Location Aware Routing Protocols for Mobile Adhoc Networks in TinyOS

Developed by

Salma Basharat
Saadia Khan

Supervised by

Dr. Shoab Ahmad Khan
Dr. M. Sikander Hayat Khiyal

Department of Computer Sciences
Faculty of Applied Sciences
International Islamic University Islamabad

2007

بسم الله الرحمن الرحيم

A dissertation submitted to the

Department of Computer Science,

Faculty of Applied Sciences,

International Islamic University, Islamabad, Pakistan,

as a partial fulfillment of the requirements for the award of the degree of

MS Computer Science

To
The Holiest Man Ever Born,
Prophet Muhammad (صلى الله عليه وسلم)
&
To
Our Parents and Families
We are most indebted to our parents and siblings, whose affection has always been the source of encouragement for us, and whose prayers have always been a key to our success.
&
To
Those Holy Seekers
Who give away their lives to make the stream of life flow
Smoothly and with Justice.
&
To
Our Honorable Teachers
Who have been a beacon of knowledge and a constant source of inspiration, for our whole life span.

Declaration

We hereby declare and affirm that this software, neither as a whole nor as a part. thereof has been copied out from any source. It is further declared that we have developed this software and accompanied thesis entirely on the basis of our personal efforts, made under the sincere guidance of our teachers. If any part of this project is proven to be copied out or found to be a reproduction, we shall stand by the consequences.

No portion of the work presented in this thesis has been submitted in support of an application for other degree or qualification of this or any other University or Institute of learning.

Salma Basharat
221-CS/MSCS/F04

Saadia Khan

220-CS/MSCS/F04

Acknowledgement

We bestow all praises to, acclamation and appreciation to Almighty Allah, The Most Merciful and Compassionate, The Most Gracious and Beneficent, Whose bounteous blessings enabled us to pursue and perceive higher ideals of life, Who bestowed us good health, courage, and knowledge to carry out and complete our work. Special thanks to our Holy Prophet Muhammad (SAW) who enabled us to recognize our Lord and Creator and brought us the real source of knowledge from Allah (SWT), the Quran, and who is the role model for us in every aspect of life.

We pay our heartiest gratitude to our supervisor **Dr Shoab Ahmad Khan** who has always been a source of inspiration for us both technically and morally. We owe a lot to him and Insha Allah we would take his advices forward in our professional life. We can never repay what he has taught us but we have prayers which would always thank him for his endless support.

We express our heartiest and deepest gratitude to our supervisor **Dr. Malik Sikander Hayat Khayal** who kept our morale high by his suggestions and appreciation. His motivation led us to this success. Without his sincere and cooperative nature and precious guidance; we could never have been able to complete this task.

Finally we must mention that it was mainly due to our parent's moral support and financial help during our entire academic career that enabled us to complete our work dedicatedly. We owe all our achievements to our most loving parents, who mean most to us, for their prayers are more precious before any treasure on the earth. We are also thankful to our loving brothers, sisters, friends, and class fellows who mean a lot to us and their prayers have always been a source of determination for us.

Salma Basharat
221-CS/MSCS/F04
Saadia Khan
220-CS/MSCS/F04

International Islamic University Islamabad
Faculty of Applied Sciences
Department of Computer Sciences

Dated: 16-04-2007

FINAL APPROVAL

It is certified that we have read the project report, titled **Location Aware Routing Protocols for Mobile Ad hoc Networks in TinyOS** submitted by **Salma Basharat Reg. No. 221-FAS/MS/CS/F04** and **Saadia Khan Reg. No. 220-FAS/MS/CS/F04**. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by International Islamic University, Islamabad, for the MS Degree of Computer Sciences.

PROJECT EVALUATION COMMITTEE

External Examiner

Dr. Abdus Sattar,
House No. 143, Street No. 60,
I-8/3, Islamabad

[illegible]

[illegible]

Internal Examiner

Dr. Syed Afaaq Hussain
Head Dept. of Telecommunication Sciences,
International Islamic University, Islamabad

[illegible]

Supervisor

Dr. Malik Sikander Hayat Khiyal
Head Computer Science Department
Islamic International University Islamabad

[illegible]

Supervisor

Dr. Shoab Ahmad Khan
Faculty Member,
Centre of Advance Studies and Engineering (CASE),
Sir Syed Memorial Centre, Islamabad

[illegible]

PROJECT IN BRIEF

Project Title:	**Location Aware Routing Protocols for Mobile Ad hoc networks in TinyOS**
Organization	**International Islamic University, Islamabad**
Under Taken By:	**Salma Basharat** **Reg. No# 221/CS/MSCS/F04** **Saadia Khan** **Reg. No# 220/CS/MSCS/F04**
Supervised By:	**Dr. Malik Sikander Hayat Khiyal** **Head Computer Science Department** **Islamic International University Islamabad** **Dr. Shoab Ahmad Khan** **Centre of Advance Studies and Engineering (CASE), Islamabad**
Starting Date:	**May, 2006.**
End Date:	**Feb, 2007.**
Tools used:	**TinyOS**
System Used	**Pentium IV**

ABSTRACT

Wireless networks are getting importance day by day but the design of ad hoc routing protocols is very critical now days. Number of routing protocols has been developed so far. Besides the two famous categories of mobile ad hoc networks i.e. proactive and reactive routing schemes, there is a new division of routing algorithms on the basis of geographical location. With respect to this, there is a category of localized routing algorithm and non localized routing algorithms. Researches have proved that localized routing protocols are more beneficial in terms of packets sent and received, energy utilization, number of hops , efficient bandwidth utilization etc. This is only due to the fact that in localized routing algorithms, packets are sent and received in a particular direction hence limiting the search space of the network. Physical location of the nodes is achieved by some external means like GPS or other position based hardware device.

Other key aspect about design of routing protocols is to utilize energy of the nodes in efficient manner. In wireless sensor networks especially, the data is usually sent to a control station i.e. a sink node. Sink node acts as a data aggregator and shoulders maximum of the network load. Therefore saving energy of the sink node is very critical because if the sink node will collapse then entire network will crash. Since localized algorithm like Location Aided Routing (LAR) performs only directional flooding hence maximum energy is utilized which is an added advantage.

Normal operating systems may provide simple threading, memory and variable space, hardware resources etc but for wireless sensor networks and mobile ad hoc networks there was a dire need for such an operating system which can cater large number of nodes in a network, can accommodate lengthy calculation of the algorithms and properly simulate the hardware resources. For this purpose, TinyOS was implemented by University of California, Berkeley in cooperation with Intel. It is a component based event driven operating system and provides a modular abstraction of hardware resources. It provides an emulator, PowerTOSSIM, which is the only available emulator which can emulate a large sensor network. It calculates approximately the actual power consumption by each node in terms of hardware attached like LED's, EEPROM and Sensors etc.

To the best of out knowledge, no such comparison was made so far between localized and non localized routing algorithms in TinyOS using PowerTOSSIM as a simulation tool. We have compared the performance of both the routing protocols in terms of energy utilization and found that localized routing algorithm, LAR, conserves less energy as compared to the non localized routing algorithm, DSDV.

Table of Contents

Chapter 1

Introduction

1. Introduction

This chapter briefly discuses wireless networks and its types. Besides this, the chapter also contains some information about MANETS routing protocols and their subdivisions. Energy conservation, a critical issue in MANETS and Wireless sensor networks (WSN), is discussed as a next section. Lastly the operating system for Ad hoc networks, TinyOS, and the associated simulator are discussed.

1.1. Wireless Networks

Wireless networks have become increasingly popular since 1970. There are currently two variations of mobile wireless networks. The first is known as the infra-structured network. Wireless local area network based on IEEE 802.11 technology is the most prevalent infra-structured mobile network, where a mobile node communicates with a fixed base station, and thus a wireless link is limited to one hop between the node and the base station. In this type of network, a mobile unit connects and communicates with, the nearest base station that is within its communication radius. A "handsoff" occurs when the base station goes out of range and mobile unit is not able to communicate to the old base station. Typical applications of this type of network include office wireless local area networks (WLANs).

The second type of mobile wireless network is the infrastructure-less mobile network, commonly known as Mobile Ad hoc Network. They have no fixed routes; all nodes are capable of movement and can be connected dynamically in an arbitrary manner without any radio contact. Self route discovery and maintenance to other nodes takes place. Packet forwarding takes place in a multi-hop fashion. Example applications of ad hoc networks are emergency search-and-rescue operations, meetings or conventions in which persons wish to quickly share information, and data acquisition operations in inhospitable terrain. Fig.1-1 shows the two types of mobile wireless networks [22].
Advances in wireless communications, and wireless communication have made it possible to produce large amount of small-size, low-cost sensors which integrate sensing, processing, and communication capabilities together and form an autonomous entity. Large amount of these sensors can be quickly deployed in the field, where each sensor independently senses the environment but collaboratively achieves complex information

gathering and dissemination tasks like intrusion detection, target tracking, environmental monitoring, remote sensing, global surveillance, etc.

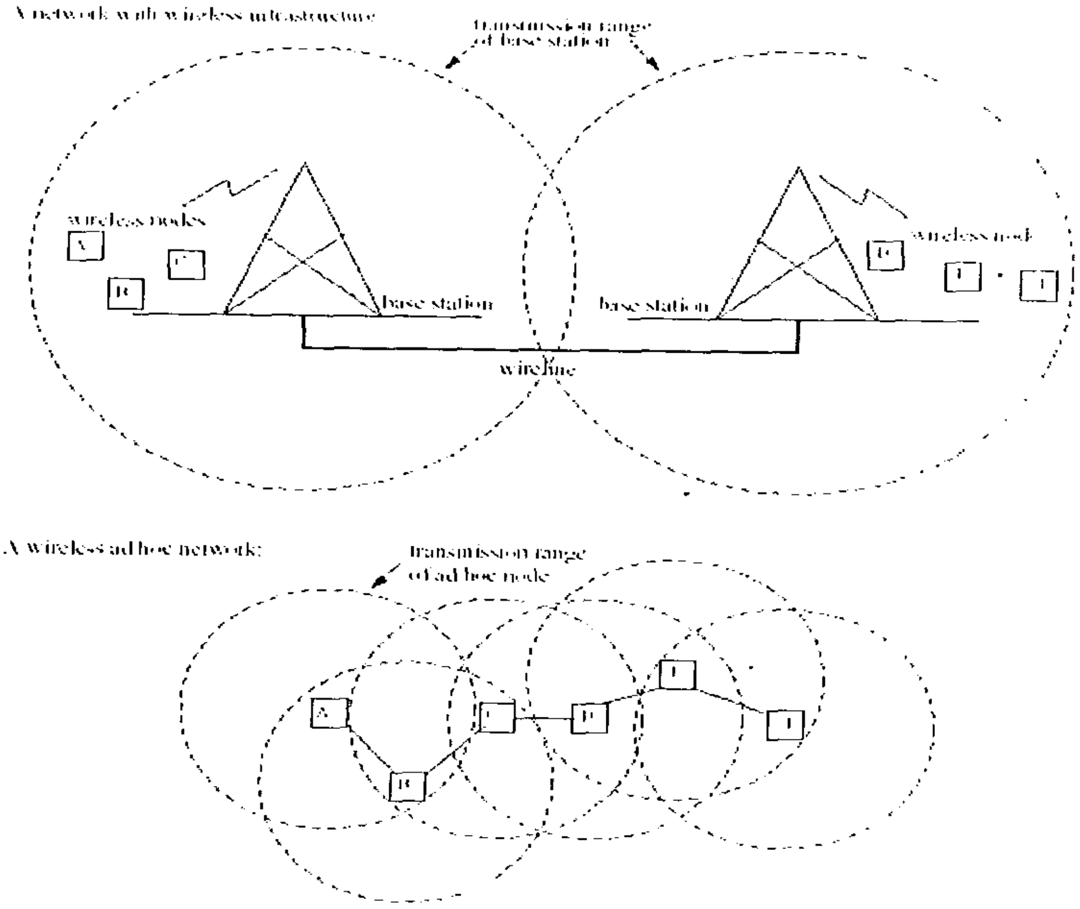

A network with wireless infrastructure
transmission range of base station
wireless nodes
base station
base station
wireless node
wireline
A wireless ad hoc network:
transmission range of ad hoc node

Figure 1-1: Types of Wireless networks. Upper figure represents infrastructures network while lower figure represents infrastructure less network.

1.1.1. Characteristics of Ad hoc networks

Following are the main characteristics of ad hoc networks

Dynamic topology: Hosts are mobile and can be connected dynamically in any arbitrary manner. Links of the network vary and are based on the proximity of one host to another one.

Autonomous: No centralized administration entity is required to manage the operation of the different mobile hosts.

Bandwidth constrained Wireless links have a significantly lower capacity than the wired ones; they are affected by several error sources that result in degradation of the received signal.

Energy constrained Mobile hosts rely on battery power, which is a scarce resource; the most important system design criterion for optimization may be energy conservation.

Limited security Mobility implies higher security risks than static operations because portable devices may be stolen or their traffic may cross insecure wireless links.

Because of the distributed, dynamic, ad-hoc, and energy-constraint nature of the sensor network, localized algorithms need to be developed for their scalability, robustness and energy - effectiveness advantages. Localized algorithms intelligently select necessary nodes for sensing, tracking, and reasoning to avoid flooding the network with useless or redundant data, and thus extend the lifetime of the sensor network. The selection of participating nodes can be most efficiently done if both the network services and the applications consider it.

Despite the exciting potentials presented by sensor networks, their unique characteristics have challenged many aspects of traditional computer networking design. Here, we summarize the advantages of sensor networks as well as the challenges from three points of view: scalability, dynamics, and stringent resource.

Scalability issue: Since the cost of sensors is relatively low, so large number of sensors can be deployed but sheer amount of sensors causes the scalability problem. In this case centralized sensor network is useless and distributes sensor network has to be established.

Dynamic issue: Due to constantly changing topology of wireless sensor networks it becomes really difficult to have a pre designed network. Furthermore, because of limited battery lifetime of a sensor node is dependent on the power supply since battery replacement is not an option in sensor networks. In order to save power, localized algorithms seem attracting since only a subset of nodes in the network is utilized for a specific task.

1.2. MANET Routing Protocols

Since the advent of Defense Advanced Research Projects Agency (DARPA) packet radio networks in the early 1970s, numerous protocols have been developed for ad hoc mobile networks. Such protocols must deal with the typical limitations of these networks, which include high power consumption, low bandwidth, and high error rates. Routing protocols in ad hoc networks are generally categorized in two groups: Proactive (Table Driven) and Reactive (On-Demand) routing[24].

1.2.1. Proactive (Table-Driven) Routing Protocols

These routing protocols are similar to and come as a natural extension of those for the wired networks. In proactive routing, each node attempts to maintain consistent, up-to-date routing information to every other node in the network. Each node has one or more tables that contain the latest information of the routes to any node in the network. Each row has the next hop for reaching a node/subnet and the cost of this route. The protocols are proactive in the sense that when a packet needs to be forwarded the route is.already known and can be immediately used. As is the case for wired networks, the routing table is constructed using either link-state or distance vector algorithms containing a list of all the destinations, the next hop, and the number of hops to each destination. Various table-driven protocols (Destination-Sequenced Distance Vector (DSDV), Fisheye State Routing (FSR) protocol etc) differ in the way the information about a change in topology is propagated through all nodes in the network[24].

1.2.2. Reactive (On-Demand) Protocols

Reactive routing is also known as on-demand routing. These protocols take a lazy approach to routing. They do not maintain or constantly update their route tables with the latest route topology. Routes are discovered only when a source node desires them. Route discovery and route maintenance are two main procedures: The route discovery process involves sending route-request packets from a source to its neighbor nodes, which then forward the request to their neighbors, and so on. Once the route-request reaches the destination node, it responds by unicasting a route-reply packet back to the source node via the neighbor from which it first received the route-request. When the route-request reaches an intermediate node that has a sufficiently up-to-date route, it stops forwarding and sends a route-reply message back to the source. Once the route is established, some form of route maintenance process maintains it in each node's internal data structure called a route-cache until the destination becomes inaccessible along the route. Note th each node learns the routing paths as time passes not only as a source or an intermediate node but also as an overhearing neighbor node. In contrast to table-driven routing protocols, not all up-to-date routes are maintained at every node. Examples of reactive routing protocols are the dynamic source Routing (DSR), ad hoc on-demand distance vector routing (AODV) and temporally ordered routing algorithm (TORA0)[24].

1.2.3. Localized and Non-localized Routing

Localization is of much importance in Mobile Ad hoc networks. Localization is the ability of mobile unit to know its physical location in terms of coordinates. GPS (Global Positing System) is one of the straight forward localization strategies. In the absence of the GPS, distance between the nodes can be estimated by the incoming signal strength. Localized algorithms use the concept of localization. Localization is the ability of a sensor to find out its physical coordinates; this is a fundamental ability for embedded networks because interpreting the data collected from the network will not be possible unless the physical context of the reporting sensors is known. Localization may be carried out in one of several ways. If the node is equipped with a Global Positioning System (GPS) card, it can determine its coordinates by receiving signals from a number of satellites. Alternative localization approaches have been proposed to allow nodes to learn their location either from neighboring nodes or from reference beacons. Since all these approaches require communication (sending, receiving or both), localization requires significant energy[25].

1.3. Problems of Energy Conservation

Building an ad hoc network poses a significant technical challenge because of the number of constraints imposed by the environment. Thus, the devices used in the field must be lightweight. Furthermore, since they are battery operated, they need to be energy conserving so that battery life is maximized. Several technologies are being developed to achieve these goals by targeting specific components of the computer and optimizing their energy consumption. For instance, low-power displays, algorithms to reduce power consumption of disk drives low-power I/O devices such as cameras etc. all contribute to overall energy savings. Other related work includes the development of low-power CPUs (such as those used in lap- tops) and high-capacity batteries. From the past few years focus is to develop strategies for reducing the energy consumption of the communication subsystem and increasing the life of the nodes. Recent studies have stressed the need for designing proto- cols to ensure longer battery life. It is observed that the average life of batteries in an idle cellular phone is one day. Some studies revealed that even in Sleep mode the power consumption ranged between 150- 170 mW while in Idle state the power consumption went up by one order of magnitude. In transmit mode the power consumption typically doubled [1].

In Figure 1-2, node A's transmission to node B is overheard by node C because C is a neighbor of A. Node C thus expends energy in receiving a packet that was not sent to it. In this case, clearly, node C needs to be powered off for the duration of the transmission in order to conserve its energy [21].

Energy consumption by mobile nodes takes place during active communication as well as inactive communication. Active communication takes place when the nodes are in wake state and actively listening, sending and receiving the packets while in inactive communication, mobile units consume energy even in sleep state. In conventional routing algorithms, shortest path between two nodes is selected which mainly results in immediate energy depletion. Therefore main goal of the energy efficient routing protocols is to save the energy both during the active and inactive communication therefore increasing the efficiency of the whole network.

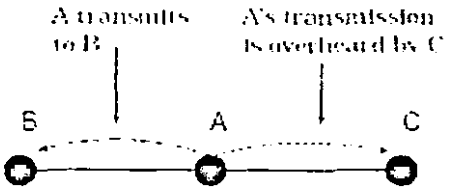


Figure 1-2: Unnecessary Power consumption

1.4. TinyOS: Operating System Designed for Wireless Sensor Networks

An important role of any operating system is to manage and implement reliable application of software while ensuring secure abstraction of hardware resources. Normally OS allocates threads to a processor, map virtual addresses in real memory, manipulating devices, drives and network on behalf of the running application. The mentioned service provided by OS is very common for the applications running on servers and personal computers but for the area of sensor networks, where there is limited hardware and specific application are being run, this service no more seems useful. Wireless sensor network (WSNs) is an emerging area of interest as a result of advancement in RF communication. WSNs are embedded but general-purpose, supporting a variety of applications. WSNs incorporate heterogeneous components, and are capable of rapid deployment in new environments. Besides normal data acquisitions operations processing nodes, there are certain important features of sensor node which are:

- Small Physical Size (as small as a square inch)
- Low Power Consumption
- Concurrency Intensive Operation
- Limited Physical Parallelism and Controller Hierarchy
- Diversity in Design and Usage
- Robust Operation

TinyOS was designed specifically for WSNs. It has a structured event-driven execution model and a component-based software design that supports a high degree of concurrency, enhances robustness and minimizes power consumption while facilitating implementation of sophisticated protocols and algorithms. TinyOS was designed to support concurrency intensive operations required by network sensors with minimal hardware requirements. It combines sensing, communication, and computation into single architecture complete systems on a chip. TinyOS was initially developed by the U.S. Berkeley EECS department. It runs on custom 'mote' hardware [2].

TinyOS is written using a language called NesC (network embedded system C). NesC is similar to the C programming language. It is a new language for programming structured component-based applications. The nesC language is primarily intended for embedded systems such as sensor networks. It has a C-like syntax, but supports the TinyOS concurrency model, as well as mechanisms for structuring, naming, and linking together software components into robust network embedded systems.

