Anycast Routing in Wireless Sensor Networks

Peizhong Zhao

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science

at Concordia University

Montreal, Quebec, Canada

September 2007

Your file Votre référence
ISBN: 978-0-494-34658-7
Our file Notre référence
ISBN: 978-0-494-34658-7

Abstract

Anycast Routing in Wireless Sensor Networks

Peizhong Zhao

An emerging new type of ad hoc network, a wireless sensor network, has great potential to be used in many application areas. While each sensor network application has its own particular technical issues, network topology formation and multi-hop routing are two problems that are common to and very important for all applications. The key question is how to organize and keep the network in an optimal topology so that sensor data can be reliably and efficiently sent to sink nodes.

In this thesis, we present a new approach to construct a wireless sensor network topology with a long network lifetime and high data delivery rate. Specifically, we designed an anycast routing protocol named Dynamic Anycast Routing to form anycast trees that enable a simple and efficient routing scheme. The protocol was implemented using the sensor network-oriented operating system TinyOS. The protocol was tested both in a simulation environment and on real sensor hardware. Experiments using the TOSSIM simulator demonstrate that our protocol can significantly prolong network lifetime and at the same time maintain a high data delivery rate.

Acknowledgements

I would like to thank my supervisor, Professor Lata Narayanan, for giving me insightful advice and guidance. Whenever I had questions, Dr. Lata Narayanan always helped me kindly. Without that great help, my research and this thesis would be impossible to finish.

Also, I am grateful to Dr. William Atwood for giving us great help when we were installing the sensor network hardware in the labs. His help made the installation process as smooth as possible.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ABS	Anycast Basestation Selection
AODV	Ad-hoc On-demand Distance Vector
CVS	Concurrent Versions System
DAR	Dynamic Anycast Routing
GPS	Global Positioning System
HAR	Hierarchy Based Anycast
HC	Hop Count
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
LAN	Local Area Network
MAC	Media Access Control
MANET	Mobile Ad hoc NETwork
MT	Minimum Transmission
NesC	Networked Embedded Systems C
NS	Network Simulator
OSPF	Open Shortest Path First
PAN	Personal Area Network
RAM	Random-Access Memory
RFID	Radio Frequency IDentification
RIP	Routing Information Protocol
ROM	Read-Only Memory
RSSI	Received Signal Strength Indicator
SARP	Sink based Anycast Routing Protocol
TCL	Tool Command Language
TCP	Transmission Control Protocol
TinyOS	Tiny Operating System
TOSSIM	TinyOS SIMulator

TTL	Time To Live
UAV	Unpiloted Aerial Vehicle
UDP	User Datagram Protocol
WMEWMA	Window Mean Exponentially Weighted Moving Average
WSN	Wireless Sensor Network

1 Introduction

Wireless Sensor Networks (WSN) are a new emerging type of ad hoc network. An ad hoc network is a network without any fixed supporting infrastructure. It is the transitory association among wireless devices which just happen to be in a common geographic location. The unique characteristic of the network is that any device in the network has no prior knowledge of other devices surrounding it. Connection and disconnection are determined by the wireless communication quality among nodes, and their willingness to participate in the formation of the transitory communication system. All nodes could be mobile or fail suddenly and most of them rely on each other to forward packets to destinations not directly in their own transmission ranges [31]. Meanwhile, wireless communication quality among these devices could be changing any time. Due to the above facts, the topology of the ad hoc network may keep changing accordingly.

A MANET (Mobile Ad Hoc Network) is a particular kind of ad hoc network in which nodes are mobile. Typical nodes in a MANET are notebook computers. Usually the total number of devices in a MANET is pretty small and these devices have very high computing and communicating power. In contrast, a wireless sensor network is formed by a large number of ultra low-power and low-cost *sensor nodes* placed in a specific geographic area to monitor certain events and interests. A sensor node is the result of seamlessly integrating sensing, special-purpose computing, and wireless communications functionalities into one thumb-sized device. Placement of new sensor nodes may take place on demand at any time at specified locations or at random in designated areas. Once

deployed, these sensor nodes must form a larger network by themselves without any prior knowledge of the neighboring sensor nodes. Once the network is formed, sensor nodes must work unattended to provide users meaningful data in real time. Recent advances in computer science, wireless communication, and micro-electronics have enabled the development of this kind of sensor node and network [23, 24].

1.1 Sensor network applications

Wireless sensor networks can be used in various applications such as environment monitoring, home appliance controlling, and military battlefield surveillance systems. A sensor network can be homogeneous or heterogeneous. A homogeneous type network contains the same kind of sensor, usually in a very large quantity, to perform a single monitoring task. On the other hand, a heterogeneous network consists of different types of sensor and actuator hardware such as temperature sensors, humidity sensors, acoustic sensors, GPS (Global Positioning System), and electronic relays. This type of network can perform more sophisticated operations including monitoring and controlling, to some extent resembling traditional feedback control systems.

Environment monitoring is one of the first seen wireless sensor network applications. A sensor network can track the movements of wild animals; can monitor environmental conditions such as pollution or forest fires; can detect local weather conditions. The project Habitat Monitoring on Great Duck Island is one typical example of this kind of application. "In the spring of 2002, the Intel Research Laboratory at Berkeley initiated a collaboration with the College of the Atlantic in Bar Harbor and the University of California at Berkeley to deploy wireless sensor networks on Great Duck Island, Maine.

These networks monitor the microclimates in and around nesting burrows used by the Leach's Storm Petrel. ... As of mid-October 2002, nearly 1 million readings have been logged from 32 motes deployed on the island" [34]. Precision agriculture deployment is another example of an environment monitoring application. In the desired locations of the field, many low-cost sensor nodes can be placed to form a wireless sensor network to send real-time environment data such as soil temperature and moisture to the control center of the farm [23]. Volcano activity monitoring is also a good example of wireless sensor network application in the environment monitoring area. In August 2005, researchers deployed wireless sensor nodes on the active volcano Volcan Reventador in northern Ecuador. The sensor nodes are equipped with microphones and seiesmometers to collect seismic and acoustic information on volcanic activity. There were 230 eruptions successfully detected in the sensor network's 19 day deployment period [45]. From the above examples, we can see that wireless sensor networks can be deployed very efficiently in harsh environments where fixed infrastructure is very difficult to set up. The twin features of wireless and low cost enabled users to monitor environment events remotely that would not have been possible before. Since sensor nodes can be strategically deployed, people can put them in locations where most attention is needed. Since the sensors are supposed to be running for years unattended, *battery life* becomes a very important criterion for this type of application.

Figure 1.01: Possible deployment of WSN for precision agriculture [23]

Military applications are another important application area for sensor networks. Battlefield surveillance information is vital for modern military operations. A very large quantity of small and smart sensors could be dropped from airplanes to be deployed in battlefields to monitor the enemy army's movement [23]. Since the sensors are small, they can not be discovered easily. By using sensors networks effectively in battlefields, many soldiers' lives can be saved. The following is an actual example, as described in [23], "As part of an experiment with the US marines, the motes (wireless sensor nodes) were deployed to detect vehicle activity at an isolated intersection in the desert near Palm Springs, California. A collection of nodes were dropped in a line along the side of a road. They were dropped from a small UAV (Unpiloted Aerial Vehicle) that was flying autonomously based on a GPS flight plan. The plane was pre-configured with a multi-point flight plan which included a low-altitude "bombing run". The motes were released from approximately 100 feet along a track parallel to the road. Once deployed, the nodes configured themselves into a multi-hop network and synchronized internal clocks so discrete sensor readings could be correlated across multiple nodes. As vehicles passed the

network, individual nodes used magnetometers to detect deviations in the magnetic field caused by metal contained in the vehicles. Each node determined the closest point of approach for the vehicle and then assigned a time stamp for the vehicle event. Events from each sensor were then communicated to neighboring nodes. Once a node collected 5 readings, it performed regression analysis to determine the velocity and direction of the vehicle. The high-level vehicle track information was then logged in persistent storage."

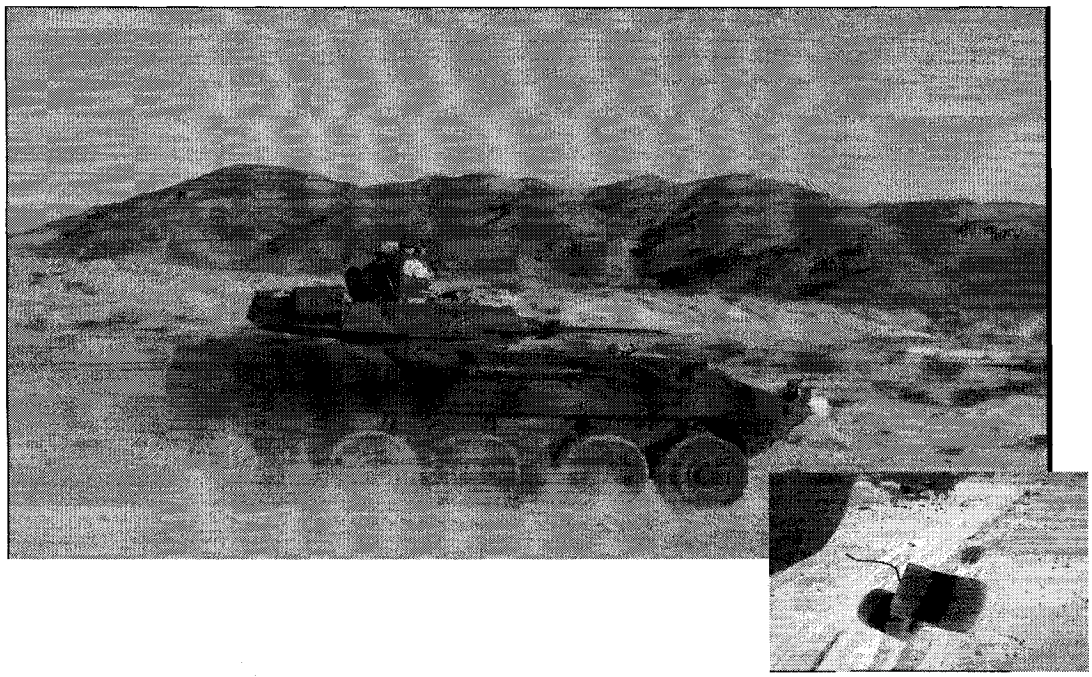


Figure 1.02: WSN in battlefield
"Motes dropped out of an airplane self-assemble onto an ad-hoc network in order to monitor for vehicle activity at a remote desert location" [23]

Home appliance controlling and supermarket article tracking are other emerging markets for wireless sensor networks. Zigbee [9] and RFID (Radio Frequency Identification) are two examples.

When sensor networks eventually integrate with IPv6 technology in the future, the way we view the world will be completely changed. All people will have the whole world's real-time all-dimension information at their fingertips. Due to limited computing power in current available sensor nodes, it is still not feasible to implement IPv6, even IPv4, on current wireless sensor networks [17]. But Moore's law (rapid increase of computing capability) will make that feasible soon. Foreseeing this future, the IETF (Internet Engineering Task Force) has created a new working group named 6lowpan (IPv6 over Low power WPAN) to define the transport of IPv6 over IEEE 802.15.4 low-power wireless personal area networks [33]. In the future, billions of all kinds of tiny sensor nodes will be deployed all over the world and each one will have its IPv6 addresses.

Realizing the great potential of wireless sensor networks, researchers all over the world are now paying more and more attention to the relatively new and challenging research areas in wireless sensor networks.

1.2 Overview of the problem

While each wireless sensor network application has its own unique technical challenges, topology formation and routing are two common and very important issues facing all sensor network applications. The problem here is how to organize and maintain the sensor nodes in a desired topology so that sensor data can be routed to users using optimal paths efficiently. In most wireless sensor networks, it is not possible for sensor nodes to transmit messages directly in a single hop to the base station because nodes are far away from the base station and their radio transmission range is very limited. Thus, routing

messages in a multi-hop fashion to their correct recipient is a complex task involving various intermediate routing nodes. Topology formation and routing are not trivial tasks in wireless sensor networks due to the following unique characteristics:

i) Sensor nodes join and leave the network dynamically at unpredictable times.

ii) Sensor nodes' radio communication is prone to collision if the number of nodes in an area is large.

iii) Sensor nodes' radio communication is sensitive to environmental noise from wireless signals because the radio in sensor nodes operate at a very low power (sensor nodes typically are AA-size or even button battery powered).

iv) Sensor nodes' radio communication may be blocked by obstructions.

v) The topology must be formed quickly to provide service to users on time.

vi) Routing must be fast so to provide real-time sensor data to users with short latency requirements.

vii) Sensor nodes must use efficient computing and routing algorithms to reduce power consumption.

viii) Sensor nodes have a very small amount of memory, usually in the range of 1KB-16KB RAM and 32KB-256KB Flash ROM, thus the routing algorithm must have low memory requirements.

Topology formation and routing are two related issues. Once a topology type is designed for the network, there are limited numbers of viable routing algorithms for that topology. Wireless sensor networks present unique challenges to topology formation and routing algorithms due to the aforementioned characteristics. Traditional algorithms developed

for Internet and Mobile Ad Hoc Networks are not suitable for wireless sensor networks. Those algorithms are computing and communication intensive, thus they can not be afforded in low-power wireless sensor networks. Also, those algorithms often are not scalable enough to be used in large scale and dense wireless sensor networks. Since wireless sensor networks are still a very young research area, there are no mature topology formation and routing algorithms developed yet although many algorithms have been proposed.

Topology formation and routing algorithms are to be judged not merely by their simplicity or efficiency, but primarily by the resulting performance of the network. This can be evaluated by several metrics, the most important of which are perhaps *network lifetime* and *data delivery rate*. Network lifetime is normally defined as the time when the first sensor node in a network runs out of battery. Data delivery rate is the percentage of successful packets delivered. Existing protocols have not addressed these two issues adequately.

1.3 Motivation

Anycast is a network routing problem in which data from a source node is to be routed to one and only one of many possible destination nodes. The one destination node is chosen according to certain merits such as number of hops, quality of links, security, etc. In large scale wireless sensor networks, often more than one base station is desired because of the dense nature of this type of network. If multiple base stations are installed, it is best to connect them by a dedicated high speed communication channel so that they can work in a coordinated manner to share collected real-time data from individual sensor nodes.

This way, a sensor node can simply send the data to any base station it views as best. When a base station wants to retrieve data from specific sensor nodes, it can get the data quickly from other base stations if they already received the desired sensor data. Thus, this scheme can lead to great simplifications in routing algorithms running on sensor nodes. Since sensor nodes have limited computing and communication power, this simplification is appreciated and can prolong the life time of the sensor network. As such, we can see that anycast routing is very suitable for large scale wireless sensor networks. To our knowledge, there is not much research on this topic yet. Our goal in this thesis is to explore anycast on wireless sensor network and its potential benefits in terms of improving the performance of topology formation and routing algorithms.

1.4 Scope of the study

We study anycast routing in large scale wireless sensor networks with multiple base stations. It is assumed that these base stations are connected by a high speed communication channel such as an Ethernet network, satellite or microwave communication channel. Sensor nodes communicate with each other and to base stations using low-power low-speed wireless connections. We consider a typical data-collecting application, that is, all sensor nodes send their sensor data to base stations periodically. Sensor nodes do not accept requests from base stations. We focus on how to construct the network topology and how to route sensor data from sensor nodes to base stations efficiently.

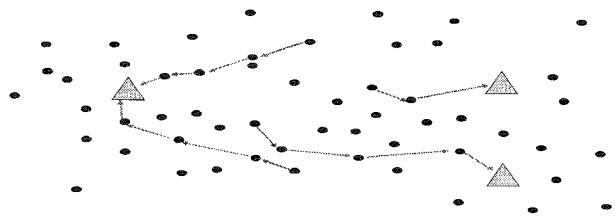


Figure 1.03: A data-collecting WSN (with 3 base stations)

Our goal is to form the network topology with routing paths that are both short and of high quality. When these two performance metrics are improved, the network can be expected to exhibit short routing latency, high delivery rate, and long network lifetime.

1.5 Contributions

In this research, we designed an anycast routing protocol named DAR (Dynamic Anycast Routing) to form anycast trees that enable employing a simple and efficient routing algorithm. Our approach is completely different from traditional anycast algorithms tuned for Internet or other traditional wired networks. The DAR protocol segregates network traffic and greatly simplifies multi-hop routing. The protocol was tested both in a simulation environment for sensor networks called TOSSIM and on real sensor hardware Tmote Sky [2] motes, where it was implemented using a sensor network-oriented operating system called TinyOS [1].

Performance of the protocol was compared against existing protocols. The simulation results demonstrate that our protocol can prolong network lifetime significantly and at the same time maintain a high data delivery rate.

An important aspect of our contribution is that all of our source code and source code history of changes is publicly available from Subversion version control system [4] hosted on www.SourceForge.net [3]. Details of how to use the source code is given in the appendix. According to our past experience, rarely students put their source code under version control system and made them publicly downloadable, thus, their work could not be re-used by others.

1.6 Organization of the thesis

The rest of the thesis is organized as follow:

- Chapter 2 is a survey of related work. We studied current mainstream topology formation and multi-hop routing algorithms and protocols.
- In Chapter 3, we explain the algorithm and protocol details of our proposed DAR protocol.
- Chapter 4 presents the design choices we made and the implementation details of our DAR protocol, including the programming environment and tools.
- Chapter 5 gives details of how we tested our protocol in the TOSSIM simulator.
- In Chapter 6, details of how we tested DAR on Tmote Sky sensor hardware are explained.
- In Chapter 7, we evaluated the performance of the DAR protocol and compared it with other protocols.
- Chapter 8 concludes our work and discusses the possible future research directions.

2 Literature Review

Various topology construction algorithms and routing protocols exist for wireless sensor networks. Some of them are adapted from early algorithms/protocols for Internet and MANET. In this chapter, we survey the important ones that lead to our proposed DAR protocol.

2.1 Ad hoc network topologies and routing

There exist various kinds of topologies for wireless ad hoc networks. The common types are:

- Bus
- Star
- Peer to peer
- Mesh
- Tree
- Cluster tree

The widely used wireless MAC and PHY layers standard, IEEE 802.15.4 [46], supports the star, peer to peer, and cluster tree topologies. The IEEE 802.15.1 standard, which is based on Bluetooth, supports a star topology in its Piconet format, and supports a combination of star with bus, mesh, and tree topologies in its Scatternet format. Currently, most sensor nodes are using IEEE 802.15.4 or IEEE 802.15.1 radios. Usually, the physical connectivity graph among nodes is a mesh topology. But the routing protocol may only use a subset of the mesh topology to construct a desired topology, i.e., peer to peer or tree, to simplify the routing protocol.

Unlike wired networks, which have relatively static topologies, topologies in wireless ad hoc networks are constantly changing. This fact makes the routing protocols in ad hoc networks more complex than Internet oriented routing protocols such as RIP and OSPF.

