/ Products/Product_pdf_files/Wireless_pdf/ 6020-0042-04_B_MICA2.pdf

Crossbow. *Mica2 and Mica2Dot Manual.* Retrieved February 29, 2004, from http://www. xbow.com/Support/ Support_pdf_files/MPR-MIB_Series_User_Manual_7430-0021-05_A.pdf

Crossbow. *Mica2Dot Datasheet.* Retrieved February 29, 2004, from http://www.xbow. com/Products/Product_pdf_files/Wireless_pdf/6020-0043-03_A_MICA2DOT.pdf

Culpepper, J., Dung, L., & Mon, M. (2003). Hybrid Indirect Transmissions (HIT) for Data Gathering in Wireless Micro Sensor Networks with Biomedical Applications [Electronic version]. *Proc. of IEEE Computer Communications Workshop*, October 2003.

Department of Electrical Engineering and Computer Science, University of California, Bekeley. TinyOS. www.tinyos.net

Heninzelman, W. R., Chandrakasan, A. & Balakrishnan, H. (2000). Communication Protocol for Wireless Microsensor Networks [Electronic version]. *Proceedings of the Hawaii International Conference on System Sciences*, 2000.

Kurose, J.F. & Ross, K.W. (2003). *Computer Networking: A Top-Down Approach Featuring the Internet* (2nd ed.). Addison-Wesley.

Lindsey, S. & Raghavendra C. S. (2002). PEGASIS: Power-Efficient Gathering in Sensor Information Systems [Electronic version]. *Proceedings of IEEE Aerospace Conference*, 2002.

Lindsey, S., Raghavendra C. S., & Sivalingam K (2002). Data Gathering in Sensor Networks using the Energy*Delay Metric [Electronic version]. *IEEE Transactions on Parallel and Distributed Systems*, September 2002, pp. 924-935.

Losee, R. (1997). *RF System, Components, and Circuits Handbook.* Norwood, MA: Artech House.

Manjeshwar, A., Zeng, q., & Agrawal, D.P. (2002). An Analytical Model for Information

Retrieval in Wireless Sensor Networks Using Enhanced APTEEN Protocol [Electronic version]. *IEEE Transactions on Parsallel and Distributed Systems*, Volume: 13, Issue: 12, pp. 1290- 1302, December 2002.

Peterson, L.L. & Davie, B.S. (2000). *Computer Networks: A System Approach* (2nd ed.). San Francisco, CA: Morgan Kaufmann. Pearson Education.

Raghunathan, V., Schurgers, C., Park, S., & Srivastava, M. B. (2002). Energy Aware Wireless Sensor Networks [Electronic version]. *IEEE Signal Processing Magazine*, Vol.19, No.2, pp. 40-50, March 2002.

Schurgers, C., & Srivastava, M. B. (2001). Energy efficient routing in wireless sensor networks [Electronic version]. *Communications for Network-Centric Operations: Creating the Information Force*, IEEE, Volume: 1, pp. 357-361.

Shah, R. C., & Rabaey, J. M., (2002). Energy Aware Routing for Low Energy Ad Hoc Sensor Networks [Electronic version]. *IEEE Wireless Communications and Networking Conference* (WCNC), March 17-21, 2002

Smithm A.A., Jr. (1998). *Radio Frequency Principles and Applicatoins*. New York, NY: IEEE.

Sohrabi, K., Gao, J., Ailawadhi, V., & Pottie G. J. (2000). Protocols for Self-Organization of a Wireless Sensor Network [Electronic version]. *IEEE Personal Communications*, vol. 7, no. 5, pp. 16-27, October 2000.

Tanenbaum, A.S. (2003). *Computer Networks* (4th ed.). Upper Saddle River, NJ: Pearson Education.

Walke, B.H. (1999). *Mobile Radio Networks*. New York, NY: Wiley.

Ye, W., Heidemann, J., & Estrin D. (2002). An Energy-Efficient MAC Protocol for Wireless Sensor Networks [Electronic version]. *Proceedings of the IEEE Infocom*, pp. 1567-1576, June 2002.

Ye, W., Heidemann, J., & Estrin D. (2003). Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks [Electronic version]. *Technical Report* ISI-TR-567, USC/Information Sciences Institute, January, 2003.

Appendix A

Source code of program used to test TOSSIM

```
//tossim_test.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/stat.h>
#include <sys/errno.h>

#define MAXLEN 1046

extern int errno;

enum __nesc_unnamed4277 {
  AM_DEBUGMSGEVENT,
  AM_RADIOMSGSENTEVENT,
  AM_UARTMSGSENTEVENT,
  AM_ADCDATAREADYEVENT,
  AM_TOSSIMINITEVENT,
  AM_TURNONMOTECOMMAND,
  AM_TURNOFFMOTECOMMAND,
  AM_RADIOMSGSENDCOMMAND,
  AM_UARTMSGSENDCOMMAND,
  AM_SETLINKPROBCOMMAND,
  AM_SETADCPORTVALUECOMMAND
};

int main(int argc, char* argv[]) {
   struct sockaddr_in srv_addr;
   struct hostent *host;
   int cli_sock;
   int addr_size, nbyte;
   unsigned char buf[MAXLEN + 1];
   unsigned char header[15];
   unsigned char data[256*256+1];
   int i;
   int cnt;
   unsigned short type, id, size;
   unsigned long long time;
   int verbose = 0;


   if (argc > 1)
      if (strcmp(argv[1], "-v") == 0)
         verbose = 1;

   /* create socket */
```

```
   if((cli_sock = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
      fprintf(stderr, "Fail to create socket: %s.\n", strerror(errno));
      exit(1);
   }

   /* prepare server address */
   host = gethostbyname("localhost");
   memcpy((char*)&srv_addr.sin_addr, (char*)host->h_addr, host-
>h_length);
   srv_addr.sin_port = htons((u_short) 10585);
   srv_addr.sin_family = AF_INET;

   addr_size = sizeof(srv_addr);

   /* connect to the server */
   if (connect(cli_sock, (struct sockaddr *) &srv_addr,
sizeof(srv_addr))<0) {
      fprintf(stderr, "Fail to connect %s :\n", strerror(errno));
      exit(1);
   }

   printf("connected to TOSSIM\n");

   do {
      cnt = 0;
      do {
         if ((nbyte = recv(cli_sock, &(header[cnt]), 14-cnt, 0)) < 0)
         {
            printf("Fail to receive : %s\n", strerror(errno));
            exit(1);
         }
         cnt += nbyte;
      }
      while ((cnt<14) && (nbyte>0));
      type = header[0] * 0x100 + header[1];
      id   = header[2] * 0x100 + header[3];
      time = header[4] * 0x100000000000000 +
             header[5] * 0x1000000000000 +
             header[6] * 0x10000000000 +
             header[7] * 0x100000000 +
             header[8] * 0x1000000 +
             header[9] * 0x10000 +
             header[10] * 0x100 +
             header[11];
      size = header[12] * 0x100 + header[13];

      cnt = 0;
      do {
         if ((nbyte = recv(cli_sock, &(buf[cnt]), size-cnt, 0)) < 0){
            printf("Fail to receive : %s\n", strerror(errno));
            exit(1);
```

```
        }
        cnt += nbyte;
     }
     while ((cnt<size) && (nbyte>0));
     memcpy(data, buf, size);
     data[size]='\0';

     if (verbose == 0) {
        if (type==0)
           printf("%d: %s", id, data);
     }
     else {

        printf("type=%d, ", type);
        printf("id=%d, ", id);
        printf("time=%ld, ", time);
        printf("size=%d\n", size);

        if (type==0)
           printf("data=%s\n", data);
        else {
           for (i=0;i<size;i++)
              printf("%02x ", data[i]);
           printf("\n\n");
        }
     }

     send(cli_sock,"",1,0);

  } while(nbyte>0);

  close(cli_sock);
  return 0;
}
```

Appendix B

Source code of program used to test Serial Forwarder

```
//sf_test.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/stat.h>
#include <sys/errno.h>

#define MAXLEN 1024*1024

extern int errno;

int main(int argc, char* argv[]) {
   struct sockaddr_in srv_addr;
   struct hostent *host;
   int cli_sock;
   int addr_size, nbyte;
   unsigned char buf[MAXLEN + 1];
   int i;
   unsigned int len1;
   unsigned int addr, hid, gid, len2;
   unsigned char payload[257];
   int cnt;

   /* create socket */
   if((cli_sock = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
      fprintf(stderr, "Fail to create socket: %s.\n", strerror(errno));
      exit(1);
   }

   /* prepare server address */
   host = gethostbyname("localhost");
   memcpy((char*)&srv_addr.sin_addr, (char*)host->h_addr, host-
>h_length);
   srv_addr.sin_port = htons((u_short) 9001);
   srv_addr.sin_family = AF_INET;

   addr_size = sizeof(srv_addr);

   /* connect to the server */
   if (connect(cli_sock, (struct sockaddr *) &srv_addr,
sizeof(srv_addr))<0) {
      fprintf(stderr, "Fail to connect %s :\n", strerror(errno));
      exit(1);
   }

   printf("connected to Serial Forwarder\n");
```

```
   send(cli_sock, "T ", 2,0);
   //get 0x54 0x20
   if ((nbyte = recv(cli_sock, buf, 1, 0)) < 0) {
      printf("Fail to receive : %s\n", strerror(errno));
      exit(1);
   }
   if (nbyte > 0)
      if ((nbyte = recv(cli_sock, buf, 1, 0)) < 0) {
         printf("Fail to receive : %s\n", strerror(errno));
         exit(1);
      }

   while (nbyte > 0) {
      if ((nbyte = recv(cli_sock, &len1, 1, 0)) < 0) {
         printf("Fail to receive : %s\n", strerror(errno));
         exit(1);
      }
      cnt = 0;
      do {
         if ((nbyte = recv(cli_sock, &(buf[cnt]), len1-cnt, 0)) < 0)
         {
            printf("Fail to receive : %s\n", strerror(errno));
            exit(1);
         }
         cnt += nbyte;
      }
      while ((cnt<len1) && (nbyte>0));
/*
      for (i=0;i<nbyte;i++)
         printf("%02x ", buf[i]);
*/
      addr = buf[0] + buf[1]*0x100;
      hid = buf[2];
      gid = buf[3];
      len2 = buf[4];
      memcpy(payload, &(buf[5]), len2);

      printf("addr=%d, ", addr);
      printf("handleID=%d, ", hid);
      printf("groupID=%d, ", gid);
      printf("payload_size=%d\n", len2);
      for (i=0;i<len2;i++)
         printf("%02x ", payload[i]);
      putchar('\n');
      putchar('\n');
   }
   close(cli_sock);
   return 0;
}
```

Appendix C

Stack Trace of Serial Forwarder when using with TOSSIM-radio

From TOSSIM to user:

```
net.tinyos.packet.SFProtocol.writeSourcePacket();
net.tinyos.packet.AbstractSource.writePacket();
net.tinyos.sf.SFClient.packetReceived();
net.tinyos.packet.PhoenixSource.dispatch();
net.tinyos.packet.PhoenixSource.packetDipatchLoop();
net.tinyos.packet.AbstractSource.readPacket();
net.tinyos.sim.packet.TossimSource.readSourcePacket();
net.tinyos.sim.packet.TossimRadioSource.readTossimPacket();
net.tinyos.sim.SimProtocol.readEvent();
```

From user to TOSSIM :

```
net.tinyos.packet.SFProtocol.readN();
net.tinyos.packet.SFProtocol.readSourcePacket();
net.tinyos.packet.AbstractSource.readPacket();
net.tinyos.sf.SFClient.readPacket();
net.tinyos.packet.PhoenixSource.writePacket();
net.tinyos.packet.AbstractSource.writePacket();
net.tinyos.sim.packet.TossimSource.writeSourcePacket();
net.tinyos.sim.packet.TossimRadioSource.writeTossimPacket();
net.tinyos.sim.SimProtocol.writeCommand();
```

Appendix D

Source Code of the Simulator

List of files:

1. CMPE.java
2. CMPE_TDMA_Comparator.java
3. CMPE_TDMA_Node.java
4. CMPE_TDMA_Tree_Node.java
5. Direct.java
6. Display.java
7. HIT.java
8. LEACH.java
9. Log.java
10. Node.java
11. NodeCMPE.java
12. NodeDirect.java
13. NodeHIT.java
14. NodeLEACH.java
15. NodePEGASIS.java
16. PEGASIS.java
17. Protocol.java
18. Queue.java
19. ReceivePkt.java
20. SendPkt.java
21. Simulator.java
22. UI.java

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: CMPE.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;

public class CMPE
    extends Protocol {

  NodeCMPE[] nodes;
  NodeCMPE[] heads;
  CMPE_TDMA_Tree_Node [] treeIndex = new CMPE_TDMA_Tree_Node [Simulator.
      no_of_nodes];
  CMPE_TDMA_Tree_Node  head;

  int[] TDMA2 = new int[Simulator.no_of_nodes];

  static final int BASE_STATION_ADV = 1;
  static final int DISCOVER = 2;
  static final int UP_STREAM = 3;
  static final int TDMA = 4;
  static final int DATA = 0;

  static final boolean LONGFORM = true;
  public int TYPE;
  float f;

  static final double range = Math.sqrt( ( (double) (Simulator.width *
      Simulator.height)) / ( (double) Simulator.no_of_nodes)) * 2;

  public CMPE(double[][] points, boolean stepByStep, int TYPE, float f) {
    super();

    int i;

    display.name = new String("CMPE");
    display.f.setTitle("CMPE");
    this.stepByStep = stepByStep;
    this.TYPE = TYPE;
    this.f = f;

    nodes = new NodeCMPE[points.length];
    for (i = 0; i < points.length; i++) {
      nodes[i] = new NodeCMPE(i, points[i][0], points[i][1], this, TYPE, f);
    }

    NAME = "CEMP " + TYPE;

    super.nodes = nodes;

    nodes[0].send( -1, "", 100000, BASE_STATION_ADV, 0);
    processSend();
  }

  public void setup() {
    tempDist = 0;
```

```
int i, j = 0;
ArrayList headList = new ArrayList();

display.clean();

for (i = 0; i < nodes.length; i++) {
  nodes[i].reset();
}
setupEnergy = 0;