Network sensors are application specific. Therefore, only one application runs on a sensor. Since single application runs on a sensor so TinyOS does not support memory management or process management. Since memory manager is not part of TinyOS therefore, it discourages applications from allocating or using dynamic memory.

TOSSIM is a simulator for TinyOS wireless sensor networks. It can capture network behavior at a high fidelity while scaling to thousands of nodes. TOSSIM remain's simple and efficient by using a probabilistic bit error model for the network. TOSSIM is a discrete event simulator which can simulate thousands of motes running complete applications. The nesC compiler (ncc) was modified to support compilation from TinyOS component graphs into the simulator framework. With the change of a compiler option, an application can be compiled for simulation instead of mote hardware, and vice versa.

Most of the simulation environments like ns-2 and others do provide varying degree of scalability and other features but do not cater the power consumed by each node. No doubt overall power usage can be derived from estimates of node duty cycle and communication rates but none of the simulators have catered the requirements of CPU, radio, sensors and other hardware devices. PowerTOSSIM is based on TOSSIM and is also a discrete event simulator which provides a scalable simulation environment for the TinyOS applications [3]. PowerTOSSIM is the first scalable simulation environment for sensor networks that provides accurate power consumption data. It was demonstrated that PowerTOSSIM obtains very accurate power consumption results for a wide range of TinyOS applications and exhibits very little overhead above that of the TOSSIM environment upon which it is based.

Chapter 2

Basic Concepts

2. Basic Concepts

This chapter discusses in detail the basic concepts of the selected protocols i.e. Location Aided Routing (LAR) and Destination Sequenced Distance-Vector (DSDV) routing protocol. Besides this, chapter includes basic introduction to TinyOS, an event driven operating system for Mobile ad hoc networks, TOSSIM and PowerTOSSIM which are the simulation tools for TinyOS.

2.1 Location Aided Routing Protocol (LAR)

Existing MANET routing algorithms do not take into account the physical location of a destination node. Location Aided Routing (LAR) is an approach to decrease overhead of route discovery by utilizing location information for mobile hosts.

Such location information may be obtained using the global positioning system (GPS). Then the location information will be used by means of two Location-Aided Routing (LAR) protocols for route discovery. The LAR protocol uses location information to reduce the search space for a desired route and limits the search space results in fewer route discovery messages. In MANETs, host mobility causes major change in network topology and the task of discovering and maintaining routes is nontrivial. In that case Flooding is a brute force way of discovering routes. LAR reduces the full impact of flooding by forwarding route discovery messages in a selective manner by using location information.

2.1.1 Route Discovery through Flooding

Figure 2-1 shows the basic flooding procedure. When a sender node, S, needs to find a route to a destination node, D, node S broadcasts a route request message to all its neighbors. When the neighbors receive the route request, they will forward it to all their neighbors. If the same route request is received more than once by the same node, it will simply be discarded. As the route request is propagated to various nodes, the path followed by the request is included in the route request packet. On receiving the route request the destination responds by sending a route reply message to the sender. The route reply packet follows a path that is obtained by reversing the path followed by the route request received by the destination node, D. If the route request message does not get to the destination due to transmission error, the sender needs to be able to re-initiate the route discovery. Therefore, when a sender initiates route discovery, it sets a timeout interval. If during the timeout interval, a route reply is not received, then a new route discovery is initiated. Route discovery is initiated either when the sender S detects that a previously determined route to node D is broken, or if S does not know a route to the destination [13].

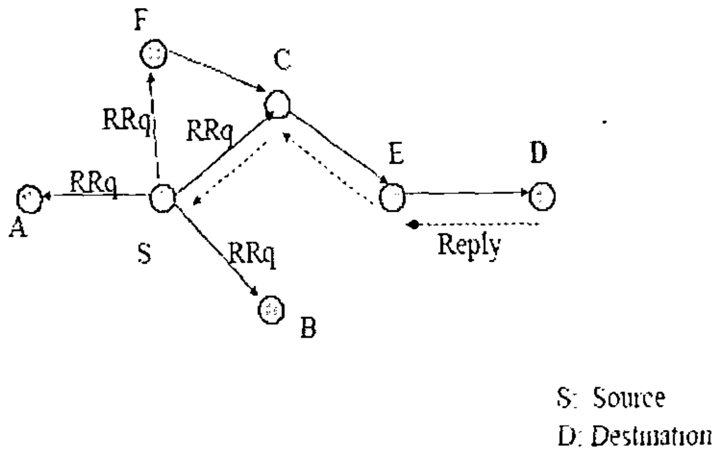


Figure 2-1: Basic flooding procedure

2.1.2. Prerequisites for LAR

2.1.2.1 Location Information

As LAR requires location information of the nodes hence it may be provided by Global Positioning System (GPS). In reality, position information provided by GPS includes some amount of error, which is the difference between GPS-calculated coordinates and the real coordinates. But for the sake of simplicity, this error is neglected.

2.1.2.2. Expected Zone

Consider a node S that needs to find a route to a node D. Assume that node S knows that node D was at location L at time t_0 and that the current time is t_0. The expected zone of node D from the viewpoint of node S at time t_1, is the region that node S expects to contain node D at time t_1.Node S can determine the expected zone based on the knowledge that node D was at location L at time t_0. If node S knows that node D travels with average speed V, then S may assume that the expected zone is the circular region of radius V (t_1 - t_0), centered at location L as shown in figure 2-2. This is an estimate, since if the actual speed is larger than the average; the destination may be outside the expected zone at time t_1. Also, if S does not know a previous location of node D, the entire region occupied by the ad hoc network is assumed to be the expected zone [13]. Having more information about the location of the destination node can help in smaller expected zone for example if the source node knows that the destination node is moving in north direction then the circle shown in figure 2–2(a) will be reduced to semicircle as shown in figure 2–2(b).

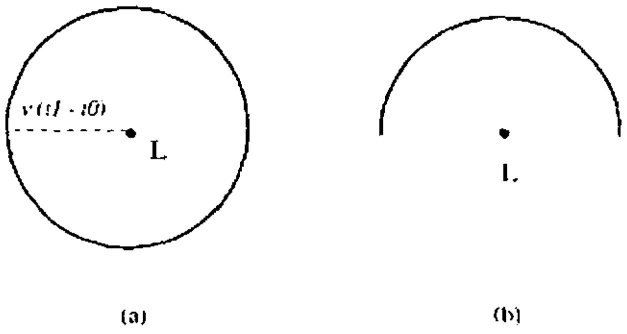


Figure 2–2: Expected Zone

2.1.2.3. Request Zone

Consider a node S that needs to find a route to a node D. Node S defines a request zone for the route request. A node forwards a route request only if it belongs to the request zone. To increase the probability that a route request will reach node D, the request zone should include the expected zone. The request zone may also include additional regions around the expected zone. Still there is no guarantee that a path can be found consisting only of the hosts in a chosen request zone. Therefore, if a route is not discovered within a suitable timeout period, this protocol allows S to initiate a new route discovery with an expanded request zone. The probability of finding a path in the first attempt can be increased by increasing the size of the initial request zone. However, route discovery overhead also increases with the size of the request zone. Thus, there is a trade off between latency of route determination and the message overhead [13]. Therefore request zone can be any of the three as shown in Figure 2–3.

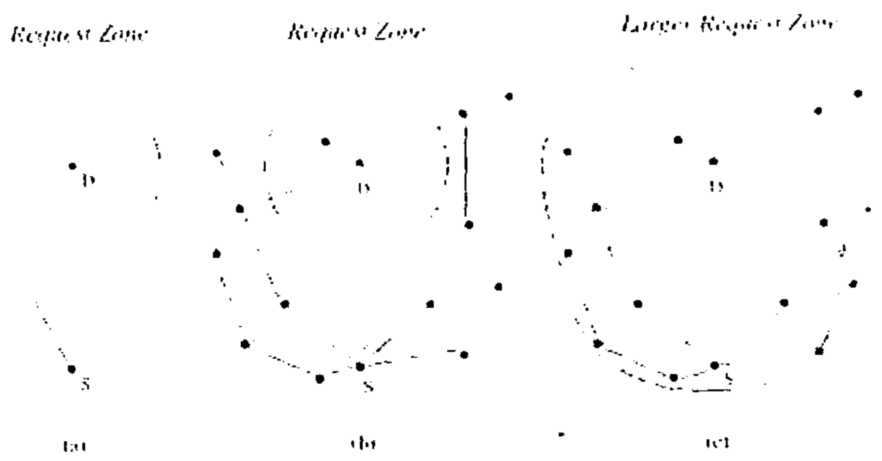


Figure 2-3: Request Zones

2. 1.2.4 Determining Membership of Request Zones

Implementing LAR algorithm requires that a node should be able to determine if it is in the request zone for a particular route request or not. For this purpose [13] presents two schemes of LAR algorithm.

a) LAR Scheme I

For scheme I, it is assumed that node S knows that node D was at location (X_d, Y_d) at time t_0. At time t_1, node S initiates a new route discovery for destination D. It is also assumed that node S also knows the average speed v with which D can move. Using this, node S defines the expected zone at time t_1 to be the circle of radius $R = v (t_1 - t_0)$ centered at location (X_d,Y_d).

In this scheme [13] the request zone is defined to be the smallest rectangle, as shown in Figure 2-4. It includes the current location of the sender node S and the expected zone, such that the sides of the rectangle are parallel to the X and Y axes. Thus, the node S can determine the four corners of the expected zone. S includes their coordinates with the route request message transmitted when initiating route discovery. When a node receives a route request, it discards the request if the node is not within the rectangle specified by the four corners included in the route request. For example node J will discard the message as it is not in the request zone whereas node I will accept and forward it.to next neighboring node. When node D receives the route request message, it replies by sending a route reply message. However, node D includes its current location and current time in the route reply message. Node S can thus use this information to determine the request zone for a future route discovery.

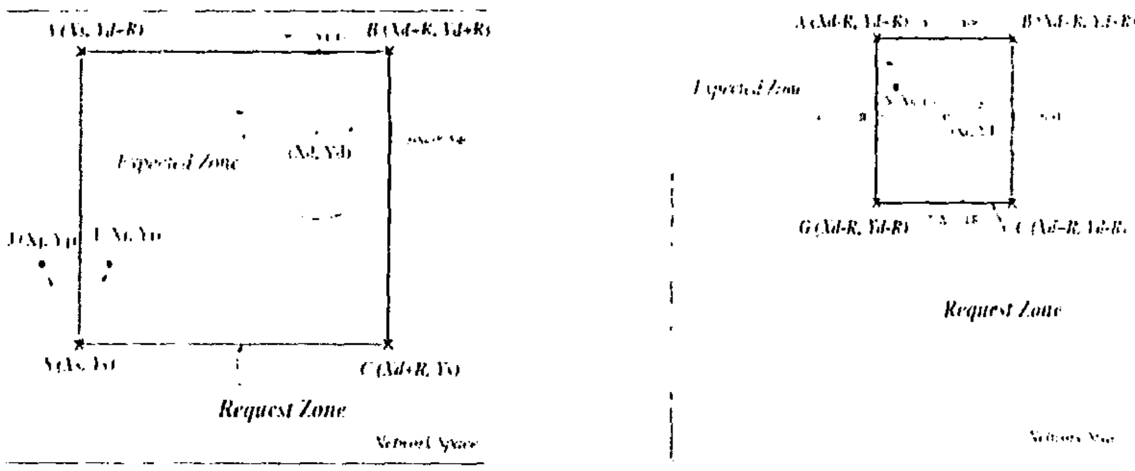


Figure 2–4: LAR Scheme I

b) LAR Scheme II

In this scheme [13] node S includes two pieces of information with its route request: its distance from node D and the coordinates of node D. Assume that node S knows the location (X_d, Y_d) of node D at time t_0. Note that the time at which route discovery i initiated by node S is t_1, where $t_1 >= t_0$. Let $DIST_s$ denote the distance between node S and node D as calculated by S. When node I receives the route request from the sender node S, node I calculates its distance from location (X_d, Y_d) denoted as $DIST_i$. For some parameter, δ, if $DIST_s + \delta >= DIST_i$, then node I forwards the request to its neighbors. When node I forwards the route request, it now includes $DIST_i$ (replacing the value $DIST_s$ received) and (X_d, Y_d) in the route request before forwarding the route request. Else if $DIST_s + \delta < DIST_i$, then node I discards the route request. Figure 2–5 depicts LAR Scheme II.

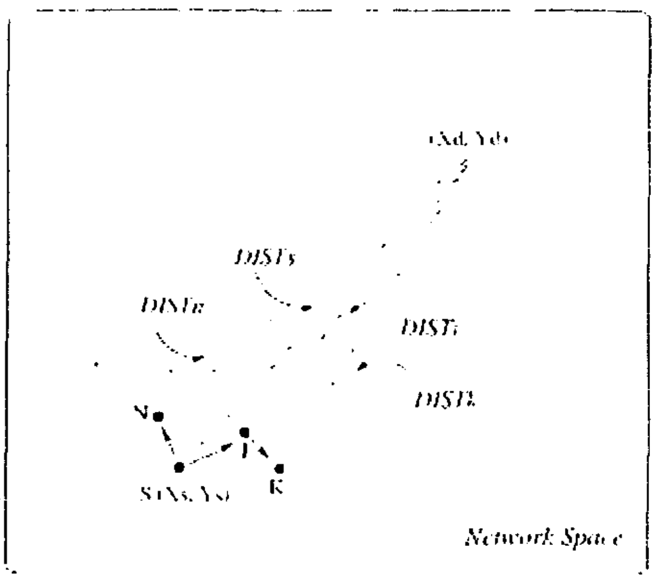


Figure 2–5: LAR Scheme II

2.2 Destination Sequenced Distance Vector Routing Protocol (DSDV)

For routing in a distributed network environment, each node maintains for each destination a preferred neighbor. The packet moving in a network at any given time contains the identifier for the destination in its header. When a node receives a packet, it forwards it to the preferred neighbor for its destination. The way in which routing tables are updates and maintained depends on different types of next-hop routing methods. Basically there are two famous routing methods. One is Link Cost and other is Distance Vector [14]. In link Cost approach, each node maintains the information about network topology in the form of cost for each link. For consistency, each node broadcasts the link cost information periodically. Node receiving the updates applies the shortest path algorithm accordingly to choose next hop for the destination. In Distance Vector approach, every node '*i*' maintains, for each destination '*x*', a list of distances $\{d^{x}_{ij}\}$ where '*j*' ranges over the neighbors of '*i*'. Node '*k*' is treated as next neighbor destined for '*x*' only if d^{x}_{ik} equals $\min_j \{d^{x}_{ij}\}$. This method will lead to the destination '*x*' along the shortest path. Up to date destination estimates are broadcasted in the network [14].

Destination Sequenced Distance Vector Routing is based on Bellman-Ford routing mechanism. Each node maintains routing table that records all possible destinations and the number of hops to those destination. Each entry is marked with a sequence number assigned by the destination node. These sequence numbers are used to distinguish "old" versus "new" routes. Routing tables are periodically transmitted to maintain table consistency. A "full dump" record reports on the entire routing table. This may result in lots of traffic, since each table is probably larger than the network protocol data unit. Beside this, incremental updates provide information since last "full dump". A new route broadcasts contains four parameters:

(1) The address of the destination

(2) The number of hops to reach the destination

(3) The sequence number of the information

(4) Sequence number unique to this broadcast.

Route labeled with most recent sequence number is always used. If two event have the same sequence number, the route with the smaller metric is used. Hosts also keep track of the settling time of routes, or the weighted average time for which routes to a destination will fluctuate before the route with the best metric is received. When a node decides that a route is broken, it increments the sequence number of the route and advertises it with infinite metric. Basic working of DSDV is as follows. Scenario considered is the one when node X receives information from Y about a route to Z. Let destination sequence number for Z at X be S(X), S(Y) is sent from Y. If S(X) > S(Y), then X ignores the routing information received from Y. If S(X) = S(Y), and cost of going through Y is smaller than the route known to X, then X sets Y as the next hop to Z. If S(X) < S(Y), then X sets X as the next hop to Z and S(X) is updated to equal S(Y).

2.3 TinyOS

TinyOS is an operating system which provides an open-source development environment It is an interrupt driven event based operating environment. Hence it uses CPU efficiently. TinyOS system is composed of state machines where each state machine is a TinyOS "component". In TinyOS there is no differentiation between kernel or user space. Main ideology of this operating system is to work only when there is a maximum work load and to sleep as often as possible to save power. For this purpose, TinyOS has power aware scheduler. Scheduler puts the processor into sleep state when queue is empty. It works on simple FIFO scheme. Furthermore it has the added advantage of high concurrency.

2.3.1. Features of TinyOS

This section describes some of the core features of TinyOS in detail.

a) Resource Constrained Concurrency

TinyOS manages several devices, e.g., ADCs, sensors, Flash memory, and radio. Typically, an operation is started on a device, which then runs concurrently with the main processor until generating a response. Meanwhile, other devices may also need service, requiring the system to juggle several event streams. The thread dedicated to a device issues a command and then sleeps or polls until the operation completes. The OS switches among threads by saving and restoring their registers, and threads coordinate with others by using shared variables as flags and semaphores.

b) Structured Event-Driven Execution

TinyOS provides a structured event-driven model. A complete system configuration is formed by 'wiring' together a set of components for a target platform and applicatio domain. Components are restricted objects with well-defined interfaces, internal state, and internal concurrency. Primitive components encapsulate hardware elements, e.g., radio, ADC, timer, or bus. Their interface reflects the hardware operations and interrupts.

c) Components and Bidirectional Interfaces

TinyOS is written in ANSI standard C with support for component composition, system-wide analysis, and network data types. A component has a set of bidirectional command/ event interfaces implemented either directly or by wiring a collection of subcomponents. The compiler optimizes the entire hierarchical graph, validates that it is free of deadlocks and sizes resources.

TinyOS uses network types with a completely specified representation; the compiler provides efficient access to packet fields on embedded nodes, as well as Java and XML methods for packet handling on conventional computers and gateways.

To avoid delay in response long latency function calls are executed in split-phase. Split-phase operations are a typical use of bidirectional interfaces. This operation is accomplished using commands, events and tasks. A higher-level component issues a command to initiate activity in a hardware or software component. The command returns immediately, indicating the status of the request, even though the operation takes some time to complete. When done, the operating component signals an event to components that will take further action. Meanwhile, the processor may service other tasks and events, or sleep if no tasks are pending. Thus, interleaved execution and power management is provided systematically throughout the entire set of TinyOS components. For example, for sending a packet the application issues a send call which will schedule the actual sending process to be executed at a later time. When this packet will be transmitted, the application is notified regarding the successful transmission.

d) Communications

Communications and networking have driven the TinyOS design as available radios and microcontroller interfaces evolved. The communication subsystem has to meet real-time requirements and respond to asynchronous events while other devices and processes are serviced. Early designs modulated the radio channel bit-by-bit in software, while the Mica and Mica2 generations used a byte-level interface. The TinyOS 2.0 radio component provides a uniform interface to the full capabilities of the radio chip. The link-level component provides rich media access control (MAC) capabilities, including channel activity detection, collision avoidance, data transfer, scheduling, and power management, allowing networking components to optimize for specific protocols. Packet transmission and reception is reflected in the TinyOS execution model; send is split-phase with a send-done event, whereas receive generates a packet-arrival event.

e) Embedded Application Services

Networking The TinyOS community has developed networking layers targeted at different applications with different techniques for discovery, routing, power reduction, reliability, and congestion control. These networking layers provide higher-level communication services to embedded applications with a TinyOS programming interface. The most widely used services for WSNs are reliable dissemination, aggregate data collection, and directed routing.

Reliable dissemination provides guaranteed delivery of messages from a source. It is used for network configuration and control, so it doesn't depend on any pre-established neighbor discovery, routing, or connection setup. To collect and aggregate data from the WSN, nodes cooperate in building and maintaining one or more collection trees, rooted at a gateway or

data sink. Similarly, a node can establish a route to any other node in the network and have packets forwarded hop-by-hop to the destination. Such directed routing is most often used to issue commands to a particular node, transfer bulk data, and in network management [20].

2.3.2. Design of TinyOS

2.3.2.1. Data Memory Model

Usually application running in TinyOS is assigned the memory in following order. For executable code 128K program flash can be used where as för data or program constant same 128k flash memory is used but for variables 4k static RAM is used. Figure 2-6. shows a pictorial representation of static memory. There is absence of heap and function pointers and stack grows down in the free space. Global variables are available on a per-frame basis whereas local variables are saved on the stack and declared within a method.