2.2 Routing Algorithms for MANET

Much research work has been done and many routing protocols have been proposed for ad hoc wireless networks. Routing protocols in ad hoc networks are usually either proactive or reactive though some of them are hybrid. Proactive routing protocols determine routing paths before any data packet is to be sent. The routing path is maintained throughout the network lifetime. The advantage of proactive routing protocols is that packet delivery latency is low since routing paths are always available. However, the routing path maintenance overhead will be high if the network topology is volatile. So proactive routing protocols are suitable for networks where nodes are relatively stationary. On the other hand, reactive routing protocols create routing paths only if necessary. If no nodes have data to be sent, there are no paths maintained in any nodes. At the time a node wants to send data packets, it will initiate a process to construct a routing path to the destination node. Obviously the network latency is high in this type of architecture since the routing path construction takes time. The advantage of reactive protocols is that the route maintenance overhead is low, and thus it is very suitable for networks in which nodes are moving constantly.

Some algorithms and protocols are suitable for special types of networks. For example, Geographic Distance Routing (GEDIR) protocol requires that sensor nodes know their geographic locations (e.g., by using GPS). Most protocols, such as DSDV, DSR, AODV,

and TORA, are general purpose and can be used in all MANET networks. Among the general purpose protocols, AODV (Ad-hoc On-Demand Distance Vector) [35] (RFC3561) is considered the overall best routing protocol. Its strength is that it can handle the mobility and scalability reasonably well and at the same time maintain an acceptable latency and memory consumption.

2.2.1 AODV protocol

As its name implies, AODV is a type of Distance-Vector routing protocol with the property of reactive on-demand route setup. A route is to be set up only when a node (the source) wants to send messages to another node (the destination). The route from the source node to the destination node is set up by using the Path Discovery procedure. A Path Discovery consists of two links' setup: a backward link setup from all intermediate routers to the source node and a forward link setup from all intermediate routers to the destination node. Routers use these two links to carry the packets back and forth between the source and the destination nodes.

The backward link is set up after the source node sends out a Route Request message because the source node wants to find a path to the destination node. The source node's neighbors will forward this message again to their neighbors. The Route Request message contains the source node's address and a Broadcast ID, which uniquely identifies the message. Upon receiving this Route Request message, routers set up the backward entry in their routing tables pointing to the source node. Duplicated messages received by any node are dropped. Eventually, the Route Request message will reach the destination node and the destination node will send back a Route Reply message to the source.

The Route Reply message will be travelling back to the source in the reverse direction and during this time all routers set up a forward entry in their routing tables pointing to the destination node. Finally the Route Reply message reaches the source node and the path discovery process is finished.

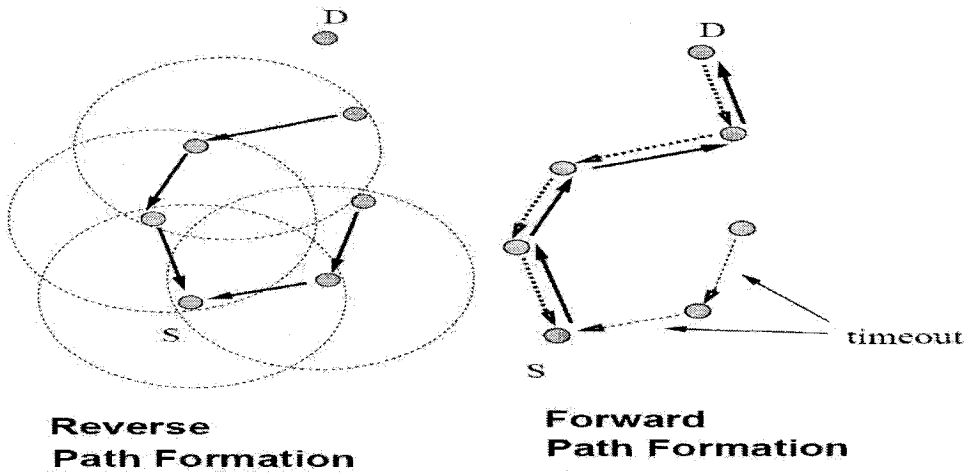


Figure 2.01: AODV reverse and forward path formation [35]

When the route is set up successfully, the source can start to send packets to the destination node. The data packets do not need to carry the intermediate routers' information since the route is already set up. This keeps the data packet overhead low compared to the source routing protocols. The route is maintained by using timeout detection. After the route has been idle for a while, the route will be considered broken by routers. Routers can also generate a Route Error message and send it to the source node to indicate that the destination node is becoming unreachable [35].

From the above characteristics, we can see that the AODV is a protocol suitable for MANET but not sensor networks. The routes in AODV are peer to peer and are set up on-demand. The peer to peer route is not suitable for sensor networks that require routes of many to one (from sensor nodes to base station). It is best for sparse networks in which most traffic is of a bursty type. It also assumes the presence of symmetric links in the medium, and disregards any pair of nodes that don't establish a symmetric link [36]. Also, it uses memory consuming routing tables to record backward and forward paths.

2.3 Topology Formation and Reactive Routing in WSN

In this section, we review some of the recent routing protocols that have been proposed specifically for wireless sensor networks. All the algorithms described here can deal with dynamic sensor networks (i.e., nodes joining/leaving the network or just moving in the network).

2.3.1 Directed Diffusion

Directed Diffusion [49] is a reactive and data-centric routing algorithm for wireless sensor networks. It is reactive because the routing paths are established on demand. It is data-centric because all communication is for named data. The Directed Diffusion algorithm is designed to let sink nodes in a wireless sensor network collect desired sensor data from sensor nodes.

The first step of routing path establishment in Directed Diffusion is by the sink node initially and repeatedly broadcasting an *interest* message to its neighbor nodes. This is called an *exploratory event*. In Directed Diffusion, an interest message is a query that

specifies what kind of sensor data a user wants to collect. It also contains the transmission rate at which the source node should send the sensor data.

After a node received an interest from its neighbor, it examines its cache to see if there is an entry for this interest or not. If no entry is found, a new entry will be created for this interest. This entry has the information of how to send back sensor data to the sink through its 1-hop neighbor. The received interest message is also locally broadcast by this node to a subset of its neighbors so that the interest message eventually reaches the sensor nodes that have the desired sensor data. In Directed Diffusion, this step is called *gradient establishment*.

When a sensor node detects a matching interest, it sends back the exploratory event to the sink. Since each node has multiple neighbors, the messages will be sent along multiple paths towards the sink. When the sink receives these events, it starts a *reinforce procedure* to draw down sensor data from a particular neighbor.

We can see that Directed Diffusion is similar to the AODV algorithm in the way of setting up routing paths. It is suitable for routing in query-response type of wireless sensor networks.

2.3.2 Rumor Routing

Rumor routing [50] is another reactive data-centric routing algorithm for wireless sensor networks. The network comprises densely deployed wireless sensor nodes by which unique events are recorded. Rumor routing lets the user send a query from the sink node

to a sensor node that detected a specific event. The Rumor routing algorithm does not rely on any geographic related criteria. The sensor nodes do not need to know their geographic coordinates.

One unique feature of the Rumor routing algorithm is that it uses agents to travel the network and propagate event information. The agent is a packet with a large TTL (Time To Live) value. It has an event table and this table is used to synchronize the event information with all sensor nodes the agent visits. In this way, distant sensor nodes learn the events detected by other nodes. When a query is sent out from the sink node, sensor nodes use the information gathered by synchronizing with the agents to route the query to the target node.

Rumor routing is a kind of variant of Directed Diffusion. Compared to Directed Diffusion, the Rumor routing algorithm maintains only one path between sink and target nodes where Directed Diffusion uses multiple paths between the nodes.

2.3.3 SARP Routing Protocol

Chalermek Intanagonwiwat in [21] proposed a hop count metric-based tree topology construction protocol named Sink based Anycast Routing Protocol (SARP). This protocol borrows ideas from the AODV protocol to reactively discover a path between a base station (sink) and a sensor node on demand. A routing path is established only when a base station is willing to get certain kind of sensor data (also known as an interest) from sensors. So this protocol is suitable for the query-response type sensor network, not for data-gathering type sensor network.

In SARP, the route establishment is very similar to that in AODV. A sink broadcasts its interest to all of its neighbors. The neighbors then rebroadcast the interest packet after increasing the hop count field in the packet. This process is repeated and eventually the interest packet is propagated to all nodes in the network. All nodes then know how far they are from a sink and how to route a packet to a sink. Nodes keep only the information for the nearest sink (having lowest hop count) when there are multiple sinks. The target sensor node that can generate the desired interest will send back the sensor data using the reverse path to the sink.

SARP assumes the low level MAC protocol is IEEE 802.11. Link breakage detection is done by probing the underlying IEEE 802.11 MAC layer indication that a packet could not be sent to its next hop node. A Route Recovery process will be performed at a node that detects a route failure when it is trying to forward the sensor data to a sink. The Route Recovery is done by the router node detecting the route failure, not the source sensor node. The router node floods a Route Request packet through the network back to a sink and the sink will re-establish a path to the router node.

2.4 Topology Formation and Proactive Routing in Static WSN

In this section, we review some of the proactive routing protocols that have been proposed for static wireless sensor networks. None of the algorithms described here deal with dynamic sensor networks.

2.4.1 Single base station tree topology formation

The basic topology desired in data-gathering wireless sensor networks is a spanning tree, since the traffic is in the form of many-to-one flows [13]. The advantage of this topology is that sensor nodes do not need to maintain routing tables. After the spanning tree topology is formed, routing is a very simple task. Any node simply sends its message to its parent and then eventually the messages reach the base station.

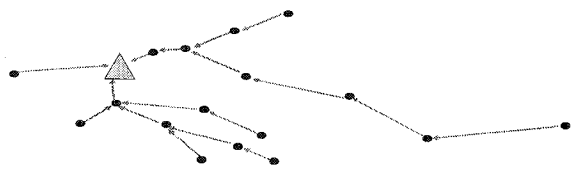


Figure 2.02: Static tree topology

Much work has been done to construct a spanning tree topology assuming that nodes are not dynamically added to or removed from the network. This kind of topology generation is a two-phase process: a flood initiated by the root node (base station), followed by a parent selection process by all sensor nodes.

In the flooding phase, the base station broadcasts its hop count as 0. All of its neighbors that hear this message learn that they are 1 hop away from the base station. Then, all these nodes increase the hop count value by 1 and re-broadcast the message. If a node receives a message containing a hop count value greater than the hop count it already learned, the message is ignored. This assures that the final topology is loop free. This process is

repeated until all sensor nodes in the network learn their hop counts. Flooding is commonly considered a costly operation because all nodes are involved in the broadcasting process and thus energy consumption is high and it also creates many redundant packets. But if no sensor nodes are added or removed dynamically during the whole network lifetime, there is only one flooding process needed at the beginning to generate the network topology. Thus, this flooding does not affect the network performance significantly.

In the second phase, a sensor node selects a node with lower hop count as its parent in one of a few mechanisms. In [13], Congzhou Zhou surveyed the commonly used mechanisms for parent selection in the second phase of the topology construction, and summarized that there are four ways for sensor nodes to select parents:

- Earliest parent selection
- Randomized parent selection
- Nearest-first parent selection
- Weighted-randomized parent selection

These mechanisms differ in the manner in which the parent selection takes place and result in qualitatively different tree structures. The characteristics of the above four parent selection schemes are summarized in the following paragraphs.

In the earliest-first parent selection scheme, a node selects the earliest node from which it receives the first flooding message as its parent. So the metric used here for parent selection is time. As a result, a node that broadcasts the flooding message first will be selected as parent node by all of its neighbors that are one more hop away from the base

station. The implementation of this scheme is very simple and straight forward. The node does not even need to track other up-level nodes that are sending the flooding messages to it since they will not be considered parent candidates [13, 52, 53].

In the randomized parent selection scheme, a node randomly chooses one node from all up-level nodes as its parent. All candidates have equal chances to be selected. The implementation of this scheme is a little bit more complicated than the earliest-first parent selection scheme since the node has to track all up-level nodes that sent the flooding messages to it and then select one as its parent later [13, 52, 53].

In the third scheme, nearest-first parent selection scheme, a node will choose the geographically nearest up-level node as its parent. Thus the metric used here is distance. Usually, this should result in very good link quality among nodes since the wireless communication distance is short. In this scheme, the node has to have a method to measure its distances to the parent candidates. The commonly used method is to estimate physical distance by measuring the RSSI (received signal strength indicator) [13, 52, 53].

Finally, in the weighted-randomized parent selection scheme, each parent candidate node is assigned a weight according to its number of neighbors. A candidate parent node with more neighbors will be assigned a smaller weight. Then the parent selection metric is based on randomization of these weights. The result is that a candidate node with more neighbors will be less likely to be selected as a parent by all of its next level neighbors. The purpose of this scheme is to try to balance the number of children a parent node has. In other parent selection schemes, a node with dense next level neighbors will have many

children, and conversely a node with sparse next level neighbors will have very few children [13, 52, 53].

Finally in [13], these four schemes are evaluated in a simulator (the name of the simulator and the details of the simulation configuration are not given) and the topologies formed by these four schemes are found have the following different characteristics: "The earliest-first and nearest-first schemes produce a data-gathering tree with low network reliability, high data aggregation ability, and long response time to an event. Randomized and weighted-randomized schemes, on the other hand, construct a balanced data-gathering tree with high network reliability, low data aggregation ability, and short response time to an event. In addition, the nearest-first scheme outperforms the other three schemes in channel quality. In all cases the differences in performance are exaggerated most when the communication range is large (when the densities and therefore possible choices for each mechanism are high)".

2.4.2 ABS Routing Algorithm for WSNs

Thomas Hou proposed an algorithm ABS (Anycast Basestation Selection) [12] to route messages from a sensor node to a selected base station. The algorithm assumes the network has a static mesh topology. The primary goal of the algorithm is to maximize the static network life time, which is defined in [12] as the time when the first (any one) sensor node runs out of battery power.

The ABS algorithm considers a two-tier architecture for wireless sensor networks. There are three types of nodes in the network: micro-sensor node (MSN) at the lowest level for

actually monitoring the environment and generating sensor data, base station (BS) at the highest level for displaying and storing sensor data, and the last, aggregation and forwarding node (AFN), which is in the intermediate level between MSN and BS. MSN nodes are deployed in groups at strategic locations for sensing applications. Each group of MSN has a central AFN which is to aggregate and forward the sensor data from this group of MSN to a BS. Using AFNs effectively makes the dense sensor network work like a sparse network. Thus the deployment of the AFNs can greatly reduce the global traffic because only AFNs are sending data to BS.

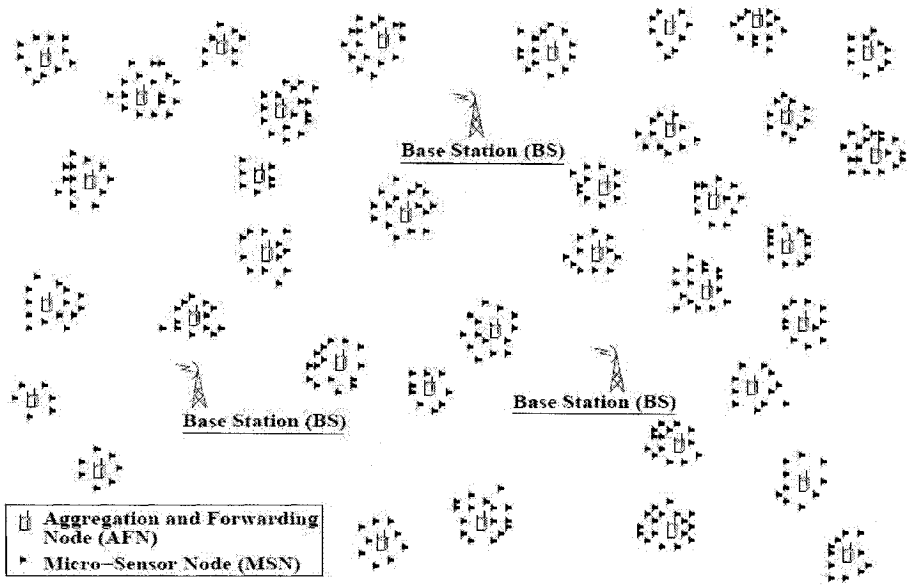


Figure 2.03: Physical network layout in ABS [12]

Due to the fact that only AFNs will be sending data to BS, the ABS algorithm only needs to consider the upper tier wireless multi-hop communications among the AFNs and base stations. The key feature of the ABS algorithm is to balance the traffic from a given AFN node, so that all intermediate AFN routing nodes evenly participate in the routing task. Thus, the power consumption of each AFN is balanced and no one node would run out of

battery significantly earlier than others. When messages generated by an AFN must be relayed to the same base station, the bit stream can be split into sub-flows and sent to the same base station through different paths. The following figure illustrates the traffic split from AFNs.

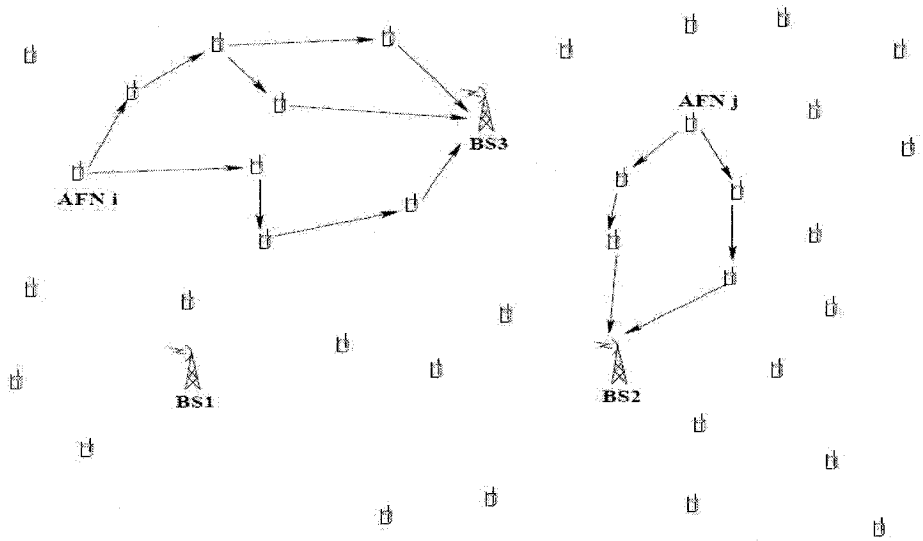


Figure 2.04: ABS routing paths [12]

2.5 Topology Formation and Proactive Routing in Dynamic WSN

Realistically, the topology in a sensor network is constantly changing because sensor nodes will be added to the network anytime and some nodes will fail due to various reasons such as battery running out, heavy noise in communication link, software bugs, and hardware failure. Routing protocols for static wireless sensor networks may be useful for research in labs, but only the topology construction algorithms and routing protocols that can deal with those real-life issues could be deployed in real applications.