/* start simulation */
/* step one: election of cluster head */
Log.write("CMPE 1.1");
/* random election */
int id;
heads = null;
headList.clear();

if (aliveSet.size() > ( (nodes.length * Simulator.clusterHeadPercent ) / 100)) {
  i = ( (nodes.length * Simulator.clusterHeadPercent ) / 100);

  while (i-- > 0) {
    do {
      id = (int) (Math.random() * nodes.length);
    }
    while (!nodes[id].isAlive && !nodes[id].isHead);

    nodes[id].isHead = true;
    nodes[id].cluster = j;
    nodes[id].upstream = 0;
    nodes[id].upstreamDistance = nodes[id].BS_Distance;
    j++;
    headList.add(nodes[id]);
    Log.write("node " + id + " became the cluster head, distance to BS " +
              nodes[id].upstreamDistance);

  }
}
else {
  Iterator iterator = aliveSet.iterator();
  while (iterator.hasNext()) {
    id = ( (Integer) iterator.next()).intValue();
    nodes[id].isHead = true;
    nodes[id].cluster = j;
    nodes[id].upstream = 0;
    nodes[id].upstreamDistance = nodes[id].BS_Distance;
    j++;
    headList.add(nodes[id]);
    Log.write("node " + id + " became the cluster head, distance to BS " +
              nodes[id].upstreamDistance);
  }
}
heads = (NodeCMPE[]) headList.toArray(new NodeCMPE[headList.size()]);
/**/
/* base on node id * /
heads = null;
headList.clear();
for (i = 1; i < nodes.length; i++) {
  if (nodes[i].isAlive)
    if ( (i + Simulator.round) % (100 / Simulator.clusterHeadPercent) == 0)
//base on node-id
```

```
        //if (Math.random()<((double)Simulator.clusterHeadPercent/(double)100))
//random choosing
        {
          nodes[i].isHead = true;
          nodes[i].cluster = j;
          nodes[i].upstream = 0;
          nodes[i].upstreamDistance = nodes[i].BS_Distance;
          j++;
          headList.add(nodes[i]);
          Log.write("node " + i + " became the cluster head, distance to BS " +
                    nodes[i].upstreamDistance);
        }
        else {
          nodes[i].isHead = false;
        }
    }
    heads = (NodeCMPE[]) headList.toArray(new NodeCMPE[headList.size()]);
    /**/

     /* part two: route setup */
     /* phase one: cluster setup */
     /* calculate diestance form cluster head */
    Log.write("CMPE 2.1");

    for (i = 0; i < heads.length; i++) {
      switch (TYPE) {
        case 1:
          heads[i].send( -1, "0", range, DISCOVER, 30);
          break;
        case 2:
          heads[i].send( -1, "0", range, DISCOVER, 30);
          break;
        case 3:
          heads[i].send( -1, "0", range, DISCOVER, 30);
          break;
        case 4:
          heads[i].send( -1, "0,0", range, DISCOVER, 40);
          break;
      }
    }
    processSend();

    /* part three: TDMA scheduling */
    /* phase one: downstream regisration */
    Log.write("CMPE 3.1");
    for (i = 1; i < nodes.length; i++) {
      //if (nodes[i].isHead == false)
      {
        nodes[i].send(nodes[i].upstream,
                      new Integer(nodes[i].upstream).toString(),
                      nodes[i].upstreamDistance , UP_STREAM, 28);
      }
    }
    processSend();

    /* part three: TDMA scheduling */
    /* phase two: send downstream/blocking list */
    Log.write("CMPE 3.2");
    for (i = 1; i < nodes.length; i++) {
      if (nodes[i].downstream.size() == 0) {
        if (LONGFORM)
```

```
          nodes[i].send(nodes[i].upstream, nodes[i].id + ";0;0;;",
                         nodes[i].upstreamDistance , TDMA, 36); // for debug, more
message sent
        else
          nodes[i].send(nodes[i].upstream, nodes[i].id + ";0;;",
                         nodes[i].upstreamDistance , TDMA, 36); // for saving byte and
energy
      }
    }
    processSend();

    /* phase three: TDMA schedule */
    Log.write("CMPE 3.3");
    TDMA_Schedule();

    // find time slot needed
    maxtime = 0;
    for (i = 1; i < nodes.length; i++) {
      if (maxtime < nodes[i].timeSlot)
        maxtime = nodes[i].timeSlot;
    }
    maxtimes.add(new Integer(maxtime));

    /* phase four: schedule broadcast */
    Log.write("CMPE 3.4");

    // "reverse" the time slot
    Iterator iterator;
    CMPE_TDMA_Tree_Node tempNode;
    treeIndex[0].timeSlot = -1;
    for (i = 1; i < treeIndex.length; i++) {
      treeIndex[i].timeSlot = maxtime - treeIndex[i].timeSlot;
      iterator = treeIndex[i].associate.iterator();
      while (iterator.hasNext()) {
        tempNode = (CMPE_TDMA_Tree_Node ) iterator.next();
        tempNode.timeSlot = maxtime - tempNode.timeSlot;
      }
    }

    nodes[0].timeSlot = -1;
    for (i = 1; i < nodes.length; i++)
      nodes[i].timeSlot = maxtime - nodes[i].timeSlot;

      /* step four: data transmission */
    Log.write("CMPE 4");

//    printNodeInfo();
//    printRouteTree();
//    printTDMATree();
//    printSchedule();

    /* evaluation */
    Log.write(NAME + " : max delay = " + maxtime);
    Log.write(NAME + " : setup energy = " + setupEnergy);

    display.markAll(nodes);
    display.label.setText("CMPE " + TYPE);
  }

  private void TDMA_Schedule() {
    /*
```

```
        // Pseudo code for the TDMA scheduling
        add the cluster head to QUEUE;
        while (QUEUE is not empty) {
                UPSTREAM_NODE = first node in QUEUE;
                TIME_SLOT = the assigned time slot of UPSTREAM_NODE + 1;
                for (each DOWNSTREAM_NODE in UPSTREAM_NODE) {
                        while (TIME_SLOT is blocked by another node) {
                                add dummy
                                TIME_SLOT++;
                        }
                        assign TIME_SLOT to DOWNSTREAM_NODE;
                        if (DOWNSTREAM_NODE blocks BLOCK_NODE)
                        // i.e., BLOCK_NODE is blocked by DOWNSTREAM_NODE
                                shift BLOCK_NODE;
                        add DOWNSTREAM_NODE to QUEUE;
                        TIME_SLOT++;
                }
        }
     */
    int i, j;
    int time = 0;
    int parent, child;
    int id;
    //    boolean isBlock = false;
    boolean isHead = false;
    ArrayList queue = new ArrayList();
    Iterator iterator;
    String tempString = "";
//    boolean verbose = false;

    CMPE_TDMA_Tree_Node currentParent = null;
    CMPE_TDMA_Tree_Node currentChild = null;
//    CMPE_TDMA_Tree_Node tempNode = null;

    for (i = 0; i < nodes.length; i++) {
      TDMA2[i] = -1;
      treeIndex[i] = new CMPE_TDMA_Tree_Node(i);
    }
    treeIndex[0].timeSlot = 0;
    TDMA2[0] = 0;
    nodes[0].timeSlot = 0;
    head = treeIndex[0];

    // build the BLOCK, BLOCKED_BY first
    // BLOCKED-BY(m) = nodes[m].BLOCKED_BY;
    // BLOCK(m) = nodes[m].BLOCK
    // all nodes in BLOCKED_LIST block all nodes in downstream
    // all nodes in downstream are blocked by all nodes in BLOCKED_LIST
    for (i = 0; i < nodes.length; i++) {
      iterator = nodes[i].BLOCK_LIST.iterator();
      while (iterator.hasNext()) {
        // for each node n in BLOCKED_LIST, add all the element in downstream to
n.BLOCKED_BY
        nodes[ ((Integer) iterator.next()).intValue()].BLOCKED_BY.addAll(nodes[
            i].downstream);
      }

      iterator = nodes[i].downstream.iterator();
      while (iterator.hasNext()) {
        nodes[ ((Integer) iterator.next()).intValue()].BLOCK.addAll(nodes[i].
            BLOCK_LIST);
```

```
      }
    }

    queue.add(head);
    while (!queue.isEmpty()) {
      currentParent = ( (CMPE_TDMA_Tree_Node ) queue.remove(0));
      parent = currentParent.id;
      time = currentParent.timeSlot;
      isHead = true;
      time++;
      Parameters parameters = new Parameters(isHead, currentChild);
      for (i = 0; i < nodes[parent].sortedNodeList.length; i++) {
        child = nodes[parent].sortedNodeList[i].id;

        // check if blocked by others, add dummy if so
        time = assignTimeSlot(time, child, queue, null, currentParent,
                              parameters);
      }
    }
  }

//globle
//globle
  private class Parameters {
    boolean isHead;
    CMPE_TDMA_Tree_Node currentChild;
    Parameters(boolean isHead, CMPE_TDMA_Tree_Node currentChild) {
      this.isHead = isHead;
      this.currentChild = currentChild;
    }

  }

  private int assignTimeSlot(int time, int child, ArrayList queue,
                             Vector queue2, CMPE_TDMA_Tree_Node currentParent,
                             Parameters parameters) {
    int j;
    int id;
    String tempString = "";
    Iterator iterator;
    CMPE_TDMA_Tree_Node tempNode = null;
    boolean verbose = false;
    boolean isBlock = true;

    // check if blocked by others, add dummy if so
    while (isBlock == true) {
      isBlock = false;
      iterator = nodes[child].BLOCK.iterator();
      while (iterator.hasNext()) {
        id = ( (Integer) iterator.next()).intValue();

        if (nodes[id].timeSlot == time) { // node id blocks this node
          if (parameters.isHead == true) { // head in sibling
            parameters.currentChild = currentParent.child = tempNode = new
                CMPE_TDMA_Tree_Node(id, time);
            currentParent.from = time;
            parameters.isHead = false;
          }
          else {
            parameters.currentChild = parameters.currentChild.sibling =
                tempNode = new
```

```
                CMPE_TDMA_Tree_Node (id, time);
          }
          isBlock = true;
          tempNode.flat = 0;
          tempNode.dummy = true;
          tempNode.associate.add(treeIndex[id]);
          treeIndex[id].associate.add(tempNode);

          if (verbose) Log.write(child + " is blocked by "  + id);
        } // end if (nodes[id].timeSlot == time)
      }
      if (isBlock) time++;
    }

    //found an (eariest) avaliable timeslot
    TDMA2[child] = time;
    nodes[child].timeSlot = time;
    treeIndex[child].timeSlot = time;

    if (parameters.isHead == true) {
      parameters.currentChild = currentParent.child = tempNode = treeIndex[
          child];
      currentParent.from = time;
      parameters.isHead = false;
    }
    else {
      parameters.currentChild = parameters.currentChild.sibling = tempNode =
          treeIndex[child];
    }
    tempNode.flat = 0;
    currentParent.to = time;
    queue.add(tempNode);

    if (verbose) {
      for (j = 0, tempString = ""; j < time; j++, tempString += "\t");
      Log.write(tempString + child + " " + time);
    }

    iterator = nodes[child].BLOCKED_BY.iterator();
    while (iterator.hasNext()) {
      id = ( (Integer) iterator.next()).intValue();
      if (nodes[id].timeSlot == time) { // this node blocks node id
        if (verbose) Log.write(child + " blocks " + id);
        if (queue2==null) shift(id, time);
        else queue2.add(new Integer(id));
      }
    }
    time++;
    return time;
  }

  public void shift(int id, int time) {
    ArrayList queue = new ArrayList();
    Vector queue2 = new Vector();
    CMPE_TDMA_Tree_Node currentParent = null, currentChild = null;
    CMPE_TDMA_Tree_Node currentNode;
    boolean isHead;
    int parent, child;
    int i, j;

    currentParent = treeIndex[nodes[id].upstream];
```

```java
currentChild = currentParent .child;

if (currentParent .child == treeIndex [id])
  isHead = true;
else {
  isHead = false;
  while (currentChild .sibling != treeIndex [id])
    currentChild = currentChild .sibling;
}

// first shift the sibling of the node
Parameters parameters = new Parameters (isHead, currentChild );
for (currentNode = treeIndex [id]; currentNode != null;
     currentNode = currentNode .sibling) {
  child = currentNode .id;
  time = assignTimeSlot (time, child, queue, queue2, currentParent ,
                         parameters );
  while (currentNode .sibling != null && currentNode .sibling.dummy == true) {
    currentNode .sibling = currentNode .sibling .sibling;
  }
}

// then shift the downstream nodes of the node
while (!queue .isEmpty ()) {
  currentParent = ( (CMPE_TDMA_Tree_Node ) queue .remove (0));
  parent = currentParent .id;
  time = currentParent .timeSlot ;
  isHead = true;
  time++;
  parameters = new Parameters (isHead, currentChild );
  for (i = 0; i < nodes [parent ].sortedNodeList .length; i++) {
    child = nodes [parent ].sortedNodeList [i].id;
    time = assignTimeSlot (time, child, queue, queue2, currentParent ,
                           parameters );
  }
}

// also shift the nodes that blocked by the nodes just shift.
while (!queue2 .isEmpty ()) {
  id = ( (Integer) queue2 .remove (0)).intValue ();
  time = TDMA2 [id];

  currentParent = treeIndex [nodes [id].upstream ];
  currentChild = currentParent .child;

  if (currentParent .child == treeIndex [id])
    isHead = true;
  else {
    isHead = false;
    while (currentChild .sibling .id !=
           id) {
      currentChild = currentChild .sibling;
    }
  }

  // shift the sibling nodes
  parameters = new Parameters (isHead, currentChild );
  for (currentNode = treeIndex [id]; currentNode != null;
       currentNode = currentNode .sibling) {
    child = currentNode .id;
    time = assignTimeSlot (time, child, queue, queue2, currentParent ,
```

```
                              parameters);
        while (currentNode.sibling != null && currentNode.sibling.dummy == true) {
          currentNode.sibling = currentNode.sibling.sibling;
        }
      }

      // lastly shift the downsteram nodes
      while (!queue.isEmpty()) {
        currentParent = ((CMPE_TDMA_Tree_Node) queue.remove(0));
        parent = currentParent.id;
        time = currentParent.timeSlot;
        isHead = true;
        time++;
        parameters = new Parameters(isHead, currentChild);
        for (i = 0; i < nodes[parent].sortedNodeList.length; i++) {
          child = nodes[parent].sortedNodeList[i].id;
          time = assignTimeSlot(time, child, queue, queue2, currentParent,
                                parameters);
          while (queue2.remove(new Integer(child)));
        }
      }
    }
  }

  public void printNodeInfo() {
    int i, j;
    Iterator iterator;
    System.out.println("--- node info ---");
    for (i = 0; i < nodes.length; i++) {
      System.out.println(i);