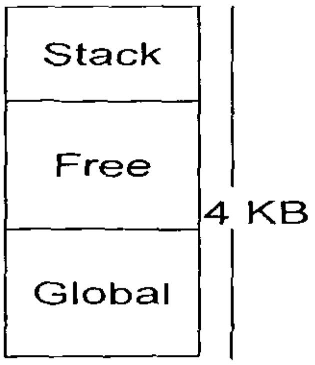


Figure 2-6 Data Memory Model of TinyOS

2.3.2.2. Programming Model

Figure 2-7 represents a complete system configuration of TinyOS which consists of a tiny scheduler and a graph of components. A component has four interrelated parts: a set of command handlers, a set of event handlers, an encapsulated fixed-size frame, and a bundle of simple tasks. Tasks, commands, and handlers execute in the context of the frame and operate on its state. To facilitate modularity, each component also declares the commands it uses and the events it signals. These declarations are used to compose the modular components in a per-application configuration. The composition process creates layers of components where higher level components issue commands to lower level components and lower level components signal events to the higher level components. Physical hardware represents the lowest level of components [16].

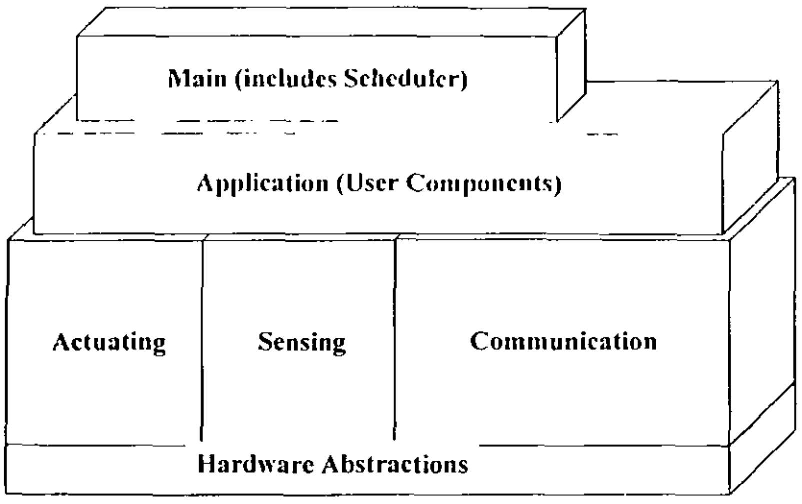


Figure2-7 Programming Model of TinyOS

In the following sections we will discuss corresponding design details.

a) Events

Events are upcalls in the component graph. As events are often the result of interrupts (and have can correspondingly have interrupts disabled), it is critical that they be as short as possible. Hardware events are interrupts, caused by a timer, sensor, or communication device.

b) Commands

Commands represent downcalls in the component graph; a component calls commands on components lower than it in the component graph. Similarly, as commands can be called by events, they must be short to ensure an event does not run for a long time.

c) Tasks

The abstraction known as the "Tasks" is a mechanism for asynchronous, long-running computation. They run synchronously in respect to one another (are never preempted), but can be preempted by events. Therefore, if a TinyOS task has real-time requirements (e.g. is posted every 100 ms and must complete within 40ms of being posted), tasks must not execute for a long period of time.

Tasks perform all the major work like computation. They can be pre-empted by events but vice-versa does not happen. It can call lower level commands, signal higher level events and schedule other tasks. Furthermore it can simulate concurrency within components. It provides means to incorporate arbitrary computation into the event driven model. Tasks are a form of deferred procedure call that allows a hardware event or task to postpone processing. Tasks are posted to a queue. As tasks are processed, interrupts can trigger hardware events that preempt tasks. When the task queue is empty, the system goes into a sleep state until the next interrupt. If this interrupt queues a task, TinyOS pulls it off the queue and runs it. If not, it returns to sleep. Tasks are atomic with respect to each other.

2.3.2.3. Component Model

a) Components

There are two types of components, Modules and Configurations. Modules implement the application behavior while configurations wire the components together. A component does not care if another component is a module or configuration and a component may be composed of other components

A named component encapsulates some fixed-size state and a number of tasks. Components interact with each other strictly via function call interfaces, which are related groups of commands and events. The set of interfaces a component uses and provides define its external namespace.

b) Commands

Commands typically represent requests to initiate some action. Commands flow downwards and the control is returned to caller. They are non-blocking requests made to lower level components. Typically, a command will deposit request parameters into its frame and conditionally post a task for later execution. It may also invoke lower commands, but it must not wait for long or indeterminate latency actions to take place. A command must provide feedback to its caller by returning status indicating whether it was successful or not.

c) Events

Events flow upwards and the control is returned to the signaler. Events can call commands but not vice versa. Events represent the completion of a request or something triggered by the environment, e.g., message reception. Both explicitly return error conditions, such as the inability to service the request. A specific set of events are bound to hardware interrupts. A programmer assembles an application by specifying a set of components and "wiring" their interfaces together.

d) Interfaces

Interface is like a header file that defines APIs using which components will interact. It specifies the functionality to the outside world. Interfaces tell the outside world that what type of commands can be called and what type of handling is needed for events. Interfaces are bidirectional. They specify a multi-function interaction channel between two components, the provider and the user. The interface specifies a set of named functions, called commands, to be implemented by the interface's provider and a set of named functions, called events, to be implemented by the interface's user.

2.4 NesC

NesC is an extension to C designed to embody the structuring concepts and execution model of TinyOS. The basic concepts behind nesC are:

1. Components are wired to form whole programs. Components define two scopes, one for their specification (containing the names of their interface instances) and one for their implementation. Threads of control may pass into a component through its interfaces. These threads exist either in a task or a hardware interrupt.
2. Interfaces may be provided or used by the component. The provided interfaces are intended to represent the functionality that the component provides to its user; the used interfaces represent the functionality the component needs to perform its job.
3. Interfaces are bidirectional; they specify a set of functions to be implemented by the interface's provider (commands) and a set to be implemented by the interface's user (events). All lengthy commands in TinyOS are non-blocking and their completion is signaled through an event.
4. Components are statically linked to each other via their interfaces. This increases runtime efficiency, encourages robust design, and allows for better static analysis of programs.
5. nesC is designed under the expectation that code will be generated by whole-program compilers. This allows for better code generation and analysis [17].

2.4.1 Nesc Interfaces and components

a) Interfaces

NesC specify a multi-function interaction channel between two components, the provider and the user. The interface specifies a set of named functions, called commands, to be implemented by the interface's provider and a set of named functions, called events, to be implemented by the interface's user. Interfaces are specified by interface types and in following manner:

Interface identifier {declaration-list}

This identifier has global scope and belongs to a separate namespace, the component and interface type namespace. This declaration-list must consist of function declarations with the command or event storage class [17].

b) Component specification

A nesC component is either a module or a configuration.

nesC-file:

    includes-list of module

    includes-list of *configuration*

module:

    module identifier specification module-implementation

configuration:

    configuration identifier specification configuration-implementation

Component's names are specified by the identifier. The specification lists the specification elements (interface instances, commands or events) used or provided by this component. A component must implement the commands of its provided interfaces and the events of its used interfaces.

Each specification element has a name (interface instance name, command name or event name).

These names belong to the variable namespace of the per-component-specification scope.

specification:

```
        { uses-provides-list }
uses-provides-list:
        uses-provides
        uses-provides-list uses-provides
uses-provides:
        uses specification-element-list
        provides specification-element-list
specification-element-list:
        specification-element
        { specification-elements }
specification-elements:
        specification-element
        specification-elements specification-element
```

c) Modules

Modules implement a component specification with C code:

```
module-implementation:
        implementation { translation-unit }
```

where translation-unit is a list of C declarations and definitions. The top-level declarations of the module's translation-unit belong to the module's component implementation scope. These declarations have indefinite extent and can be: any standard C declaration or definition, a task declaration or definition, a commands or event implementation [17].

d) Tasks

A task is an independent locus of control defined by a function of storage class task returning void and with no arguments:

```
task void myTask() { ... }
```

A task can also have a forward declaration, e.g., task void myTask (). Tasks are posted by prefixing a call to the task with post, e.g., post myTask(). Post returns immediately; its return value is 1 if the task was successfully posted for independent execution, 0 otherwise. The type of a post expression is unsigned char [17].

e) Atomic statements

Atomic statement guarantee that the statement is executed "as-if" no other computation occurred simultaneously. It is used to implement mutual exclusion, for updates to concurrent data structures, etc. Atomic sections should be short; to help meet this requirement, nesC forbids calling commands or signaling events inside atomic statements. The following constructions are also forbidden inside atomic: goto, return, break, continue, case, default and labels [17].

atomic-stmt:

atomic statement

f) Configurations

Configurations implement a component specification by connecting, or wiring, together a collection of other components:

configuration-implementation:

implementation { component-list connection-list }

The component-list lists the components that are used to build this configuration; the connection-list specifies how these components are wired to each other and to the configuration's specification.

The component-list specifies the components used to build this configuration. These components can be optionally renamed within the configuration. Wiring is used to connect specification elements (interfaces, commands, events) together.

connection:

endpoint = endpoint

endpoint -> endpoint

endpoint <- endpoint

endpoint:

identifier-path

identifier-path [argument-expression-list]

identifier-path:

identifier

Wiring statements connect two endpoints. The identifier-path of an endpoint specifies a specification element. The argument-expression-list optionally specifies interface parameter values. Wiring is attaching two components together and specifying the interface using which they will exchange information [17].

For example,

BadgeM.Leds → LedsC.Leds;

The arrow direction indicates the command calling and event signalling flow. Here BadgeM will call commands on LedsC and LedsC will signal events to BadgeM.

g) Comments

nesC allows // comments in C, interface type and component files.

2.4.2. Compilation Phase

Figure 2-8 specifies the compilation phase of the nesc application. The application is compiled in following steps.

NesC compiler first converts the application into a c file (app.c. present in build directory of each application). TinyOS kernel written in C language, while TinyOS libraries written in nesC also; act as an input in nesC compiler. All commands and events are replaced with function. Code for clock interrupt service routine is generated. C file is compiled into an exe format. avr-objcopy application converts it into a flash programmable format [17].

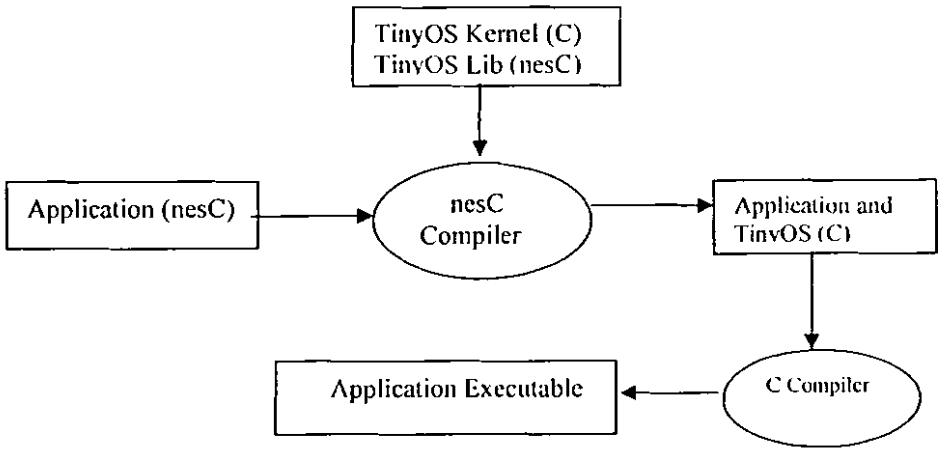


Figure2-8: Compilation in TinyOS

2.4.3. Concurrency Model

TinyOS executes only one program consisting of selected system components and custom components needed for a single application. There are two threads of execution: tasks and hardware event handlers. Tasks are functions whose execution is deferred. Once scheduled, they run to completion and do not preempt one another. Hardware event handlers are executed in response to a hardware interrupt and also run to completion, but may preempt the execution of a task or other hardware event handler. Commands and events that are executed as part of a hardware event handler must be declared with the ***async*** keyword. Because tasks and hardware event handlers may be preempted by other asynchronous code, nesC programs are susceptible to certain race conditions. Races are avoided either by accessing shared data exclusively within tasks, or by having all accesses within atomic statements. The nesC compiler reports potential data races to the programmer at compile-time. It is possible the compiler may report a false positive. In this case a variable can be declared with the ***norace*** keyword. The ***norace*** keyword should be used with extreme caution [17].

2.5 TOSSIM

2.5.1 TOSSIM Architecture

The TOSSIM architecture is composed of five parts: support for compiling TinyOS component graphs into the simulation infrastructure; a discrete event queue; a small number of re-implemented TinyOS hardware abstraction components; mechanisms for extensible radio and ADC models; and communication services for external programs to interact with a simulation [18]. Figure 2-9 shows a complete architecture of TOSSIM.

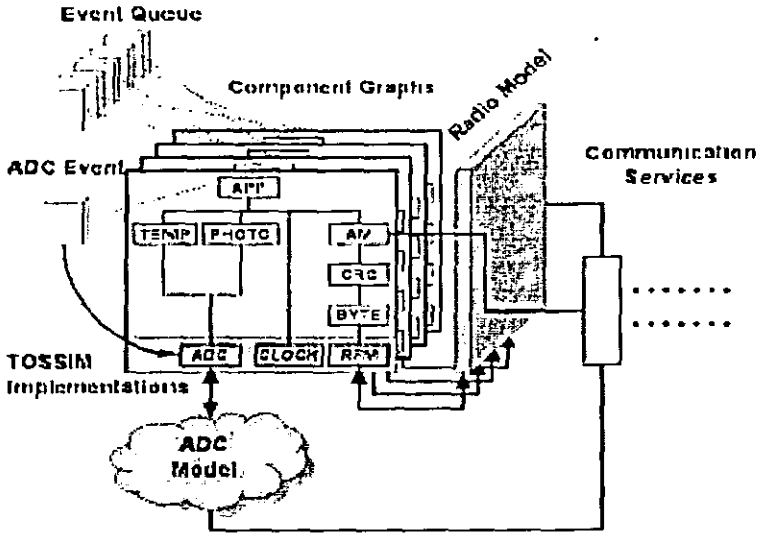


Figure 2-9 TOSSIM Architecture

TOSSIM takes advantage of TinyOS structure and whole system compilation to generate discrete-event simulations directly from TinyOS component graphs. It runs the same code that runs on sensor network hardware. By replacing a few low-level components, TOSSIM translates hardware interrupts into discrete simulator events; the simulator event queue delivers the interrupts that drive the execution of a TinyOS application. The remainder of TinyOS code runs unchanged. The simulator engine provides a set of communication services for interacting with external applications. These services allow programs to connect to TOSSIM over a TCP socket to monitor or actuate a running simulation. Details of the ADC and radio models, such as readings and loss rates, can be both queried and set. Programs can also receive higher level information, such as packet transmissions and receptions or application-level events. As it is a discrete event simulation, users can set debugger breakpoints and step through what is normally real-time code (such as packet reception) without disrupting operation. It also provides mechanisms for other programs to interact and monitor a running simulation; by keeping monitoring and interaction external to TOSSIM, the core simulator engine remains very simple and efficient[18]. TOSSIM compiles directly from TinyOS code. Built with make pc, the simulation runs natively on a desktop or laptop. TOSSIM can simulate thousands of nodes simultaneously. Every mote in a simulation runs the same TinyOS program [19]. It provides run-time configurable debugging output, allowing a user to examine the execution of an application from different perspectives without needing to recompile. **TinyViz** is a Java-based GUI that allows you to visualize and control the simulation as it runs, inspecting debug messages, radio and UART packets, and so forth. The simulation provides several mechanisms for interacting with the network; packet traffic can be monitored, packets can be statically or dynamically injected into the network [19].

2.5.2. Building and Running an Application

TOSSIM is compiled by typing make pc in an application directory. In addition to the expected TinyOS components, a few simulator-specific files are compiled. TOSSIM version of the application is build with ***make pc***. The TOSSIM executable is **build/pc/main.exe. build/pc/main.exe --help** can be used to see a brief summary of its command-line usage. TOSSIM has a single required parameter, the number of nodes to simulate. By default, TOSSIM prints out all debugging information [19]. Four DBG modes are reserved for application components and debugging use: usr1, usr2, usr3, and temp. In TinyOS code, debug message commands have this syntax:

```
dbg(<mode>, const char* format, ...);
```

2.6 Power TOSSIM

Simulation environments, such as ns2 [20], TOSSIM [18] etc, provide varying degrees of scalability, realism, and detail for understanding the behavior of sensor networks. To date, however, none of these tools have considered one of the most important aspects of sensor application design: that of power consumption. While simple approximations of overall power usage can be derived from estimates of node duty cycle and communication rates, these techniques often fail to capture the detailed, low-level energy requirements of the CPU, radio, sensors, and other peripherals[3].

Power TOSSIM is based on TOSSIM. It is a scalable simulation environment which provides accurate estimation of power consumption by each node in wireless sensor network. In PowerTOSSIM, TinyOS components corresponding to specific hardware peripherals (such as the radio, EEPROM, LEDs, and so forth) are instrumented to obtain a trace of each device's activity during the simulation run. PowerTOSSIM runs applications as a native executable to scale up to large numbers of sensor nodes and does not directly simulate each node's CPU at the instruction level.

PowerTOSSIM tracks the power state of each hardware component of the simulated motes by generating specific power state transition messages that are logged during the simulation run. This is accomplished by instrumenting the TOSSIM simulated hardware components with calls to a new component, PowerState, which tracks hardware power states for each mote and logs them to a file during the run. CPU profiling is accomplished by mapping the basic blocks executed by the simulation code to cycle counts in the corresponding mote binary. The power state data generated by PowerTOSSIM can be combined with a power model to determine per-mote and-per-component energy usage.

2.6.1. Characteristics of PowerTOSSIM

Efficiency and flexibility are important characteristics of PowerTOSSIM which will be discussed in following section.

a) Efficiency: PowerTOSSIM inherits from TOSSIM very high efficiency in simulating large networks that scale to thousands of nodes. To preserve this efficiency, it is important to avoid introducing high overheads into the simulation itself. Logging hardware state transition messages at runtime introduces very low overhead. Likewise, allowing the simulation to run as a native binary avoids the overhead of instruction-level simulation [3].

b) Flexibility: PowerTOSSIM provides a high degree of flexibility for capturing and modeling the power state of the mote. With the decoupled design it is possible to evaluate the power efficiency of potential hardware designs only by plugging a new power model into the analysis tools; the simulation itself need not be re-executed.
The following subsections and figure 2-10 present the design of each part of the PowerTOSSIM architecture in detail [3].

Chapter 3

Literature Survey

2.6.2. Architecture Details

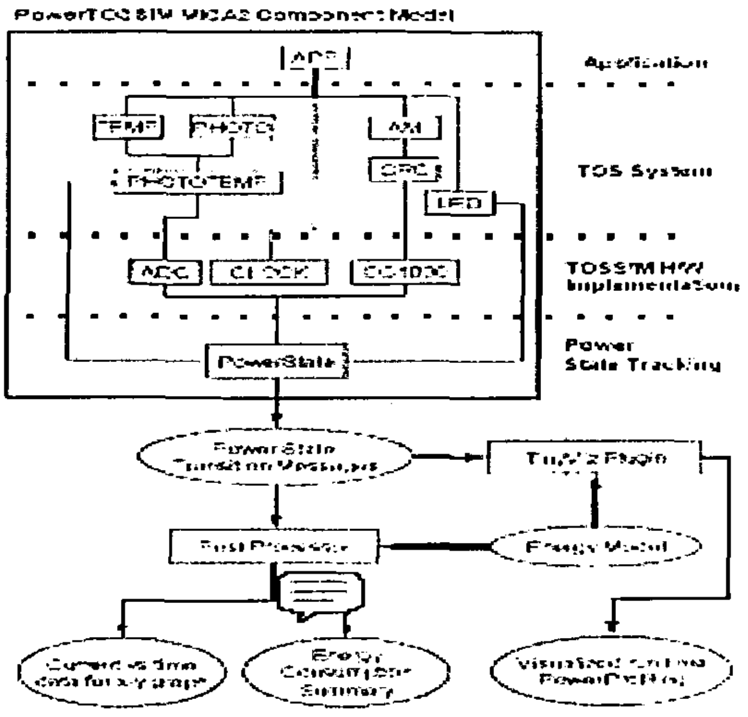


Figure 2-10 PowerTOSSIM Architecture

PowerTOSSIM leverages this design by instrumenting each of the simulated hardware drivers with power state transition messages that are logged during the simulation. This is accomplished by adding calls from each hardware driver to a new module, called PowerState, which emits log messages when the Powerstate of each hardware device changes [3].

CPU profiling is achieved in following manner. PowerTOSSIM binary is instrumented to obtain an execution count for each basic block executed by the simulated CPU. Then each basic block is mapped to its corresponding assembly instructions in the AVR binary. As a next step, the number of CPU cycles for each basic block using simple instruction analysis is determined and in the end the simulation basic block execution counts are combined with their corresponding cycle counts to obtain the total CPU cycle count for each simulated mote. At the end of the simulation run, PowerTOSSIM logs basic block execution counters that are processed offline in the steps mentioned above to obtain CPU cycle count totals.