2.5.1 WMEWMA Algorithm and MT Protocol

In the context of a dynamic sensor network for data collection, Alec Woo [37] proposed an algorithm WMEWMA (Window Mean Exponentially Weighted Moving Average) to periodically monitor the link quality change and a protocol MT (Minimum Transmission) to maintain a tree topology dynamically.

First, the WMEWMA algorithm computes an average radio transmission success rate over a time window T. The formula for the success rate is: (Packets Received in time T) / (Packets Expected in time T). The success rate is then smoothed by the EWMA algorithm to get the final WMEWMA value. The higher the WMEWMA value, the higher the link quality between a pair of nodes. The tuning parameters are T and the history of the estimator. Link quality estimation messages are sent from each node at a fixed interval driven by a periodic timer. This way, the estimator can calculate the success rate by counting the number of received packets.

The authors consider link quality to be the most important criterion for selecting the parent. The authors argue that, "with lossy links, as found in many sensor networks, link-level retransmission is critical for reliable transport, as each hop may require one or more retransmissions to compensate for the lossy channel. With links of varying quality, a longer path with fewer retransmissions may be better than a shorter path with many retransmissions. That is, the best path is the one that minimizes the total number of transmissions (including retransmissions) in delivering a packet over potentially multiple

hops to the destination." Thus, this approach can be said to use the Minimum Transmission metric.

A node passively monitors packets sent by neighbors to perform neighbor discovery. Each node has a neighborhood table in memory to keep track of its neighbors. The aforementioned link estimation is then used to calculate link quality and determine which nodes should be kept in the neighborhood table. Because the deployment of a sensor network is often dense, a node usually can hear from many nearby nodes. For the sake of keeping memory consumption low, only neighbors with good link quality are kept in the neighborhood table. In the MT protocol, the link quality message includes a parent address, estimated routing cost to the sink, and a list of reception link estimations of neighbors.

Besides a periodic timer for broadcasting, there is another one running at a slower rate for parent selection. Once in a while, the parent selection process is running to identify the best one (with the highest WMEWMA routing quality to the base station) of the neighbors as the node's parent. Sensor data from the node are put in a queue for sending to the parent to forward to the base station. Since the MT protocol only uses the WMEWMA metric and not the hop count metric as the criterion for parent selection, a node may select a node with bigger hop count value as its parent. Thus it is possible to form cycles in the MT protocol. To deal with this issue, whenever the node receives an incoming sensor data packet from its child node, the source node address is checked and the corresponding neighbor table entry is labelled as a child to detect cycles in parent selection. When a cycle is detected, a new parent selection process will be launched to break the cycle.

The MT protocol has been implemented on motes and deployed in the UC Berkeley campus [37]. It has also been used in commercial wireless sensor network systems such as Crossbow's Environment Monitoring solution [54].

2.5.2 HAR Routing Protocol

HAR (Hierarchy-Based Anycast) protocol [11] is a protocol building a tree topology in which the base stations are root nodes. The overall structure of the protocol is, to some extent, similar to the mechanism used in Bluetooth scatternet topology formation such as the one described in [47].

Unlike SARP, but like the MT protocol, the HAR protocol is a proactive routing protocol. The tree topology is formed before there is any sensor data traffic. In the HAR protocol, initially, when the base stations are powered on, they start the tree topology construction by flooding Child Request messages to all nodes in the network. A sensor node caches a few Child Request messages it received from its neighbors. These neighbors are then stored in the node's Parental Candidate table. After that, each node selects the neighbor node from which it first received a Child Request message to be its parent. We can see that, in HAR, the parent selection metric is the time stamp, which is the same as the earliest-first parent selection scheme. There is no evaluation of the link quality in HAR. After a parent is chosen, the node also replies with a Child Reply message to the selected parent so that the parent node is aware of the event. The formed topology will stay static without any change unless there are sensor nodes that are joining or leaving the network.

Sensor nodes can be added to the existing network. A new sensor node attempts to find a parent by using the parent discovery process. The new node broadcasts a Parent Request packet to its neighboring nodes. Any node in the existing tree that heard the Parent Request packet sends back a Child Request packet to the new node. The new node then caches these replies and also records these neighbors in its Parental Candidate table. Again, the first node that replies to the Child Request is chosen as the parent.

When a node previously in the tree fails, the fact will be detected when its children nodes try to send sensor data to it for forwarding. The HAR protocol assumes that the low-level MAC protocol provides a mechanism to detect link breakages. When the sensor data from a node cannot be sent successfully to its parent, the child node detects it and then chooses a new parent from its Parental Candidate table. If no available node can be chosen as a new parent, the node executes the joining algorithm described above.

2.6 Summary

In this chapter, we surveyed several algorithms and protocols that are related to our research work. Some protocols are proactive and some are reactive. Furthermore, some dynamically evaluate and change network topology according to certain metrics while others do not. The difference in the characteristics make them suitable for different application scenarios. These existing algorithms and protocols lead us to propose our own algorithm and protocol.

3 Proposed Algorithm and Protocol

In this chapter, we propose a new anycast protocol called DAR (Dynamic Anycast Routing) for dynamic wireless sensor networks. We are interested in applications that require regular reporting of sensor data by sensor nodes, which moreover, may join and leave the network at will. This implies the use of a proactive topology formation algorithm, which must deal with issues of dynamicity. The characteristics and requirements of the sensor networks we consider are summarized below:

- Large scale homogeneous network.
- The network is for data-gathering application.
- Nodes may join and leave the network at any time.
- Dense node deployment.
- Lossy radio links.
- Multiple base stations and they may join and leave the network at any time.
- Sensor data can be sent to any of several available base stations.
- Sensor data is sent to a base station periodically from sensor nodes.

We do not consider nodes that are mobile in this thesis.

3.1 Traditional Anycast Scheme

The anycast problem in our sensor network is to find a method to send sensor data from every sensor node to exactly one of the base stations. In a traditional anycast scheme, a source node has to discover all possible routes from it to target destinations, and then choose one best route from them. In wireless sensor networks, the available routes from a

sensor node to base stations are subject to change at any time, so the route discovery has to be done whenever messages are to be sent. For a periodical data-gathering application, this kind of route discovery overhead is too high. Moreover, this onus on the source node makes anycast routing computing-expensive and memory-hungry. Because the source node in a wireless sensor network is a tiny sensor, which lacks this kind of computing resource, the traditional anycast scheme is not suitable for sensor network applications.

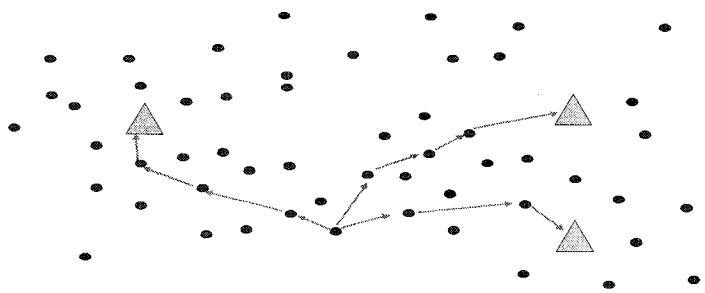


Figure 3.01: Traditional Anycast Scheme

3.2 Desired Topology

We consider a tree topology as a favorable topology for wireless sensor networks especially in data-gathering applications. From the literature survey, we can see that the tree topology is chosen in many protocols such as SARP, HAR, and MT. The merit of this topology is that the routing is very simple: sensor nodes just need to send their data to the single parent and the data will reach the base station. Since the application is a data-gathering type application in which sensor nodes send data periodically, the underlying protocol has the chance to monitor the network status periodically and then maintain the

network in the tree topology. Thus, route discovery is not needed when sending sensor data. This is an advantage compared to AODV, which needs to discover a route whenever messages are queued to be sent. We adopt the idea of building multiple trees rooted at base stations from the HAR [11] protocol. In this way, the anycast routing scheme is achieved very simply and efficiently. Figure 3.02 is a sample of node distribution and network topology in HAR and DAR.

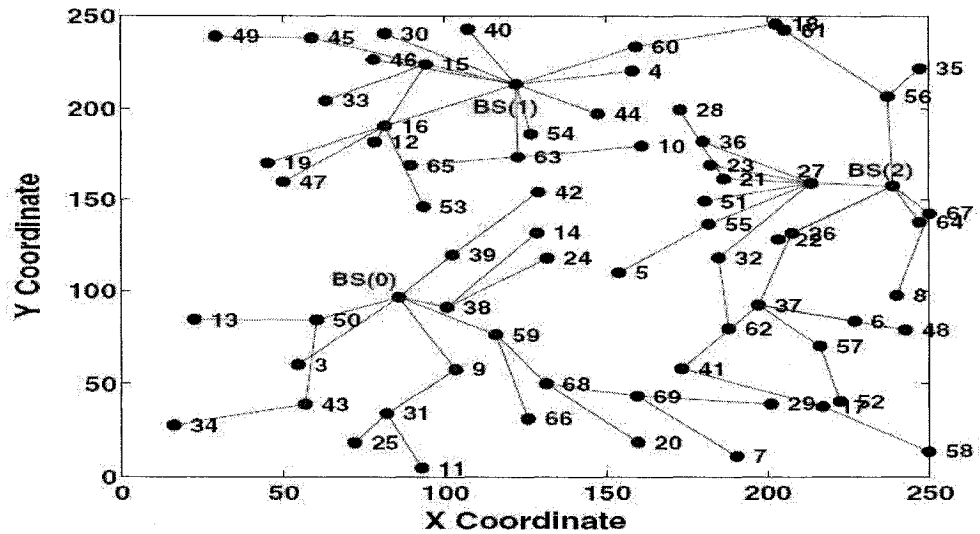


Figure 3.02: Nodes Distribution and Topology [11]

There are some sensor networks in which sensor nodes are mobile. However, in most sensor network applications, the sensor nodes do not move to new locations frequently, thus, a proactive topology formation approach is better than the reactive approach. Besides that, since sensor data are sent to the base station periodically, the reactive scheme will be having a much higher overhead. For these reasons, the DAR protocol

constructs the topology proactively similar to the schemes used in the MT and HAR protocols.

3.3 Routing Cost Evaluation

The core of a protocol for tree topology formation is selecting a node's parent according to a certain routing cost metric. The routing cost from a sensor node to the selected base station is the sum of the routing cost of its parent and the local routing cost (the routing cost from the sensor node to its parent). In traditional Distance-Vector routing protocols, hop count is the only metric. Hop count is also considered the only metric in the SARP and HAR protocols. We argue that lossy links constitute the actual situation for real sensor network deployment, so that link quality must be considered as a metric as well. The MT protocol considers the link quality as the only metric for route selection. A better approach is to consider both of them. This is the approach that will be used in our DAR protocol.

Like the WMEMWA algorithm used in the MT protocol, in our protocol, each node in the network locally broadcasts its known routing cost to a base station periodically. Each node also calculates the local routing costs according to the observed local link quality to its neighbors. Its neighbors thus infer their routing costs according to the above two parameters. The routing cost propagation process is a recursive process starting from base stations. So the most important issue in determining the routing cost is evaluating the local link quality. Estimating local link quality in wireless sensor network is a complex and time consuming process in itself and is still a hot topic for research [10]. We are not concentrating deeply in studying the link quality estimation, rather, our DAR protocol

will be using an algorithm derived from the WMEWMA [37] algorithm. We consider the WMEWMA algorithm to be a good local link quality evaluation scheme. The merits of this algorithm include the following:

- Estimating link quality by observing packet success and loss events, the scheme fits into periodical data-gathering applications well.
- Takes advantage of the broadcast nature of the wireless medium to passively estimate link quality by snooping on the channel.
- Simple implementation.
- Low in memory consumption.
- Reacts quickly to potentially large changes in link quality.
- No dependency on specific hardware such as RSSI (Received Signal Strength Indicator), so it can be used on any hardware platform. Using RSSI to measure local link quality is a widely used method, but it relies on the availability of the hardware in the radio.

The drawback of the MT routing protocol is that it is not suitable for unreliable networks. We define an unreliable network as a network does not need to re-transmit any lost packet. In other words, an unreliable network uses a UDP type datagram communication scheme. In contrast, the MT protocol assumes the upper layer applications require TCP type reliable communication. We argue that the unreliable transmission scheme is more suitable for data-gathering applications for the following two reasons: 1) sensor nodes are densely deployed. 2) sensor data is read periodically. If a sensor reading from a specific node at a specific time is not received, the sensor data management/analysis system

(usually a database system running on a resource rich server) can use some kind of interpolation operation to calculate an approximate value upon user's request. The interpolation operation is possible because the sensor data from the sensor's neighbors and the sensor data from the same node at different times are available. In summary, an unreliable network is acceptable by taking advantage of the fact there is some redundancy in the collected sensor data.

A router node consumes more energy than a non-router (leaf) node or a router node with fewer children because a router node has to forward all messages from its children. A shortcoming commonly seen in previous protocols is that a node with good radio link quality will be always selected by its neighbors as their parents and thus its battery will be consumed rapidly. The result is that a node with better radio link quality runs out of energy much earlier than nearby nodes with slightly worse radio link quality. To resolve this uneven power consumption problem, we borrow the idea of wear-leverage from the smart control firmware for Flash EEPROM. A Flash chip contains several sectors and each of them can be erased/written independently. The total number of times a sector can be erased/written is limited. In a typical usage scenario without wear-leverage firmware, it is the first sector will be always used while the others rarely used. So the first sector will be worn out very quickly and the chip is considered to be at the end of its life. When a wear-leverage firmware is used, all sectors will be used evenly because the firmware will use others sectors when it detected the first sector have been erased/written too frequently. Thus the life of the chip is prolonged significantly. Inspired by this feature seen in Flash control firmware, we propose that the energy that has been consumed by a node should be

considered as well at the time of forming the network topology. If this metric is considered, when a node has been in a router role for a while, the protocol should increase its estimated routing cost so that its nearby nodes will have a chance to be router nodes. This way, the energy consumption by all nodes will be balanced, which should prolong the network life.

3.3.1 Routing cost formula

Based on the above reasons, we propose a new routing cost metric used in DAR based on three variables: hop count, local link quality, and energy. In DAR, each node locally broadcasts its hop count and energy consumed, to its neighbors. This node is then a Parent Candidate to its neighbors. Each of its neighbors calculates its routing cost according to the following formula. Let P_1, P_2, ... P_n be the Parent Candidates of a node v. Then, the routing cost for v via Parent Candidate P_i is denoted $R_v(P_i)$ and is given by the following equation:

$$R_v(P_i) = H(P_i) + \alpha * \left(\beta * L_v(P_i) + (1-\beta) * E(P_i) \right)$$

Where:

- $R_v(P_i)$: the total routing cost from node v to a selected base station through this Parent Candidate P_i.
- $H(P_i)$: the hop count of this Parent Candidate P_i. The range of this value is from 0 (a base node) to 15 in our simulation.
- $L_v(P_i)$: the loss rate of radio link between this node v and the specific Parent Candidate P_i. This value is calculated using the WMEWMA algorithm and is represented as a percentage of packet loss. The value range is from 0% (perfect radio link) to 100% (unreachable). The common practice is to set a threshold for the loss rate and if the radio link is too weak, the link is not to be used for routing. We set the threshold to be 80%, that is, if the loss rate is larger than 80%, this link is not considered.
- $E(P_i)$: the energy that has been consumed by this Parent Candidate. This value is in the range of 0% (new battery) and 99% (unusable battery).
- α: tuning parameter to control the weight distribution between the hop count and a combined measure of link quality and energy consumption in the final routing cost. The value of it is 0, 1, 2, ... etc.
- β: tuning parameter to control the weight distribution between link quality and energy consumption. The value is in the range of 0% to 100%.

When the tuning parameter α is 0, the above DAR routing cost is the same as the traditional pure hop count-based algorithms.

When the tuning parameter α is not 0, evaluation of link quality and energy consumption is enabled in the calculation of the final routing cost. From the above value ranges, when α is 1, we get the value range of the DAR routing cost $R_v(P_i)$:

Minimum($R_v(P_i)$) = H(P_i), in which case the radio link is perfect and the battery is new.

Maximum($R_v(P_i)$) ≈ H(P_i) + 1, in which case the radio link quality is the threshold value and the battery is almost completely used up.

Note that $R_v(P_i)$ is always < H(P_i) + 1

This routing cost metric is designed so as to let the routing cost be mainly based on hop count. Using hop count as the main routing metric has the advantage of forming a loop-free tree topology quickly and the network tends to have a lower average hop count. If α=1, the maximum value of DAR routing cost is less than H+1 so that a router that has lower hop count value will always have a lower routing cost. The other two variables radio link quality and energy consumed are used to let the nodes with the same hop count values be chosen as routers fairly and evenly. The purpose of this design is to maximize the network lifetime. By using values of α>1, we enable the choice of longer, better quality paths over shorter, lower-quality ones.

The variable E (energy consumed) represents the willingness of a node to act as a router according to its energy consumed. The energy consumed by a node is modeled by its radio activities because a major part of energy consumption is radio transmission.

The tuning parameter β is designed to distribute the weight between the link quality and energy consumption. When β is 0%, link quality is not counted in the routing cost evaluation, and, when β is 100%, energy consumption is not counted in the routing cost evaluation. Various values of α and β have to be evaluated to find the optimal configuration to let the network achieve the best performance according to the metrics desired in the application.

3.3.2 Choice of tuning parameters

In this thesis, we are interested in both increasing the network lifetime and well as increasing the data delivery rate. Choosing $\alpha=0$ gives a pure hop count based protocol (called HC from now on). However, while the networks generated by the HC protocol have shorter routing paths and longer network lifetime, the data delivery rate is very low. The reason is that the HC protocol prefers parent candidates closer to base stations and it does not consider the link quality of the routing paths. On the other hand, the MT protocol always chooses nearby parent candidates with high level of link quality. Thus, the routing paths are longer but the delivery rate is high. Longer routing paths make the nodes near the base station heavily loaded and this makes the network lifetime short. It is clear that if we want to design a better algorithm, the merits from both the HC and MT protocols have to be combined.

When the parameter α is large enough, a node with higher hop count and lower loss rate has the possibility to have a lower final routing cost than a node that has lower hop count and higher loss rate. Of course, in this situation, the protocol is no longer hop count based. Assuming that a difference in link quality of l is enough to overcome the advantage of a routing path shorter by h hops, in a network with n nodes, the value of α should therefore be chosen as

$$\alpha > \frac{h}{(\beta * l)}$$

Note that h should probably be a function of the size n of the network. In a small network, a difference at most one hop between two routing paths may be considered in choosing a larger path over a shorter path. But in a longer network, it may be more acceptable to choose a path that is even 3 hops longer in the interest of a higher data delivery rate. The exact values of l and h will be determined experimentally, as shown in Chapter 7.

3.4 Adding and Removing Base Stations

Base stations are the target nodes of our anycast routing protocol. The MT protocol was working with only one base station. HAR protocol does not deal with the issue of dynamically adding or removing base stations. One of the goals of our protocol DAR is to allow base stations to be added and removed any time and the network topology will be changed automatically. Supported by this capability, the user can power up/down a base station for maintenance without service interruption. Mobile base stations that make periodic stops can also be supported by this capability.