      System.out.println("   timeslot = " + nodes[i].timeSlot);

      System.out.println("   upstream = " + nodes[i].upstream);

      System.out.println("   total down/block = " +
                        (nodes[i].totalDownstream + nodes[i].totalBlocking));

      System.out.print("   downstream = ");
      for (j = 0; j < nodes[i].sortedNodeList.length; j++) {
        System.out.print(nodes[i].sortedNodeList[j].id + ", ");
      }
      System.out.println();

      iterator = nodes[i].BLOCK_LIST.iterator();
      System.out.print("   BLOCK_LIST = ");
      while (iterator.hasNext()) {
        System.out.print((Integer) iterator.next() + ", ");
      }
      System.out.println();

      iterator = nodes[i].BLOCK.iterator();
      System.out.print("   BLOCK = ");
      while (iterator.hasNext()) {
        System.out.print((Integer) iterator.next() + ", ");
      }
      System.out.println();

      iterator = nodes[i].BLOCKED_BY.iterator();
      System.out.print("   BLOCKED_BY = ");
      while (iterator.hasNext()) {
```

```
      System.out.print( (Integer) iterator.next() + ", ");
    }
    System.out.println();

    System.out.println("   upstream distance = "  + nodes[i].upstreamDistance );
  }
}

public void printRouteTree () {
  HashSet pending = new HashSet();
  Log.write("--- route tree ---" );
  printRouteTree (nodes[0], 0, pending, true);
}

public void printRouteTree (NodeCMPE n, int level, HashSet pending,
                            boolean last) {
  int i, j;
  String tempString = "";
  CMPE_TDMA_Node  ns[] = n.sortedNodeList ;

  /* expended mode, takes up more lines but easier to read */
  for (j = 0; j < level; j++) {
    if (pending.contains(new Integer(j)))
      System.out.print("|");
    else
      System.out.print(" ");
    System.out.print("\t");

  }
  System.out.println( (last ? "\\" : "+") + "-------" + n.id);

  pending.add(new Integer(level + 1));
  for (i = 0; i < ns.length; i++) {
    if (i == ns.length - 1) pending.remove(new Integer(level + 1));
    printRouteTree (nodes[ns[i].id], level + 1, pending, (i == ns.length - 1));
  }
  /**/

  /* compressed mode, takes less lines but more difficult to read * /
   System.out.print( ( (n.downstream.size() > 1) ? "+" : " ") + n.id + "\t");

       for (i = 0; i < ns.length; i++) {
    if (i == 0) pending.add(new Integer(level));
    if (i == ns.length - 1) pending.remove(new Integer(level));
    printRouteTree(nodes[ns[i].id], level + 1, pending, false);
    if (i < ns.length - 1) {
      for (j = 0; j < level; j++) {
        if (pending.contains(new Integer(j)))
          System.out.print("| ");
        else
          System.out.print("  ");
        System.out.print("\t");

      }
      System.out.print( ( (i == ns.length - 2) ? "\\" : "+") + "-------");
    }

       }
       if (ns.length == 0) {
    System.out.println();
       }
```

```
        /**/
  }

  public void printTDMATree () {
    int i;
    String tempString = "";
    HashSet pending = new HashSet();
    Log.write("--- TDMA tree ---" );

    for (i = maxtime; i > 0; i--)
      System.out.print("| " + i + "\t");
    System.out.println();

    if (head != null)
      printTDMATree(head, 0, pending);
    System.out.println();
  }

  public void printTDMATree(CMPE_TDMA_Tree_Node  n, int level, HashSet pending) {
    int i, j;

    System.out.print( ( (n.child == null) ? " " : "+") + (n.dummy ? "d" : " ") +
                      n.id + "." + n.timeSlot + "\t");
    if (n.child != null)
      pending.add(new Integer(level));

    if (n.sibling != null) {
      printTDMATree(n.sibling, level + 1, pending);
    }
    if (n.child != null) {
      System.out.println();
      for (j = 0; j < level; j++)
        if (pending.contains(new Integer(j)))
          System.out.print("|\t");
        else
          System.out.print(" \t");
      System.out.print("\\-------" );
      pending.remove(new Integer(level));

      printTDMATree(n.child, level + 1, pending);

    }
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: CMPE_TDMA_Comparator.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;

public class CMPE_TDMA_Comparator
    implements Comparator {

  public int compare(Object o1, Object o2) {
    CMPE_TDMA_Node n1 = (CMPE_TDMA_Node ) o1;
    CMPE_TDMA_Node n2 = (CMPE_TDMA_Node ) o2;

    if ( (n1.totalDownstream + n1.totalBlocking ) <
        (n2.totalDownstream + n2.totalBlocking ))
      return 1;
    else if ( (n1.totalDownstream + n1.totalBlocking ) >
             (n2.totalDownstream + n2.totalBlocking ))
      return -1;
    else // if ((n1.totalDownstream + n1.totalBlocking) == (n2.totalDownstream +
n2.totalBlocking))
      return 0;
  }

}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: CMPE_TDMA_Node.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

public class CMPE_TDMA_Node {
  int id = -1;
  int totalDownstream = 0;
  int totalBlocking = 0;
  String list = "";

  public CMPE_TDMA_Node (int id, int totalDownstream , int totalBlocking , String list)
{
    this.id = id;
    this.totalDownstream = totalDownstream ;
    this.totalBlocking = totalBlocking ;
    this.list = list;
  }

  public String toString() {
    return Integer.toString(id);
  }

}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: CMPE_TDMA_Tree_Node.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;

public class CMPE_TDMA_Tree_Node  {

  public CMPE_TDMA_Tree_Node  sibling = null;
  public CMPE_TDMA_Tree_Node  child = null;
  public ArrayList block = null;
  public boolean dummy = false;
  public ArrayList associate = new ArrayList();

  public int id = -1;
  public int timeSlot = -1;
  public int from = -1, to = -1; //for child
  public int flat;

  public CMPE_TDMA_Tree_Node (int id) {
    this.id = id;
  }

  public CMPE_TDMA_Tree_Node (int id, int timeSlot) {
    this.id = id;
    this.timeSlot = timeSlot;
  }

  public String toString() {
    return Integer.toString(id);
  }

  public boolean equals(Object o) {
    return ( ( (CMPE_TDMA_Tree_Node ) o).id == id);
  }

}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: Direct.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

public class Direct
    extends Protocol {

  NodeDirect [] nodes;
  NodeDirect [] heads;
  static final int DATA = 0;
  static final int BASE_STATION_ADV = 1;

  public Direct(double[][] points, boolean stepByStep) {
    super();

    int i;

    display.name = new String("PEGASIS");
    display.f.setTitle("PEGASIS");

    this.stepByStep = stepByStep;

    nodes = new NodeDirect[points.length];
    for (i = 0; i < points.length; i++)
      nodes[i] = new NodeDirect(i, points[i][0], points[i][1], this);

    super.nodes = nodes;

    NAME = "Direct";

    nodes[0].send( -1, "", 100000, BASE_STATION_ADV , 0);
    processSend();
  }

  public void setup() {
    int i, j = 0;

    display.clean();

    for (i = 0; i < nodes.length; i++) {
      nodes[i].reset();
    }

    for (i = 1; i < nodes.length; i++) {
      nodes[i].upstream = 0;
      nodes[i].upstreamDistance = nodes[i].BS_Distance;
      nodes[i].timeSlot = i;
    }
    maxtime = aliveSet.size()-1;

    /* data transmission */
//    printNodeInfo();

    /* evaluation */
    Log.write(NAME + " : max delay = " + maxtime);
    Log.write(NAME + " : setup energy = " + setupEnergy);
```

```
    display.markAll(nodes);
    display.label.setText("Direct ");
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: Display.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class Display /* extends JPanel */ {
//  private Display display = null;
  JLabel label;
  JFrame f;
  JPanel p, q;
  Point point = new Point(0, 0);
  Dimension preferredSize;
//  Node nodes[];
  int width, height;
  int xScale = 3, yScale = 3;
  int CH_DotSize = 9, dotSize = 3;
  String name;

  public Display(int width, int height, String name) {
    this.width = width;
    this.height = height;
    this.name = name;

    preferredSize = new Dimension(width * xScale, height * yScale);
    f = new JFrame("WSN Simulator : " + name);
    p = new JPanel();
    q = new JPanel();
    label = new JLabel("WSN Simulator : " + name);

    Container container = f.getContentPane();
    /**/
    p.setPreferredSize(new Dimension(200, 200));
    p.setBackground(Color.WHITE);
    f.setBackground(Color.WHITE);

    container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
    container.add(p);
    container.add(label);

    //Align the left edges of the components.
    p.setAlignmentX(p.LEFT_ALIGNMENT);
    label.setAlignmentX(p.LEFT_ALIGNMENT); //redundant

    f.pack();

    /**/
    f.setVisible(true);

    p.addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent e) {
```

```
        int x = e.getX();
        int y = e.getY();
        //label.setText("("+getWidth()+","+getHeight()+")");
      }
    });

    p.addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseMoved(MouseEvent e) {
        label.setText("(" + e.getX() / 5 + "," + e.getY() / 5 + ")");
      }
    });

    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
  }

  public void clean() {
    p.getGraphics().clearRect(0, 0, p.getWidth(), p.getHeight());
  }

  public void markNodes(Node nodes[]) {
    int i;
    Graphics g = p.getGraphics();

    for (i = 0; i < nodes.length; i++) {
      if (nodes[i].isHead == true)
        g.fillRect(x(nodes[i].x) - CH_DotSize / 2,
                   y(nodes[i].y) - CH_DotSize / 2, CH_DotSize, CH_DotSize);
      else
        g.fillRect(x(nodes[i].x) - dotSize / 2, y(nodes[i].y) - dotSize / 2,
                   dotSize, dotSize);
    }

    //label.setText("("+getWidth()+","+getHeight()+")");
  }

  public void markConnections(Node nodes[], Node heads[]) {
    int tempDist = 0;
    int i, j;
    Graphics g = p.getGraphics();

    for (i = 1; i < nodes.length; i++) {
      if ( (nodes[i].upstream != -1) && (nodes[i].upstream != 0)) {
        g.drawLine(x(nodes[i].x), y(nodes[i].y),
                   x(nodes[nodes[i].upstream].x),
                   y(nodes[nodes[i].upstream].y));
        tempDist +=
            ( (nodes[i].x - nodes[nodes[i].upstream].x) *
             (nodes[i].x - nodes[nodes[i].upstream].x) +
             (nodes[i].y - nodes[nodes[i].upstream].y) *
             (nodes[i].y - nodes[nodes[i].upstream].y));
      }
    }
//    Log.write(name + ": Distance square=" + tempDist);
//    label.setText("ok");
  }

  public void markAll(Node nodes[]) {
```

```
    int i;
    Graphics g = p.getGraphics ();
    for (i = 1; i < nodes.length; i++) {
      if (nodes[i].isHead == true)
        g.fillRect (x(nodes[i].x) - CH_DotSize / 2,
                    y(nodes[i].y) - CH_DotSize / 2, CH_DotSize, CH_DotSize);
      else
        g.fillRect (x(nodes[i].x) - dotSize / 2, y(nodes[i].y) - dotSize / 2,
                    dotSize, dotSize);

      if ( (nodes[i].upstream != -1) && (nodes[i].upstream != 0)) {
        g.drawLine (x(nodes[i].x), y(nodes[i].y),
                    x(nodes[nodes[i].upstream].x),
                    y(nodes[nodes[i].upstream].y));
      }
    }
  }

  public void markNeighbors (Node nodes[]) {
    int i;
    Graphics g = p.getGraphics ();
    Iterator iterator;
    int id;

    Color oldColor = g.getColor ();
    g.setColor (Color.YELLOW);

    for (i = 1; i < nodes.length; i++) {
      iterator = nodes[i].neighbors.iterator ();
      while (iterator.hasNext ()) {
        id = ( (Integer) iterator.next ()).intValue ();
        g.drawLine (x(nodes[i].x), y(nodes[i].y),
                    x(nodes[id].x),
                    y(nodes[id].y));
      }
    }
    g.setColor (oldColor);
  }

  public int x(double x) {
    return (int) (x * ( ( (double) p.getWidth ()) / (double) width));
  }

  public int y(double y) {
    return ( - ( (int) (y * ( ( (double) p.getHeight ()) / ( (double) height)))) +
            p.getHeight () - 1);
  }

  public void displayName () {
    label.setText (name);
  }

  public static void main(String[] args) {
    Display display = new Display(200, 200, "");
  }

  public Dimension getPreferredSize () {
    return preferredSize;
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: HIT.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;

public class HIT
    extends Protocol {

  NodeHIT[] nodes;
  NodeHIT[] heads;
  static final int BASE_STATION_ADV = 1;
  static final int CH_ADV = 2;
  static final int MEMBER = 3;
  static final int MY_UPSTREAM = 4;
  static final int BLOCKDOWN = 5;
  static final int DATA = 0;

  static final double range = Math.sqrt(2) * Simulator.width;

  ArrayList SEND = new ArrayList();

  public HIT(double[][] points, boolean stepByStep) {
    super();
    int i;

    display.name = new String("HIT");
    display.f.setTitle("HIT");

    this.stepByStep = stepByStep;

    nodes = new NodeHIT[points.length];
    for (i = 0; i < points.length; i++)
      nodes[i] = new NodeHIT(i, points[i][0], points[i][1], this);
    super.nodes = nodes;
    NAME = "HIT";

    nodes[0].send( -1, "", 100000, BASE_STATION_ADV , 0);
    processSend();
  }

  public void setup() {
    tempDist = 0;
    int i, j = 0;
    ArrayList headList = new ArrayList();
    int minHead = -1;
    double dist, minDist;
    Iterator iterator;
    String tempString;
    int id;
    Integer ID;