3. Literature Survey

Following section includes the literature survey in three aspects. First section describes the literature survey regarding the position based routing in mobile ad hoc networks while section two describes the work related to energy efficient routing in MANETS. The last section describes the work associated with the simulating tools provided for the sensor or mobile ad hoc networks.

3.1. Position Based Routing in Mobile Ad hoc Networks

There may be two divisions of ad hoc networks. One is static ad hoc network and other is mobile ad hoc networks. In static ad hoc network, the position of the node in the network may not change once it has entered in the network while in the mobile ad hoc network, the node position will change frequently. The topology of this type of network will keep on fluctuating thereby resulting in routing challenges. There exist different approaches in routing. One is topology based routing and other is position based routing. In *topology based routing* packets are forwarded on the basis of the links existing between the nodes in the network where as in *position based routing* , physical position of the nodes is available either through GPS or through other means. Packet forwarding takes place by knowing the physical location of the source and the destination beside other additional information. Topology based routing can be further divided into three schemes i.e. Proactive, Reactive and Hybrid. Proactive algorithms maintain the record of the available paths even if they are not active in the network. Reactive algorithms only maintain the record of the network paths that are currently being used hence eliminating the unnecessary burden of the resources. Hybrid algorithms are amalgam of local proactive routing and global reactive routing. In the following, we give the study of few research papers of the position based routing algorithms.

3.1.1. A Survey on Position Based Routing in Mobile Ad hoc Networks

Martin et al., [4] presented in this paper an overview of ad hoc routing protocol that make forwarding decision based on the geographical location. Authors conclude that for forwarding the packet, beside source position, destination position and the position of one- hop neighbor is required. As a result of this position based routing performs well when the topology is changing dynamically. Beside position based forwarding strategies, authors have discussed location services strategies.

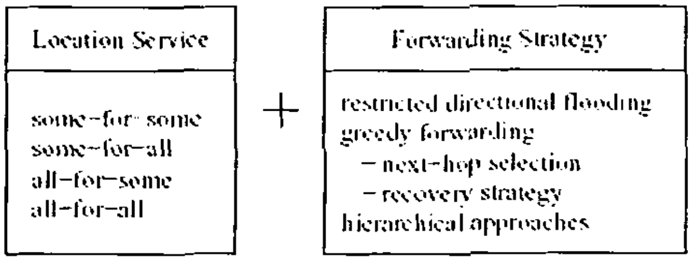


Figure 2-1: Building blocks and criteria for classification

Considering the combinations of the location services mentioned in Figure XYZ, authors have discussed four location services namely distance routing effect algorithm, quorum based location service, home zone and grid location service. Under the umbrella of forwarding strategies three approaches have been discussed where each approach discusses the routing protocols falling in that strategy. For example, in restricted directional flooding approach, DREAM (Distance Routing Effect Algorithm for Mobility) and LAR (Location Aided Routing) have been discussed. Paper compares the above mentioned location services under time, communication and implementation complexity. Furthermore, forwarding strategies were compared under the evaluation criteria of communication and implementation complexity, robustness etc. DREAM and LAR have communication complexity of O (n) and does not scale to a large network for high data transmission. Both approaches perform well against the failure of the individual nodes and position inaccuracy while implementation of both the algorithm is simple. Greedy, terminodes and grid have communication complexity of O (√n) and their implementation is medium, high and high respectively.

3.1.2. Position-Based Multicast Routing for Mobile Ad-Hoc Networks

Widmer et al., [5] presented a new protocol known as position based multicast routing protocol (PBM). It is a multicast protocol where the forwarding node forwards the packets on the basis of the position of the destination and its own neighbors to determine the position of the destination. It is adaptation of position based unicast routing protocols such as face-2 and GPSR (Greedy Perimeter Stateless Routing) to multicast routing. Authors claim that both, tree- and mesh-based multicast routing protocols, need to maintain state information about the distribution structure whereas in position-based multicast (PBM) does neither require the maintenance of state about a distribution structure nor does it resort to flooding of the data packets. Instead each node that forwards a multicast packet autonomously determines the neighbors that it should forward the packet to. This decision is based on information about the position of the destination nodes, the position of the forwarding node, and the position of the forwarding node's neighbors. Without the use of dedicated simulation environment, authors have selected C++ as a tool for simulation as they claim that large number of nodes, large number of simulation runs or large area cannot be simulated on ns-2 or GloMoSim. Multiple node densities were simulated for three types of areas small, medium and large. Results showed that delivery rate of packets was successful fro static network while for dynamic network, due to routing loops, packets may drop. PBM is very well suited for highly dynamic networks without resorting to flooding of the data packets. In addition the rule for splitting a multicast packet includes a parameter I that may be used to adapt the algorithm to different application scenarios by controlling the tradeoff between latency and bandwidth.

3.2. Power Aware Routing Protocols for MANETS

A key technical challenge in designing protocols for the ad hoc networks has been energy savage. Energy is a valuable source in a network where the nodes consumes ample amount of energy while on the other hand battery life is limited. Power dissipation of the network is directly proportional to the power consumption at different entities of the network. Therefore there is a need for developing power aware schemes at different layers of the networks. A number of routing algorithms have been proposed so far. The literature surveyed for the proposed power aware routing protocols is as follows:

3.2.1. Probabilistic Geographic Routing Protocol for Ad Hoc and Sensor Networks

Tanya et al., [6] presented a novel power aware routing algorithm for wireless ad hoc and sensor networks. For probabilistically forwarding the packet to the next hop, the protocol uses local information. A set of candidate nodes were selected for forwarding and then assigned a probability proportional to their residual energy and the link reliability. Using the residual energy in the cost function ensures that nodes with more reliable links are not drained out of energy too quickly. This will in turn increase the lifetime of the network. PGR locally minimizes the retransmissions which results in energy saving. Authors have compared the performance of the proposed protocol (PGR) and GPSR on NS-2 and concluded that PGR improves the throughput by 40%, increases the life time of the network by 30% and decreases the overall end to end delay

3.2.2. A Simulation Study of an Energy Efficient Routing Protocol for Mobile Ad hoc Networks

Xiaojiang, [7] proposed a novel energy efficient protocol for dense networks named as Energy Efficient Cell Relay Routing (EECR). Author have compared the performance of EECR with LAR (Location Aided Routing) and through simulation proved that the protocol has good performance and good scalability in large network. EECR divides the entire routing area into the cells and selects that particular node with maximum energy in the cell and even for the same source-destination pair; the participating node from a cell may be different for different sessions. This balances the energy of different nodes and reduces the chance of draining out certain nodes too soon, and thus increase the lifetime of the whole network therby increasing the lifetime of the network. Simulation results showed that routing overhead increases for both the protocol as mobility increases. Furthermore the routing overhead decreases for both protocols when node transmission range becomes large. LAR causes network saturation when the traffic load is heavy while delay of LAR is low when traffic load is light ad vice versa.

3.2.3. Power Aware Localized Routing in Wireless Networks

Ivan et al., [8] proposed that localized power, cost and power-cost efficient routing algorithms are loop free and their efficiency was proved by experiments. Authors defined a new metric for wireless networks based on the remaining energy at the nodes. The newly defined metric is hybrid of cost and power metric and named as power-cost metric based on the combinations of node's lifetime and the distance based power metric. Authors have proved by theorems that localized power-cost efficient routing protocols based on the power-cost aware metric are loop free and energy efficient. A new routing scheme known as Nearest Closer (NC) was proposed. According to this scheme, packet is forwarded to the nearest neighbor of the destination rather than the source. For large values of the square size of the network, NC method is superior to GEDIR and DIR since it simulates retransmissions in the best possible way.

3.2.4. Minimum Energy Mobile Wireless Networks

Volkan et al., [9] described a distributed position-based network protocol optimized for minimum energy consumption in mobile wireless networks that support peer-to-peer communications. This network protocol reconfigures the links dynamically as nodes move around, and its operation does not depend on the number of nodes in the system. It was assumed that each mobile node have a portable set with transmission, reception, and processing capabilities. Besides this, each has a low-power global positioning system (GPS) receiver on board, which provides position information. A local optimization procedure that finds the minimum energy links and dynamically updates them was proposed. The protocol was self-reconfiguring in mobile scenarios. In peer to peer communications each nodes acts as an information sender and information receiver so a strong connectivity must be provided by the protocol. Second focus was on the design of a self-reconfiguring protocol that consumes least amount of energy. Authors have considered one of the nodes as an information sink node and termed it as master-site. Authors claimed that a protocol that solves the minimum energy problem with a single master-site simultaneously solves the general peer-to- peer communications problem because each node can independently be taken as a master-site, and the optimal topologies can be superimposed. Paper included the simulation of a stationary network and a mobile network. Results of static network showed that as the number of nodes grows larger, the average power decreases toward its asymptote while for mobile network the average power consumption per node was significantly low.

3.2.5. A Location-aided Power Aware Routing Protocol in Mobile Ad Hoc Networks

Yuan et al., [10] proposed a location-aided power-aware routing (LAPAR) protocol that dynamically makes local routing decisions so that a near-optimal power-efficient end-to-end route is formed for forwarding data packets. In this protocol basically a relay region, on the basis of neighbor's position, was constructed by a forwarding node and packet is forwarded to the node whose relay region covers the destination. Greedy decision is made if there is more than one node which covers the destination in their relay region. Authors proposed an alternative backup algorithm where the greedy algorithm fails to discover a power efficient route. Algorithm made local decisions based on the geographical location of the neighboring nodes and considered only unidirectional links. Authors simulated the LAPAR algorithm for stationary environment and the nodes were uniformly distributed over a square region of 400m on each side. Results proved that as the number of nodes deployed in this region increased, the density and connectivity increased. The success rate increased accordingly. Furthermore authors compared the power consumption of LAPAR with GPSR and the results showed that LAPAR consumes less power as compared to GPSR. Results of LAPAR's energy consumption levels with node mobility showed that the average power consumption per unit distance was significantly lower and the motion of nodes does not significantly affect the power consumption.

3.3. Wireless Sensor Networks and Mobile Ad hoc Networks Simulators

The goal for any simulator is to accurately model and predict the behavior of a real world environment. In sensor and mobile networks it is very difficult to buy the hardware and physically implement the proposed protocol .Similarly testing the protocol in desired environment is a tedious and time consuming task thereby simulation based testing is a wise step to follow. A number of published papers proved their ideas and concepts through simulations. Simulation provides advantages that include lower cost, ease of implementation, time saving, practicality of implementing large scale networks etc. Following is a literature survey in this regard.

3.3.1. A Survey of Simulation in Sensor Networks

Curren, [11] presented the paper which aids the developer in selection of specific simulation tools. Author has listed simulators and emulators for wireless sensor networks and Mobile ad hoc networks and provided the developer with the advantage and disadvantages of each. Author claims that selection of tool depends upon the environment for which it has to be used. Paper includes thirteen different simulators (ns-2. TOSSIM. EmStar, GloMoSim, OPNET etc) depending on their features, their publishing results and popularity. Author proposed to select the simulator or emulator as desired and if developer decides to build a simulator or emulator he may follow bottom up or top down approach. Paper has two purposes. First, by knowing the strengths and weaknesses of a number of different simulators were valuable because it allows users to select the one most appropriate for their testing. Second, the developers of new simulators were well served knowing what has worked in previous simulators and what has not.

3.3.2. Real-World Experiences with an Interactive Ad Hoc Sensor Network

Mark et al., [12] described a practical application of DSDV on TinyOS. Variants of DSDV were straight DSDV, DSDV with asymmetric link detection, DSDV with link quality monitoring are studied. Techniques for reducing packet loss, quality-based routing, passive acknowledgment, at link and network layer, were explored in this paper. Results of 24 and 48 node experiments showed that both the quality-based routing and asymmetric link detection become less effective at reducing the packet loss rate as the size of the network increases.

3.4. Problem Statement

In the broadcasting task, a message originated from a source node needs to be forwarded to all the other nodes in the network. Solutions are globalized, meaning that each node needs global network information. Mobility of nodes or changes may cause global changes. Therefore topology changes must be propagated throughout the network for any globalized solution. This may result in extreme and unacceptable communication overhead for ad-hoc networks. Therefore, due to limited resources of mobile nodes, it is ideal that each node should decide on its own behavior based only on the information of its one or two neighbors and distances to them. Such routing protocols like Location Aided Routing Protocol reduces the search space and can efficiently save the number of packets sent by the node. As a result of this, less energy will be consumed by the node while flooding the route request messages. Directional flooding will help in power savage as energy depletion is critical factor in the lifetime of the node in the network, for mobile wireless network. Conserving power and carefully sharing the cost of routing packets will ensure that node and network life are increased. The network lifetime can be maximized only by incorporating energy awareness into every stage of wireless sensor network design MANET routing protocols proposed in early stage do not considered energy savage as a major concern. Later on, few researchers introduced power aware metrics like energy consumed per packet, total energy of sent and received packets etc but there exists no comparison between localized and non localized algorithms specifically on a simulation tool specifically designed for mobile ad hoc networks which can simulate large number of nodes and calculates energy consumption per node. Simulators like ns2 and others may cater the energy consumption but either the simulation tools are not suitable for scalable networks or do not calculate the energy consumption at grass root level.

4. Methodology

As discussed in the problem statement in section 3.4, broadcasting the messages in entire network causes a number of disadvantages. It includes wastage of resources, lack of energy conservation, downfall in the lifetime of the network, inefficient bandwidth utilization, unnecessary storage and maintenance of large routing tables etc. therefore there is a dire need for such routing schemes that minimizes the number of packets sent and received and hence can prevent from all sort of disadvantages which broadcasting have. In context to this, localized routing protocol provides the best solution which implements directional flooding hence limiting the network search space and efficiently utilizing the network resources. Therefore we have selected the localized routing protocol and have compared the performance with a non localized routing protocol. Selected protocols were Location Aided Routing protocol and Destination Sequenced Distance Vector Routing Protocol.

Location Aided Routing Protocol is selected due to the fact that it is the first basic routing protocol proposed in the category of geographical location routing protocols. Other location based routing protocol were based on LAR or have other novel ideas. Beauty of LAR lies in the fact that local computation takes place and node only has to know three parameters that is its own location, destination location and the neighbor location. As advancement in hardware technology is also taking place so it is not difficult to capture the physical location of the nodes in the network like GPS and others. After capturing the location information of the nodes and calculating request and expected zones as discussed in Section 2.1.2, there are two LAR schemes. Initially in our research work, we have implemented the LAR scheme I. The basis algorithm is already discussed in section 2.1.2.4. In this scheme source node only has to know the average speed with which the destination node is moving and the coordinates of destination, its own and the neighbor node. Depending on the expected zone and request zone, source node floods the message to the neighboring nodes. The nodes in the expected zone forward the route request sent by source and other discards it. Eventually the message follows the path which directs towards the destination node. As a result the destination node sends the route reply to source node which contains the coordinates of destination nodes besides other relevant information.

Destination Sequenced Distance Vector is the second selected protocol which falls in the category of non localized routing algorithm and hence information about all nodes and edges is required. Basic working of DSDV is already discussed in section 2.2. It is also a proactive routing protocol so periodic knowledge of active nodes status is also mandatory. Overhead of maintaining routes to each destination is also involved in case of DSDV. While implementing the protocol, maximum number of hops was selected to be 127 and it supports sending data to only one DSDV destination. 8 bit sequence number for transmitted route announcements was generated.

Besides establishing correct and efficient route to the destination, an important goal or the routing protocol is to maintain the network life for greater time interval. As discussed in section 1.3 and 3.4, this goal can be achieved if the mobile node's energy can be saved not only when it is active but also when it is inactive. In Wireless sensor network, data is mostly sunk into a central node or the base station. Hence we can say that sink nodes acts as a data aggregator and shoulders maximum of the workload. Therefore it is very important to minimize the power consumed by the sink node because if the sink node will crash, entire network will die out. Realizing the importance and dependency of sink node on entire network, we have considered the performance and evaluation of sink node. For both protocols i.e. LAR and DSDV, we have taken the values and readings for the sink node especially.

As discussed earlier there are number of sensor network simulators which provide different set of services. As our main focus is on power savage.we will just consider the simulators which provide some sort of services for power calculations. Some of the famous simulators do not cater the power calculations inherently. Some incorporate power usage but do not appear to be validated against actual hardware applications. PowerTOSSIM is the only simulation tool for the TinyOS applications which performs well for scalable network and gives an accurate per node power estimation. PowerTOSSIM includes a detailed model of hardware energy consumption based on the Mica2 sensor node platform. In PowerTOSSIM, TinyOS components corresponding to specific hardware peripherals like the radio, EEPROM, LEDs, CPU, ADC and Sensor are instrumented to obtain a trace of each device's activity during the simulation run. To the best of our knowledge, there exists no comparison between the local zed and non localized routing algorithm using TinyOS as an operating system and PowerTOSSIM as a simulation tool. DSDV is one of the implemented proactive protocols in TinyOS but performance of nodes was never evaluated before using PowerTOSSIM. We have implemented for the first time, a localized routing protocol i.e. Location Aided Routing Protocol and then compared its performance with DSDV using PowerTOSSIM as a simulation tool.

Chapter 5

Implementation

5. Implementation

This chapter will list all the components and interfaces used by the two applications. One is lar1_0.4 and other is DSDV. Before discussing the details of components and interfaces, it is important to tell the method of reading and analyzing the component graphs which will be listed for the specific application.

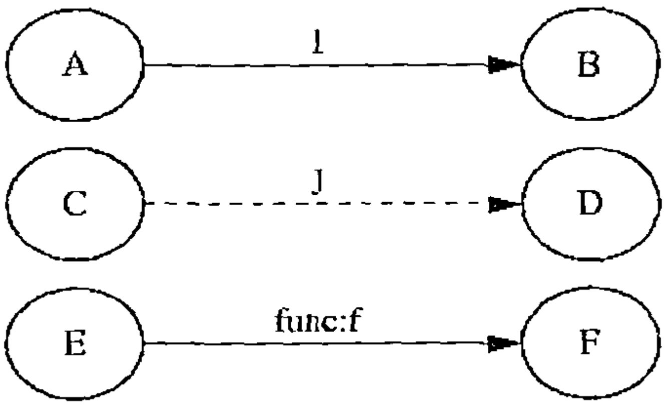


Figure 5-1: Help of Component Graph

In Figure 5-1, the first part shows the situation when **A** requires interface **I** and **B** provides **I**. For this purpose **A** and **B** are wired together. Second part of the figure explains when **C** and **D** both require or both provide **J**. The direction of the arrow indicates that the original wiring is "**C = D**". Third part of the figure shows when **E** requires a function **f** and **F** provides function **f**.

5.1. Application # 1: lar1_0

Application # 1 i.e. lar1_0 has used the components at different levels. First level is the list of components coming under the umbrella of the application. Second list is about the platform specific components. In our case it is PC and not others like mica2, mica. Third list is about the System specific components. We will discuss in detail each list in the following sections.

5.1.1. Components used by Application

This section will include the components graph for the entire application as well as the component details and their functionalities. Interface with their description will also be discussed in this section.