3.4.1 Network address allocation

Like most protocols for current wireless sensor networks, our protocol uses simple numeric numbers as addresses for all sensor nodes and base stations. For simplicity, DAR allocates one base station address for every fixed number (let it be N) of sensor nodes. So, the addresses of 0, N, 2N, 3N, etc. are allocated to base stations. Other addresses are for sensor nodes.

3.4.2 Protocol running on base station

When a base station device is powered on, it first checks its network address. If the address is a base station address, it executes the base station algorithm. For a base station, its hop count and routing cost are fixed to the minimum value of 0. The algorithm executed continuously by the base station is:

- Broadcast periodically: hop count = 0, energy consumed = 0.
- Constantly listen on the wireless receiving port.
- If any sensor packet is received over the air, forward it to the local data collecting port.

There will be other programs listening on the local data collecting port to store and process the sensor data.

When a base station is powered down, the algorithm running on sensor nodes will detect the disappearance of the base station after a while and start to look for a new base station.

3.5 Adding and Removing Sensor Nodes

3.5.1 Initialization

When a sensor node is turned on, it will execute an initialization routine and then try to find a nearby node as its parent to join an existing tree.

The initialization process is as follows:

- Set its hop count to infinity (-1 in our simulation).
- Read battery meter.
- Set its parent to NULL.
- Clear neighbor table.
- Start a Periodic Transmission Timer.

3.5.2 Hop count and energy usage broadcast

The purpose of the Periodic Transmission Timer is to broadcast the node's hop count and energy consumed to the nodes nearby so that they can calculate the link quality between them and this node. The broadcast here is a local broadcast, not a flooding, since the message is not forwarded by any node that heard it. So the Periodic Transmission Timer executes the following task:

- Broadcast the hop count and energy consumed periodically to nearby nodes.

3.5.3 Neighborhood management

Upon receiving any packet over the air, the source address is examined to determine whether the node is a known neighbor or not. A known neighbor is a nearby node that has

already been recorded in the neighborhood table. In a dense sensor node deployment application, a node may be able to receive from many nodes. However, only a small number of them can be kept in the neighborhood table since sensor nodes have limited size of memory. When the neighborhood table is full, the node with worst link quality metric will be removed from the table. The neighborhood management module executes the following protocol:

Examine the source address

If (the source node is not in the neighborhood table)

If (the neighborhood table is not full yet)

Insert the source node to the neighborhood table

Else

Find the node with the maximal routing cost in the neighborhood table

Replace this node with the new node

Else the source node is already in the neighborhood table

Calculate the local radio link quality to this node

Calculate the total routing cost using the DAR algorithm

3.5.4 Parent selection

Once in a while, a parent selection algorithm is executed to choose a node in the neighborhood table to be the parent for routing sensor data. To allow the routing cost estimation algorithm to calculate a relatively stable link quality of its neighbors, the parent selection interval must be much larger than the interval of routing cost broadcast. In other words, the parent selection algorithm is running at a slower rate than the link quality estimation algorithm. Only the neighbor nodes with local link quality higher than

a certain threshold are considered candidate parents. The node with the lowest routing cost among the candidate parents is chosen as the parent. When a parent is selected, all sensor readings will be sent to this parent periodically. Upon receiving a sensor reading packet, the parent node will forward it to its own parent and eventually the data reaches a base station.

As we stated before, when the parameter α is larger than 1, the protocol is not hop count based. In this case, it is possible that the protocol creates routing loops. To deal with this issue, a loop detection and break mechanism is employed: whenever a node receives a data packet originated from itself (this means a loop is formed), it invalidates its current parent and selects a new one.

3.5.5 Dealing with node removal

A sensor node may run out of battery, and a base station may be turned off for maintenance. Its neighbor nodes must have the ability to deal with this dynamically. Since the DAR algorithm is monitoring the link quality periodically, if a node is powered down, its neighbors no longer hear the routing cost broadcast from it. Thus, its link quality records in its neighbor's neighborhood table will be degraded over time and eventually be considered dead. If the removed node is a leaf in the tree topology, its neighbors do not need to do anything besides degrading its link quality. If the removed node is an intermediate router in the tree topology, its children will be downgrading its link quality and new parents will be chosen when its link quality drops to not be the best one in its children's neighborhood tables.

There exists a situation that a node cannot find a proper new parent when its current parent was powered off. For example, a node may discover, on losing its parent, that all of its other neighbors are its children. In this case, the node will change its routing cost to infinite. Over time, its children will downgrade its link quality and choose other nodes as their parents. After that, the node will be able to choose one of its original children to be its new parent.

3.6 Summary

In this chapter, we presented our routing protocol DAR and the rationale behind our proposal. The DAR protocol consists of an innovative routing cost estimation algorithm, the base station addition/removal protocol, and the sensor nodes addition/removal protocol. The following table shows the comparison of features of DAR and other protocols:

	AODV	**SARP**	**ABS**	**HAR**	**MT**	**DAR**
Suitable for WSN	No	Yes	Yes	Yes	Yes	Yes
Possible to add/remove sensors	N/A	No	No	Yes	Yes	Yes
Multiple bases	N/A	Yes	No	Yes	No	Yes
Possible to add/remove bases	N/A	Yes	No	No	No	Yes
Proactive or reactive	React.	React.	Pro.	Pro.	Pro.	Pro.

Table 1: DAR features compared to that of other protocols

4 Design and Implementation

After the proposal of our protocol, we seek to implement it on both a simulation environment and real sensor hardware platform to test our work. In this chapter, we survey several development environments available for wireless sensor network development. Then we choose the proper toolset to design and implement our protocol.

4.1 Wireless Sensor Network Hardware

Over the last few years, several types of wireless sensor nodes, or *motes,* have been developed by the University of California at Berkeley. These motes are now widely used in both research institutes and industry. A mote is a battery powered device with very small physical size but integrates a micro processor, Flash ROM, RAM, EEPROM, sensors, a radio, an antenna, and a programming interface all together. The power consumption of a mote is very low and the memory size of a mote is very small. The following table lists the motes available when this thesis was started:

Mote	**Dot**	**Mica**	**Mica2**	**MicaZ**	**Telos**	**TelosB (Tmote)**
Year	2000	2001	2002	2004	2004	2005
CPU	ATmega163	ATmega103	ATmega128	ATmega128	MSP430	MSP430
ROM (KB)	16	128	128	128	60	48
RAM (KB)	1	4	4	4	2	10
Radio	TR1000	CC1000		CC2420, IEEE 802.15.4 compliant		

Table 2: Berkeley Motes

Among the above motes, the TelosB is the newest and the most attractive. This mote has a marketing name Tmote Sky designated by its producer Moteiv Corporation [2].

According to [2], "Tmote Sky is the next-generation mote platform for extremely low power, high data-rate, sensor network applications designed with the dual goal of fault tolerance and development ease. Tmote Sky boasts the largest on-chip RAM size (10kB) of any mote, the first IEEE 802.15.4 radio, and an integrated on-board antenna providing up to 125 meter range. Tmote Sky offers a number of integrated peripherals including a 12-bit ADC and DAC, Timer, I2C, SPI, and UART bus protocols, and a performance boosting DMA controller. Tmote Sky offers a robust solution with hardware protected external flash (1Mb in size), applications may be wirelessly programmed to the Tmote Sky module. In the event of a malfunctioning program, the module loads a protected image from flash. Toward development ease, Tmote Sky provides an easy-to-use USB protocol for programming, debugging and data collection."

The Tmote Sky mote also has a low price tag at about USD110 each. It was selected to be our sensor node for this research. About 40 of them were purchased by the Department of Computer Science. The following figures show the Tmote Sky hardware:

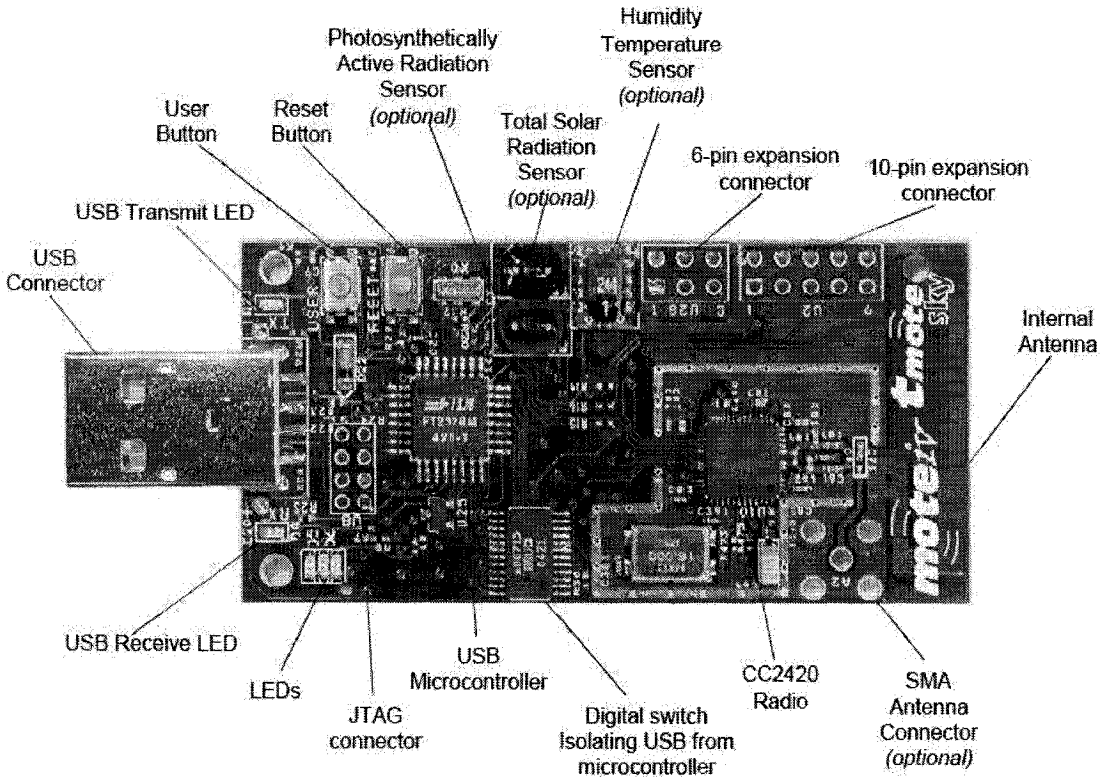


Figure 4.01: Tmote Sky mote front side [2]

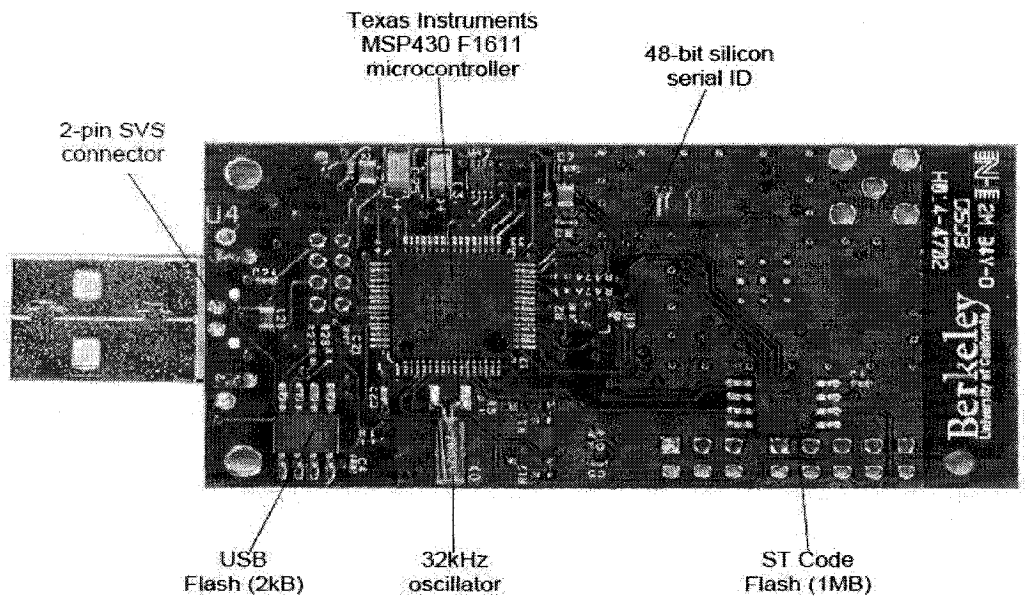


Figure 4.02: Tmote Sky mote back side [2]

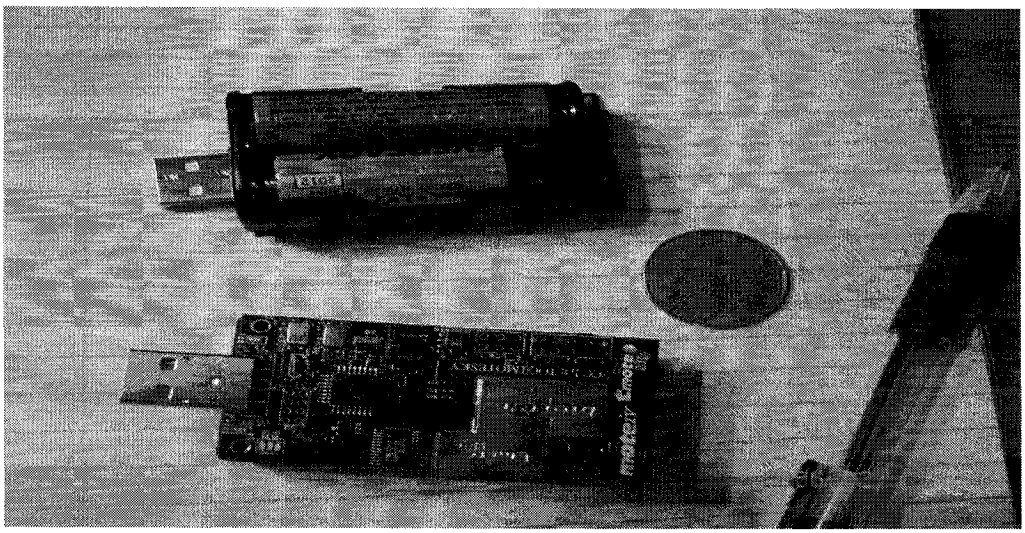


Figure 4.03: The Tmote Sky motes used in this research

4.2 Operating Systems and Programming Languages for WSN

An operating system is the most important systems software in a system. It provides a framework to manage hardware effectively and supports concurrency operations and provides a uniform programming interface. There exist many traditional embedded RTOS (real-time operating systems) long before operating systems specifically for wireless sensor networks are available. Commonly seen RTOSes such as uITRON, pSOS, VxWorks, UC/OS, etc., require memory footprints larger than the Tmote Sky mote can provide. Currently, to the best of our knowledge, there are four operating systems specifically designed for sensor nodes: TinyOS, Contiki, NutOS, and AmbientRT. Among them, TinyOS [1] and Contiki [40] support the Tmote Sky mote. For our research, the TinyOS is chosen as the operating system because it is the most commonly used one.

4.2.1 TinyOS and NesC

TinyOS is an operating system specifically developed for running applications in the tiny size sensor nodes. This open-source project was started at the University of California, Berkeley a few years ago and is still under active development. It has become the most commonly used operating system for wireless sensor networks [39].

TinyOS is tightly coupled with a programming language NesC (Networked Embedded Systems C) and most of the features of the operating system are implemented by the unique features of the NesC. NesC is an extension of C and it is specifically designed for networked embedded systems programming. Programming in NesC is a two-step process: first, individual function blocks called modules are developed with well-defined interfaces; then these modules are assembled together by a configuration to generate a monolithic application and this step is called wiring. By separating the building process into two steps, the development of individual modules becomes very independent of each other. Developers can simply use different configurations to re-use existing modules.

A module is either a functionality user or a provider and they communicate with each other through an interface An interface specifies the commands that have to be implemented by a provider module and specifies the events that have to be implemented by a user module. The NesC wiring glues modules to make a TinyOS application [1].

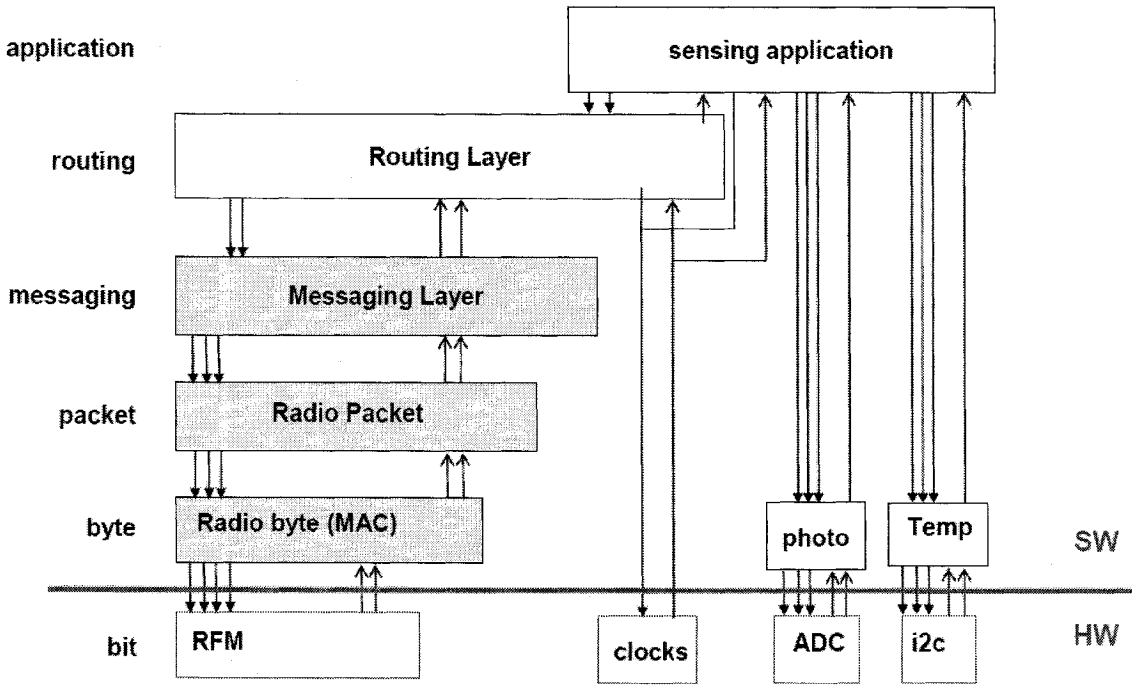


Figure 4.04: A sample TinyOS application wiring graph [24]

When designing the overall structure of the DAR protocol, we followed the convention of NesC and other good practices to separate the modules and configurations. Our work is made smoother thanks to the large user base and plenty of sample applications especially the SurgeTelos that existed in TinyOS (though they are not well documented). This is the snapshot of the code to implement the DAR protocol module:

```
includes AM;
includes DarMultiHop;

module DarSelectM
{
    provides
    {
        interface StdControl;
        interface DarRouteSelect;
        interface RouteControl;
    }
    uses
    {
```

```
        interface Leds;
        interface Timer;
    }
}

implementation
{
    ......
    typedef struct RPEstEntry
    {
        uint16_t id;
    } __attribute__ ((packed)) RPEstEntry;

    TOS_Msg routeMsg;
    bool gfSendRouteBusy;

    uint8_t findEntry (uint16_t id)
    {
        uint8_t i = 0;
        for (i = 0; i < ROUTE_TABLE_SIZE; i++)
        {
            if ((NeighborTbl[i].flags &
            NBRFLAG_VALID)
                && NeighborTbl[i].id == id)
            {  return i;
            }
        }
        return ROUTE_INVALID;
    }
```

The following is the snapshot of the code to implement the configuration to use the DAR protocol module:

```
includes DarMultiHop;

configuration DarRouter {

  provides {
    interface StdControl;
    interface Receive[uint8_t id];
      ......
  }

  uses {
    interface ReceiveMsg[uint8_t id];
  }
```

```
}

implementation {
  components MultiHopEngineM, DarSelectM,
GenericCommPromiscuous as Comm,    QueuedSend,
TimerC, LedsC;

  ReceiveMsg = MultiHopEngineM;

  MultiHopEngineM.SubControl ->
QueuedSend.StdControl;
  MultiHopEngineM.SubControl ->
DarSelectM.StdControl;
  MultiHopEngineM.RouteSelectCntl ->
DarSelectM.RouteControl;
  MultiHopEngineM.DarRouteSelect ->
DarSelectM.DarRouteSelect;

  MultiHopEngineM.SendMsg -> QueuedSend.SendMsg;
  DarSelectM.Timer ->
     TimerC.Timer[unique("Timer")];
   ......
}
```

From the above code, we can see that the NesC language's unique feature is modularization. NesC compiler is actually a translator, not a compiler, as it translates all NesC application source files and necessary libraries to an ordinary C file. Then, this C file is compiled by the standard and widely available gcc.