    SEND.clear();

    display.clean();
```

```
    for (i = 0; i < nodes.length; i++) {
      nodes[i].reset();
    }

    /* start simulation */
    /* phase one: election of cluster head */
    Log.write("HIT 1");
    /* random election */
    heads = null;
    headList.clear();

    if (aliveSet.size() > ( (nodes.length * Simulator.clusterHeadPercent ) / 100)) {
      i = ( (nodes.length * Simulator.clusterHeadPercent ) / 100);

      while (i-- > 0) {
        do {
          id = (int) (Math.random() * nodes.length);
        }
        while (!nodes[id].isAlive && !nodes[id].isHead);

        nodes[id].isHead = true;
        nodes[id].cluster = j;
        nodes[id].upstream = 0;
        nodes[id].upstreamDistance = nodes[id].BS_Distance;
        nodes[id].diH = 0;
        nodes[id].diu = nodes[id].BS_Distance;
        j++;
        headList.add(nodes[id]);
        Log.write("node " + id + " became the cluster head, distance to BS "  +
                  nodes[id].upstreamDistance);
      }
    }
    else {
      iterator = aliveSet.iterator();
      while (iterator.hasNext()) {
        id = ( (Integer) iterator.next()).intValue();
        nodes[id].isHead = true;
        nodes[id].cluster = j;
        nodes[id].upstream = 0;
        nodes[id].upstreamDistance = nodes[id].BS_Distance;
        nodes[id].diH = 0;
        nodes[id].diu = nodes[id].BS_Distance;
        j++;
        headList.add(nodes[id]);
        Log.write("node " + id + " became the cluster head, distance to BS "  +
                  nodes[id].upstreamDistance);
      }
    }
    heads = (NodeHIT[]) headList.toArray(new NodeHIT[headList.size()]);
    /**/
    /* base on node id * /
         heads = null;
         headList.clear();
         for (i = 1; i < nodes.length; i++) {
      if (nodes[i].isAlive)
      if ( (i + Simulator.round) % (100 / Simulator.clusterHeadPercent) == 0) //base
on node-id                                                                        Ã
      //if (Math.random()<((double)Simulator.clusterHeadPercent/(double)100))
//random choosing                                                                 Ã
      {
        nodes[i].isHead = true;
```

```
        nodes[i].cluster = j;
        nodes[i].upstream = 0;
        nodes[i].upstreamDistance = nodes[i].BS_Distance;
        nodes[i].diH = 0;
        nodes[i].diu = nodes[i].BS_Distance;
        j++;
        headList.add(nodes[i]);
        Log.write("node " + i + " became the cluster head");
      }
      else {
        nodes[i].isHead = false;
      }
         }
         heads = (NodeHIT[]) headList.toArray(new NodeHIT[headList.size()]);
     /**/

     /* phase two: cluster head advertisement */
    Log.write ("HIT 2");
    for (i = 0; i < heads.length; i++) {
      heads[i].send( -1, "0", range, CH_ADV, 20);

      heads[i].send(0, "", heads[i].BS_Distance , CH_ADV, 20);
    }
    processSend ();

    /* phase three: cluster setup */
    /* the class node will send out message after receiving message form cluster head
*/
    Log.write ("HIT 3");
    for (i = 1; i < nodes.length; i++) {
      nodes[i].send( -1, String.valueOf(nodes[i].diH), range, MEMBER, 30);

    }
    processSend ();

    /* phase four: Route Setup */
    /* send upstream id and the distance */
    Log.write ("HIT 4");
    for (i = 1; i < nodes.length; i++) {
      nodes[i].send( -1, nodes[i].upstream + ";" + nodes[i].diu, range,
                     MY_UPSTREAM, 38);
    }
    processSend ();

    /* phase five: Blocking Set Computation */
    /* send the blocking list */
    Log.write ("HIT 5");
    for (i = 1; i < nodes.length; i++) {
      iterator = nodes[i].BLOCK_LIST.iterator();
      tempString = "";
      while (iterator.hasNext()) {
        tempString += (Integer) iterator.next() + ",";
      }
      nodes[i].send( -1, tempString, range, BLOCKDOWN,
                     20 + tempString.length() * 4);

    }
    processSend ();

    /* phase six: MAC Schedule Setup */
    /* TDMA Scheduling */
```

```
    Log.write("HIT 6");
    TDMA_Schedule();
    maxtime = SEND.size();

    /* data transmission */
    Log.write("HIT 7");

//    printSchedule();
//    printRouteTree();

    /* evaluation */
    Log.write("HIT : max delay = "  + maxtime);
    Log.write("HIT : setup energy = "  + setupEnergy);

    display.markAll(nodes);
    display.label.setText("HIT ");
  }

  private void TDMA_Schedule() {
    int i;
    Integer ID, ID2;
    Iterator iterator, iterator2;
    TreeSet READY = new TreeSet(); // the set of nodes that have a message to send
    TreeSet WAIT = new TreeSet(); // the set of ndoes that are ready, but need to
wait due to blocking conditions                                                   Â
    TreeSet SENDk;
    int time = 0;

    /*
     //Pseudo code for HIT
     DOWN(j) ?The set of downstream neighbors of j.
     BLOCKED-BY(j) ?The set of blocked nodes when node j is transmitting.
     BLOCK(j) ?The set of nodes that block j from transmitting.
     READY ?The set of sensors that have a message to send.
     WAIT ?The set of sensors that are ready, but need to wait due to blocking
conditions.                                                                       Â
     SENDk ?The set of nodes that will send at time slotk of the TDMA schedule.

     while (READY is not empty) {
            for (l = 0 ; l < size(READY) ; l++) {
                    //a node in READY is placed in WAIT if all of its downstream
                    nodes have been scheduled to transmit in earlier slots. //
                    m = READY[l]
                    if (DOWN(m) is empty) {
                            add m into WAIT
                    }
            }
            for (l = 0 ; l < size(WAIT) ; l++) {
                    //a node in WAIT is placed in SEND; all its blocking
                    nodes are removed from WAIT to avoid collisions. //
                    m = WAIT[l];
                    remove m from WAIT
                    remove m from READY
                    add m into SENDk
                    remove m from DOWN(um)
                    remove BLOCKED-BY(m) from WAIT
                    remove BLOCK(m) from WAIT
            }
            // All the nodes in SENDk may transmit simultaneously in time slot
            k of the TDMA schedule. //
            k++;
```

```
    }
    */

   // build the BLOCK, BLOCKED_BY first
   // BLOCKED-BY(m) = nodes[m].BLOCKED_BY;
   // BLOCK(m) = nodes[m].BLOCK
   // all nodes in BLOCKED_LIST block all nodes in downstream
   // all nodes in downstream are blocked by all nodes in BLOCKED_LIST
   // all nodes in downstream block other nodes in downstream
   // all nodes in downstream are blocked by other nodes in downstream

   for (i = 0; i < nodes.length; i++) {
     if (nodes[i].isAlive) {
       iterator = nodes[i].BLOCK_LIST.iterator();
       while (iterator.hasNext()) {
         // for each node n in BLOCKED_LIST, add all the element in downstream to
n.BLOCKED_BY
         nodes[ ( (Integer) iterator.next()).intValue()].BLOCKED_BY.addAll(
             nodes[i].downstream);
       }

       iterator = nodes[i].downstream.iterator();
       while (iterator.hasNext()) {
         nodes[ ( (Integer) iterator.next()).intValue()].BLOCK.addAll(nodes[i].
             BLOCK_LIST);
       }

       iterator = nodes[i].downstream.iterator();
       while (iterator.hasNext()) {
         ID = (Integer) iterator.next();
         nodes[ID.intValue()].BLOCKED_BY.addAll(nodes[i].downstream);
         nodes[ID.intValue()].BLOCKED_BY.remove(ID);
         nodes[ID.intValue()].BLOCK.addAll(nodes[i].downstream);
         nodes[ID.intValue()].BLOCK.remove(ID);
       }
     }
   }

   // add all nodes to READY
   for (i = 1; i < nodes.length; i++) {
     if (nodes[i].isAlive)
       READY.add(new Integer(i));
     READY.addAll(nodes[i].downstream);
   }

   // build WAIT (nodes with no childs)
   while (!READY.isEmpty()) {
     iterator = READY.iterator();
     WAIT.clear();
     while (iterator.hasNext()) {
       ID = (Integer) iterator.next();
       if (nodes[ID.intValue()].downstream.isEmpty())
         WAIT.add(ID);
     }

     SENDk = new TreeSet();
     iterator = WAIT.iterator();
     while (iterator.hasNext()) {
       ID = (Integer) iterator.next();
       iterator.remove();
       READY.remove(ID);
```

```
        SENDk.add(ID);
        nodes[ID.intValue()].timeSlot = time;
        nodes[nodes[ID.intValue()].upstream].downstream.remove(ID);

        //remove BLOCCKED_BY(m) from WAIT
        iterator2 = nodes[ID.intValue()].BLOCKED_BY.iterator();
        while (iterator2.hasNext()) {
          WAIT.remove(iterator2.next());
        }

        //remove BLOCK(m) from WAIT
        iterator2 = nodes[ID.intValue()].BLOCK.iterator();
        while (iterator2.hasNext()) {
          WAIT.remove(iterator2.next());
        }
        iterator = WAIT.iterator(); // java specific, must update iterator after the
underlying treeset is changed
      }
      SEND.add(SENDk);
      time++;
    }
  }

  private void printNodeInfo() {
    Integer ID;
    int i;
    Iterator iterator;
    //print the downstream, BLOCK, BLOCKED_BY list
    System.out.println("--- node info ---");

    for (i = 0; i < nodes.length; i++) {
      System.out.println(i);

      System.out.println("    timeslot = " + nodes[i].timeSlot);

      System.out.println("    isalive = " + nodes[i].isAlive);

      System.out.println("    upstream = " + nodes[i].upstream);

      System.out.println("    upstreamDistance = " +
                         nodes[i].upstreamDistance);

      iterator = nodes[i].downstream.iterator();
      System.out.print("    downstream = ");
      while (iterator.hasNext()) {
        ID = (Integer) iterator.next();
        System.out.print(ID + ", ");
      }
      System.out.println();

      iterator = nodes[i].BLOCK.iterator();
      System.out.print("    BLOCK = ");
      while (iterator.hasNext()) {
        ID = (Integer) iterator.next();
        System.out.print(ID + ", ");
      }
      System.out.println();

      iterator = nodes[i].BLOCKED_BY.iterator();
      System.out.print("    BLOCKED_BY = ");
      while (iterator.hasNext()) {
```

```
        ID = (Integer) iterator.next();
        System.out.print(ID + ", ");
      }
      System.out.println();

      iterator = nodes[i].BLOCK_LIST.iterator();
      System.out.print("    BLOCK_LIST = " );
      while (iterator.hasNext()) {
        ID = (Integer) iterator.next();
        System.out.print(ID + ", ");
      }
      System.out.println();
    }
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: LEACH.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;

public class LEACH
    extends Protocol {

  NodeLEACH[] nodes;
  NodeLEACH[] heads;
  static final int BASE_STATION_ADV = 1;
  static final int CH_ADV = 2;
  static final int DATA = 0;
  static final double range = Math.sqrt(2) * Simulator.width;

  public LEACH(double[][] points, boolean stepByStep) {
    super();
    int i;

    display.name = new String("LEACH");
    display.f.setTitle("LEACH");

    this.stepByStep = stepByStep;

    nodes = new NodeLEACH[points.length];
    for (i = 0; i < points.length; i++)
      nodes[i] = new NodeLEACH(i, points[i][0], points[i][1], this);

    super.nodes = nodes;
    NAME = "LEACH";

    nodes[0].send( -1, "", 100000, BASE_STATION_ADV , 0);
    processSend();
  }

  public void setup() {
    tempDist = 0;
    int i, j = 0;
    ArrayList headList = new ArrayList();

    display.clean();

    for (i = 0; i < nodes.length; i++) {
      nodes[i].reset();
    }

    /* start simulation */
    /* phase one: election of cluster head */
    Log.write("LEACH 1");
    /* random election */
    int id;
    heads = null;
    headList.clear();

    if (aliveSet.size() > ( (nodes.length * Simulator.clusterHeadPercent ) / 100)) {
```

```
      i = ( (nodes.length * Simulator.clusterHeadPercent ) / 100);

      while (i-- > 0) {
        do {
          id = (int) (Math.random() * nodes.length);
        }
        while (!nodes[id].isAlive && !nodes[id].isHead);

        nodes[id].isHead = true;
        nodes[id].cluster = j;
        nodes[id].upstream = 0;
        nodes[id].upstreamDistance = nodes[id].BS_Distance;
        j++;
        headList.add(nodes[id]);
        Log.write("node " + id + " became the cluster head, distance to BS " +
                  nodes[id].upstreamDistance);
      }
    }
    else {
      Iterator iterator = aliveSet.iterator();
      while (iterator.hasNext()) {
        id = ( (Integer) iterator.next()).intValue();
        nodes[id].isHead = true;
        nodes[id].cluster = j;
        nodes[id].upstream = 0;
        nodes[id].upstreamDistance = nodes[id].BS_Distance;
        j++;
        headList.add(nodes[id]);
        Log.write("node " + id + " became the cluster head, distance to BS " +
                  nodes[id].upstreamDistance);
      }
    }

    heads = (NodeLEACH[]) headList.toArray(new NodeLEACH[headList.size()]);
    /**/
    /* base on node id * /
         heads = null;
         headList.clear();
         for (i = 1; i < nodes.length; i++) {
           if (nodes[i].isAlive)
             if ( (i + Simulator.round) % (100 / Simulator.clusterHeadPercent) == 0)     Ã
//base on node-id
             //if (Math.random()<((double)Simulator.clusterHeadPercent/(double)100))     Ã
//random choosing
             {
               nodes[i].isHead = true;
               nodes[i].cluster = j;
               nodes[i].upstream = 0;
               nodes[i].upstreamDistance = nodes[i].BS_Distance;
               j++;
               headList.add(nodes[i]);
               Log.write("node " + i + " became the cluster head");
             }
             else {
               nodes[i].isHead = false;
             }
         }
     heads = (NodeLEACH[]) headList.toArray(new NodeLEACH[headList.size()]);
     /**/
     /* phase two: cluster head advertisement */
    Log.write("LEACH 2");
```

```
    for (i = 0; i < heads.length; i++) {
      heads[i].send( -1, "0", range, CH_ADV, 20);
    }

    /* phase three: cluster setup */
    /* each node calculates diestance form cluster head */
    /* asociates to the cluster head with shortest distance */
    Log.write("LEACH 3");
    processSend();

    /* simple schedle */
    for (i = 1; i < nodes.length; i++) {
      if (nodes[i].upstream != -1) {
        nodes[i].timeSlot = nodes[nodes[i].upstream].downstream.size();
        nodes[nodes[i].upstream].downstream.add(new Integer(i));
      }
    }
    maxtime = 0;
    for (i = 0; i < heads.length; i++) {
      if (maxtime < heads[i].downstream.size())
        maxtime = heads[i].downstream.size();
    }
    for (i = 0; i < heads.length; i++) {
      heads[i].timeSlot = maxtime;
      maxtime++;
    }

    maxtimes.add(new Integer(maxtime));