NoUART, SingleHopManager M, IFlood, IFloodM, lar1_0M, SysTimeM,PromiscuousCommNoUART, SingleHopManager, SingleTimer, SingleTimerM and lar1_0 are the main components used by the application. .Each component may have the list of interfaces provided, interfaces required, Function Index and its own Component Graph. Now we will discuss each component in detail.

a) lar1_0

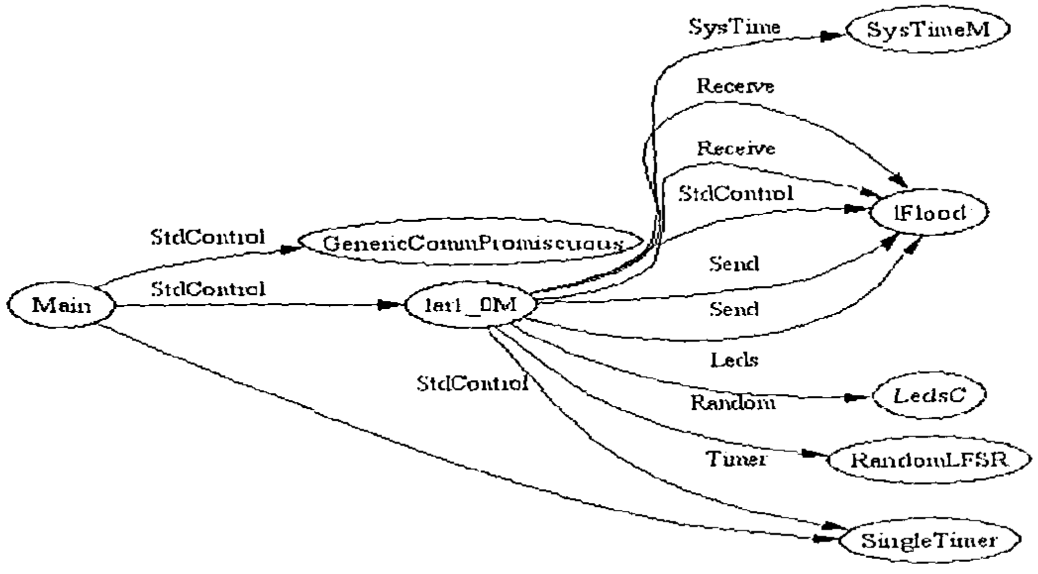


Figure 5–2: Component Graph

b) lar1_0M

Required Interfaces

Major interfaces required by application#1 are:

- **Timer** and **SysTime** to generate system time twice.
- **Send** and **Receive** to send and receive route requests and route replies.
- **StdControl** to provide the initial and final control

Variables

Three Message Buffers of type **TOS_Msg** were declared and used while two message pointers of type **TOS_MsgPtr** were also used. Structures for **LAR_Rreq_Msg** and **LAR_Rreply_Msg** were defined by us according to the algorithm requirement. Besides all these, number of variables was declared to fulfill the protocol needs like:

- uint32_t **time0**[1000]

- uint32_t **time1**[1000]
- uint8_t **ExpectedZoneRange**[1000]

Function Index

command result_t **StdControl.init** (void)

This is an initialization command.

void **calRZone**(int node1, int node2)

This function calculates the request zone as required by LAR scheme I.

uint8_t **sublinkload**(TOS_MsgPtr msg, uint8_t **buf)

It gets the pointer to the TOS_Msg data.

uint8_t **linkload**(TOS_MsgPtr tosmsg, uint8_t **flood)

It gets the pointer to the SHop_Msg data.

uint8_t **larload**(TOS_MsgPtr msg, uint8_t **lar)

It gets the pointer to the **Flood_Msg** data, i.e., actually the LAR RREQ message.

void **getLARPtr** (TOS_MsgPtr rrqMsg)

It gets the pointer to the LAR_Rreq_Msg.

task void **sendRreq**(void)

This is a task for sending the route request

task void **frwrdRreq**(void)

This task forwards the route request.

task void **sendRreply**(void)

This task sends route reply.

task void **frwrdRreply**(void)

This task forwards the route reply.

command result_t **StdControl.start** (void)

This command starts the necessary controls.

command result_t **StdControl.stop** (void)

This command stops the necessary controls.

event result_t **Timer.fired** (void)

This event fires the timer.

event TOS_MsgPtr **ReceiveReq.receive** (TOS_MsgPtr pMsg, void *payload, uint16_t payloadLen)

This event receives event for both route requests and route replies.

event TOS_MsgPtr **ReceiveRreply.receive** (TOS_MsgPtr pMsg, void *payload, uint16_t payloadLen)

This event is for an intermediate node.

event result_t **SendRreq.sendDone** (TOS_MsgPtr sentBuffer, bool success)

This is a send done event for sending route request.

event result_t **SendRreply.sendDone** (TOS_MsgPtr sentBuffer, bool success)

This is a send done event for sending route reply.

void **getLARPtr**(TOS_MsgPtr rrqMsg)

This function gets the pointer to the **LAR_Rreq_Msg**. This is a wrapper function over the payload functions.

c) NoUART

Provided Interfaces

StdControl, **SendVarLenPacket**, **BareSendMsg** and **ReceiveMsg** are the major interfaces provided by this component.

Function Index

Besides normal send, major function of this component is:

command result_t **SendVarLenPacket.send** (uint8_t *packet, uint8_t numBytes)

d) PromiscuousCommNoUART

Required Interfaces

result_t **sendDone**(void)

Provided Interfaces

Major interfaces provided by this component are: **CommControl**, **SendVarLenPacket** **SendMsg** and **ReceiveMsg**.

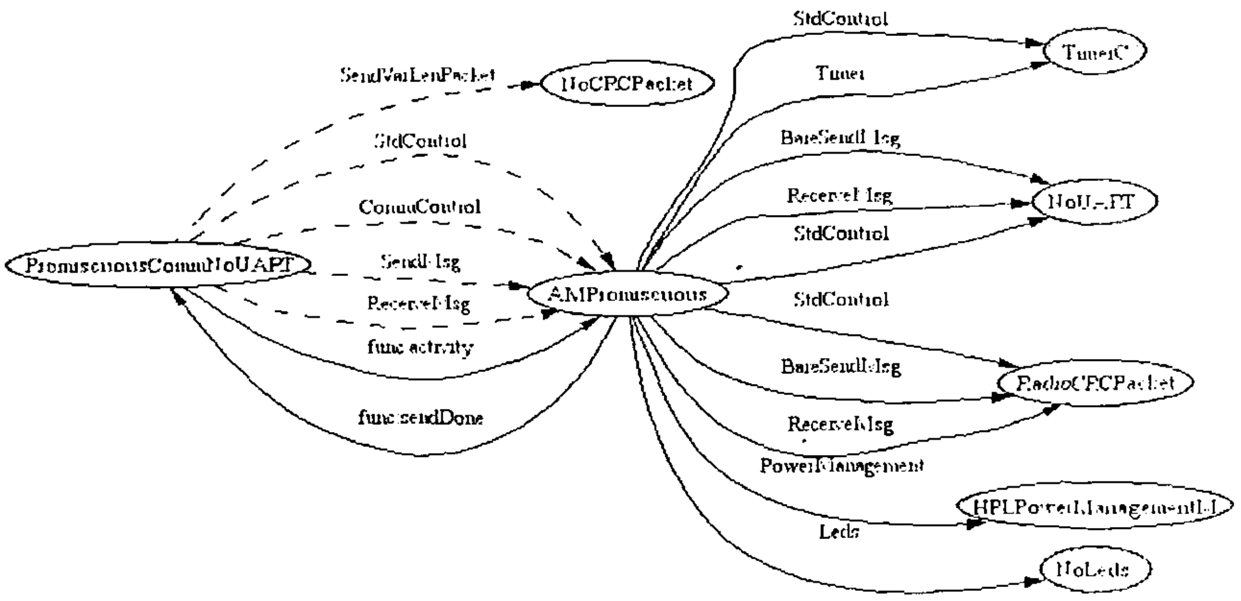


Figure 5–3: Component Graph

e) SingleHopManagerM

Required Interfaces

Major interfaces required by this component are: **CommControl**, **SendMsg**, **ReceiveMsg** and **Payload**.

Provided Interfaces

Major Interfaces provided by this component are **SequenceNumber** and **NetStat**.

Function Index

Major functions are:

event void **SequenceNumber.updateSeqNum** (wsnAddr addr, uint8_t seqNum)

command uint16_t **NetStat.sentMessages** (void)

command uint16_t **NetStat.receivedMessages** (void)

f) SingleHopManager

Provided Interfaces

This component provides the major interfaces of **Payload**, **SequenceNumber** and **SingleHopMsg**.

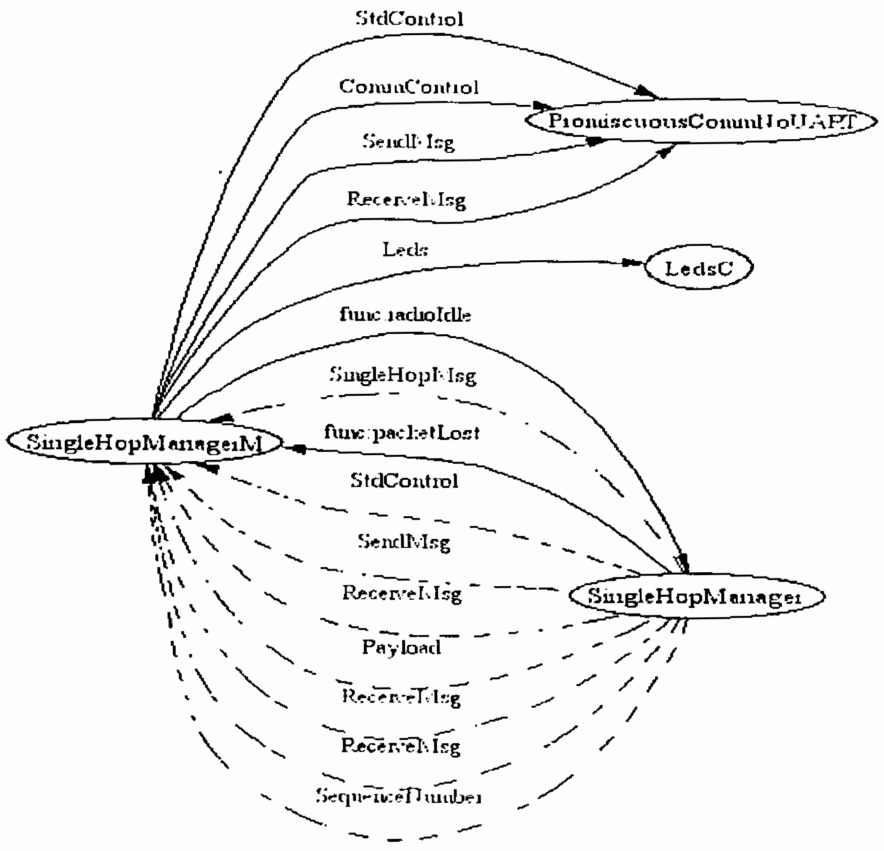


Figure 5–4: Component Graph

g) SingleTimer

Provided Interfaces

Two interfaces i.e. **Timer** and **StdControl** are provided by this component.

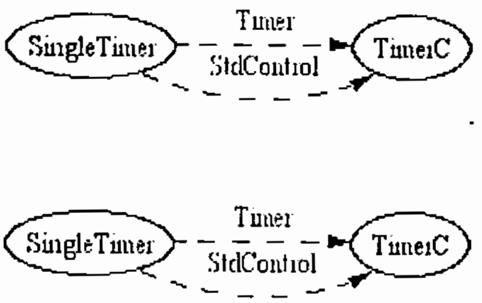


Figure 5–5: Component Graph

h) SysTimeM

Provided Interfaces

This component just provides the interface for **SysTime**.

Function Index

This component uses two functions to get 16 and 32 bit system times so that the returned vales may be used to calculate the expected zone for LAR.

i) IFlood

Required Interfaces

result_t **radioIdle**(void)

Provided Interfaces

Major interfaces provided by this component are **Send**, **Receive**, **Intercept**. **SingleHopMsg**, **MultiHopMsg** and **FloodMsg**.

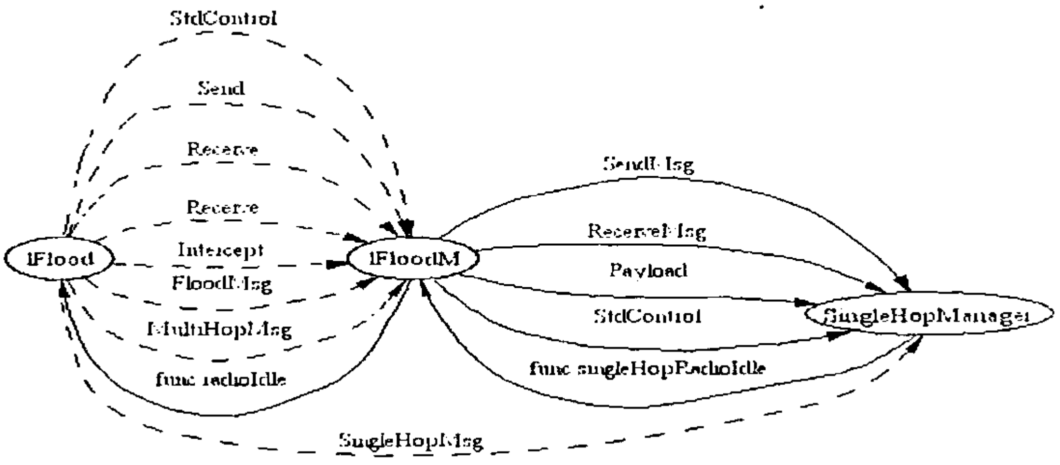


Figure 5–6: Component Graph

j) IFloodM

Required Interfaces

StdControl is to initiate and stop the controls.

RoutingControl, **Payload**, **SendMsg** and **ReceiveMsg** are used to perform basic flooding mechanism.

Provided Interfaces

Main interfaces provided by this file are **Send**, **SendMHopMsg**, **Receive**, **Intercept**, **MultiHopMsg** and **FloodMsg**.

result_t **singleHopRadioIdle**(void)

Function Index

Major functions used by the module files are: **getFloodPtr**, all functions provided by **MultiHopMsg**, **FloodMsg.getSequenceNum**, **FloodMsg.getTTL**, **Intercept.intercept**.

SendMHopMsg.sendTTL command prepares a message for transmission using the flooding protocol and sends it.

Function Description

task void **retryForwardFloodMsg**(void)

This procedure handles the processing of an incoming flooded message received from a neighbor. It compares the message against the packet signature cache and rebroadcasts if TTL > 0 and returns a free message pointer.

5.1.2. Components Used By Platform i.e. pc

Following are the components which are specific for the pc platform and can be studied in detail by going through directory tos/platform/pc.

ChannelMonC ,HPLClock, ChannelMonC ,HPLClock ,HPLPotC , InjectMsg, LedsC, LedsM, HPLPowerManagementM , UARTNoCRCPacketM , MicaHighSpeedRadioM, UARTFramedPacket, RadioCRCPacket, RadioTimingC,SecDedEncoding, SpiByteFifoC, PowerStateM, TimerC , Nido, Main

5.1.3. Components used by System

Following are the components which are specific for the pc platform and can be studied in detail by going through directory tos/system.

- AMPromiscuous
- ClockC
- CrcFilter

- GenericCommPromiscuous
- NoCRCPacket
- NoLeds
- PotC
- PotM
- RandomLFSR
- Time

5.1.4 Interfaces used by Application

Following is the list of the interfaces used by the application components mentioned above. This section will briefly list the purpose of the used interfaces.

Table 5-1: Interfaces used by Components in the Application

No.	Interface Name	Purpose
a)	BareSendMsg.nc	Functionality for sending a raw packet buffer; unaware of message structure (besides length). This is in contrast to SendMsg, which takes parameters for message headers.
b)	ByteComm.nc	A byte-level communication interface. It signals byte receptions and provides a split-phased byte send interface. txByteReady states that the component can accept another byte in its queue to send, while txDone states that the send queue has been emptied.
c)	Clock.nc	The hardware clock interface.
d)	CommControl.nc	This interface defines commands for controlling aspects of the communication layer.
e)	HPLPot.nc	Interface to a variable potentiometer. Calls to increase and decrease must be actualized with a call to finalise. Because this is a direct hardware interface, it does not maintain state; checks for potentiometer bounds must be performed by a higher-level interface.
f)	Intercept.nc	Signals that a message has been received, which is supposed to be forwarded to another destination.

g)	Leds.nc	Abstraction of the LEDs.
h)	Pot.nc	The Pot interface allows users to adjust the potentiometer on the input to the RFM radio which controls the RF transmit power or connectivity range.
i)	PowerManagement.nc	Adjust the power state of a component.
j)	Random.nc	This is the interface to a simple pseudorandom number generator. Currently this interface is implemented by the RandomLFSR, which uses a linear feedback shift register to generate the sequence and mote address to initialize the register.
k)	Receive.nc	Received a message buffer addressed.
l)	ReceiveMsg.nc	TinyOS AM packet reception interface.
m)	Send.nc	Send a message buffer with a data payload of a specific length.
n)	SendMsg.nc	Basic interface for sending AM messages. Interface to the basic TinyOS communication primitive.
o)	SendVarLenPacket.nc	Interface for sending arbitrary streams of bytes.
p)	StdControl.nc	The TinyOS standard control interface. All components that require initialization or can be powered down should provide this interface. start() and stop() are synonymous with powering on and off, when appropriate. On boot, the init() of all wired components must be called.
q)	SysTime.nc	This interface provides access to a free-running CPU timer that is started at startup. The current value of this timer is NOT supposed to be changed. On the MICA2 platform the current implementation uses a 1/8 prescaler that results in a 921.6 KHz clock frequency.
r)	Timer.nc	This interface provides a generic timer that can be used to generate events at regular intervals.

5.2. Application # 2: DSDV

This application was available in the contributed code provided by TinyOS. Application uses some of interfaces provided by the contributed code as well as the interfaces provided by the TinyOS. Basic functionality of DSDV is maintained in a library provided by contributed code. We will discuss only the application specific components and interfaces.

5.2.1. TraceRouteTest

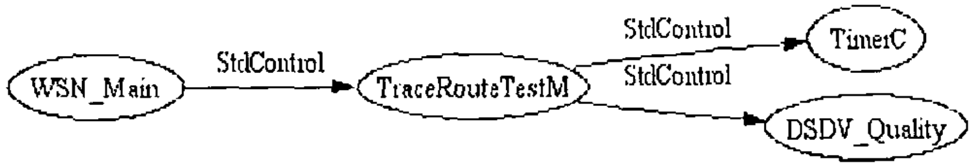


Figure 5–7: Component Graph

5.2.2. TraceRouteTest

Required Interfaces

Major interface required by **TraceRoutTestM** is only of **StdControl**.

Provided Interfaces

Major interface required by **TraceRoutTestM** is only of **StdControl**.

Function Index

Major functions only include the **init** ( ), **start** ( ) and **stop** ( ) functions provided by **StdControl**.

5.2.3. DSDV_Quality

Required Interfaces

Major interfaces required by this interface are **Send**, **Receive**, **Intercept**, **SingleHopMsg** **MultiHopMsg** and **DSDVMsg**.

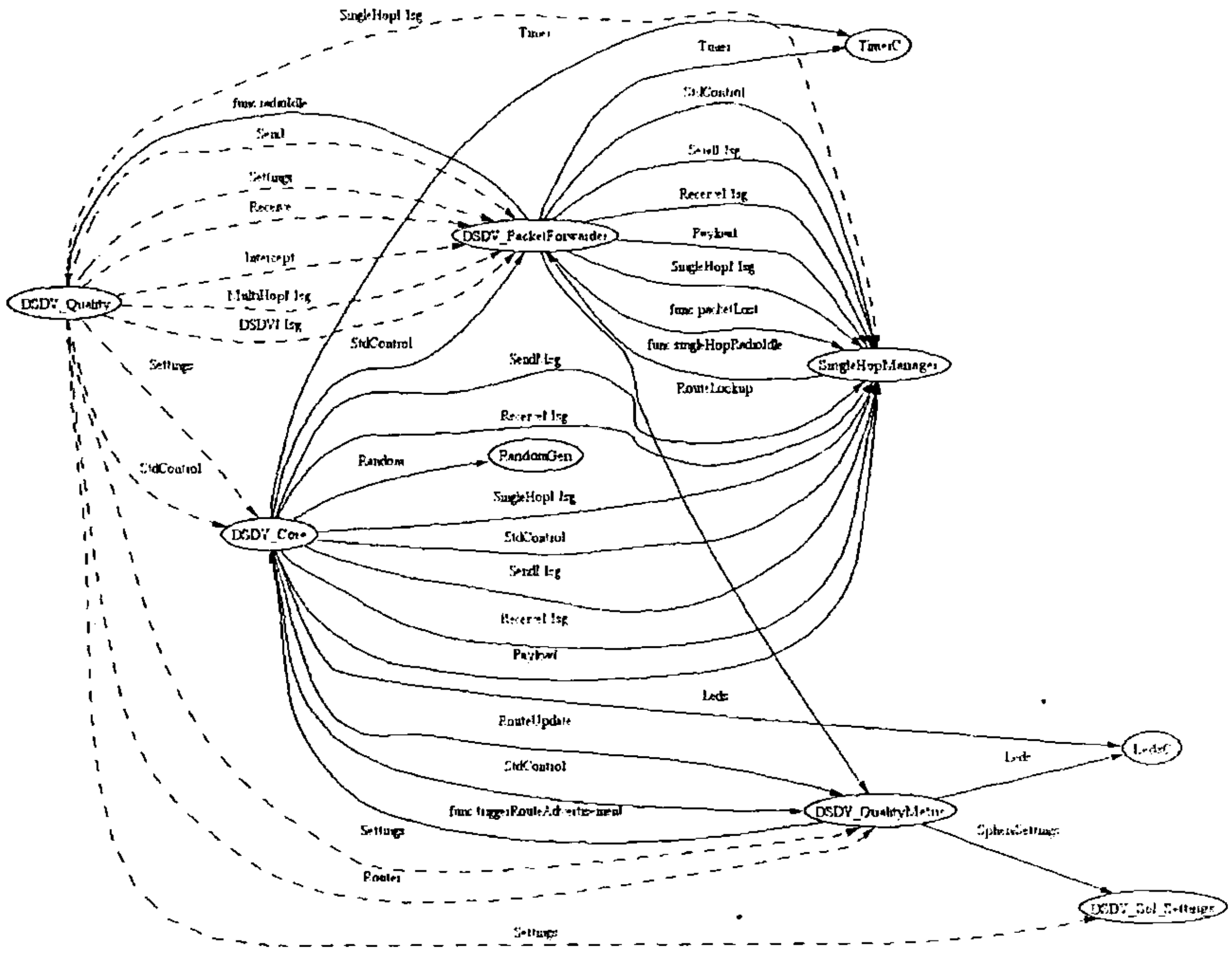


Figure 5–8: Component Graph

5.2.4. DSDV_Core

Required Interfaces

Major interfaces required are **RouteUpdate**, **SendMsg**, **Payload** and **ReceiveMsg**.