NesC is a new programming language, existing development IDEs do not recognize its syntax. This may make developers frustrated since most of them are used to the rich features provided by modern IDEs including syntax coloring, auto code-completion, etc,. To relieve this pain, a TinyOS plugin [42] has been developed for the popular Eclipse IDE to provide those features.

TinyOS has an event-driven kernel because a sensor network application is naturally event-driven for sensor event processing [23]. It supports two levels of concurrency: tasks and events. A task is similar to the concept of task in a traditional RTOS and it is for executing non real-time computation in FIFO style without preempting each other. An event is to handle a real-time interrupt coming from low-level hardware. Events are naturally asynchronous and have higher priorities over the tasks, thus they can interrupt the execution of tasks. This simple concurrency model of TinyOS allows high event-handling throughput while keeping the overhead significantly lower than in the traditional thread-based approaches [39].

TinyOS is a free and open-source software. Its source code is in CVS (Concurrent Version System) reversion control system hosted on Sourceforge [3]. The details of TinyOS internals are systematically described in [23].

4.2.2 Contiki

Contiki is also a widely used operating system for sensor networks and other embedded devices. According to [40], "Contiki is an open source, highly portable, multi-tasking operating system for memory-constrained networked embedded systems written by Adam Dunkels at the Networked Embedded Systems group at the Swedish Institute of Computer Science. Contiki is designed for embedded systems with small amounts of memory. A typical Contiki configuration is 2 kilobytes of RAM and 40 kilobytes of ROM. Contiki consists of an event-driven kernel on top of which application programs are dynamically loaded and unloaded at runtime. Contiki processes use light-weight protothreads that provide a linear, thread-like programming style on top of the event-driven kernel. Contiki

also supports per-process optional preemptive multi-threading, interprocess communication using message passing through events, as well as an optional GUI subsystem with either direct graphic support for locally connected terminals or networked virtual display with VNC or over Telnet."

The programming language used in Contiki is C. Usually the memory footprint for the same application in Contiki is slightly larger than that in TinyOS [39].

4.3 Simulators for Wireless Sensor Networks

A simulator is a very useful tool in any research area. It enables a developer to verify algorithms and protocols before they are actually running on real hardware. Developing protocols in a simulator is also easier because a simulator provides a way to study the system implementation in a controlled environment, explores system configurations that are difficult to physically construct, and observes interactions that are difficult to capture in a real-life system [41]. Another benefit of simulation is that it enables developers to simulate the execution of the protocols with a very large number of units which may not available in real deployment.

4.3.1 TOSSIM

TOSSIM (TinyOS SIMulator) [41] is a simulator specifically designed for simulating motes running TinyOS. Tossim is used in our research because it can execute the same code (source code level compatible) written for TinyOS motes, which is a tremendous advantage.

TOSSIM simulates the TinyOS behavior by re-implementing the lowest part of the TinyOS functionality on a Personal Computer environment (Unix and Cygwin systems). It compiles the exact same application source code for TinyOS into a binary executable on Unix, thus the developers do not to write different code for the simulator and the real hardware. Hardware interrupts in TinyOS are simulated by discrete simulator events in TOSSIM. The simulator captures the behavior of the radio in bit level which is in a very fine grain. TOSSIM also provides a rich toolchain running on the PC host to help the developers to debug the TinyOS applications. The major part of the TOSSIM functionality is implemented in JAVA and Python.

4.3.2 NS-2 Simulator

Before special simulators specifically for sensor networks were available, the popular generic network simulator NS-2 (Network Simulator version 2) [6] was widely used in all kinds of network research areas including LAN, Internet, MANET, and WSN (with special extensions). NS-2 provides rich support for simulation of TCP, routing, and multicast protocols over wired and wireless networks [6]. Some routing protocols we surveyed previously, such as SARP and HAR, are implemented and simulated in this simulator.

The programming language used in NS-2 for implementing protocols is C++. A simple script language TCL is used to configure and interact with the C++ implemented modules. User implemented function modules are integrated to be part of the NS-2 simulator and the resulting simulator in a sense is a TCL script interpreter for executing user provided

TCL scripts. The NS-2 simulator is supported by various systems including Linux, Unix, and Windows.

4.4 Reversion Control System

Any non-trivial software development will benefit from putting the source code under a reversion control system. Before selecting a specific reversion control system, we want the reversion control system to have the following features:

- Free and open-source.
- Widely used and supported.
- Supports reversion history, tags, and branches.
- Multiple developers can work on the same files concurrently (non-locking mode).
- Supports merges.
- Supports Internet based repositories.

Based on the above criteria, we determined that CVS and Subversion [4] are our candidates. CVS is the most widely used reversion control system nowadays. TinyOS is hosted on Sourceforge's CVS system. The Subversion is a relatively new system that was developed initially intended to replace CVS. Whether it achieved this goal or not is left to be debated. Since it uses the approach of database-based repository to organize the reversion files, it is not totally backward compatible with CVS, which uses separate text files.

The Sourceforge web site provides free CVS and Subversion hosting for open-source software development. We tested both the CVS and Subversion systems on Sourceforge

and found that using Subversion is more smooth and straightforward on Sourceforge. So we selected the Subversion to be the reversion control system for our research. All of our work is then tracked by this system on the Sourceforge site. The code is available to the general public and anyone at anytime can retrieve it from anywhere through Internet access.

4.5 Architecture

We designed and implemented the DAR protocol. At the same time, we also implemented a demo application named DARapp to use the DAR protocol. The DARapp application uses the network constructed by DAR to send messages to the base stations. The relationship of the components is as in the following diagram.

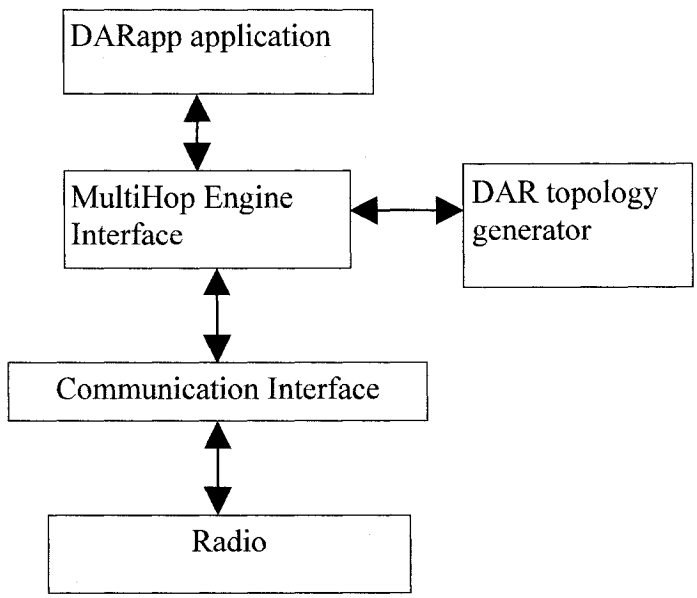


Figure 4.05: Architecture of DAR and DARapp

The DARapp application linked with DAR and TinyOS library will be the final executable. All motes including base station nodes and sensor nodes are loaded with the

same binary image. A mote executes base station or sensor node protocol according to its network address. The outline of the program is like this:

If (network address is base station address)
Run base station protocol
Else
Run sensor node protocol

There are two reasons that we chose to use the same binary image for both base station and sensor node:

1. TOSSIM requires that all nodes execute the same program.
2. Loading the same image to motes hardware is easier.

4.6 Summary

In this chapter, we surveyed the tools suitable for developing sensor network applications. We designed and implemented our protocol using the selected tool set.

5 Experiments in a Simulation Environment

After the implementation, first we need to verify the correctness of our protocol in the simulator TOSSIM. There are two reasons we use a simulator here: 1) Debugging and verifying algorithms and protocols are easier in a controlled, reproducible environment provided by a simulator; 2) We want to simulate more motes than we actually have.

5.1 Topology Display Tool

The primary goal of the simulation is to find out whether the DAR protocol generates the network topology as we designed. The TOSSIM simulator itself does not provide any means to display network topology, instead it provides a flexible GUI framework in which plugins can be added by users to implement desired functionality. The GUI framework in TOSSIM is called TinyViz, which is a Java visualization environment. "Users can write new plugins, which TinyViz can dynamically load. A simple event bus sits in the center of TinyViz; simulator messages sent to TinyViz appear as events, which any plugin can respond to. For example, when a mote transmits a packet in TOSSIM, the simulator sends a *packet send message* to TinyViz, which generates a packet send event and broadcasts it on the event bus. A networking plugin can listen for *packet send events* and update TinyViz node state and draw an animation of the communication. Plugins can be dynamically registered and deregistered, which correspondingly connect and disconnect the plugin from the event bus. A plugin hears all events sent to the event bus, but individually decides whether to do anything in response to a specific event; this keeps the event bus simple, instead of having a content-specific subscription mechanism." [41]

Following TinyViz plugin's convention, we designed and implemented a plugin named DarTopoPlugin in Java to display the network topology generated by the DAR protocol. The main data structure in the plugin is an array holding each mote's parent address. The plugin learns each mote's parent by monitoring and analyzing the messages sent by the DARapp application. In DARapp, the message is always sent from a child to a parent. Whenever this type of message is received from TinyViz's event bus, we extract the source and destination addresses and update the source mote's parent address record accordingly. Then, we draw an arrow from the source (child) node to the destination (parent) node. Thus, the network topology is visually presented on the TinyViz panel.

.

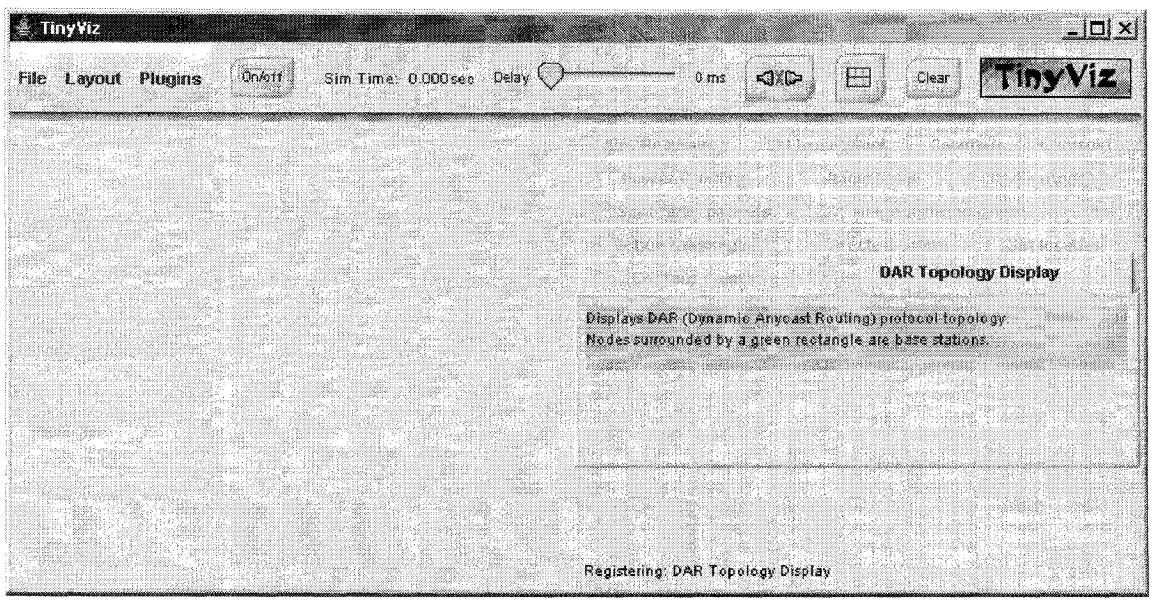


Figure 5.01: DAR topology plugin in TinyViz

5.2 Radio Model

Besides implementing a topology display tool, we also need to determine the layout, and most importantly, the radio model of the simulated nodes before the simulation can be actually started. In TOSSIM, the radio model is used to represent the radio link quality (the loss rate) between each pair of motes.

According to [41], there exist two radio models in TOSSIM: simple and lossy. In the "simple" radio model, all nodes can hear from each other and any bit transmitted is assumed received without error. Although there is no receiving error caused by radio transmission, there is still a chance that nodes receive corrupted packets because more than one node can start transmitting at the same time. In that situation, every node hears the overlap of the signals. The simple model is useful for testing single-hop protocols and TinyOS components for correctness.

In the "lossy" radio model, the radio link between any pair of nodes is lossy and there is a loss rate associated with this lossy radio link. The value of the loss rate is in the range from 0% to 100%, representing the probability a bit sent by the source node will be corrupted when the destination node hears it. For example, if the loss rate value is 0.05, any bit transmitted will be having a 5% chance of being received corrupted. In TOSSIM, the lossy radio model only considers interference and corruption. The environmental noise is not taken into consideration. In real life situations, the noise will affect the transmission error probability as well.

To simulate a multi-hop protocol execution, the lossy radio model must be used. TOSSIM has a Java utility (net.tinyos.sim.LossyBuilder) to generate a lossy radio model from the physical layout of motes. This utility models loss rates observed empirically in the experiments performed in [37]. In TOSSIM, each mote is assumed to have an effective transmission radius of 50 feet, with the transmission error rate increasing with distance from the center.

For this research, we used 49 motes in a layout of 7 by 7 grid with a grid space of 22 feet. Grid space is the horizontal and vertical distance between two neighboring nodes. As mentioned earlier, there is a tool provided in the simulator for building a radio model. The typical lossy rate (using the node 0 as an example) as given in the text file output by the tool is:

0:1:0.0
0:2:0.5
0:7:0.029419
0:8:0.024351
0:9:0.5
0:14:0.5
0:15:0.5

In this example, packets sent from node 0 to node 1 will be received with 0% loss rate and packets sent from node 0 to node 2 will be received with 50% loss rate. If the loss rate value between a pair of nodes is not given in the output file, the loss rate is 100% between them. In the above example, the loss rate from node 0 to node 3 is not shown; this means that node 3 can not hear from node 0 at all.

The physical layout of node 0 and its neighbors in the 7x7 grid is illustrated in Figure 5.02:

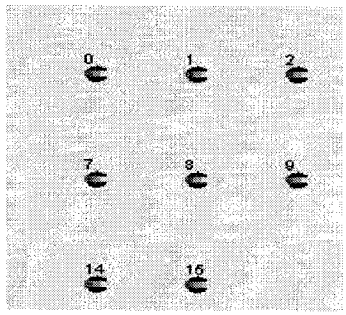


Figure 5.02: Node 0 and its neighbors

Among those 49 motes, nodes 0, 20, and 40 are set to be base stations in the simulation. (Nodes 20 and 40 are not neighbors of node 0 and are therefore not shown in Figure 5.02)

5.3 Execution Result

Using the above radio model and the DAR topology display tool, we run the simulation and observe the process of the anycast tree creation. The formation of anycast trees started after a few seconds of estimating routing costs and is shown in Figures 5.03, 5.04, and 5.05. Figure 5.04 shows more and more motes joining the anycast trees gradually. Figure 5.05 shows the final topology; all motes are in the anycast trees.