    /* phase four: data transmission */
    Log.write("LEACH 4");

//    printNodeInfo();
//    printSchedule();
//    printRouteTree();

    /* evaluation */
    Log.write(NAME + " : max delay = " + maxtime);
    Log.write(NAME + " : setup energy = " + setupEnergy);

    display.markAll(nodes);
    display.label.setText("LEACH ");
  }

  public void printNodeInfo() {
    int i, j;
    Iterator iterator;
    System.out.println("--- node info ---");
    for (i = 0; i < nodes.length; i++) {
      System.out.println(i);
      System.out.println("   timeslot = " + nodes[i].timeSlot);
      System.out.println("   upstream = " + nodes[i].upstream);
      System.out.println("   upstream distance = " + nodes[i].upstreamDistance);
      System.out.print("   total downstream = " + nodes[i].downstream.size());
      System.out.print("   downstream = ");

      iterator = nodes[i].downstream.iterator();
      while (iterator.hasNext()) {
        System.out.print( (Integer) iterator.next() + ", ");
      }
```

```
      System.out.println();
    }
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: Log.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

public class Log {
  public Log() {
  }

  static void write(String s) {
    System.out.println(s);
  }

  static void write(int i) {
    System.out.println(i);
  }

  static void write(Object o) {
    System.out.println(o.toString());
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: Node.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;

public abstract class Node {
  public int id;
  public double x, y;
  public double energy=100;//*1000000000;
  public int upstream = -1;
  public int cluster = -1;
  public HashSet neighbors = new HashSet();
  public HashSet downstream = new HashSet();
  public boolean isHead = false;
  public Protocol protocol;
  public double BS_Distance = Double.MAX_VALUE;
  public double Eelec = 5.0E-8, Eamp = 1.0E-10;
//  public double Eelec = 0.00000005 /*50*10^-9*/, Eamp = 0.0000000001 /*100*10^-12
*/;
  public int timeSlot;
  double upstreamDistance = Double.MAX_VALUE;
  boolean isAlive=true;
  int oldTime = -1, oldSource = -1, oldTarget = -1;
  int deadtime;

  protected Node(int id, double x, double y, Protocol protocol) {
    this.id = id;
    this.x = x;
    this.y = y;
    this.protocol = protocol;
  }

  public void reset() {
    cluster = -1;
    isHead = false;
    neighbors.clear();
    downstream.clear();
    upstream = -1;
    timeSlot=-1;
    upstreamDistance = Double.MAX_VALUE;
    oldTime = -1;
    oldSource = -1;
    oldTarget = -1;
  }

  public boolean send(int targetId, String payload, double distance, int type, int
length) {
    SendPkt pkt = new SendPkt(id, targetId, payload, distance, type, length);
    pkt.cluster = cluster;
    protocol.queue.enqueue(pkt);
    protocol.tempDist += distance * distance;

    return false;
  }
```

```
  public abstract boolean receive(ReceivePkt pkt);
  public abstract void sendData(int time);

  public String toString() {
    return ("node " + id + "  ");
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: NodeCMPE.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;

public class NodeCMPE
    extends Node {

  double cost = Double.MAX_VALUE;

  TreeSet BLOCK = new TreeSet(); //the set of nodes that block this node form
transmitting                                                                          Ä
  TreeSet BLOCKED_BY = new TreeSet(); //the set of blocked node when this node is
transmitting                                                                          Ä
  TreeSet BLOCK_LIST = new TreeSet(); // the list of nodes that block the dwonstream
neighbors of this donw                                                                Ä

  int downstreamCounter = 0;
  int totalDownstream = 0;
  int totalBlocking = 0;

  String buf = "";
  HashSet TDMA_Node_Set = new HashSet();
  String TDMA_List = "";
  CMPE_TDMA_Node [] sortedNodeList = new CMPE_TDMA_Node [0];
  int TYPE;
  float f;

  public NodeCMPE(int id, double x, double y, Protocol protocol, int TYPE,
                  float f) {
    super(id, x, y, protocol);
    this.TYPE = TYPE;
    this.f = f;
  }

  public void reset() {
    super.reset();
    cost = Double.MAX_VALUE;
    downstream.clear();
    BLOCK_LIST.clear();
    BLOCK.clear();
    BLOCKED_BY.clear();
    downstreamCounter = 0;
    totalDownstream = 0;
    totalBlocking = 0;
    buf = "";
    TDMA_List = "";
    TDMA_Node_Set.clear();
    sortedNodeList = new CMPE_TDMA_Node [0];
  }

  public boolean receive(ReceivePkt pkt) {
    if (pkt.type == CMPE.BASE_STATION_ADV)
      BS_Distance = pkt.distance;
    else if (pkt.type == CMPE.DISCOVER)
```

```
    receiveDiscover (pkt);
  else if (pkt.type == CMPE.UP_STREAM)
    receiveUpStream (pkt);
  else if (pkt.type == CMPE.TDMA) {
    if (pkt.targetId == id)
      receiveTDMA (pkt);
  }
  else if (pkt.type == CMPE.DATA) {
    receiveDATA (pkt);
  }
  return true;
}

public void receiveDiscover (ReceivePkt pkt) {
  double newCost;

  switch (TYPE) {
    case 1:
      newCost = 2 * Eelec * 100 + Eamp * 100 * pkt.distance * pkt.distance +
          (new Double(pkt.payload).doubleValue());

      if ( (newCost < cost) && (isHead == false)) {
        cost = newCost;
        send( -1, String.valueOf(cost), CMPE.range, CMPE.DISCOVER, 30);
        neighbors.add(new Integer(pkt.sourceId));
        upstream = pkt.sourceId;
        upstreamDistance = pkt.distance;
      }
      break;

    case 2:
      newCost = pkt.distance * pkt.distance +
          (new Double(pkt.payload).doubleValue());

      if ( (newCost < cost) && (isHead == false)) {
        cost = newCost;
        send( -1, String.valueOf(cost), CMPE.range, CMPE.DISCOVER, 30);
        neighbors.add(new Integer(pkt.sourceId));
        upstream = pkt.sourceId;
        upstreamDistance = pkt.distance;
      }
      break;

    case 3:
      newCost = pkt.distance * pkt.distance +
          (new Double(pkt.payload).doubleValue()) * f;

      if ( (newCost < cost) && (isHead == false)) {
        cost = newCost;
        send( -1, String.valueOf(cost), CMPE.range, CMPE.DISCOVER, 30);
        neighbors.add(new Integer(pkt.sourceId));
        upstream = pkt.sourceId;
        upstreamDistance = pkt.distance;
      }
      break;

    case 4:
      String[] costs = pkt.payload.split("[,]");
      double cost1;
      double newCost1 = (new Double(costs[0]).doubleValue()); //cost for this node
```

```
        double newCost2 = (new Double(costs[1]).doubleValue()); //cost for upstream
node
        newCost = newCost1 * f + newCost2 + pkt.distance * pkt.distance;

        if ( (newCost < cost) && (isHead == false)) {
          cost = newCost;
          cost1 = newCost1 + pkt.distance * pkt.distance;
          send( -1, String.valueOf(cost1 + "," + cost), CMPE.range,
               CMPE.DISCOVER, 40);
          neighbors.add(new Integer(pkt.sourceId));
          upstream = pkt.sourceId;
          upstreamDistance = pkt.distance;
        }
        break;
    }
  }

  public void receiveUpStream(ReceivePkt pkt) {
    int pktUpStreamId = Integer.parseInt(pkt.payload);

    if (pktUpStreamId == id) {
      downstream.add(new Integer(pkt.sourceId));
    }
    else {
      BLOCK_LIST.add(new Integer(pkt.sourceId));
    }
  }

  public void receiveTDMA(ReceivePkt pkt) {

    String[] parseString = pkt.payload.split("[;]");

    totalDownstream += Integer.parseInt(parseString[1]);
    if (CMPE.LONGFORM)
      totalBlocking += Integer.parseInt(parseString[2]);

    TDMA_Node_Set.add(new CMPE_TDMA_Node(pkt.sourceId,
                                         Integer.parseInt(parseString[1]),
                                         Integer.parseInt(parseString[2]),
                                         pkt.payload));

    buf = buf + ":" + pkt.payload; // not used yet

    downstreamCounter++;

    if (downstreamCounter == downstream.size()) { // received all the list form
downstream, send the list to upstream
      totalDownstream += downstream.size();
      totalBlocking += BLOCK_LIST.size();

      ArrayList arrayList = new ArrayList(TDMA_Node_Set);
      Collections.sort(arrayList, new CMPE_TDMA_Comparator());
      sortedNodeList = (CMPE_TDMA_Node[]) arrayList.toArray(new CMPE_TDMA_Node[
          arrayList.size()]);

      // packet format = id;total_no_of_downstream_node;otal_no_of_blocking_node;down
stream_node_list;blocking_node_list

      TDMA_List = "";

      // prepare the id, total no of downstream and total no of blocking node
```

```
      if (CMPE.LONGFORM)
        TDMA_List = id + ";" + totalDownstream + ";" + totalBlocking + ";";
      else
        TDMA_List = id + ";" + (totalDownstream + totalBlocking) + ";";

      // prepare the downstream list
      if (sortedNodeList.length > 0)
        TDMA_List += sortedNodeList[0].id;
      for (int i = 1; i < sortedNodeList.length; i++)
        TDMA_List += "," + sortedNodeList[i].id;

      TDMA_List += ";";

      // prepare the blocking list
      Iterator iterator = BLOCK_LIST.iterator();
      if (iterator.hasNext())
        TDMA_List += ( (Integer) iterator.next()).toString();
      while (iterator.hasNext())
        TDMA_List += "," + ( (Integer) iterator.next()).toString();

      TDMA_List += ":";

      // prepare the downstream list of the downstream node
      if (sortedNodeList.length > 0)
        TDMA_List += sortedNodeList[0].list;
      for (int i = 1; i < sortedNodeList.length; i++)
        TDMA_List += ":" + sortedNodeList[i].list;

      //Log.write(TDMA_List);
      send(upstream, TDMA_List, upstreamDistance, CMPE.TDMA,
           (20 + TDMA_List.length() * 4));

    }

  }

  public void receiveDATA(ReceivePkt pkt) {
    // check if data collide
    if (oldTime == Integer.parseInt(pkt.payload)) {
      if (oldTarget == id || pkt.targetId == id)
        Log.write(id + " Error: two pkt at the same time: " + oldSource + " " +
                  oldTime + " " + pkt.sourceId + " " + pkt.payload);
    }

    oldTime = Integer.parseInt(pkt.payload);
    oldSource = pkt.sourceId;
    oldTarget = pkt.targetId;
  }

  public void sendData(int time) {
    if (time == timeSlot) {
      send(upstream, time + "", upstreamDistance, CMPE.DATA, CMPE.PACKET_LEN);
    }
  }

}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: NodeDirect.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

public class NodeDirect
    extends Node {

  public NodeDirect (int id, double x, double y, Protocol protocol) {
    super(id, x, y, protocol);
  }

  public void reset() {
    super.reset();
    isHead = true;
  }

  public boolean receive(ReceivePkt pkt) {
    if (pkt.type == LEACH.BASE_STATION_ADV )
      BS_Distance = pkt.distance;
    else if (pkt.type == LEACH.DATA)
      receiveDATA(pkt);

    return true;
  }

  public void receiveDATA(ReceivePkt pkt) {
    // check if data collide
    if (oldTime == Integer.parseInt(pkt.payload)) {
      if (oldTarget == id || pkt.targetId == id)
        Log.write(id + " Error: two pkt at the same time: "  + oldSource + " " +
                  oldTime + " " + pkt.sourceId + " " + pkt.payload);
    }

    oldTime = Integer.parseInt(pkt.payload);
    oldSource = pkt.sourceId;
    oldTarget = pkt.targetId;
  }

  public void sendData(int time) {
    if (time == timeSlot) {
      send(upstream, time + "", upstreamDistance , Direct.DATA,
           Direct.PACKET_LEN);
    }
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: NodeHIT.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;

public class NodeHIT
    extends Node {

  double diH = Double.MAX_VALUE; // distance from cluster head
  double diu = Double.MAX_VALUE; // distance form upstream neighbor

  TreeSet BLOCK = new TreeSet(); //the set of nodes that block this node form      Ä
transmitting
  TreeSet BLOCKED_BY = new TreeSet(); //the set of blocked node when this node is     Ä
transmitting
  TreeSet BLOCK_LIST = new TreeSet(); // the list of nodes that block the dwonstream    Ä
neighbors of this donw

  public NodeHIT(int id, double x, double y, Protocol protocol) {
    super(id, x, y, protocol);
  }

  public void reset() {
    super.reset();
    diH = Double.MAX_VALUE;
    diu = Double.MAX_VALUE;