Provided Interfaces

Major interface provided by DSDV_CORE is **Settings**.

Function Index

Major functions used by this component are the one provided by **StdControl,SendMsg** which is a Basic interface for sending AM messages. Besides this, functions of interface **ReceiveMsg** and **Timer** are used.

5.2.5. DSDV_PacketForwarder

Required Interfaces

Major interfaces required by this component are **SingleHopControl**, **SendMsg**, **ReceiveMsg** and **Payload**

Provided Interfaces

Interfaces provided by this component are **Send**, **Receive**, **Intercept** and **MultiHopMsg**.

5.2.6. DSDV_QualityMetric

Required Interfaces

Only two interfaces are required by this component and they are **Leds** and **SphereSettings**.

Provided Interfaces

RouteUpdate and **RouteLookup** are the two major interfaces provided by this component.

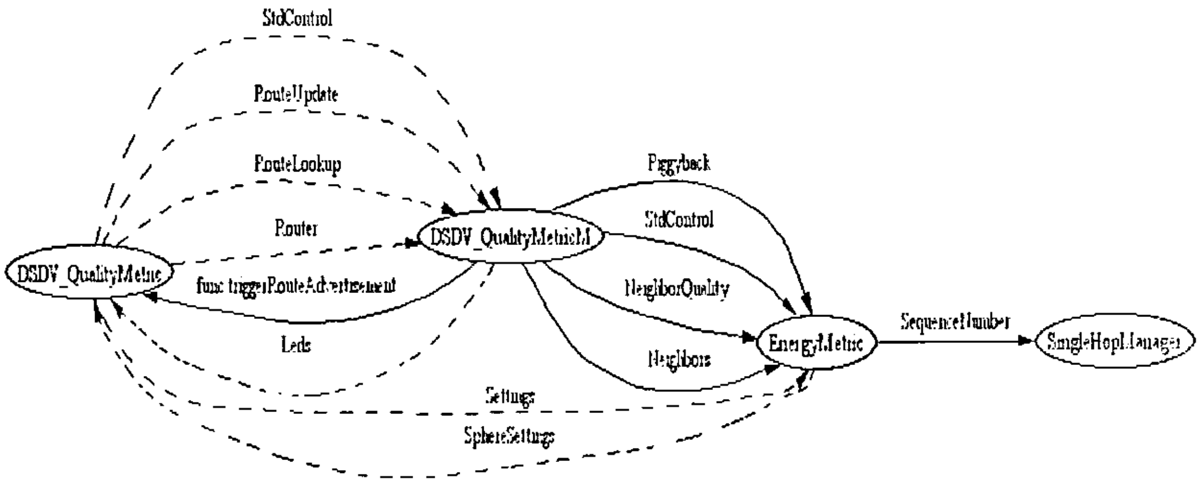


Figure 5–9: Component Graph

5.2.7. EnergyMetric

Required Interfaces

SequenceNumber and **SphereSettings** are the two major interfaces required by this component.

Provided Interfaces

Neighbors, **NeighborQuality** and **Piggyback** are the interfaces provided by this interface.

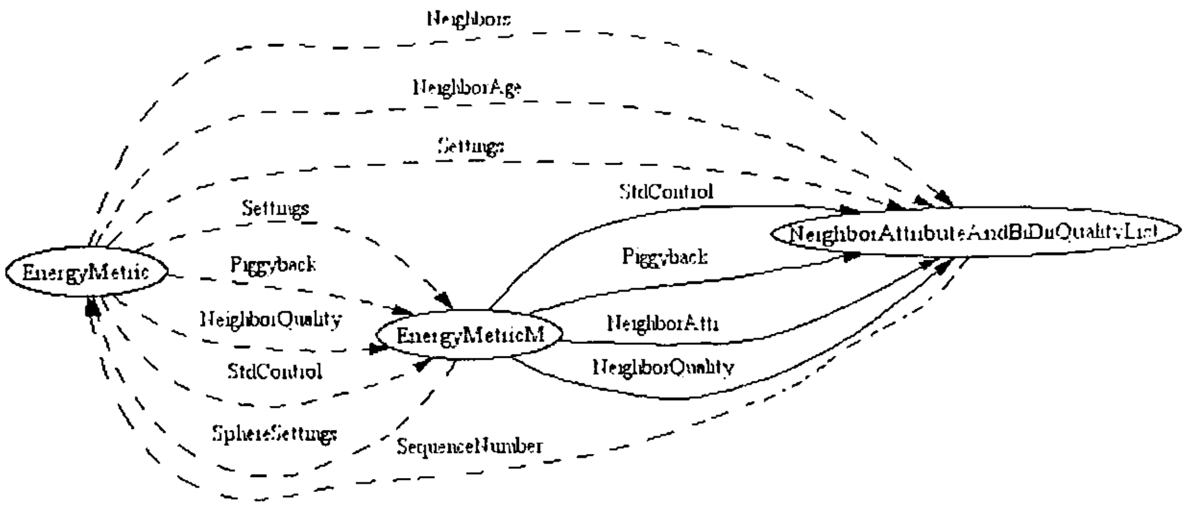


Figure 5–10: Component Graph

5.2.8. PowerStateM

Require Interfaces

PowerState is the only interface required by the component **PowerStateM**.

Function Index

List of functions includes the function for mica2 hardware i.e. separate functions are implemented for ADC, LEDS, Sensors, Radio state, CPU state, EEPROM. Each function used the commands provided by interface **PowerState** which is an interface for the PowerState functions.

Chapter 6

Results

6. Results

This chapter describes the simulation parameters and simulation results obtained after implementing the protocols in TinyOS.

6.1. Simulation Parameters

TinyOS gives the freedom to select the number of nodes in a network hence we started with 10 nodes in a network and simulated till 50 nodes. However, to cross check the scalability of PowerTOSSIM as well as the energy savage by greater number of nodes, we have simulated till 100 nods. Figure 6-4 depicts the results. Mote 0 was selected as a default sink node and performance of the protocols was compared using the average values of the sink node i.e. mote0. Duration of each simulation run was 35 virtual seconds. 20 simulation runs had been carried out. Energy model for mica2 [3], provided by PowerTOSSIM, was considered. This model computes the energy consumed by the radio, CPU, EEPROM, ADC, sensors and LEDs separately for each simulated mote. At the end, it gives the total energy consumed by the mote. For DSDV we used the contributed code which was available with TinyOS version 1.x [15], whereas LAR scheme 1 was implemented by us in the same TinyOS version.

6.2. Simulation Results

Averages values for the sink node in case of LAR and DSDV was considered for two types of results. In first case, average energy consumed by the sink node, in each of the selected protocol, was considered where as in second case, average power saved by LAR and DSDV was catered. Both the cases will be discussed in the section below.

6.2.1. Average Energy Consumed by Selected Protocols

Figure.6-1 shows the energy consumption of the sink node in case of both LAR and DSDV. Average values of energy were plotted against the number of nodes in the network. The results indicates that LAR consumes less power as compared to DSDV due to its location based routing scheme. In LAR, flooding is confined to the number of the nodes existing in request zone hence eliminating unnecessary overhead of flooding messages to all the nodes in the network where as in DSDV, messages are flooded to each node in the network hence increasing the load on the sink node. Therefore we can say that when energy of the network is critical, we should adopt a routing scheme which involves less routing overhead as in LAR. From simulations, it can be inferred that LAR performs better in route establishment as compared to DSDV since it is used as and when required. Moreover, due to the directional flooding used in the route establishment, it performs better than DSDV that uses periodic route updates.

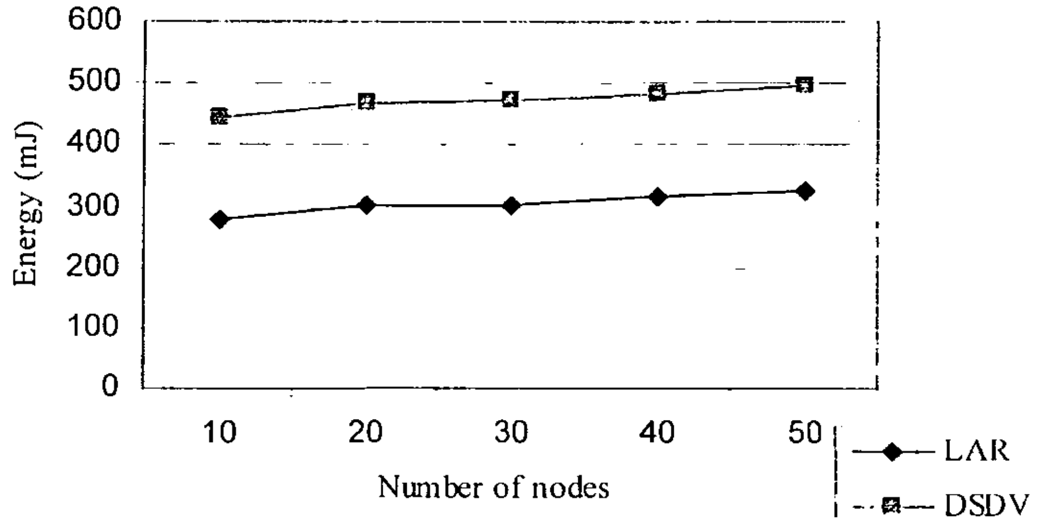


Figure 6–1: Average energy consumed at Sink Node by LAR and DSDV

6.2.2. Average Energy Saved by Selected Protocols

Figure.6-2 illustrates the percentage of average power saved by LAR against different node densities where percentage is calculated as:

$$\frac{\textbf{Power used by DSDV} - \textbf{Power used by LAR}}{\textbf{Power used by LAR}}$$

The results indicate that power consumption is directly proportional to the number of the nodes in the network. Therefore the percentage of power saved is more at less number of the nodes and vice versa. It can be inferred from these results that LAR saves 59% of available energy for a small sized network. As the network density increases, energy savage by LAR decreases by approximately 7% and drops to 52%.

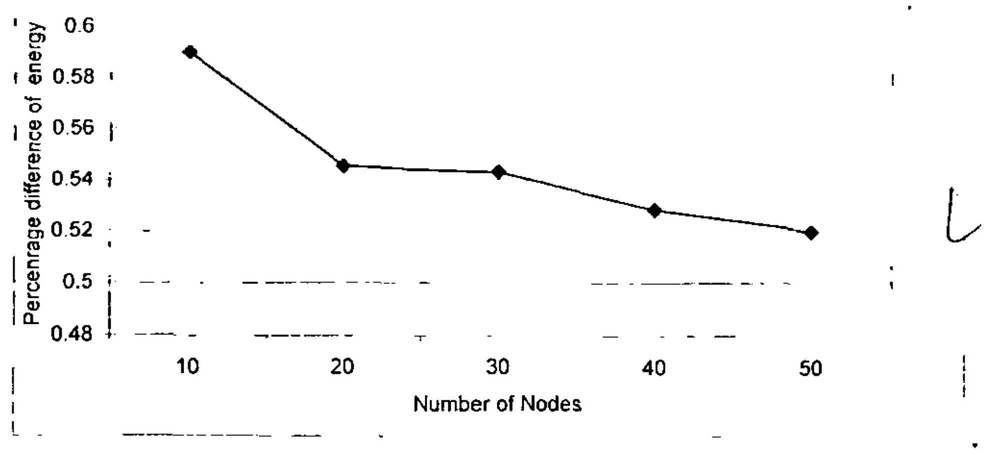


Figure 6–2: Average energy saving in LAR for different node densities

On the other hand, as illustrated in Figure 6-3, the energy conservation is about 37% in case of DSDV for a small scale network. However, this value drops about 34 % for a medium sized network topology, which in contrast to LAR is a relatively less variation. These statistics show that LAR conserves more energy, i.e., approximately 22% for small scaled networks and 18 % for medium scaled networks as compared to DSDV.

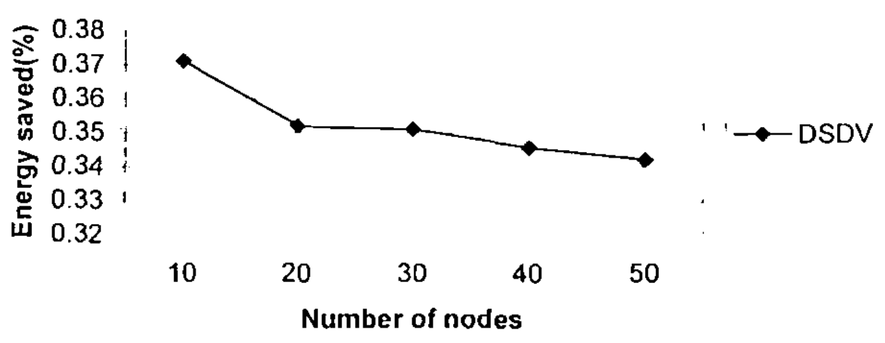


Figure 6–3: Average energy saving in DSDV for different node densities

Figure 6-4 depicts the comparisons of the percentages of energy saved by both LAR and DSDV. Number of nodes ranged from 10 to 100. The results shows that percentage of energy saved reduced with increase in number of nodes. However, energy saved by LAR is still more as compared to DSDV. There is almost a difference of 12% between the energy saved by DSDV and LAR. From the results, it is also proved that PowerTOSSIM scales well as the number of nodes increases.

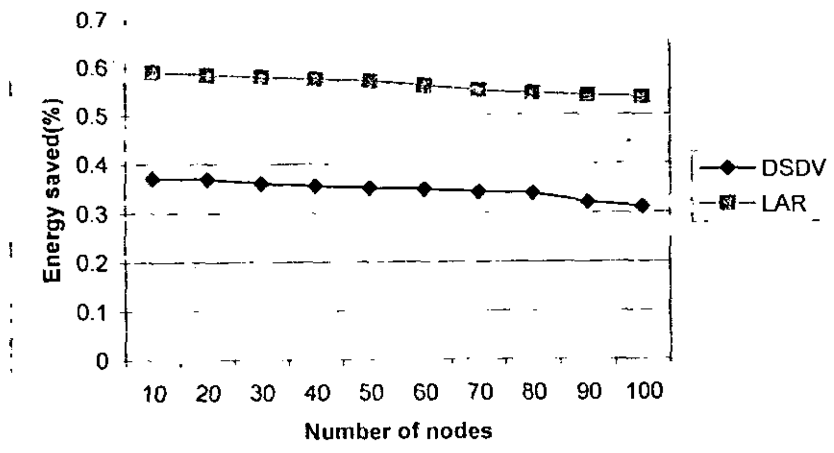


Figure 6–4: Average energy saving in LAR and DSDV for node densities

Chapter 7

Conclusions and Future Enhancements

7. Conclusion and Future Enhancements

This chapter contains the conclusions drawn form our research work. Furthermore it narrates our future plans for enhancements.

7.1. Conclusions

In the thesis, we compared the performance of LAR, a localized ad hoc routing protocol, with DSDV, a non localized routing protocol, by characterizing them to a sensor network scenario. The LAR protocol utilizes the location information for the mobile hosts to reduce the search space for a desired route. As a result the overhead of route discovery can be reduced. The location information may be found by using the global positioning system. LAR is fully distributed where message flooding is only directional and according to pre- calculated request zone area while in DSDV message is just flooded to all the neighboring nodes. We have shown by simulation results that performance of LAR protocol in terms of energy consumption is better as it consumes less energy. However, the performance of said protocols for large scale networks is still to be analyzed. In addition to this, we intend to explore the effects of nodes mobility on the performance of LAR and DSDV in TinyOS using Power TOSSIM as a possible future work.

7.2. Future Enhancements

We plan to implement second LAR Scheme, LAR II, and compare its performance with LAR Scheme I where energy conservation will be the metric. Further, we intend to compare both LAR schemes with DSDV.

Beside this, we intend to expand the horizon of our research work by bringing more location based protocols under the concerned umbrella and then comparing their performance with non localized routing protocols as well as among each other. The performance criteria will be energy consumption by the concerned nodes.

References

References

—

[1] **Power aware routing in mobile ad hoc networks,"** Suresh Singh, Mike Woo, CS Raghavendra, MOBICOM, 1998.

[2] **System architecture directions for networked sensors,"** J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, In Architectural Support for Programming Languages and Operating Systems, pp. 93 –104, Boston, MA, USA, Nov. 2000.

[3] **Simulating the power consumption of large-scale sensor network applications,"** Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh, In Proc. of Conference on Embedded Networked Sensor Systems, 2004.

[4] **A survey on position based rout ing in mobile ad hoc networks,"** Martin Mauve, Jogg Widmer and Hanees Hartenstein. *IEEE Network Magazine*, 15(6):30 - 39, November 2001.

[5] **Position-based multicast routin g in ad-hoc wireless networks,"** Jörg Widmer, Martin Mauve, Hannes Hartenstein, and Holger Füßler. In *ACM MobiHoc '03*, Annapolis, MD, USA, June 2003.

[6] **Probabilistic Geographic Routing in Ad Hoc and Sensor Networks,"** T. Roosta, M. Menzo, and S. Sastry, International Workshop on Wireless Ad-hoc Networks, 2005.

[7] **A Simulation Study of an Energy Effi cient Routing Protocol for Mobile Ad Hoc Networks,"** Xiaojiang Du, In Proc. of Annual Simulation Symposium 2004, pp. 125 – 131.

[8] **Power-Aware Localized Routing in Wireless Networks,"** Ivan Stojmenovic, Xu Lin, IEEE Transactions on Parallel and Distributed Systems, vol. 12 no.11, pp.1122 – 1133, November 2001.

[9] **Minimum Energy Mobile Wireless Networks,** " Volkan Rodoplu and Teresa H. Meng, IEEE journal on Selected Areas in Communications, vol. 17, no. 8, August 1999.

[10] **"A Location-aided Power-aware Routing Protocol in Mobile Ad Hoc Networks,"** Yuan Xue, Baochun Li, In Proc. of IEEE Globecom 2001, San Antonio, Texas, November, 2001.

[11] **"A Survey of Simulation in Sensor Networks,"** David Curren, University of Binghamton, unpublished.

[12] **"Real-World Experiences with an Interactive Ad Hoc Sensor Network,"** Mark D. Yarvis, W. Steven Conner, Lakshman Krishnamurthy, Jasmeet Chhabra, Brent Elliott, and Alan Mainwaring, In Proc. of the International Workshop on Ad Hoc Networking (IWAHN 2002), Vancouver, British Columbia, Canada, August, 2002.

[13] **"Location-Aided Routing (LAR) in Mobile Ad Hoc Networks,"** Y.B. Ko and N.H. Vaidya, Wireless Networks, vol. 6, pp. 307 – 321, 2000.

[14] **"Highly Dynamic Destination-Sequenced Distance-Vector Routing for Mobile Computers,"** C. Perkins and P. Bhagwat ACM, In Proc. of SIGCOMM Conference on Communications Architectures, Protocols and Applications, pp. 234 – 244, 1994.

[15] **"System Architecture Directions for Networked Sensors,"** J. Hill, R. Szewcyk, A. Woo, D. Culler, S. Hollar, K. Pister, In Proc. of ASPLOS 2000.

[16] **"nesC1.1 Language Reference manual"**, D. Gay, P. Levis, D. Culler, E. Brewer May2003, www.tinyos.net/tinyos-1.x/doc/nesc/ref.pdf

[17] **"TOSSIM: Accurate and scalable simulation of entire TinyOS applications,"** P. Levis, N. Lee, M. Welsh, and D. Culler, In Proc. of the First ACM Conference on Embedded Networked Sensor Systems (SenSys) 2003, pp. 126 – 137, Nov. 2003.

[18] TinyOS Official website : www.tinyos.net

[19] **"The ns manual,"** K. Fall and K. Varadhan, url: www.isi.edu/nsnam/ns/doc/index.html

[20] Sensors, May 1, 2006 by: David E. Culler, PhD, Arch Rock Corp.

[21] **"Power Aware Routing in Mobile Adhoc Networks"**, S. Singh, M. Woo, C. S. Raghavendra, In Proceedings of Mobicom '98, Dallas, Oct. 1998.

[22] **"Implementation and Real World Evaluation of Routing Protocols for Wireless Adhoc Networks"**, H. Lundgren, Licentiate Thesis, Dec 2002, url: http://user.it.uu.se/~henrikl/publications.html

[23] **"A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks"** E. M. Royer, C. Toh, IEEE Journal on Personal Communications, April 1999.