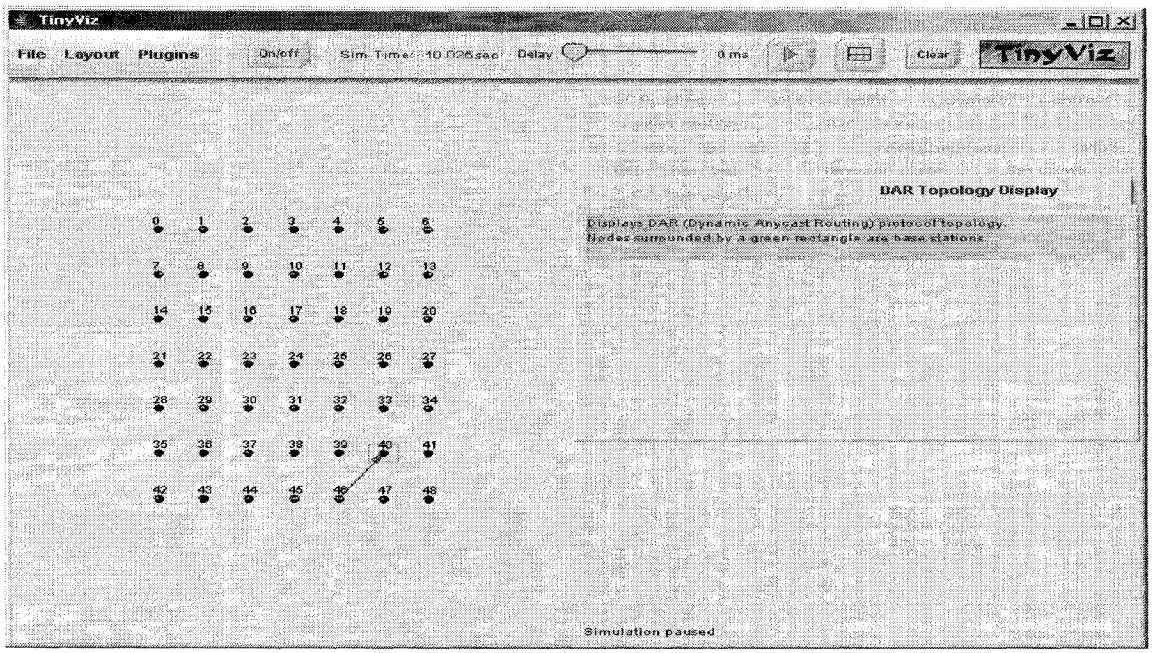


Figure 5.03: The formation of anycast trees. Nodes 0, 20 and 40 are base stations

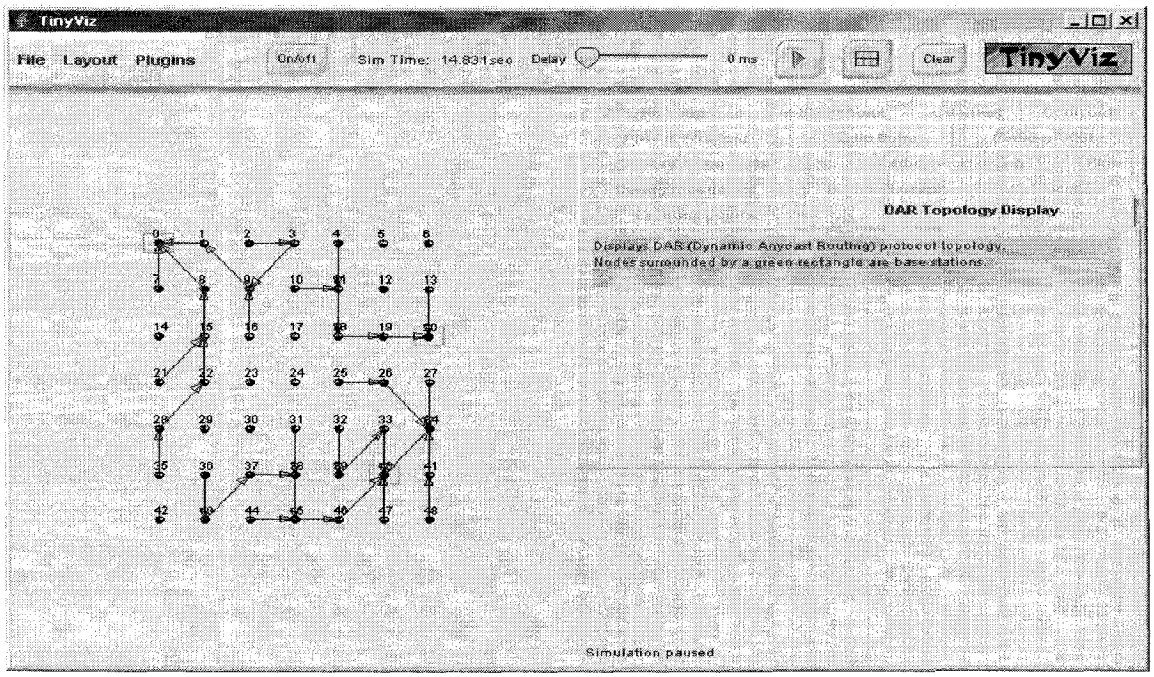


Figure 5.04: Motes joining the anycast trees

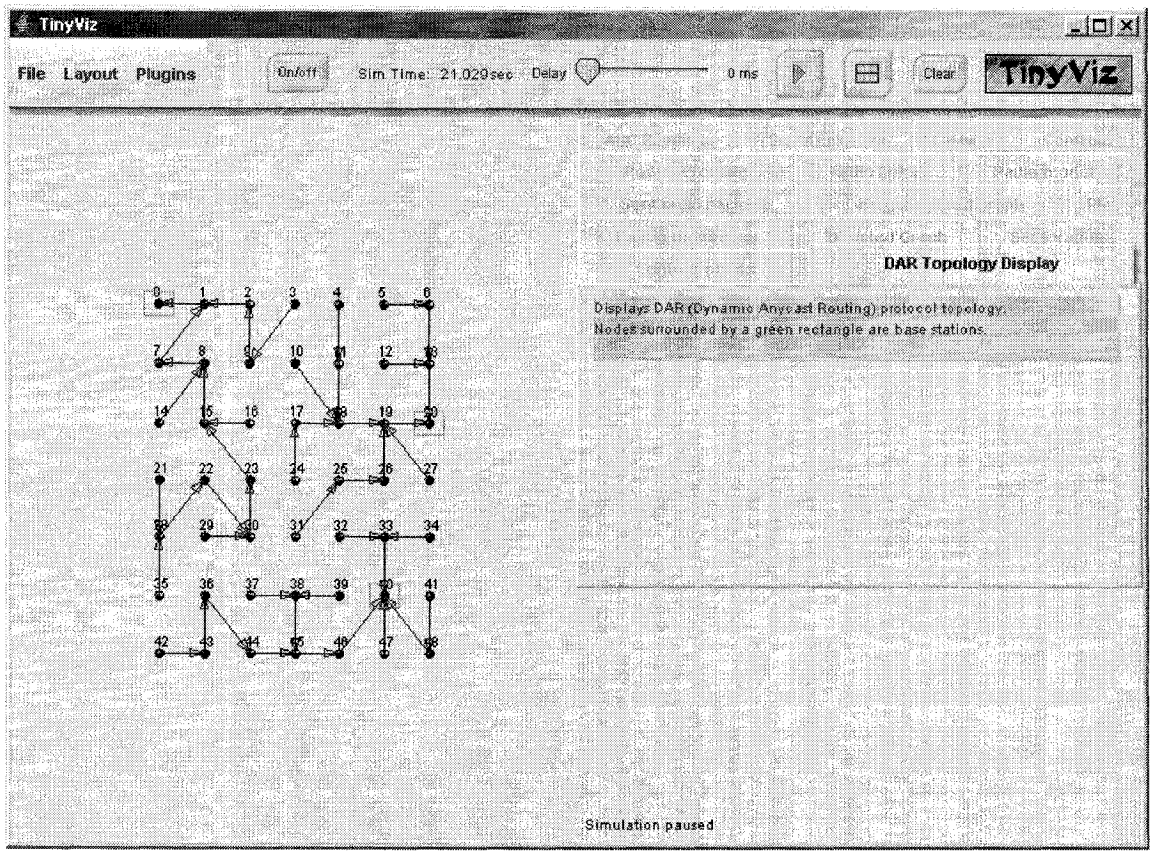


Figure 5.05: Final topology

Over time, the topology will change slightly due to the following two reasons:

- The routing cost metric will change following the motes' energy consumption over time.
- The random nature of TOSSIM radio transmission simulation.

Figure 5.06 shows a topology change that occurred due to a routing cost change.

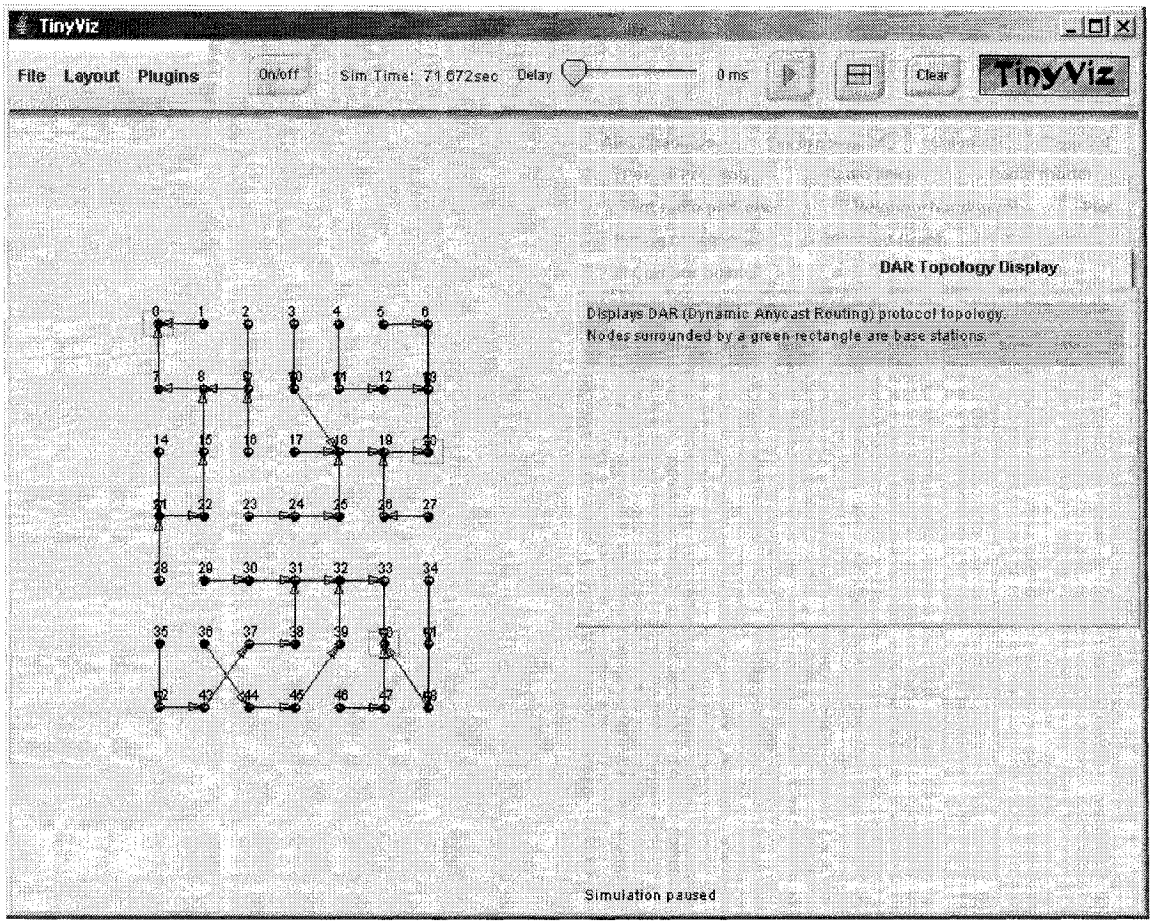


Figure 5.06: Topology change following routing cost change

5.4 Adding/Removing Base Stations

So far, we have simulated the execution in a static network. All base stations and sensor nodes are turned on at the same time and no node failure occurred. To simulate adding/removing base stations or sensor nodes, we need to use the Tython [48] scripting interface provided by TOSSIM.

Tython (TinyOS Python) provides a Python console to users for interacting with the TinyViz GUI through a scripting interface. Tython is based on Jython and Jython in turn

is a Java implementation of the Python programming language. Tython combines the features of Java and Python and thus enables users to import Java classes within Python. Since most PC side tools in TinyOS and TOSSIM are written in Java, users can access those tools from the interactive Tython interface to control TOSSIM simulator execution. The control of motes in TOSSIM is exposed by the "simcore" module in Tython [48]. So, after importing this module to Tython, we can turn on/off any mote in TOSSIM to simulate the adding/removing base station or sensor node.

Figures 5.07 and 5.08 show the toplogy resulting from turning off, in turn, base station 0, and 20 respectively, and Figure 5.09 shows the topology resulting from turning on all base stations again.

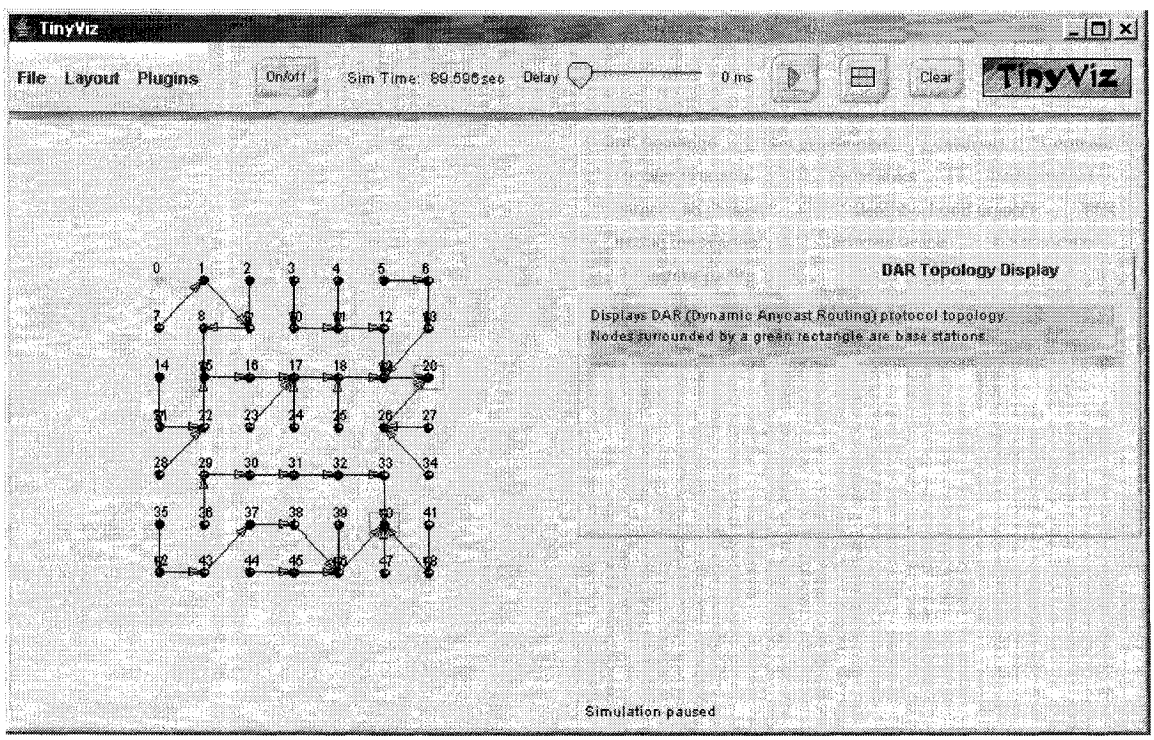


Figure 5.07: Topology after base station node 0 is turned off

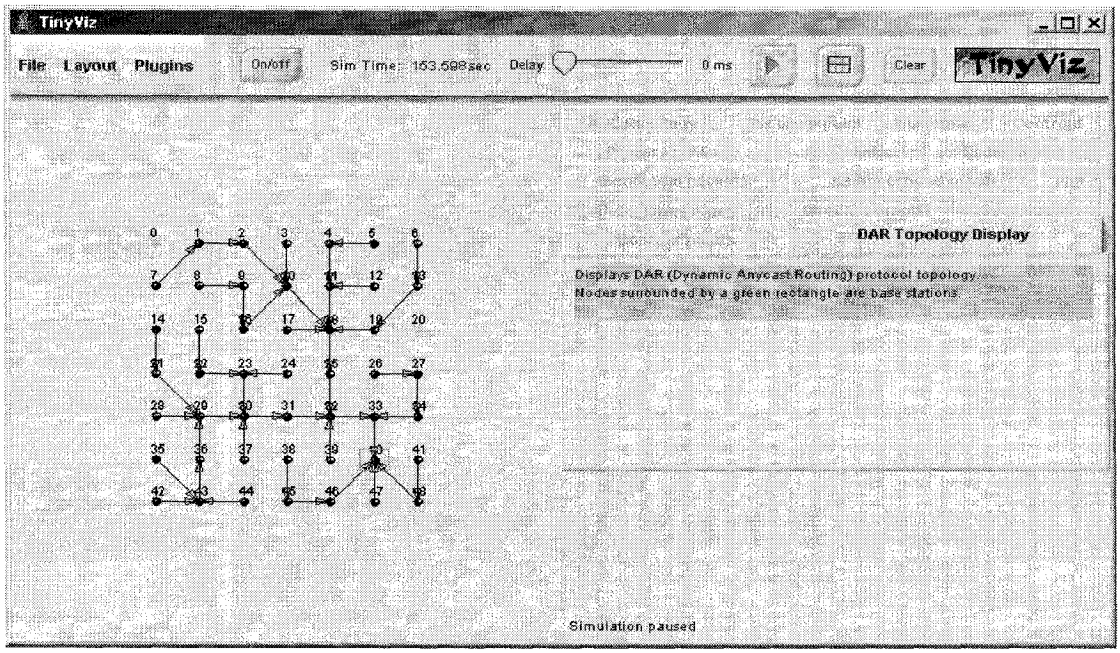


Figure 5.08: Topology after base station node 20 is also turned off

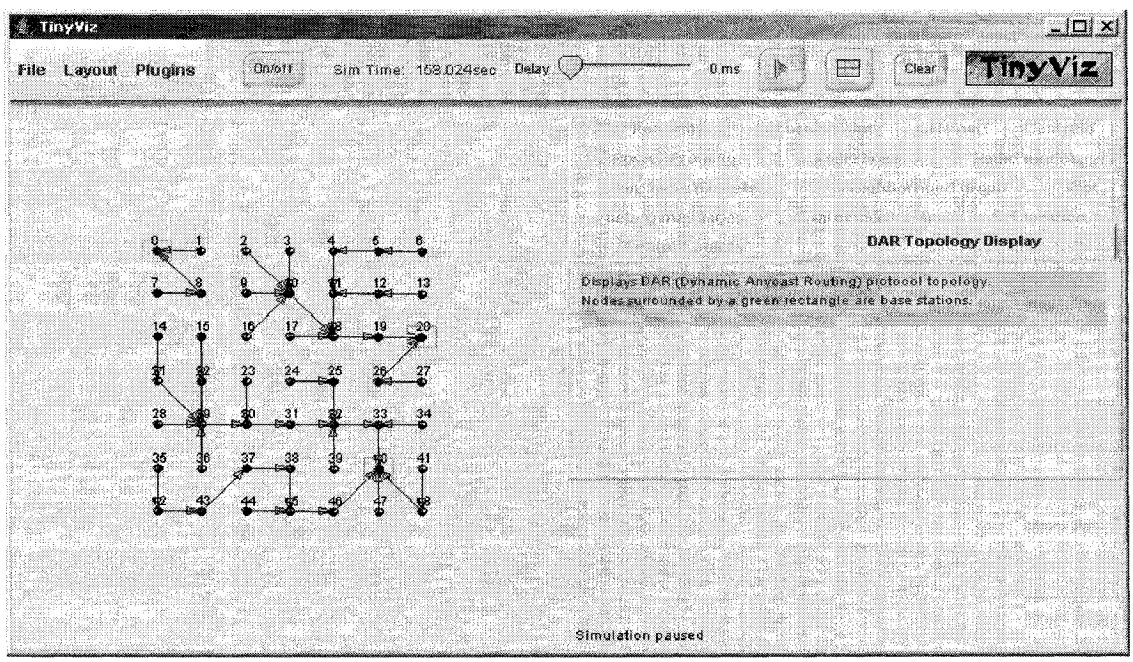


Figure 5.09: Topology after all base stations are turned back on

5.5 Adding/Removing Motes

The same method used in turning on/off base station is used to turn on/off sensor nodes. When the nodes 11 (a router node) and 28 (a leaf node) are turned off, the topology changed to the one shown in Figure 5.10. If we turn on the sensor nodes again, they will join the anycast trees after a while, as shown in Figure 5.11.