    BLOCK.clear();
    BLOCKED_BY.clear();
    BLOCK_LIST.clear();
  }

  public boolean receive(ReceivePkt pkt) {
    if (pkt.type == HIT.BASE_STATION_ADV )
      BS_Distance = pkt.distance;
    else if (pkt.type == HIT.CH_ADV) { // message form cluster head
      receiveCH(pkt);
    }
    else if (pkt.type == HIT.MEMBER) {
      receiveMEM(pkt);
    }
    else if (pkt.type == HIT.MY_UPSTREAM) {
      receiveUP(pkt);
    }
    else if (pkt.type == HIT.BLOCKDOWN) {
      // blank
    }
    else if (pkt.type == HIT.DATA) {
      receiveDATA(pkt);
    }
    return true;
  }

  public void receiveCH(ReceivePkt pkt) {
    if (pkt.distance < diH) {
```

```
      diH = pkt.distance; // d(i, H)
      diu = pkt.distance; // d(i, ui)
      cluster = pkt.cluster;
      upstream = pkt.sourceId;
      upstreamDistance = pkt.distance;
      //Log.write("node " + id + " diH=" + diH);
    }

    if (id == 0) { // add the id of cluster head if the node is the base station
      downstream.add(new Integer(pkt.sourceId));
    }
  }

  public void receiveMEM(ReceivePkt pkt) {
    double duH;

    if ( (pkt.distance < diu) && (pkt.cluster == cluster)) {
      duH = Double.parseDouble(pkt.payload); // d(u, H)
      if ( (duH < diH) && (pkt.distance < diH)) {
        diu = pkt.distance; //d(i, u)
        upstream = pkt.sourceId;
        upstreamDistance = pkt.distance;
      }
    }
  }

  public void receiveUP(ReceivePkt pkt) {
    String tempString[] = pkt.payload.split("[;]"); //j;dju
    int j = Integer.parseInt(tempString[0]);
    double dju = Double.parseDouble(tempString[1]);

    if (j == id)
      downstream.add(new Integer(pkt.sourceId));
    else {
      if (dju + 0.2 > pkt.distance)
        BLOCK_LIST.add(new Integer(pkt.sourceId));
    }
  }

  public void receiveDATA(ReceivePkt pkt) {
    // check if data collide
    if (oldTime == Integer.parseInt(pkt.payload)) {
      if (oldTarget == id || pkt.targetId == id)
        Log.write(id + " Error: two pkt at the same time: " + oldSource + " " +
                  oldTarget + " " + oldTime + " " + pkt.sourceId + " " +
                  pkt.targetId + " " + pkt.payload);
    }

    oldTime = Integer.parseInt(pkt.payload);
    oldSource = pkt.sourceId;
    oldTarget = pkt.targetId;
  }

  public void sendData(int time) {
    if (time == timeSlot) {
      send(upstream, time + "", upstreamDistance, HIT.DATA, HIT.PACKET_LEN);
    }
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: NodeLEACH.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

public class NodeLEACH
    extends Node {
  double cost = Double.MAX_VALUE;

  public NodeLEACH(int id, double x, double y, Protocol protocol) {
    super(id, x, y, protocol);
  }

  public void reset() {
    super.reset();
    cost = Double.MAX_VALUE;
  }

  public boolean receive(ReceivePkt pkt) {
    if (pkt.type == LEACH.BASE_STATION_ADV )
      BS_Distance = pkt.distance;
    else if (pkt.type == LEACH.CH_ADV) {
      double newCost = pkt.distance;

      if ( (newCost < cost) && (isHead == false)) {
        cost = newCost;
        upstream = pkt.sourceId;
        upstreamDistance = pkt.distance;
        cluster = pkt.cluster;
      }
    }
    else if (pkt.type == LEACH.DATA)
      receiveDATA(pkt);

    return true;
  }

  public void receiveDATA(ReceivePkt pkt) {
    // check if data collide
    if (oldTime == Integer.parseInt(pkt.payload)) {
      if (oldTarget == id || pkt.targetId == id)
        Log.write(id + " Error: two pkt at the same time: " + oldSource + " " +
                  oldTime + " " + pkt.sourceId + " " + pkt.payload);
    }

    oldTime = Integer.parseInt(pkt.payload);
    oldSource = pkt.sourceId;
    oldTarget = pkt.targetId;
  }

  public void sendData(int time) {
    if (time == timeSlot) {
      send(upstream, time + "", upstreamDistance, LEACH.DATA, LEACH.PACKET_LEN);
    }
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: NodePEGASIS.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

public class NodePEGASIS
    extends Node {
  public boolean isConnected = false;
  public int prevNode, nextNode;
  public double prevDist, nextDist;

  public NodePEGASIS(int id, double x, double y, Protocol protocol) {
    super(id, x, y, protocol);
  }

  public void reset() {
    super.reset();
    isConnected = false;
  }

  public boolean receive(ReceivePkt pkt) {
    if (pkt.type == PEGASIS.BASE_STATION_ADV)
      BS_Distance = pkt.distance;
    else if (pkt.type == PEGASIS.DATA) {
      receiveDATA(pkt);
    }
    return false;
  }

  public void receiveDATA(ReceivePkt pkt) {
    // check if data collide
    if (oldTime == Integer.parseInt(pkt.payload)) {
      if (oldTarget == id || pkt.targetId == id)
        Log.write(id + " Error: two pkt at the same time: " + oldSource + " " +
                  oldTime + " " + pkt.sourceId + " " + pkt.payload);
    }

    oldTime = Integer.parseInt(pkt.payload);
    oldSource = pkt.sourceId;
    oldTarget = pkt.targetId;
  }

  public void sendData(int time) {
    if (time == timeSlot) {
      send(upstream, time + "", upstreamDistance, PEGASIS.DATA,
           PEGASIS.PACKET_LEN);
    }
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: PEGASIS.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

public class PEGASIS
    extends Protocol {
  NodePEGASIS [] nodes;
  NodePEGASIS [] heads;
  static final int BASE_STATION_ADV = 1;
  static final int DISCOVERY = 2;
  static final int DATA = 0;
  int headID = 0;
  int oldSize = 0;

  public PEGASIS(double[][] points, boolean stepByStep) {
    super();
    int i;

    display.name = new String("PEGASIS");
    display.f.setTitle("PEGASIS");

    this.stepByStep = stepByStep;

    nodes = new NodePEGASIS[points.length];
    for (i = 0; i < points.length; i++)
      nodes[i] = new NodePEGASIS(i, points[i][0], points[i][1], this);
    super.nodes = nodes;
    NAME = "PEGASIS";

    nodes[0].send( -1, "", 100000, BASE_STATION_ADV, 0);
    processSend();
    oldSize = 0;
  }

  public void setup() {
    int i, j = 0;

    display.clean();

    for (i = 0; i < nodes.length; i++) {
      nodes[i].downstream.clear();
    }

    if (aliveSet.size() != oldSize) {
      chainConstruction();
      oldSize = aliveSet.size();
    }

    /* start simulation */
    /* phase three : election of cluster head */
    Log.write("PEGASIS 3");

    nodes[headID].isHead = false;
    nodes[headID].upstream = -1;
    nodes[headID].upstreamDistance = Double.MAX_VALUE;
```

```
    do {
      headID++;
      if (headID == nodes.length)
        headID = 1;
    }
    while (!nodes[headID].isAlive);

    nodes[headID].isHead = true;
    nodes[headID].upstream = 0;
    nodes[headID].upstreamDistance = nodes[headID].BS_Distance;
    nodes[0].downstream.add(new Integer(headID));
    nodes[headID].timeSlot = 0;

    Log.write("node " + headID + " became the cluster head, distance to BS " +
              nodes[headID].upstreamDistance);
    j = 1;

    for (i = nodes[headID].prevNode; i != 0; i = nodes[i].prevNode) {
      nodes[i].upstream = nodes[i].nextNode;
      nodes[nodes[i].nextNode].downstream.add(new Integer(i));
      nodes[i].upstreamDistance = nodes[i].nextDist;
      nodes[i].isHead = false;
      nodes[i].timeSlot = j++;
    }

    for (i = nodes[headID].nextNode; i != 0; i = nodes[i].nextNode) {
      nodes[i].upstream = nodes[i].prevNode;
      nodes[nodes[i].prevNode].downstream.add(new Integer(i));
      nodes[i].upstreamDistance = nodes[i].prevDist;
      nodes[i].isHead = false;
      nodes[i].timeSlot = j++;
    }

    /* step four : data transmission */
    Log.write("PEGASIS 4");

//    printRouteTree();
    maxtime = aliveSet.size() - 1;

    Log.write(NAME + " : max delay = " + maxtime);
    Log.write(NAME + " : setup energy = " + setupEnergy);

    display.markAll(nodes);
    display.label.setText("PEGASIS ");
  }

  public double dist(NodePEGASIS n1, NodePEGASIS n2) {
    return Math.sqrt( ( (n1.x - n2.x) * (n1.x - n2.x)) +
                      ( (n1.y - n2.y) * (n1.y - n2.y)));
  }

  private void chainConstruction () {
    int i, j, lastid, minid = 0;
    int head = -1;
    double dist, minDist = Double.MAX_VALUE, maxDist = 0;
    int counter = 1;

    Log.write("construct chain");

    for (i = 0; i < nodes.length; i++) {
      nodes[i].reset();
```

```
    }

    /* phase one: find the furthest node */
    for (i = 1; i < nodes.length; i++) {
      dist = dist(nodes[i], nodes[0]);
      if (dist > maxDist && nodes[i].isAlive) {
        maxDist = dist;
        head = i;
      }
    }
    nodes[head].prevNode = 0;
    nodes[head].prevDist = nodes[head].BS_Distance;
    nodes[head].isConnected = true;
    counter++;

    Log.write("head = " + head);
    lastid = head;

    /* phase two: build the chain */
    while (counter < aliveSet.size()) {
      minDist = Double.MAX_VALUE;
      for (i = 1; i < nodes.length; i++) {
        if ( (lastid != i) && (!nodes[i].isConnected) && nodes[i].isAlive) {
          dist = dist(nodes[lastid], nodes[i]);
          if (dist < minDist) {
            minDist = dist;
            minid = i;
          }
        }
      }
      nodes[lastid].nextNode = minid;
      nodes[lastid].nextDist = minDist;
      nodes[minid].prevNode = lastid;
      nodes[minid].prevDist = minDist;
      nodes[minid].isConnected = true;
      lastid = minid;
      counter++;
    }
    Log.write("tail = " + lastid);
    nodes[minid].nextNode = 0;
    nodes[minid].nextDist = nodes[minid].BS_Distance;
  }

}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: Protocol.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;

// a protocol should hold the configuration / parameter applied to all node in a
protocol sense
// and initialize the action but not the exact behavior of a node (i.e., how the
protocol work.)

public abstract class Protocol
    implements Runnable {
  int tempDist;
  public boolean stepByStep = false;
  public Queue queue = new Queue();
  public Display display = new Display(Simulator.width, Simulator.height, "");
  public double Eelec = 5.0E-8, Eamp = 1.0E-10;
  public int subround = 0;

//  public double Eelec = 0.00000005 /*50*10^-9*/, Eamp = 0.0000000001 /*100*10^-12
*/;
  public int timer = 0;
  public int maxtime = 0;
  public ArrayList maxtimes = new ArrayList();
  public HashMap roundAlive = new HashMap();
  public String NAME = "";
  public double setupEnergy = 0;
  public HashSet aliveSet = new HashSet();
  public int fdie = -1, hdie = -1, ldie = -1;

  static public int ROUND = 1000;
  static public int PACKET_LEN = 100;

  Node nodes[]; // should be overrided
  Node heads[]; // should be overrided

  boolean wait = true;

  public Protocol() {
    int i;
    for (i = 0; i < Simulator.no_of_nodes; i++) {
      aliveSet.add(new Integer(i));
    }
  }

  public boolean processSend() {
    double dist;
    int i;

    while (queue.isEmpty() == false) {
      SendPkt spkt = queue.dequeue();

      if (nodes[spkt.sourceId].isAlive || spkt.sourceId == 0) {

        nodes[spkt.sourceId].energy -= Eelec * spkt.length +
```

```
    Eamp * spkt.length * spkt.distance * spkt.distance;

if (spkt.type > 0 && spkt.sourceId != 0) {
  setupEnergy += Eelec * spkt.length +
      Eamp * spkt.length * spkt.distance * spkt.distance;
}

if (nodes[spkt.sourceId].energy <= 0 && spkt.sourceId != 0) {
  nodes[spkt.sourceId].isAlive = false;
  nodes[spkt.sourceId].deadtime = Simulator.round;
  aliveSet.remove(new Integer(spkt.sourceId));
  Log.write("node " + spkt.sourceId + " dies at " + subround + ", " +
            aliveSet.size() + " alive");
  if (aliveSet.size() == Simulator.no_of_nodes - 2)
    fdie = subround;
  else if (aliveSet.size() == Simulator.no_of_nodes / 2)
    hdie = subround;
  else if (aliveSet.size() == 1)
    ldie = subround;
}

for (i = 0; i < nodes.length; i++) {
  dist = Math.sqrt( (nodes[i].x - nodes[spkt.sourceId].x) *
                   (nodes[i].x - nodes[spkt.sourceId].x) +
                   (nodes[i].y - nodes[spkt.sourceId].y) *
                   (nodes[i].y - nodes[spkt.sourceId].y));

  /* check if the node is fall in the range of the message */
  if ( (dist < spkt.distance + 0.1) && (i != spkt.sourceId) &&
      (nodes[i].isAlive || i == 0)) {
    ReceivePkt rpkt = new ReceivePkt(spkt.sourceId, spkt.targetId,
                                     spkt.payload, dist, spkt.type,
                                     spkt.length);

    //System.out.print(nodes[i].energy+ " ");
    nodes[i].energy -= Eelec * rpkt.length;

    if (rpkt.type > 0 && spkt.sourceId != 0) {
      setupEnergy += Eelec * rpkt.length;
    }

    if (nodes[i].energy <= 0 && i != 0) {
      nodes[i].isAlive = false;
      nodes[i].deadtime = Simulator.round;
      aliveSet.remove(new Integer(i));
      Log.write("node " + i + " dies at " + subround + ", " +
                aliveSet.size() + " alive");
      if (aliveSet.size() == Simulator.no_of_nodes - 2)
        fdie = subround;
      else if (aliveSet.size() == Simulator.no_of_nodes / 2)
        hdie = subround;
      else if (aliveSet.size() == 1)
        ldie = subround;
    }

    nodes[i].receive(rpkt);

    /**/
    if (stepByStep && spkt.type != 0) {
      display.clean();
      display.markAll(nodes);
```

```
            try {
              Thread.sleep(0);
            }
            catch (Exception e) {
              e.printStackTrace();
            }
          }
          /**/
        }
      }
    }
  }
  return true;
}

public void run() {
//    test1(); // TDMA schedule test
//    test2(100);  // energy used in one round
//    test3(100);  // energy used in one round (no fusion or comparess)
//    test4(100);  // send until die
//    test5(100);  // send until die (no fusion or comparess)
  setup();
}

public abstract void setup();

public void test1() {
  int time;
  int i, j;
  while (aliveSet.size() > 1) {
    Log.write("round " + Simulator.round);
    setup();
    for (j = 0; j < ROUND; j++) {
      for (time = 0; time < maxtime; time++) {
        for (i = 1; i < nodes.length; i++) {
          nodes[i].sendData(time);
        }
        processSend();
      }
      for (i = 0; i < nodes.length; i++) {
        nodes[i].oldSource = nodes[i].oldTarget = nodes[i].oldTime = -1;
      }
      subround++;
    }
    Simulator.round++;
  }
}

public double test2(int k) { // energy used in one round
  int i;
  double totalEnergy = 0;
  setup();

  for (i = 1; i < nodes.length; i++) {
    if (nodes[i].isHead) {
      totalEnergy += Eelec * k +
          Eamp * k * nodes[i].upstreamDistance * nodes[i].upstreamDistance ;
    }
    else {
      //transmit
      totalEnergy += Eelec * k +
```