[24] **"Power-aware Source Routing Protocol for Mobile Ad Hoc Networks"**, Morteza Maleki, Karthik Dantu, and Massoud Pedram, In Proc. of the Intl Symposium on Low Power Electronics and Design, pp. 72 – 75, Monterey, California, USA, 2002.

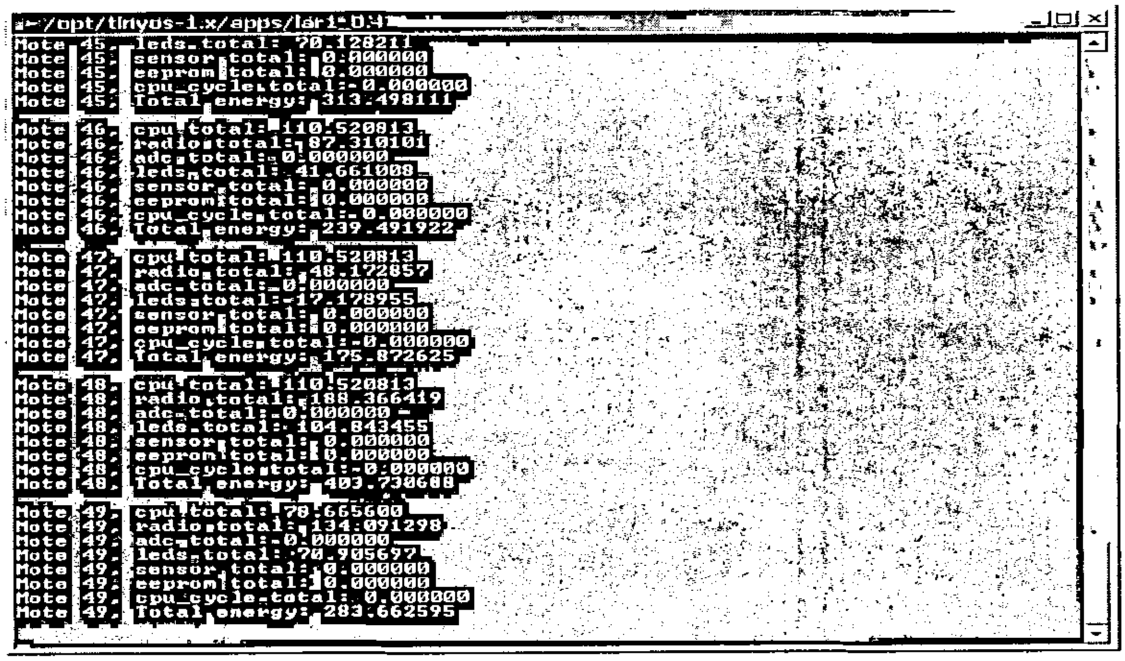
[25] **"Location-based localized alternate, disjoint and multi-path routing algorithms for wireless networks"**, X. Lin and I. Stojmenovic, In Journal of Parallel and Distributed Computing, Vol. 63, no. 1, pp. 22 – 32, 2003.

Appendices

A. Code Screen Shots

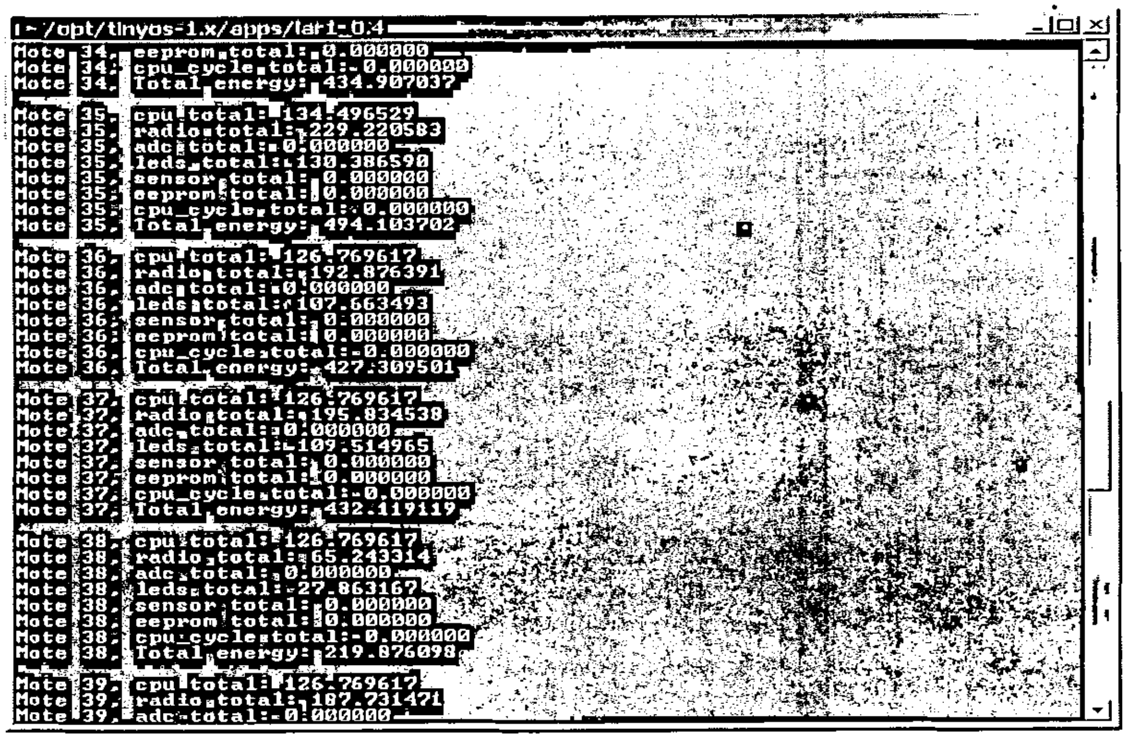
A.1. Lar 1_0.4

Setting the parameters for debugging before execution

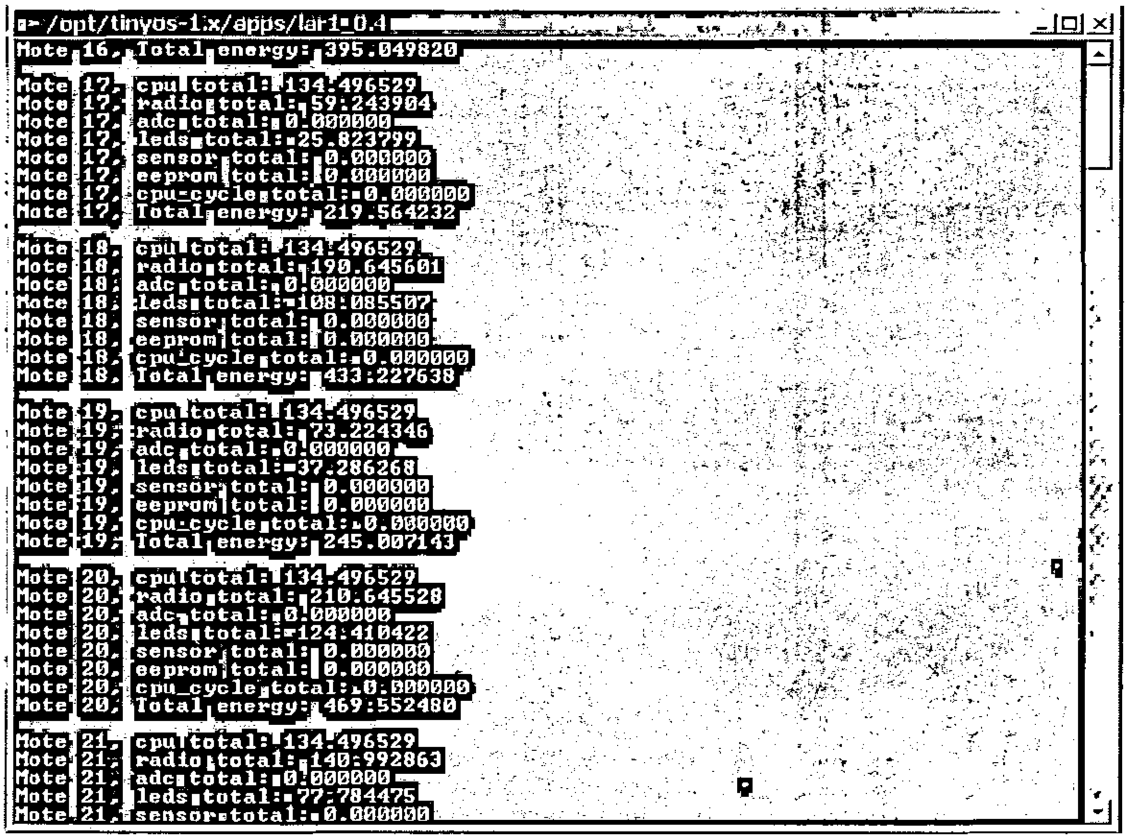


```
~/opt/tinyos-1.x/apps/lar1_0.4
Mote 45, leds_total: 70.128211
Mote 45, sensor_total: 0.000000
Mote 45, eeprom_total: 0.000000
Mote 45, cpu_cycle_total: 0.000000
Mote 45, Total energy: 313.498111

Mote 46, cpu_total: 110.520813
Mote 46, radio_total: 87.310101
Mote 46, adc_total: 0.000000
Mote 46, leds_total: 41.661008
Mote 46, sensor_total: 0.000000
Mote 46, eeprom_total: 0.000000
Mote 46, cpu_cycle_total: 0.000000
Mote 46, Total energy: 239.491922

Mote 47, cpu_total: 110.520813
Mote 47, radio_total: 48.172857
Mote 47, adc_total: 0.000000
Mote 47, leds_total: 17.178955
Mote 47, sensor_total: 0.000000
Mote 47, eeprom_total: 0.000000
Mote 47, cpu_cycle_total: 0.000000
Mote 47, Total energy: 175.872625

Mote 48, cpu_total: 110.520813
Mote 48, radio_total: 188.366419
Mote 48, adc_total: 0.000000
Mote 48, leds_total: 104.843455
Mote 48, sensor_total: 0.000000
Mote 48, eeprom_total: 0.000000
Mote 48, cpu_cycle_total: 0.000000
Mote 48, Total energy: 403.730688

Mote 49, cpu_total: 78.665600
Mote 49, radio_total: 134.091298
Mote 49, adc_total: 0.000000
Mote 49, leds_total: 70.905697
Mote 49, sensor_total: 0.000000
Mote 49, eeprom_total: 0.000000
Mote 49, cpu_cycle_total: 0.000000
Mote 49, Total energy: 283.662595
```

Few Execution Screen shots



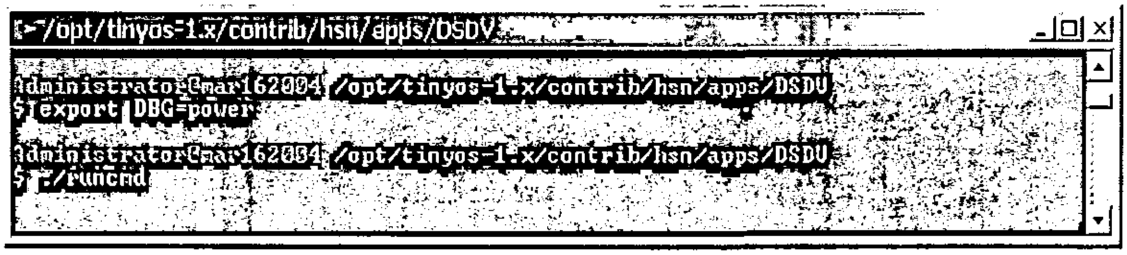
```
~/opt/tinyos-1.x/apps/lar1_0.4
Mote 34, eeprom_total: 0.000000
Mote 34, cpu_cycle_total: 0.000000
Mote 34, Total energy: 434.907037

Mote 35, cpu_total: 134.496529
Mote 35, radio_total: 229.220583
Mote 35, adc_total: 0.000000
Mote 35, leds_total: 130.386590
Mote 35, sensor_total: 0.000000
Mote 35, eeprom_total: 0.000000
Mote 35, cpu_cycle_total: 0.000000
Mote 35, Total energy: 494.103702

Mote 36, cpu_total: 126.769617
Mote 36, radio_total: 192.876391
Mote 36, adc_total: 0.000000
Mote 36, leds_total: 107.663493
Mote 36, sensor_total: 0.000000
Mote 36, eeprom_total: 0.000000
Mote 36, cpu_cycle_total: 0.000000
Mote 36, Total energy: 427.309501

Mote 37, cpu_total: 126.769617
Mote 37, radio_total: 195.834538
Mote 37, adc_total: 0.000000
Mote 37, leds_total: 109.514965
Mote 37, sensor_total: 0.000000
Mote 37, eeprom_total: 0.000000
Mote 37, cpu_cycle_total: 0.000000
Mote 37, Total energy: 432.119119

Mote 38, cpu_total: 126.769617
Mote 38, radio_total: 65.243314
Mote 38, adc_total: 0.000000
Mote 38, leds_total: 27.863167
Mote 38, sensor_total: 0.000000
Mote 38, eeprom_total: 0.000000
Mote 38, cpu_cycle_total: 0.000000
Mote 38, Total energy: 219.876098

Mote 39, cpu_total: 126.769617
Mote 39, radio_total: 187.731471
Mote 39, adc_total: 0.000000
```



```
Mote 16, Total energy: 395.049820

Mote 17, cpu total: 134.496529
Mote 17, radio total: 59.243904
Mote 17, adc total: 0.000000
Mote 17, leds total: 25.823799
Mote 17, sensor total: 0.000000
Mote 17, eeprom total: 0.000000
Mote 17, cpu_cycle total: 0.000000
Mote 17, Total energy: 219.564232

Mote 18, cpu total: 134.496529
Mote 18, radio total: 190.645601
Mote 18, adc total: 0.000000
Mote 18, leds total: 108.085507
Mote 18, sensor total: 0.000000
Mote 18, eeprom total: 0.000000
Mote 18, cpu_cycle total: 0.000000
Mote 18, Total energy: 433.227638

Mote 19, cpu total: 134.496529
Mote 19, radio total: 73.224346
Mote 19, adc total: 0.000000
Mote 19, leds total: 37.286268
Mote 19, sensor total: 0.000000
Mote 19, eeprom total: 0.000000
Mote 19, cpu_cycle total: 0.000000
Mote 19, Total energy: 245.007143

Mote 20, cpu total: 134.496529
Mote 20, radio total: 210.645528
Mote 20, adc total: 0.000000
Mote 20, leds total: 124.410422
Mote 20, sensor total: 0.000000
Mote 20, eeprom total: 0.000000
Mote 20, cpu_cycle total: 0.000000
Mote 20, Total energy: 469.552480

Mote 21, cpu total: 134.496529
Mote 21, radio total: 140.992863
Mote 21, adc total: 0.000000
Mote 21, leds total: 77.784475
Mote 21, sensor total: 0.000000
```

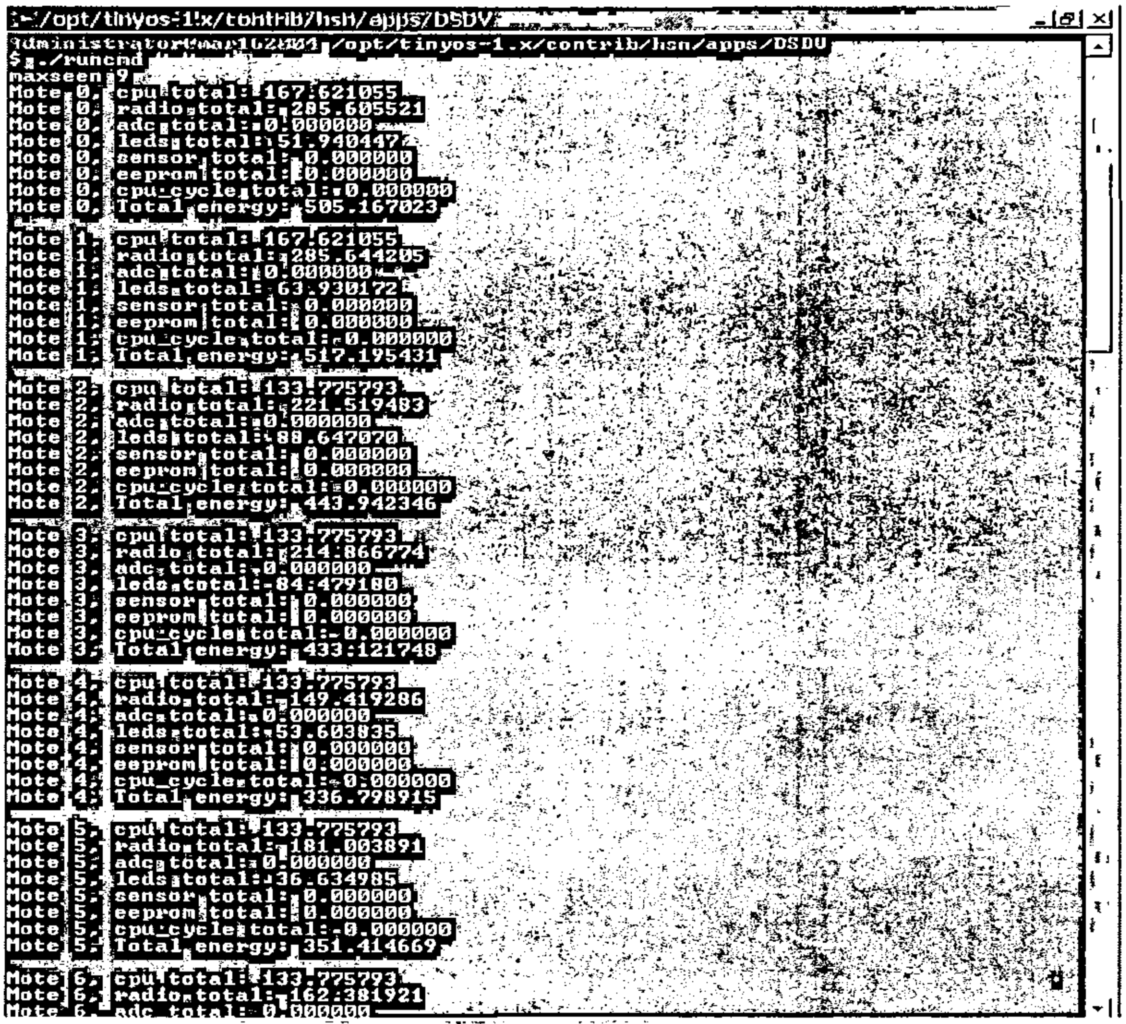
A.2. Screenshots of DSDV

Setting the parameters for debugging before execution

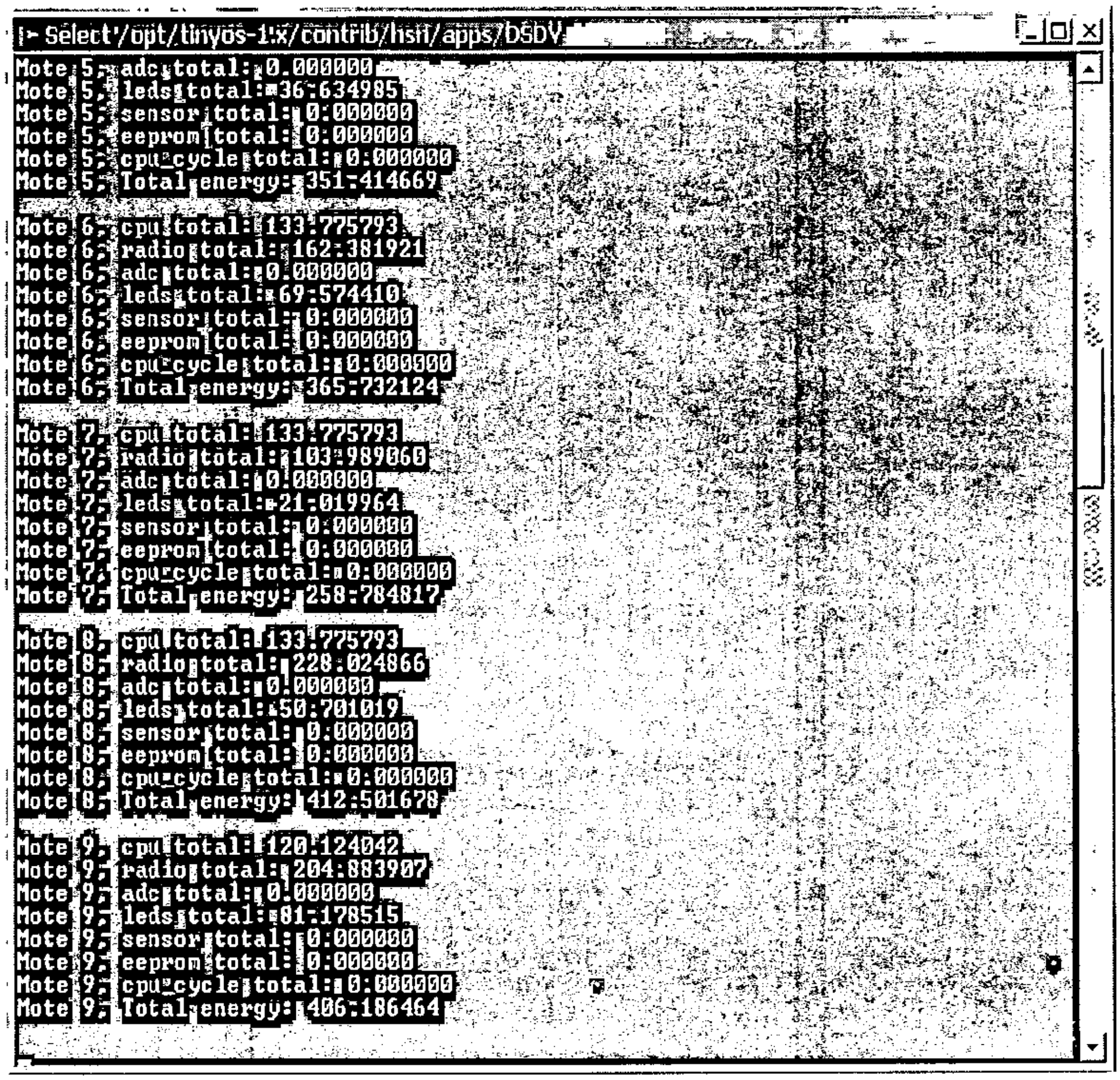


```
~/opt/tinyos-1.x/contrib/hsn/apps/DSDV

Administrator@mar162004 /opt/tinyos-1.x/contrib/hsn/apps/DSDV
$ export DBG=power

Administrator@mar162004 /opt/tinyos-1.x/contrib/hsn/apps/DSDV
$ ./runcmd
```