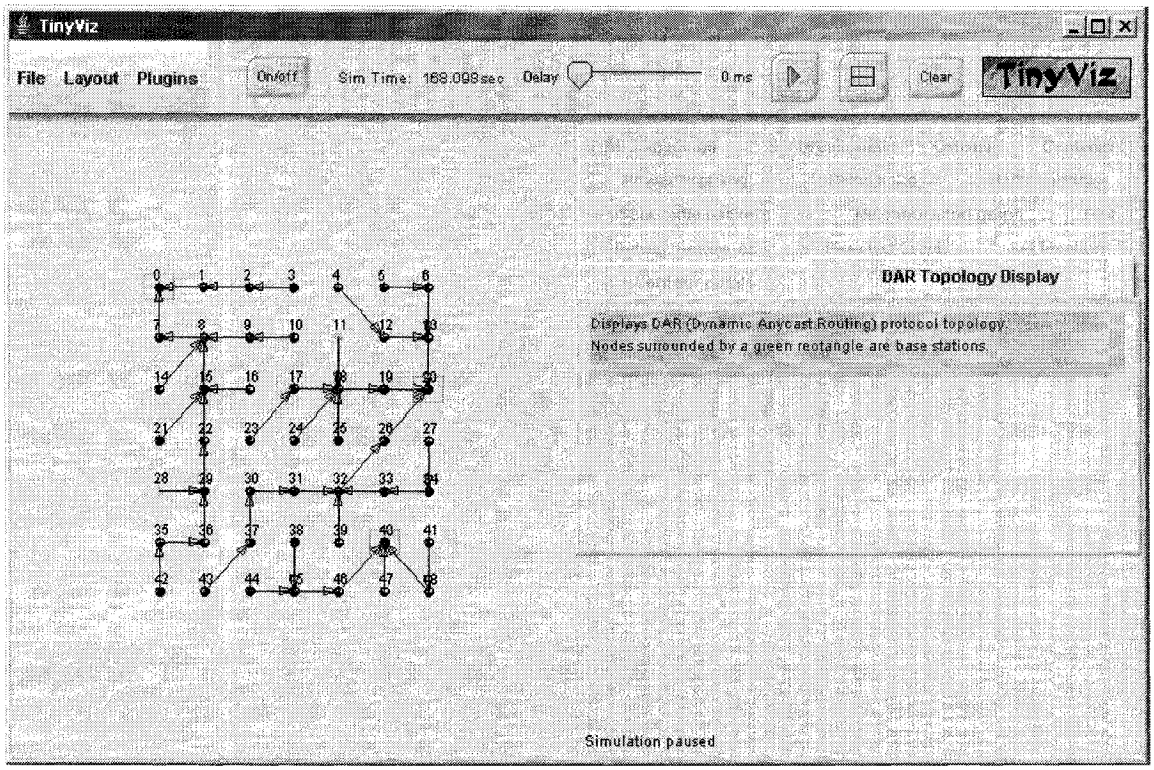


Figure 5.10: Topology after sensor nodes are removed

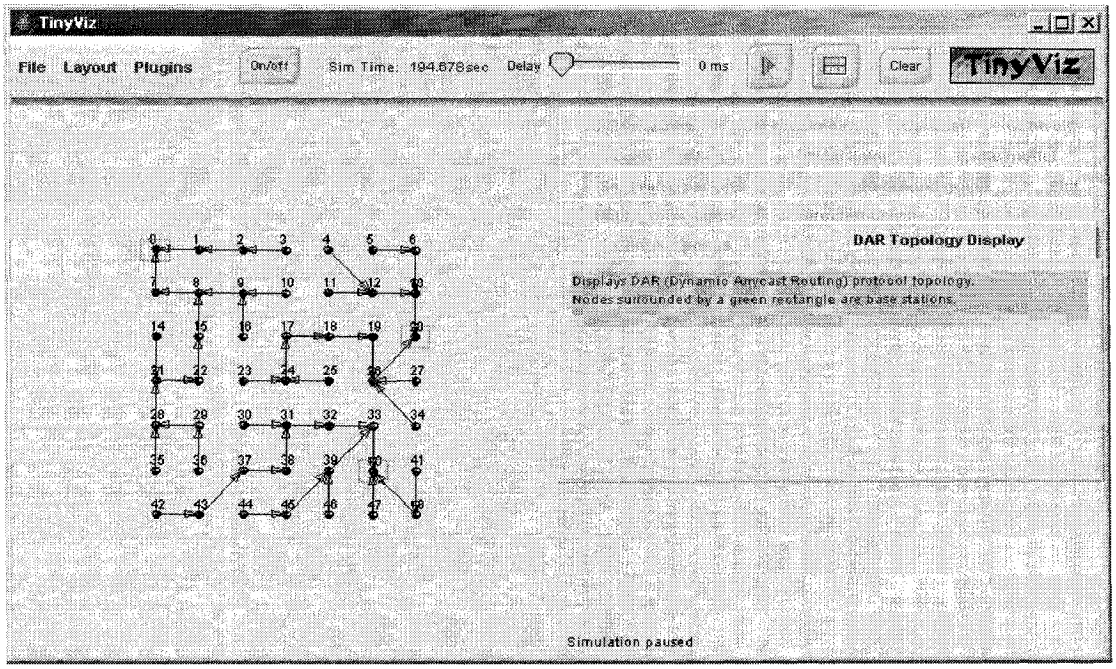


Figure 5.11: Topology after adding new sensor nodes and the base stations

5.6 Summary

In this chapter, we executed the DARapp application in TOSSIM to test the correctness of the underlying DAR routing protocol. To completely validate the correctness of the protocol in all possible situations is difficult due to the nature of the randomness of the scheduler of the simulator. Even in a simple purely hop count based protocol, a node can be blocked from hearing from its neighbors with shorter paths for a while. When this happens, the node will choose a neighbor with a longer path to be its parent. A more thorough approach is to track the states of the network and all individual nodes in the entire executing process. Thus the problem is in the field of snapshotting and analyzing the global states in distributed systems. This is outside the scope of this thesis.

6 Experiments on Real Hardware

Many protocols implemented and tested in a pure simulation environment look sound but fail to act as expected after porting to real hardware [44]. The reason behind this issue usually is that a simulator cannot provide an exact environment compared to real life hardware. Thus the flaws in protocol or implementation can not be discovered in the simulator. For this reason, we believe experimenting with real hardware is the ultimate validation of a protocol and its implementation.

6.1 Hardware Setup

The Tmote Sky motes are the hardware on which we run our test. Base stations and sensor nodes are the same mote hardware and all motes also are programmed with the same program code. A mote executes base station or sensor node protocol according to its network address.

Sensor nodes communicate with each other and to the base station through wireless 802.15.4 radio.

Each base station mote is connected to an USB port of a Tmote Connect gateway. Tmote Connect is a gateway for connecting Tmote Sky motes to a wired LAN. The motes connected to the gateway can be remotely controlled (including the functionality of re-programming, reset, reading serial data) through the gateway. Data from sensor nodes received by base station node will be forwarded to the gateway through the USB port. The

Tmote Connect gateway runs a version of embedded Linux operating system. It forwards the data received from its USB ports to network sockets to be accessed by host applications (such as topology display tool) running on a PC.

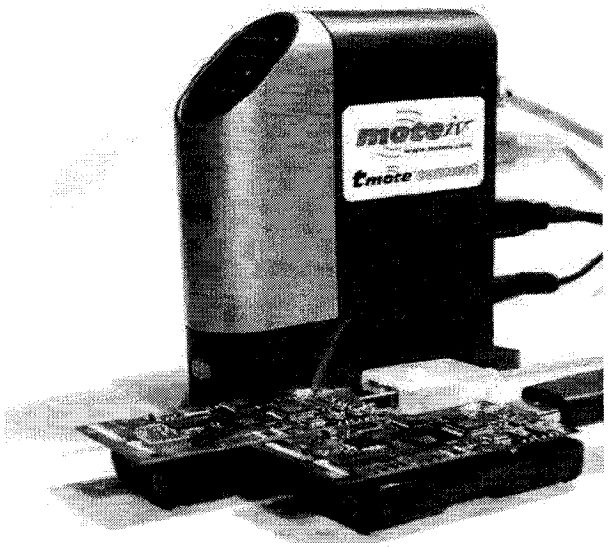


Figure 6.01: Tmote Connect Gateway with two connected motes [2]

6.2 Topology Display Tool

Just like the topology display tool we develop in TOSSIM, a topology display tool is needed here to monitor the topology generated by the DAR protocol as the primary goal of the experiment is to find out whether the DAR protocol generates the network topology as we expected. Fortunately there is already a tool named SurgeTelos written in Java in TinyOS. It can talk to the gateway through a LAN and then display the sensor network topology by analyzing data packets received by the base station.

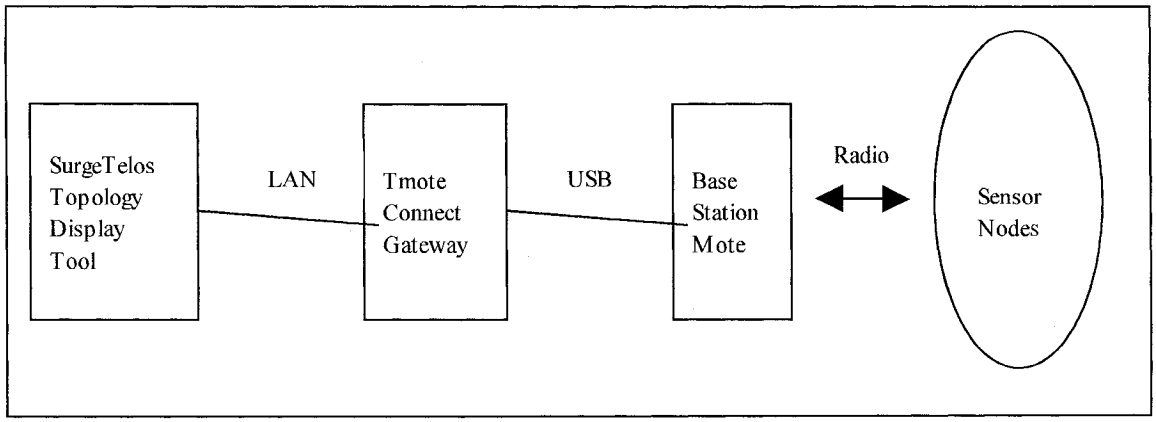


Figure 6.02: System connection

6.3 Radio Transmission Range and Motes Layout

Before determining the layout of the motes, we need to find out the actual radio transmission range of the motes in our test lab. Then, we can place the motes in a way such that they form a multi-hop network. Otherwise if all motes can hear each other, they will form a big single hop network.

We implemented a small utility program, similar to the example available in the TinyOS tutorial [1], to measure effective radio transmission range. The program executes the following code to send out a message every second and toggle an LED whenever a message is received:

Forever
If (one second elapsed)
Send a message out of the radio. Toggle the Red LED
If(a message is received over the air)
Toggle the Blue LED

This program is then loaded into two motes. The motes' Red LEDs will be blinking constantly to indicate that the programs are running. When the two motes can hear each other, their Blue LEDs should blink (changing color every second). Beyond a certain distance, a mote will no longer be able to hear from the other mote, its Blue LED will stay steady in a color. The actual radio transmission range is changing constantly due to the nature of the environment and noise from other wireless devices such as 802.11 devices, 2.4G Hz cordless phones, and Bluetooth devices just to name a few. For this reason, at a certain distance, a mote may be able to hear about 99% of the packets sent from the other mote. We consider this distance to be the effective radio transmission range.

The CC2420 radio transceiver chip used in the Tmote Sky motes can be set to work with transmission power ranging from 0 dBm (1 mw) to -25 dBm (0.003 mw). Setting up the transmission power is done by modifying the compiler option CFLAGS [2].

Using the above utility program, we measured the transmission ranges with both the maximum and the minimum transmission power. In the test, we found that in most cases, the radio transmission range is asymmetric, that is, in certain distance, the mote A can hear from the mote B, but the mote B can not hear from mote A. This table is the measurement result of the average radio transmission range:

Tx Power	Tx Range
0 dBm	58 feet
-25 dBm	3.2 feet

Table 3: Effective radio transmission range

Due to the limitation in the room size, we decided to use the minimum transmission power -25 dBm in all of our motes. This way we can place the motes close to each other and yet form a multi-hop network.

We used 12 motes placed in a 3 by 4 grid to conduct the test. The grid space is set to 1.5 feet. The node IDs for those motes are from number 0 to 10, and number 20. Among them, the nodes 0 and 20 are base stations and the others are sensor nodes. Figures 6.03 and 6.04 show the layout of motes and the actual hardware devices in the test.

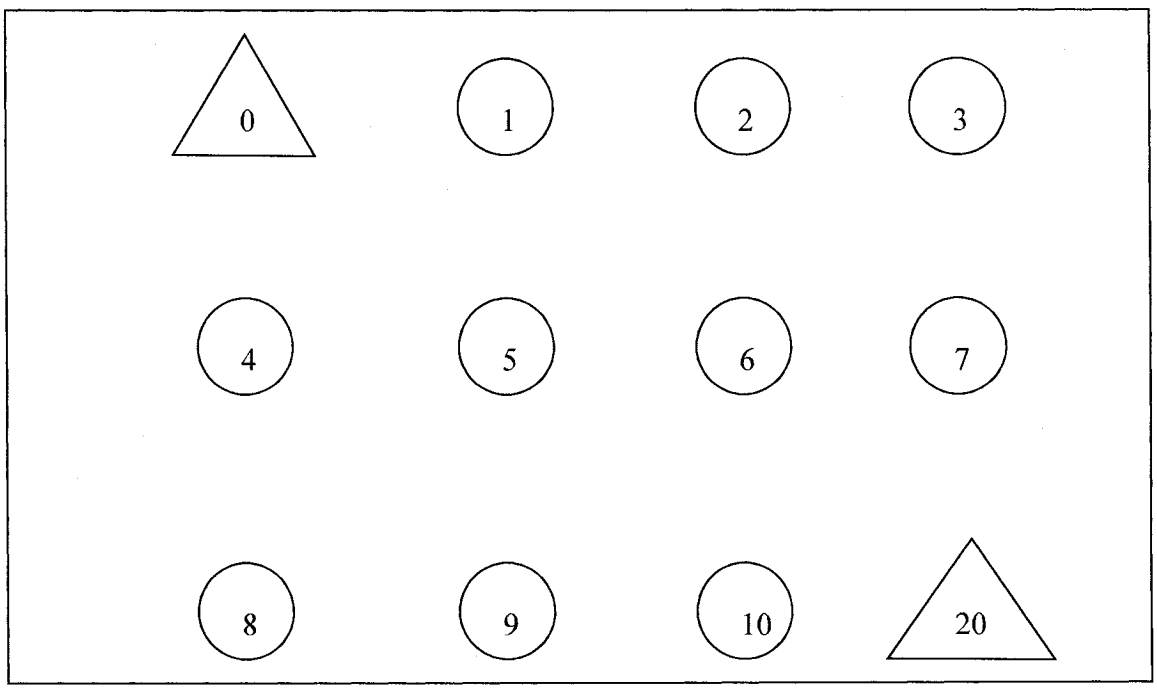


Figure 6.03: Physical layout of motes. Grid space of 1.5 feet.

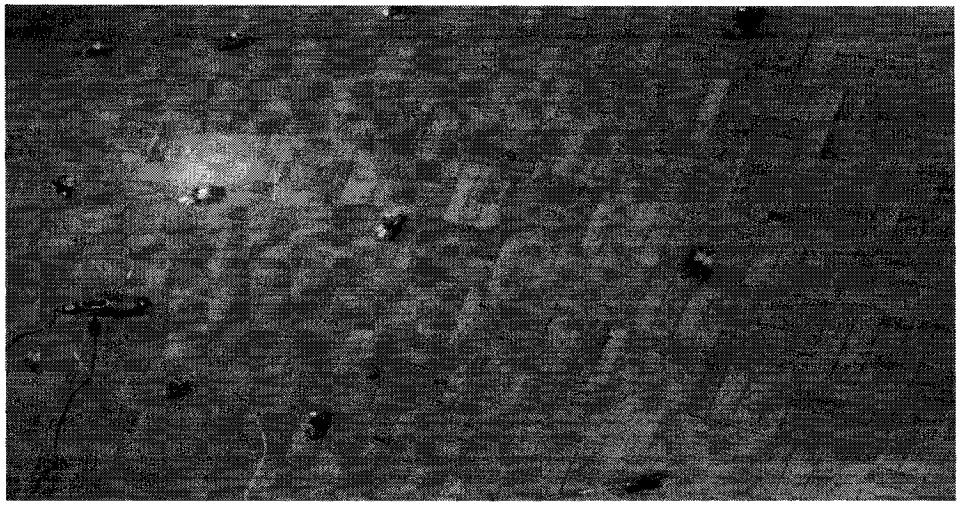


Figure 6.04: Actual devices (motes and gateway) in the test.

6.4 Execution Result

Like the execution in the simulator, the motes formed anycast trees successfully as we expected. The topologies from the base station node 0's view and base station node 20's view are shown as Figures 6.05 and 6.06 respectively.

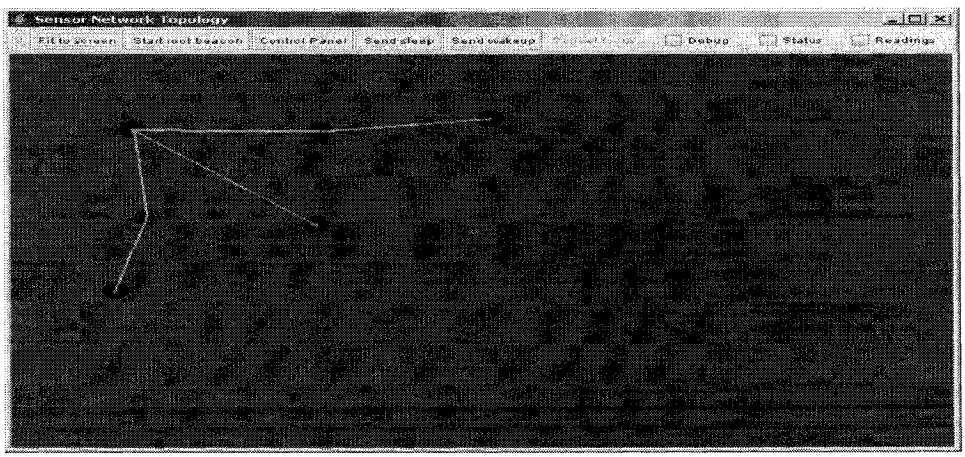


Figure 6.05: Topology rooted at base station 0.