```
          Eamp * k * nodes[i].upstreamDistance  * nodes[i].upstreamDistance ;
      //receive
      totalEnergy += Eelec * k;
    }
  }
  Log.write(NAME + " : energy used in one round = "  + totalEnergy);
  return totalEnergy;
}

public double test3(int k) { // cumulative energy in one round (no fusion or
comparess)
  int i;
  Node tempNode;
  double totalEnergy = 0;
  setup();

  for (i = 1; i < nodes.length; i++)
    for (tempNode = nodes[i]; tempNode.id != 0;
         tempNode = nodes[tempNode.upstream]) {
      if (tempNode.isHead) {
        totalEnergy += Eelec * k +
            Eamp * k * tempNode.upstreamDistance  * tempNode.upstreamDistance ;
      }
      else {
        //transmit
        totalEnergy += Eelec * k +
            Eamp * k * tempNode.upstreamDistance  * tempNode.upstreamDistance ;
        //receive
        totalEnergy += Eelec * k;
      }
    }

  Log.write(NAME + " : energy used in one round (no fusion) ="  + totalEnergy);

  return totalEnergy;
}

public void test4(int k) { //send until die
  PACKET_LEN = k;
  int i, j;
  int time;
  subround = 1;
  roundAlive.clear();
  while (aliveSet.size() > 1) {
    Log.write("round " + Simulator.round);
    setup();
    for (j = 0; j < ROUND; j++) {
      for (time = 0; time < maxtime; time++) {
        for (i = 1; i < nodes.length; i++) {
          if (nodes[i].isAlive) {

            nodes[i].energy -= Eelec * k +
                Eamp * k * nodes[i].upstreamDistance  *
                nodes[i].upstreamDistance ;

            if (nodes[i].energy <= 0 && i != 0) {
              nodes[i].isAlive = false;
              nodes[i].deadtime = subround;
              aliveSet.remove(new Integer(i));
              if (aliveSet.size() == Simulator.no_of_nodes - 2)
                fdie = subround;
```

```
            else if (aliveSet.size() == Simulator.no_of_nodes / 2)
              hdie = subround;
            else if (aliveSet.size() == 1)
              ldie = subround;
          }

          if (nodes[nodes[i].upstream].isAlive) {

            nodes[nodes[i].upstream].energy -= Eelec * k;

            if (nodes[nodes[i].upstream].energy <= 0 &&
                nodes[i].upstream != 0) {
              nodes[nodes[i].upstream].isAlive = false;
              nodes[nodes[i].upstream].deadtime = subround;
              aliveSet.remove(new Integer(nodes[i].upstream));
              if (aliveSet.size() == Simulator.no_of_nodes - 2)
                fdie = subround;
              else if (aliveSet.size() == Simulator.no_of_nodes / 2)
                hdie = subround;
              else if (aliveSet.size() == 1)
                ldie = subround;
            }
          }
        }
      }
    }
    if (subround % 100 == 0) {
      roundAlive.put(new Integer(subround), new Integer(aliveSet.size()));
    }
    subround++;
  }
  Simulator.round++;
}
Log.write("round=" + subround);
Log.write("f=" + fdie + ", h=" + hdie + ", l=" + ldie);

// print result in CSV format
System.out.print(NAME + ", ");
for (i = 0; i < roundAlive.size(); i++) {
  System.out.print(100 * (i + 1) + ", ");
}
System.out.println();
System.out.print(NAME + ", ");
for (i = 0; i < roundAlive.size(); i++) {
  System.out.print(roundAlive.get(new Integer(100 * (i + 1))) + ", ");
}
System.out.println();

}

public void test5(int k) { //send until die no fusion
  int i, j;
  Node tempNode;
  double totalEnergy = 0;
  PACKET_LEN = k;
  int time;
  subround = 1;
  roundAlive.clear();

  while (aliveSet.size() > 1) {
    Log.write("round " + Simulator.round);
```

```
  setup();
  for (j = 0; j < ROUND; j++) {
    for (time = 0; time < maxtime; time++) {
      for (i = 1; i < nodes.length; i++) {
        tempNode = nodes[i];
        while (tempNode.id != 0 && tempNode.isAlive) {

          tempNode.energy -= Eelec * k +
              Eamp * k * tempNode.upstreamDistance *
              tempNode.upstreamDistance;

          if (tempNode.energy <= 0) {
            tempNode.isAlive = false;
            tempNode.deadtime = subround;
            aliveSet.remove(new Integer(tempNode.id));
            Log.write("node " + tempNode.id + " dies at " + subround +
                      ", " + aliveSet.size() + " alive");
            if (aliveSet.size() == Simulator.no_of_nodes - 2)
              fdie = subround;
            else if (aliveSet.size() == Simulator.no_of_nodes / 2)
              hdie = subround;
            else if (aliveSet.size() == 1)
              ldie = subround;
          }
          if (nodes[tempNode.upstream].isAlive) {

            nodes[tempNode.upstream].energy -= Eelec * k;

            if (nodes[tempNode.upstream].energy <= 0 &&
                nodes[tempNode.upstream].id != 0) {
              nodes[tempNode.upstream].isAlive = false;
              nodes[tempNode.upstream].deadtime = subround;
              aliveSet.remove(new Integer(tempNode.upstream));
              Log.write("node " + tempNode.upstream + " dies at " +
                        subround + ", " + aliveSet.size() + " alive");
              if (aliveSet.size() == Simulator.no_of_nodes - 2)
                fdie = subround;
              else if (aliveSet.size() == Simulator.no_of_nodes / 2)
                hdie = subround;
              else if (aliveSet.size() == 1)
                ldie = subround;
            }
          }
          tempNode = nodes[tempNode.upstream];
        }
      }
    }
    if (subround % 100 == 0) {
      roundAlive.put(new Integer(subround), new Integer(aliveSet.size()));
    }
    subround++;
  }
  Simulator.round++;
}
Log.write("round=" + subround);
Log.write("f=" + fdie + ", h=" + hdie + ", l=" + ldie);

// print result in CSV format
System.out.print(NAME + ", ");
for (i = 0; i < roundAlive.size(); i++) {
  System.out.print(100 * (i + 1) + ", ");
```

```
    }
    System.out.println();
    System.out.print(NAME + ", ");
    for (i = 0; i < roundAlive.size(); i++) {
      System.out.print(roundAlive.get(new Integer(100 * (i + 1))) + ", ");
    }
    System.out.println();
  }

  public void printRouteTree() {
    HashSet pending = new HashSet();
    Log.write("--- route tree ---" );
    printRouteTree(nodes[0], 0, pending, true);
  }

  public void printRouteTree(Node n, int level, HashSet pending, boolean last) {
    int i, j;
    int id;
    String tempString = "";
    HashSet downstream = n.downstream;
    Iterator iterator = downstream.iterator();

    /* expended mode, takes up more lines but easier to read * /
    for (j = 0; j < level; j++) {
      if (pending.contains(new Integer(j)))
        System.out.print("|");
      else
        System.out.print(" ");
      System.out.print("\t");

    }
    System.out.println( (last ? "\\" : "+") + "-------" + n.id);

    pending.add(new Integer(level + 1));

    for (i = 0; i < n.downstream.size(); i++) {
      if (i == n.downstream.size() - 1) pending.remove(new Integer(level + 1));
      printRouteTree(nodes[ ( (Integer) iterator.next()).intValue()], level + 1,
                     pending, (i == n.downstream.size() - 1));
    }
    /**/
    /* compressed mode, takes less lines but more difficult to read */
    System.out.print( ( (n.downstream.size() > 1) ? "+" : " ") + n.id + "\t");

    for (i = 0; i < n.downstream.size(); i++) {
      if (i == 0) pending.add(new Integer(level));
      if (i == n.downstream.size() - 1) pending.remove(new Integer(level));
      printRouteTree(nodes[ ( (Integer) iterator.next()).intValue()],
                     level + 1,
                     pending, false);
      if (i < n.downstream.size() - 1) {
        for (j = 0; j < level; j++) {
          if (pending.contains(new Integer(j)))
            System.out.print("| ");
          else
            System.out.print("  ");
          System.out.print("\t");

        }
        System.out.print( ( (i == n.downstream.size() - 2) ? "\\" : "+") +
                         "-------");
```

```
      }
    }
    if (n.downstream.size() == 0) {
      System.out.println();
    }
    /**/
  }

  public void printSchedule() {
    int i, j, n;
    System.out.println("schedule");
    for (i = 0; i < maxtime; i++) {
      n = 0;
      System.out.print("slot " + i + " : ");
      for (j = 1; j < nodes.length; j++) {
        if (nodes[j].timeSlot == i) {
          n++;
          System.out.print(j + ", ");
        }
      }
      System.out.println(" Total: " + n);
    }

    /* CSV format */
    for (i = 0; i < maxtime; i++)
      System.out.print(i + ", ");
    System.out.println();
    for (i = 0; i < maxtime; i++) {
      n = 0;
      for (j = 1; j < nodes.length; j++) {
        if (nodes[j].timeSlot == i) {
          n++;
        }
      }
      System.out.print(n + ", ");
    }
    System.out.println();
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: Queue.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;

public class Queue {

  public Vector msgQueue = new Vector();

  public Queue() {
  }

  public boolean enqueue(SendPkt pkt) {
    msgQueue.add(pkt);
    return true;
  }

  public SendPkt dequeue() {
    return (SendPkt) msgQueue.remove(0);
  }

  public boolean isEmpty() {
    return (msgQueue.size() == 0 ? true : false);
  }
}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: ReceivePkt.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

public class ReceivePkt {
  public String payload;
  public int sourceId;
  public int targetId;
  public double distance;
  public int cluster;
  public int type;
  public int length;

  public ReceivePkt (int sourceId, int targetId, String payload, double distance,
                     int type, int length) {
    this.sourceId = sourceId;
    this.targetId = targetId;
    this.payload = new String(payload);
    this.distance = distance;
    this.type = type;
    this.length = length;
  }

  public String toString() {
    return ("node " + sourceId + " send to node "  + targetId + " : " + payload);
  }

}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: SendPkt.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

public class SendPkt {
  public String payload;
  public int sourceId;
  public int targetId;
  public double distance;
  public int cluster;
  public int type;
  public int length;

  public SendPkt(int sourceId, int targetId, String payload, double distance,
                 int type, int length) {
    this.sourceId = sourceId;
    this.targetId = targetId;
    this.payload = new String(payload);
    this.distance = distance;
    this.type = type;
    this.length = length;
  }

  public String toString() {
    return ("node " + sourceId + " send to node "  + targetId + " : " + payload);
  }

}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: Simulator.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import java.util.*;
import java.awt.*;

public class Simulator {
  static public int no_of_nodes;
  static public int width, height;
  static public int clusterHeadPercent ;
  static public int round = 0;
  static public Point baseStationPt ;

  public ArrayList protocols = new ArrayList();
  public ArrayList threads = new ArrayList();

//  public Point[] points;
  // public int[][] points;
  public int pointSet = 1;

  public CMPE cmpe;
  public HIT hit;
  public LEACH leach;
  public CMPE cmpe2;
  public CMPE cmpe3;
  public CMPE cmpe4;
  public PEGASIS pegasis;
  public Direct direct;

  public Simulator(int no_of_nodes, int width, int height,
                   int clusterHeadPercent ) {
    no_of_nodes++; // plus one base station
    this.no_of_nodes = no_of_nodes;
    this.width = width;
    this.height = height;
    this.clusterHeadPercent = clusterHeadPercent;

    int i;

    double[][] points;

    Log.write("setup");
    Random random = new Random();
    Random random1 = new Random(random.nextLong());
    Random random2 = new Random(random.nextLong());

    /* set up of node in simulator("placement" of sensor nodes) */

    // nodes used for CMPE routing example
    // points = new double[][] {{50, -100}, {35,30},{13, 23}, {28,8}, {43,65},{30,
76},{73,64},{86,48},{93,20},{50,50}};

    // nodes used for CMPE TDMA example
    // points = new double[][] {{50, -100},{41,25},{53,38},{30,55},{63,42},{55,59}
,{60,83},{70,78},{0,0},{43,13}};
```

```
    /* nodes used for LEACH, PEGASIS, and HIT examples
    points = new double[][] {{50,-200},
                              {69,5}, {76,94}, {97,97}, {29,93}, {13,49}, {37,10}, {51  Ã
,40}, {82,39}, {45,32}, {70,42},
                              {15,61}, {88,79}, {47,39}, {72,48}, {69,22}, {94,80},     Ã
{34,96}, {19,39}, {11,2}, {54,95},
                              {4,61}, {64,33}, {42,28}, {15,75}, {26,34}, {37,43}, {56  Ã
,85}, {14,15}, {10,14}, {12,78},
                              {64,35}, {96,23}, {84,11}, {1,52}, {51,30}, {40,11}, {82  Ã
,78}, {44,42}, {72,86}, {67,0},
                              {57,9}, {75,45}, {12,59}, {46,37}, {12,87}, {4,53}, {84   Ã
,61}, {15,83}, {4,22}, {79,66},
                              {77,24}, {97,33}, {47,20}, {40,51}, {26,15}, {70,88},     Ã
{18,40}, {56,56}, {85,21}, {2,39},
                              {54,97}, {35,99}, {55,72}, {30,7}, {86,34}, {97,75}, {14  Ã
,14}, {43,98}, {7,64}, {97,90},
                              {52,70}, {82,14}, {38,5}, {4,18}, {86,89}, {45,1}, {82    Ã
,97}, {98,19}, {16,72}, {15,11},
                              {31,70}, {25,99}, {29,77}, {49,7}, {59,31}, {15,54}, {32  Ã
,34}, {5,11}, {16,46}, {89,17},
                              {61,41}, {5,8}, {11,24}, {82,51}, {93,70}, {13,27}, {87   Ã
,85}, {0,56}, {61,75}, {61,43}};
    */
    /* random placement */
    points = new double[no_of_nodes][2];

    points[0][0] = width / 2;
    points[0][1] = -height * 2;

    if (points[0][1] < -500) points[0][1]=-500;

    for (i = 1; i < no_of_nodes; i++) {
      points[i][0] = random1.nextDouble() * width;
      points[i][1] = random2.nextDouble() * height;
    }
    /* end random placement */