Setting the parameters for debugging before execution



```
~/opt/tinyos-1.x/contrib/hsn/apps/DSDV
Administrator@mar162004 /opt/tinyos-1.x/contrib/hsn/apps/DSDV
$ ./runcmd
maxseen 9
Mote 0, cpu total: 167.621055
Mote 0, radio total: 285.605521
Mote 0, adc total: 0.000000
Mote 0, leds total: 51.940447
Mote 0, sensor total: 0.000000
Mote 0, eeprom total: 0.000000
Mote 0, cpu_cycle total: 0.000000
Mote 0, Total energy: 505.167023

Mote 1, cpu total: 167.621055
Mote 1, radio total: 285.644205
Mote 1, adc total: 0.000000
Mote 1, leds total: 63.930172
Mote 1, sensor total: 0.000000
Mote 1, eeprom total: 0.000000
Mote 1, cpu_cycle total: 0.000000
Mote 1, Total energy: 517.195431

Mote 2, cpu total: 133.775793
Mote 2, radio total: 221.519483
Mote 2, adc total: 0.000000
Mote 2, leds total: 88.647070
Mote 2, sensor total: 0.000000
Mote 2, eeprom total: 0.000000
Mote 2, cpu_cycle total: 0.000000
Mote 2, Total energy: 443.942346

Mote 3, cpu total: 133.775793
Mote 3, radio total: 214.866774
Mote 3, adc total: 0.000000
Mote 3, leds total: 84.479180
Mote 3, sensor total: 0.000000
Mote 3, eeprom total: 0.000000
Mote 3, cpu_cycle total: 0.000000
Mote 3, Total energy: 433.121748

Mote 4, cpu total: 133.775793
Mote 4, radio total: 149.419286
Mote 4, adc total: 0.000000
Mote 4, leds total: 53.603835
Mote 4, sensor total: 0.000000
Mote 4, eeprom total: 0.000000
Mote 4, cpu_cycle total: 0.000000
Mote 4, Total energy: 336.798915

Mote 5, cpu total: 133.775793
Mote 5, radio total: 181.003891
Mote 5, adc total: 0.000000
Mote 5, leds total: 36.634985
Mote 5, sensor total: 0.000000
Mote 5, eeprom total: 0.000000
Mote 5, cpu_cycle total: 0.000000
Mote 5, Total energy: 351.414669

Mote 6, cpu total: 133.775793
Mote 6, radio total: 162.381921
Mote 6, adc total: 0.000000
```



```
Mote 5, adc total: 0.000000
Mote 5, leds total: 36.634985
Mote 5, sensor total: 0.000000
Mote 5, eeprom total: 0.000000
Mote 5, cpu_cycle total: 0.000000
Mote 5, Total energy: 351.414669

Mote 6, cpu total: 133.775793
Mote 6, radio total: 162.381921
Mote 6, adc total: 0.000000
Mote 6, leds total: 69.574410
Mote 6, sensor total: 0.000000
Mote 6, eeprom total: 0.000000
Mote 6, cpu_cycle total: 0.000000
Mote 6, Total energy: 365.732124

Mote 7, cpu total: 133.775793
Mote 7, radio total: 103.989060
Mote 7, adc total: 0.000000
Mote 7, leds total: 21.019964
Mote 7, sensor total: 0.000000
Mote 7, eeprom total: 0.000000
Mote 7, cpu_cycle total: 0.000000
Mote 7, Total energy: 258.784817

Mote 8, cpu total: 133.775793
Mote 8, radio total: 228.024866
Mote 8, adc total: 0.000000
Mote 8, leds total: 50.701019
Mote 8, sensor total: 0.000000
Mote 8, eeprom total: 0.000000
Mote 8, cpu_cycle total: 0.000000
Mote 8, Total energy: 412.501678

Mote 9, cpu total: 120.124042
Mote 9, radio total: 204.883907
Mote 9, adc total: 0.000000
Mote 9, leds total: 81.178515
Mote 9, sensor total: 0.000000
Mote 9, eeprom total: 0.000000
Mote 9, cpu_cycle total: 0.000000
Mote 9, Total energy: 406.186464
```

B. Data Sheet of Mica2 Mote by Crossbow

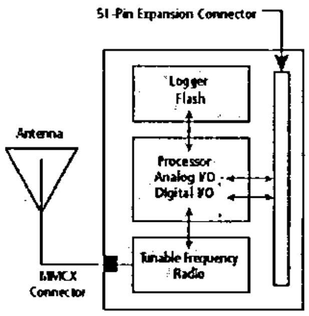
MICA2

WIRELESS MEASUREMENT SYSTEM

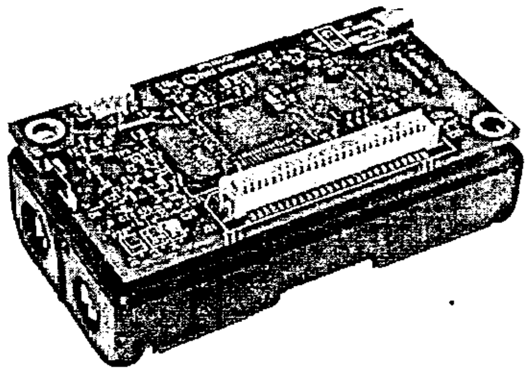
• 3rd Generation, Tiny, Wireless Platform for Smart Sensors
• Designed Specifically for Deeply Embedded Sensor Networks
•>1 Year Battery Life on AA Batteries (Using Sleep Modes)
•Wireless Communications with Every Node as Router Capability
• 868/916 MHz Multi-Channel Radio Transceiver
• Expansion Connector for Light, Temperature, RH, Barometric Pressure, Acceleration/Seismic, Acoustic, Magnetic and other Crossbow Sensor Boards

Applications

• Wireless Sensor Networks
• Security, Surveillance and Force Protection
• Environmental Monitoring
• Large Scale Wireless Networks (1000+ points)
• Distributed Computing Platform



MPR400 Block Diagram



MICA2

The MICA2 Mote is a third generation mote module used for enabling low-power, wireless, sensor networks. The MICA2 Mote features several new mprovements over the original MICA Mote. The following features make the MICA2 better suited to commercial deployment:

• 868/916 MHz multi-channel transceiver with extended range
•Supported by MoteWorks™ wireless sensor network platform for reliable, ad-hoc mesh networking
• Support for wireless remote reprogramming
• Wide range of sensor boards and data acquisition add-on boards

MoteWorks enables the development of custom sensor applications and is specifically optimized for low-power, battery-operated networks. MoteWorks is based on the open-source TinyOS operating system and provides reliable, ad-hoc mesh networking, over-the-air- programming capabilities, cross development tools, server middleware for enterprise network integration and client user interface for analysis and configuration.

Processor and Radio Platform (MPR400)

The MPR400 is based on the Atmel ATmega128L which is a low-power microcontroller. It runs MoteWorks from its internal flash memory. A single processor board (MPR400) can be configured to run your sensor application/processing and the network/radio comm stack simultaneously. The MICA2 51-pin expansion connector supports Analog Inputs, Digital I/O, I2C, SPI and UART interfaces which make it easy to connect to a wide variety of external peripherals.

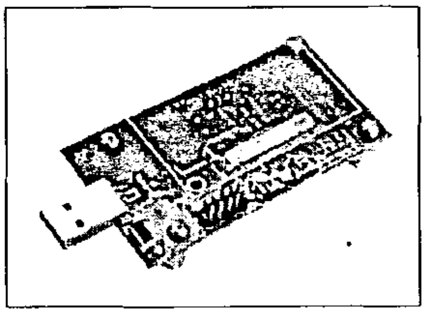
Sensor Boards

Crossbow offers a variety of sensor and data acquisition boards for the MICA2 Mote. All of these boards connect to the MICA2 via the standard 51-pin expansion connector. Custom sensor and data acquisition boards are also available.

Processor/Radio Board	MPR400CB	Remarks
Processor Performance		
Program Flash Memory	128K bytes	
Measurement (Serial) Flash	512K bytes	>100,000 Measurements
Configuration EEPROM	4K bytes	
Serial Communications	UART	0-3V transmission levels
Analog to Digital Converter	10 bit ADC	8 channel, 0-3V input
Other Interfaces	DIO,I2C,SPI	
Current Draw	8 mA	Active mode
	< 15 µA	Sleep mode
Multi-Channel Radio		
Center Frequency	868/916 MHz	ISM bands
Number of Channels	4/ 50	Programmable, country specific
Data Rate	38.4 Kbaud	Manchester encoded
RF Power	-20 to +5 dBm	Programmable, typical
Receive Sensitivty	-98 dBm	Typical, analog RSSI at AD Ch. 0
Outdoor Range	500 ft	1/4 Wave dipole, line of sight
Current Draw	27 mA	Transmit with maximum power
	10 mA	Receive
	< 1 µA	Sleep
Electromechanical		
Battery	2X AA batteries	Attached pack
External Power	2.7 - 3.3 V	Connector provided
User Interface	3 LEDs	User programmable
Size (in)	2.25 x 1.25 x 0.25	Excluding battery pack
(mm)	58 x 32 x 7	Excluding battery pack
Weight (oz)	0.7	Excluding batteries
(grams)	18	Excluding batteries
Expansion Connector	51-pin	All major I/O signals

Notes: Specifications subject to change without notice

Base Stations



A base station allows the aggregation of sensor network data onto a PC or other computer platform. Any MICA2 Mote can function as a base station when it is connected to a standard PC interface or gateway board. The MIB510/MIB520 provides a serial/USB interface for both programming and data communications. Crossbow also offers a stand-alone gateway solution, the MIB600 for TCP/IP-based Ethernet networks.

Ordering Information

Model	Description
WSN-START900CA	MICA2 Starter Kit 868/916 MHz
WSN-PRO900CA	MICA2 Professional Kit 868/916 MHz
MPR400CB	868/916 MHz Processor/Radio Board

IEEE

Proceedings

10th IEEE International Multitopic Conference 2006 (INMIC 2006)

December 23-24, 2006
Islamabad, Pakistan

Organized By

M.A.J.U.

Mohammad Ali Jinnah University, Islamabad.

Investigating Energy Consumption of Localized and Non Localized Ad hoc Routing Protocols in TinyOS

Saadia Khan[1], Salma Basharat[2], Malik Sikander Hayat Khiyal[3] and Shoab Ahmad Khan[4]
Dept. of Computer Sciences, Faculty of Applied Sciences, International Islamic University, H-10 Islamabad, Pakistan[1 2 3]
Centre for Advanced Studies in Engineering (CASE), Islamabad, Pakistan[4].

***Abstract*-This paper characterizes two ad hoc routing protocols LAR and DSDV for sensor area networking on TinyOS. To the best of our knowledge no such comparison of these selected protocols is being made on TinyOS, an event driven component based operating system for embedded sensor networks. Energy utilization in mobile ad hoc networks or sensor networks is a major concern. Minimizing energy in terms of payload, battery power, CPU and memory utilization at a node are important factors that decide the potential of a routing scheme. This paper presents the comparison of the mentioned protocols on the basis of energy being consumed by the whole network. The scenario considered was the one in which the whole information was gathered at one node, i.e., the base station or the sink node of the network. Results indicate that LAR performs better than DSDV.**

Keywords: Ad hoc networks, Localized, Non-Localized, Sensor networks, Energy consumption.

I. Introduction

Mobile computing is the foundation of present age communication systems and they range infrastructure based wireless networks to infrastructure-less wireless networks, having a wide variety of communication devices acting as nodes. An "Ad-Hoc Network" is a decentralized network of autonomous mobile nodes able to communicate with each other over wireless links. The topology of the network may rapidly be changing due to the mobility of the nodes, making it impossible to use conventional routing tables maintained at routers. However, in ad hoc networks each node acts as a router as well to determine the best route to a given destination node. Due to dynamicity in topology, route establishment in ad hoc networks differs significantly from the static routes in wired networks. Moreover, nodes may have different capabilities and may differ with respect to signal strength, available power, reliability etc.

Ad hoc networks are categorized as Mobile Ad Hoc Networks (MANETs) and Wireless Sensor Networks (WSNs). A WSN consists of a number of sensors spread across a geographical area with each sensor having wireless communication capability and some level of intelligence for signal processing and networking of the data. WSNs are characterized by large number of sensors, network self-organization, collaborative signal processing, querying ability and low energy usage [1].

Nowadays most of the research and development in the field of ad hoc networks is focused around making them cost effective, reliable, efficient and practical in terms of routing and deployment. However their diversity poses many challenges in making routing decisions. Routing schemes for ad hoc networks are mainly categorized as reactive, proactive, and hybrid. Further more they are also categorized on the basis of geographical position of the node such as the one provided by GPS or other mechanisms. Location aware or position based routing protocols come under this umbrella [2]. An important challenge in designing protocols for MANETs and sensor networks has been the problem of energy conservation. Energy is valuable to any network that has nodes with [illegible] battery power. The power dissipation in a network is due to the power consumption at different entities of the network. Many schemes have been proposed which handle power saving at routing layer, link layer etc. Examples of the cost function may be the energy metric based on the end-to-end delivery success rate, number of packets transmitted and received, and the total number of neighbors. We intend to compare the performance of localized and non localized routing algorithms in TinyOS [3], specifically, with respect to the energy conservation of whole network. Energy of the entire network depends on those nodes which shoulder the maximum load and serve as data aggregators. In sensor networks, such nodes are known as sink nodes, as they gather all the sensor information of the network. The sustenance of the sink node is crucial to the network due to the fact that if the sink node crashes, whole network will eventually break down.

There are multiple power aware route selection protocols but to our best knowledge, there is no such comparison which involves performance comparison of above mentioned protocol categories using TinyOS and having energy conservation of the whole network as the metric.

The two selected protocols were Location Aided Routing (LAR) [4] and Destination Sequenced Vector Routing (DSDV) [5].

The paper is organized as follows. Section II gives a brief account of the platform and environment used for simulation. Section III describes the routing protocols used for performance analysis. Section IV discusses related work. The simulation statistics and results are explained in section V, while section VI concludes the analysis.

II. Simulation Environment

The simulation environment used is TinyOS [3] which is a component based, event driven, embedded operating system

position information only to setup route in more efficient
nner than DREAM.

1 [14], paper describes a practical application of moderate-
e ad hoc sensor networks. While this application could be
bled by a fixed infrastructure, techniques for reducing
ket loss, including quality-based routing and passive ac-
wledgment, at link and network layer, are explored in this
er. Three variants of DSDV protocol: straight DSDV,
DV with asymmetric link detection, and DSDV with link
ility monitoring are studied and the applications are imple-
nted using TinyOS. Results of 24 and 48 node experiments
w that both the quality-based routing and asymmetric link
ection become less effective at reducing the packet loss rate
the size of the network increases.

V. Simulation Results

Number of nodes in the network was chosen to be 10,
,30,40,50 while mote0 was taken as a sink node for all the
nulations. A total of 20 simulations were carried out while
e duration for each simulation run was 35 virtual seconds.
e used the energy model for mica2 mote, provided by pow-
TOSSIM [13]. This model computes the energy consumed
the radio, CPU, EEPROM, ADC, sensors and LEDs sepa-
tely for each simulated mote. At the end, it gives the total
ergy consumed by the mote. For DSDV we used the con-
ibuted code which is available with TinyOS version 1.x [15],
hereas LAR scheme 1 was implemented by us.

Fig.1 shows the energy consumption of the sink node in
ase of both LAR and DSDV. Average values of energy were
lotted against the number of nodes in the network. The re-
ults indicates that LAR consumes less power as compared to
DSDV due to its location based routing scheme. In LAR,
looding is confined to the number of the nodes existing in
equest zone hence eliminating unnecessary overhead of
looding messages to all the nodes in the network where as in
DSDV, messages are flooded to each node in the network
ence increasing the load on the sink node. Therefore we can
ay that when energy of the network is critical, we should
dopt a routing scheme which involves less routing overhead
as in LAR. From simulations, it can be inferred that LAR
performs better in route establishment as compared to DSDV
since it is used as and when required. Moreover, due to the
directional flooding used in the route establishment, it per-
forms better than DSDV that uses periodic route updates.

Figure.2 illustrates the percentage of average power saved
by LAR against different node densities where percentage is
calculated as:

$$\frac{\text{Power used by DSDV} - \text{Power used by LAR}}{\text{Power used by LAR}}$$

The results indicate that power consumption is directly pro-
portional to the number of the nodes in the network. There-
fore the percentage of power saved is more at less number of
the nodes and vice versa. It can be inferred from these results
that LAR saves 59% of available energy for a small sized
network. As the network density increases, energy savage by
LAR decreases by approximately 7% and drops to 52%. On
the other hand, as illustrated in Fig. 3, the energy conservation
is about 37% in case of DSDV for a small scale network.
However, this value drops about 34 % for a medium sized
network topology, which in contrast to LAR is a relatively less
variation. These statistics show that LAR conserves more
energy, i.e., approximately 22% for small scaled networks and
18 % for medium scaled networks as compared to DSDV.

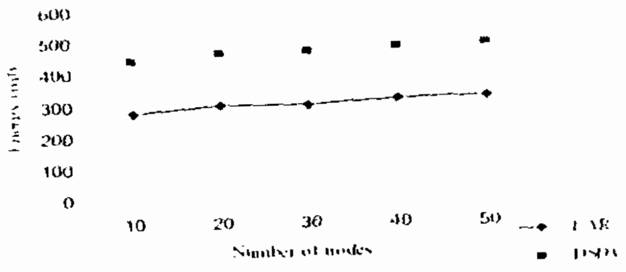


Fig. 1. Average energy consumed at Sink Node by LAR and DSDV

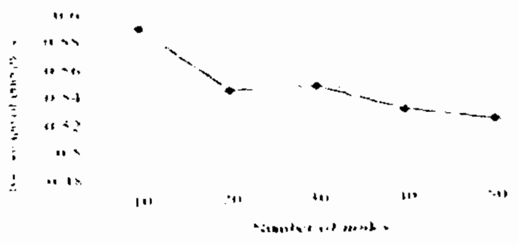


Fig. 2. Average energy saving in LAR for different node density

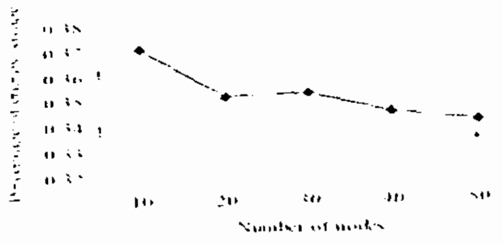


Fig. 3. Average energy saving in DSDV for different node density

VI. Conclusion and Future Work

In this paper, we compared the performance of LAR, a lo-
calized ad hoc routing protocol, with DSDV, a non-localized
routing protocol, by characterizing them to a sensor network
scenario. LAR is fully distributed where message flooding is
only directional and according to pre- calculated request zone
area while in DSDV message is just flooded to all the
neighboring nodes. We have shown by simulation results that