    /* grid placement * /
    double w = width / (2*Math.sqrt(no_of_nodes)), h = height / (2*Math.sqrt(no_of_no  Ã
des));
    points = new double[no_of_nodes][2];

    points[0][0] = width / 2;
    points[0][1] = -height * 2;

    if (points[0][1] < -500) points[0][1]=-500;

    for (i = 1; i < no_of_nodes; i++) {
      points[i][0] = w;
      points[i][1] = h;
      w += width / Math.sqrt(no_of_nodes);
      if (w >= width) {
        w = width / (2*Math.sqrt(no_of_nodes));
        h += height / Math.sqrt(no_of_nodes);
        if (h >= height)
          h = height / (2*Math.sqrt(no_of_nodes));
      }
    }
    /* end grid placement */
```

```java
    // print the position of the nodes
    System.out.println("{{" + points[0][0] + "," + points[0][1] + "},");
    for (i = 1; i < points.length - 1; i++) {
      System.out.print(" {" + points[i][0] + "," + points[i][1] + "},");
      if (i % 10 == 0)
        System.out.println();
    }
    System.out.println(" {" + points[i][0] + "," + points[i][1] + "}};");

    /* uncomment the code to add protocol(s) */
    protocols.add(cmpe = new CMPE(points, false, 1, 1));
//    protocols.add(cmpe2 = new CMPE(points, false, 2, 1));
//    protocols.add(cmpe3 = new CMPE(points, false, 3, 1));
//    protocols.add(cmpe4 = new CMPE(points, false,4, 1));
    protocols.add(hit = new HIT(points, false));
    protocols.add(leach = new LEACH(points, false));
    protocols.add(pegasis = new PEGASIS(points, false));
//    protocols.add(direct = new Direct(points, false));

    /* set the location of the window of the display */
    for (i = 0; i < protocols.size(); i++) {
      switch (i) {
        case 0:
           ( (Protocol) protocols.get(i)).display.f.setLocation(130, 0);
          break;
        case 1:
           ( (Protocol) protocols.get(i)).display.f.setLocation(130, 250);
          break;
        case 2:
           ( (Protocol) protocols.get(i)).display.f.setLocation(350, 0);
          break;
        case 3:
           ( (Protocol) protocols.get(i)).display.f.setLocation(350, 250);
          break;
        case 4:
           ( (Protocol) protocols.get(i)).display.f.setLocation(570, 0);
          break;
        case 5:
           ( (Protocol) protocols.get(i)).display.f.setLocation(570, 250);
          break;
      }
    }
  }

  public void startSim() {
    round++;

    Log.write("Simulation started" );

    for (int i = 0; i < protocols.size(); i++) {
      Protocol protocol = (Protocol) protocols.get(i);
      Thread thread = new Thread(protocol, protocol.NAME);
      threads.add(thread);
      thread.start();
    }
  }

  public void start() {
  }

  public void stop() {
```

```
  }

  public void draw() {
    for (int i = 0; i < protocols.size(); i++) {
      Protocol protocol = (Protocol) protocols.get(i);
      protocol.display.markAll(protocol.nodes);
    }
  }

  public void notice() {
    cmpe.wait = false;
  }

  public void pause() {
  }

  static public void main(String arg[]) {
    Simulator sim;

    if (arg.length != 4)
      sim = new Simulator(100, 100, 100, 5);
    else
      sim = new Simulator(Integer.parseInt(arg[0]),
                          Integer.parseInt(arg[1]),
                          Integer.parseInt(arg[2]),
                          Integer.parseInt(arg[3]));

    sim.startSim();
    return;
  }

}
```

```
/**
 * <p>Title: Cluster-Management & Power-Efficient Protocol for WSN</p>
 * <p>Description: UI.java</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * @author Shen Ben Ho
 */

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class UI  extends JFrame {
  Simulator sim;
  JButton ButStart = new JButton();
  JButton ButNext = new JButton();
  JButton ButExit = new JButton();
  JButton ButDraw = new JButton();
  JButton ButTest = new JButton();
  JLabel jLabel1 = new JLabel();

  public UI() {
    try {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String[] args) {
    UI ui = new UI();
    ui.setVisible(true);
  }

  private void jbInit() throws Exception {
    setTitle("Applet Frame");
    this.addWindowListener(new UI_this_windowAdapter(this));
    setSize(100, 200);

    ButStart.setBounds(new Rectangle(9, 9, 73, 27));
    ButStart.setVerifyInputWhenFocusTarget(true);
    ButStart.setActionCommand("Start");
    ButStart.setText("Start");
    ButStart.addActionListener(new UI_ButStart_actionAdapter(this));
    this.getContentPane().setLayout(null);
    ButNext.setBounds(new Rectangle(9, 42, 73, 27));
    ButNext.setActionCommand("Next");
    ButNext.setText("Next");
    ButNext.addActionListener(new UI_ButNext_actionAdapter(this));
    ButExit.setBounds(new Rectangle(10, 142, 73, 27));
    ButExit.setActionCommand("Exit");
    ButExit.setText("Exit");
    ButExit.addActionListener(new UI_ButExit_actionAdapter(this));
    ButDraw.setBounds(new Rectangle(9, 75, 73, 27));
    ButDraw.setToolTipText("");
    ButDraw.setActionCommand("Draw");
    ButDraw.setText("Draw");
    ButDraw.addActionListener(new UI_ButDraw_actionAdapter(this));
    ButTest.setBounds(new Rectangle(10, 109, 73, 27));
```

```java
    ButTest.setActionCommand("Test");
    ButTest.setText("Test");
    ButTest.addActionListener(new UI_ButTest_actionAdapter(this));
    jLabel1.setText("jLabel1");
    jLabel1.setBounds(new Rectangle(12, 179, 34, 17));
    this.getContentPane().add(ButStart, null);
    this.getContentPane().add(ButNext, null);
    this.getContentPane().add(ButDraw, null);
    this.getContentPane().add(ButExit, null);
    this.getContentPane().add(ButTest, null);
    this.getContentPane().add(jLabel1, null);
  }

  void ButStart_actionPerformed(ActionEvent e) {
    /* Simulator(no_of_nodes, width, height, clusterHeadPercent) */
//        sim = new Simulator(10, 10, 10, 10);
//        sim = new Simulator(10, 100, 100, 10);
//        sim = new Simulator(100, 10, 10, 1);
//        sim = new Simulator(100, 100, 100, 5);
        sim = new Simulator(100, 1000, 1000, 5);
//        sim = new Simulator(100, 500, 500, 5);
//        sim = new Simulator(1000, 1000, 1000, 10);
  }

  void ButNext_actionPerformed(ActionEvent e) {
    sim.startSim();
  }

  void ButExit_actionPerformed(ActionEvent e) {
    System.exit(0);
  }

  void ButDraw_actionPerformed(ActionEvent e) {
    sim.draw();
  }

  void this_windowClosing(WindowEvent e) {
    System.exit(0);
  }

  void ButTest_actionPerformed(ActionEvent e) {
    sim.notice();
  }
}

class UI_ButStart_actionAdapter
    implements java.awt.event.ActionListener {
  UI adaptee;

  UI_ButStart_actionAdapter(UI adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.ButStart_actionPerformed(e);
  }
}

class UI_ButNext_actionAdapter
    implements java.awt.event.ActionListener {
  UI adaptee;
```

```
  UI_ButNext_actionAdapter (UI adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.ButNext_actionPerformed(e);
  }
}

class UI_ButExit_actionAdapter
    implements java.awt.event.ActionListener {
  UI adaptee;

  UI_ButExit_actionAdapter (UI adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.ButExit_actionPerformed(e);
  }
}

class UI_ButDraw_actionAdapter
    implements java.awt.event.ActionListener {
  UI adaptee;

  UI_ButDraw_actionAdapter (UI adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.ButDraw_actionPerformed(e);
  }
}

class UI_this_windowAdapter
    extends java.awt.event.WindowAdapter {
  UI adaptee;

  UI_this_windowAdapter (UI adaptee) {
    this.adaptee = adaptee;
  }

  public void windowClosing(WindowEvent e) {
    adaptee.this_windowClosing(e);
  }
}

class UI_ButTest_actionAdapter
    implements java.awt.event.ActionListener {
  UI adaptee;

  UI_ButTest_actionAdapter (UI adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.ButTest_actionPerformed(e);
  }
}
```

Date: Wed, 21 Apr 2004 00:26:31 +0100
From: Seth Hollar <shollar@alum.mit.edu>
To: sho@email.sjsu.edu
Subject: Re: Requesting permission to reprint a figure

Dear Shen-Ben,

You have full permission to reprint the figures in your master's thesis. Thank you for letting me know.
Best regards,
Seth Hollar

At 07:27 PM 4/15/2004, you wrote:
>Dear Mr. Hollar,
>
>I am a graduate student of Computer Engineering Department of San Jos
>University. I am writing to request your permission to reprint a figu
>master thesis. The figure interested is described below:
>
>The figure of Mini Mote
>(http://www-bsac.eecs.berkeley.edu/archive/users/hollar-seth/macro_mo
>in the webpage Cots Dust
>(http://www-bsac.eecs.berkeley.edu/archive/users/hollar-seth/macro_mo
>
>I would appreciate a letter expressing the appropriate permission.
>
>Thank you for your assistance.
>
>
>Sincerely,
>
>Shen-Ben Ho
>
>P.O. Box 720074
>San Jose, CA 95172
>
>(P.S. Sorry for sending you email as I could not find your contact ad
>
>
>--
>San Jose State University IMP server -- https://web-mail.sjsu.edu/hor

Date: Thu, 03 Jun 2004 10:18:56 -0700
From: Kevin Delin <kevin.delin@jpl.nasa.gov>
To: sho@email.sjsu.edu
Subject: Re: Request for Permission to Reprint a Figure

Dear Shen,

You have permission to reprint the figure discussed below for your thesis
with appropriate attribution. I would like to get a final copy of your

thesis (either hard copy mailed to the address below or a soft PDF copy emailed to me).

I have been on extensive travel for the past two months and unfortunately
in your original letter you didn't leave an email address (only a PO box)
which delayed my response.

Thank you for following good academic practices in the use of your source
material.

Sincerely,

Kevin Delin

At 09:36 AM 6/3/2004 -0700, you wrote:
>Dear Dr. Delin
>
>Referring to my letter dated May 24, 2004 on the above subject, please be
>advised that the deadline for my submission of thesis is June 4, 2004. I have
>attached a sample of the permission letter for your easy reference.
>
>Thank you for your help.
>
>Shen Ben Ho
>
>--
>
>Dear Mr. Shen Ben Ho,
>
>Referring to your letter requesting to reprint the following figure in you
>master thesis,
>
>The figure of Sensor Webs (the figure on the left of
>http://sensorwebs.jpl.nasa.gov/
resources/images/briefingSlide1_middle.jpg) in
>the webpage JPL Sensor Webs Project Briefing (http://sensorwebs.jpl.
>nasa.gov/resources/briefing1.shtml).
>
>I hereby grant you the permission to reprint the figure.

>
>Yours sincerely,
>
>
>---
>San Jose State University IMP server -- https://web-mail.sjsu.edu/horde

Dr. Kevin A. Delin
Manager, NASA/JPL Sensor Webs Project (http://sensorwebs.jpl.nasa.gov)

Jet Propulsion Laboratory
4800 Oak Grove Drive, M/S 306-336
Pasadena, CA 91109-8099

Tel: (818) 354-9647
Email: kevin.delin@jpl.nasa.gov

UNIVERSITY OF
ROCHESTER
THE COLLEGE
ARTS • SCIENCES • ENGINEERING

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

Wendi Heinzelman
University of Rochester
Box 270126
Rochester, NY 14627-0126

Dear Shen-Ben Ho,

I am writing this letter to give you permission to reproduce Figure 1 (First order radio model) from our paper titled, "Communication Protocol for Wireless Microsensor Networks," which appeared in the Proceedings of the Hawaii International Conference on System Sciences, in your Master's Thesis. Please add an appropriate reference to the figure.

Sincerely,

Wendi Heinzelman

University of Rochester
P.O. Box 270231
Rochester, New York 14627-0231
(585) 275-4060

Dear Mr. Shen Ben Ho,

Referring to your letter requesting to reprint the following figures in you master thesis,

Figure 2-1 (photo of Mica2), figure 3-1 (photo of Mica2dot) and figure 6.1 (photo of MIB510CA) in the "MPR - Mote Processor Radio Board, MIB - Mote Interface / Programming Board User's Manual";

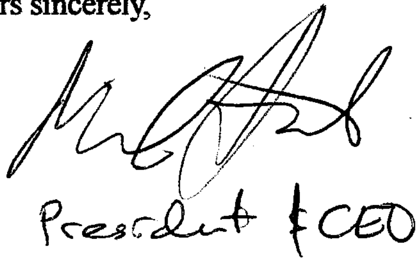
The block diagram of Mica2 in Mica2 DataSheet; and

The block diagram of Mica2dot in Mica2dot DataSheet

I hereby grant you the permission to reprint the figure.

Please reference Crossbow Technology www.xbow.com

Yours sincerely,



Information Sciences Institute

Divisions

Advanced Systems

Computational Sciences

Computer Networks

Distributed Scalable Systems

Dynamic Systems

Information Processing Center

Integration Sciences

Intelligent Systems

Silicon Systems (MOSIS)

April 16, 2004

Shen-Ben Ho
P.O. Box 720074
San Jose, CA 95172

Dear Shen-Ben,

Re: Permission to reprint a figure

Thanks for your request for the permission to reprint Figure 1 from my paper entitled "Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks." in your master's thesis. I'm happy to grant this permission.

Sincerely,

Wei Ye
Computer Scientist

University of Southern California
USC Viterbi School of Engineering
4676 Admiralty Way
Suite 1001
Marina Del Rey,
California 90292-6601
Tel: 310 822 1511
Fax: 310 823 6714

Date: Thu, 3 Jun 2004 17:05:28 -0700
From: Pottie-icsl <pottie@icsl.ucla.edu>
To: sho@email.sjsu.edu
Subject: RE: Request for Permission to Reprint a Figure

Dear Mr. Shen Ben Ho,

Referring to your letter requesting to reprint the following figure in your master's thesis,

Figure 2 (Link-layer self-organizing procedures) in the paper titled "Protocols for Self-Organization of a Wireless Sensor Network" in "IEEE Personal Communications".

Of course you have my permission to reprint the figure. Please accept my apologies for the tardiness in responding.

Regards,

Gregory J Pottie
Professor, UCLA EE Department