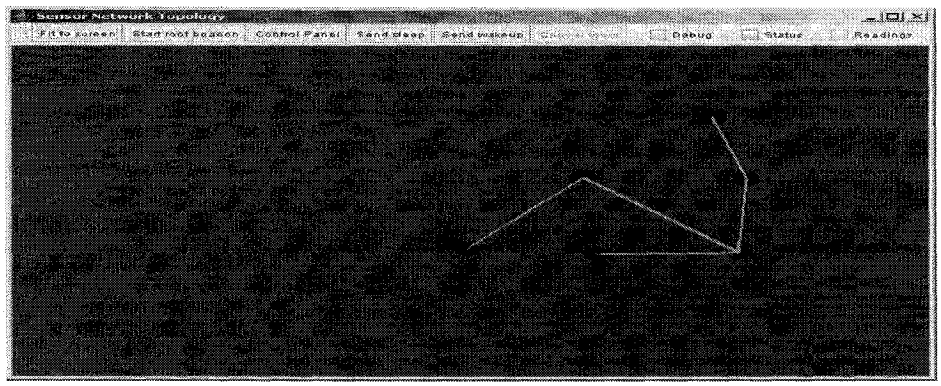


Figure 6.06: Topology rooted at base station 20.

As in the TOSSIM simulator, over time, the routing cost estimation metric will change and the topology will be changing accordingly.

6.5 Adding/Removing Base Stations and Motes

After a base station is powered down, the sensor nodes previously connected to it will gradually join another tree. For example, when base station node 20 is turned off, the network topology becomes a single tree rooted at base station node 0, as shown in Figure 6.07.

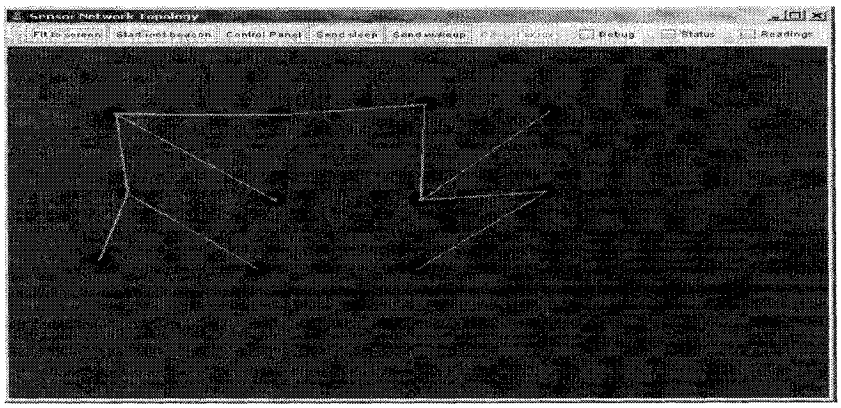


Figure 6.07: After base station node 20 is turned off.

When a router sensor node is powered off, its children gradually downgrade its link quality and eventually select new parents automatically. Figure 6.08 shows the new topology when the router node 7 is turned off.

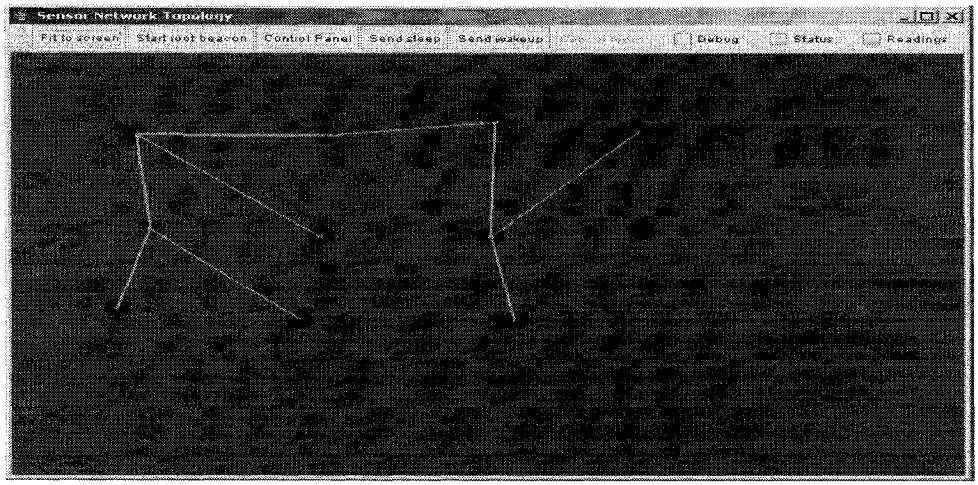


Figure 6.08: Topology after router node 7 is turned off.

When node 7 was turned back on, and node 4 was turned off, the topology changed accordingly as shown in Figure 6.09.

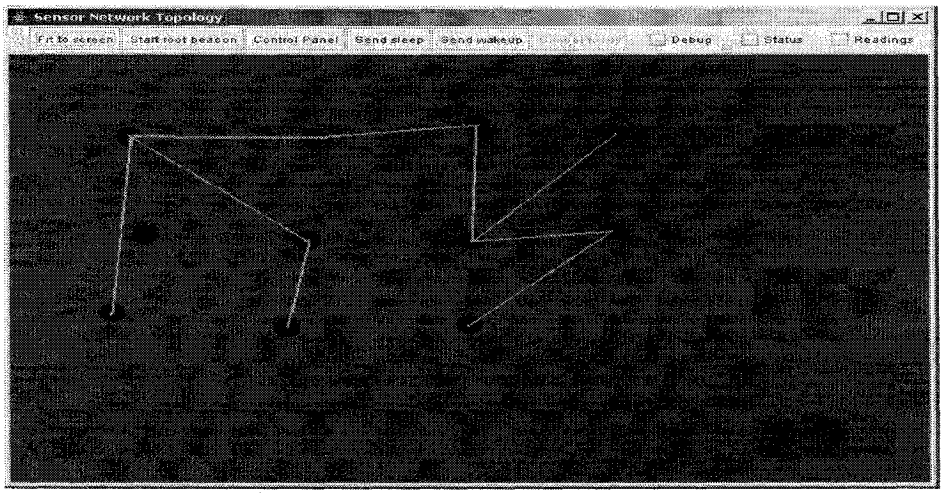


Figure 6.09: Topology after router node 7 is back and node 4 is turned off.

When the base station node 20 is turned on again, some sensor nodes close to it will be connecting to it again, and then the network topology changes to create two anycast trees rooted at both base stations, as before.

6.6 Summary

In this chapter, we tested the DAR routing protocol in real sensor hardware. We tested adding and removing base station and sensor nodes dynamically. The test results show that the DAR protocol can correctly deal with the dynamic situations and worked as we expected. For the same reason that we stated in Section 5.6, complete validation of the correctness of the protocol is difficult and out of the scope of this thesis.

7 Performance Evaluation and Comparison

Many metrics exist for evaluating performance of routing protocols in sensor networks. The commonly seen metrics include the following:

- **Network Lifetime** is normally defined as the time when the first sensor node in a network runs out of battery.
- **Delivery Rate** means the percentage of successful packets delivered. The value is: $\frac{N(b)}{N(s)}$, in which N(b) is the total number of packets successfully received by the base stations and N(s) is the total number of packets sent from the sensor nodes. The higher the quality of routing path constructed by the underlying protocol, the higher we can expect the data delivery rate to be.
- **Hop Count** is the average number of hops in the paths between sensor nodes and their chosen base stations. In a dynamic network, each node's hop count is calculated by weighting hop counts with the corresponding time it stays in that hop count status. Hop count by itself is not actually a performance metric, but it determines another performance metric, that is, data delivery latency.
- **Data Delivery Latency** is the time taken to route sensor data from their sender to the target base stations. This metric is determined by the hop count. Lower hop count value means smaller latency.

Different applications have different emphasis on performance metrics. Often, wireless sensor networks are deployed in remote and hazardous areas. This makes changing

batteries or adding new nodes difficult, if not impossible. Thus, it is very desirable to have a longer network lifetime. The main goal of the DAR protocol is to prolong network lifetime and at the same time maintain a reasonably high data delivery rate.

The data delivery latency is an important performance metric for query-response reactive type networks since user has to wait for the response coming back once a query is issued. On the other hand, for the proactive data gathering networks, the data delivery latency is not as significant because the latency only affects the arriving time of the first packet of any sensor node. After the first packet arrives, the following packets will be arriving at a constant rate no matter what the latency value is. Based on the above reasons, we will be using the delivery rate and network lifetime as the performance metrics.

7.1 Methodology

The goal of the DAR protocol is to prolong network lifetime and at the same time maintain a high data delivery rate. First, optimal values for DAR's parameters α and β are determined. Then, the DAR protocol with the optimal parameters is compared with other similar protocols. Of the other protocols surveyed in Chapter 2, the most suitable candidates for performance comparison with DAR were MT and HC protocols [37, 13]. Their common feature is that they are all beacon-based proactive protocols. On the other hand, HAR [11] is a message exchange-based protocol, requiring exchange of messages to establish parent-child relationships, and ABS [12] has a two-tier architecture in which only a subset of sensor nodes (aggregation nodes) act as routers.. For these reasons, we compare the performance of DAR with MT and HC. The MT protocol is a single base station protocol. Thus, to compare with it, the DAR protocol is set up to run with a single

base station configuration. The platform for the performance comparison is the TOSSIM simulator.

7.2 Measuring network lifetime and data delivery rate

A simple energy consumption model is used to simulate the battery power drain: a counter is increased for every radio activity (transmit or receive a packet). This simple model is good enough because the sensor node operation is event driven and the energy consumed by other components (such as CPU, etc.) is proportional to the power consumed by the radio. The network lifetime here is defined as the first sensor node running out of battery. When the first sensor node power consumption counter reaches a predefined threshold value (in our current implementation, this value is 20,000), the simulation is stopped and the simulation time is the network lifetime used in this performance evaluation.

The data delivery rate is calculated by counting the total number of sensor data packets received by the base station. The count is then divided by the total number of sensor data packets expected.

7.3 Tuning the Parameters α and β

As stated in Section 3.3.1, the formula for the routing cost in DAR is:

$$R_v(P_i) = H(P_i) + \alpha * \left(\beta * L_v(P_i) + (1-\beta) * E(P_i) \right)$$

There are two tuning parameters α and β. Before comparing DAR with other protocols, we will first have to determine optimal values for the tuning parameters. Note that when α is 0, the DAR protocol actually becomes the HC protocol. The following setup is used to evaluate the DAR performance with different values of the tuning parameters :

- 25 (5x5), 49 (7x7), and 100 (10x10) sensor nodes are placed in a grid.
- Distance between neighboring nodes is 12 feet.
- One base station at the corner.
- Radio transmission range is 50 feet.
- Parameter α : 1,2,3, ...
- Parameter β : 0% ...100%.

For each configuration, the simulations were run 50 times to get the average values of performance metrics.

First, we set the parameter $\alpha=1$ and tried the parameter β with the values of {0, 0.25, 0.5, 0.75, 1.0}. We found that there is a significant performance change after the point $\beta=$ 0.75. So, we fine tuned the evaluation points of β to be {0, 0.25, 0.5, 0.75, 0.8, 0.85, 0.9, 0.95 1.0}. At the same time, we also found that setting the increasing step of the parameter α as 1 is not significant enough to change the performance much. So, we set the increasing step to be 2, thus the parameter α is chosen from the set {1, 3, 5, ... }. The best value of β for other values of α were also studied. The experimental results are shown in Appendix 5. In all cases, β=80% was seen to be the best value in term of providing the best balance between network lifetime and delivery rate.

We found that there exist upper bounds for the parameter α and they are related to the network size (see Figure 7.02). Let us define the upper bound of the parameter α for network of size *n* as α'(*n*). Specifically, the upper bounds are 7, 9, and 13 for 5x5, 7x7, and 10x10 networks respectively. After these bounds, changing the parameter α no longer has impact on the network performance. Figures *7.01* and *7.02* summarize the changes of the performance metrics following the two tuning parameters.

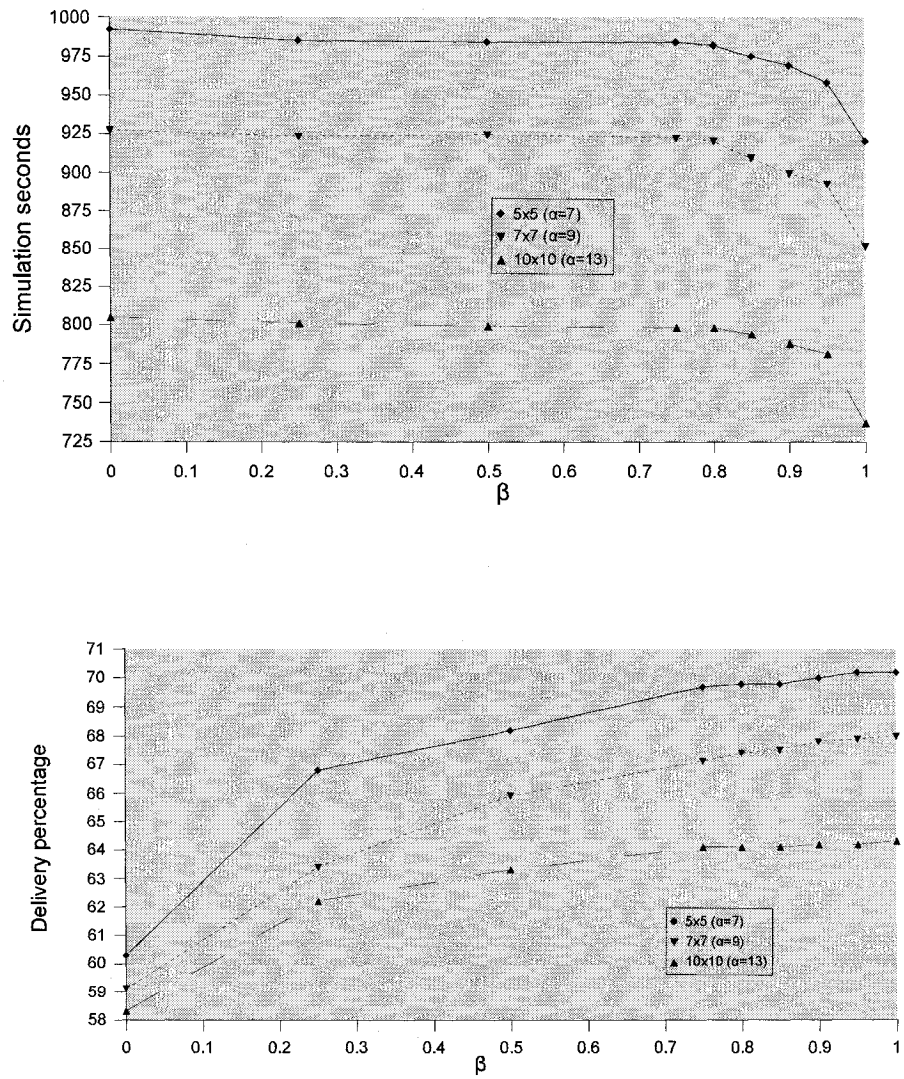
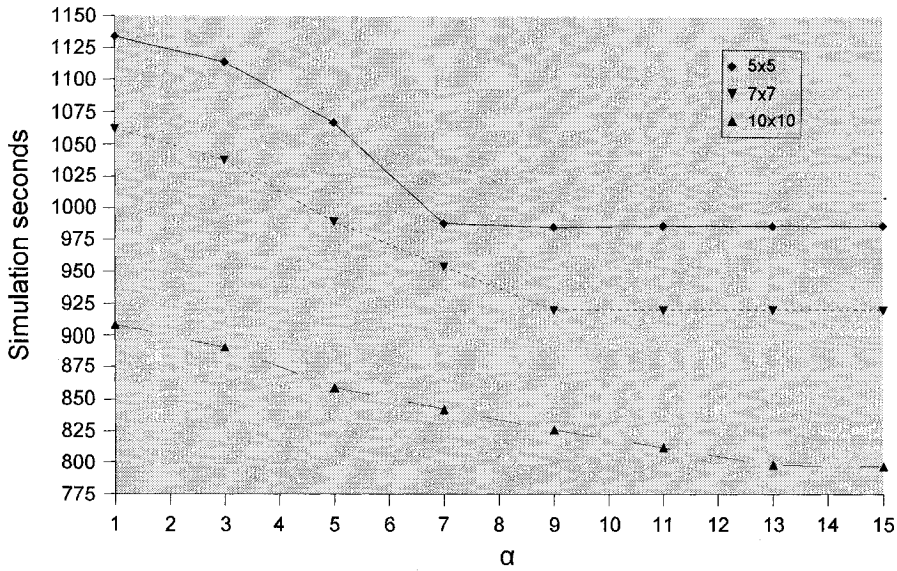


Figure 7.01: Performance of DAR with different β



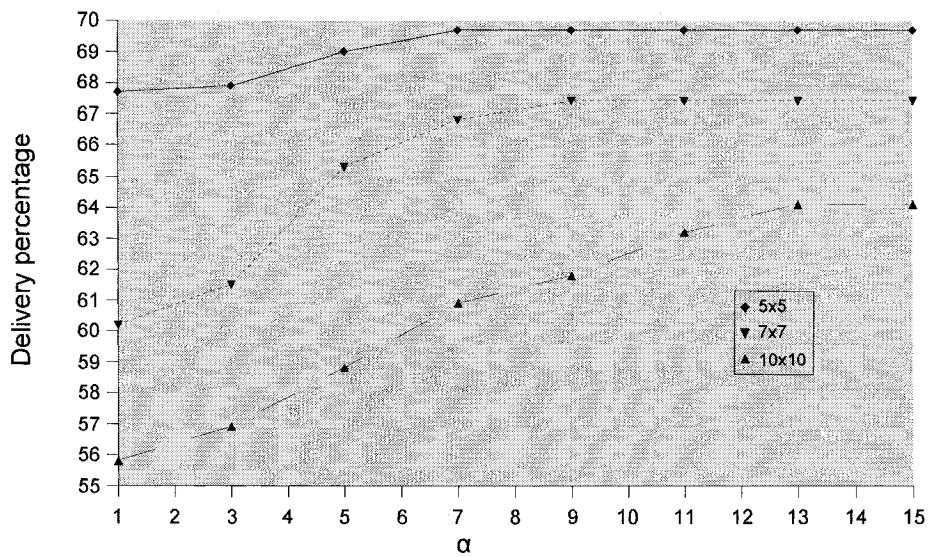


Figure 7.02: Performance of DAR with different α (β=0.8)

From the above charts, we can see that the network lifetime drops dramatically when the parameter β is set to 100% in which case the energy consumption is not considered. The turning point of the parameter β for the network lifetime is in the range of 0.75 to 0.85. The data delivery rate increases when the parameter β is increasing. Considering the two performance metrics, we think that β = 0.8 is the best configuration. On the other hand, increasing the parameter α (up to $\alpha'(n)$) will increase the delivery rate and decrease the network lifetime. The reasons for this are discussed in Section *7.4*. It is clear that choosing the optimal value of the tunning parameters α depends on the specific application. If network lifetime is the primary concern in the application, α should be set to 1. To maintain both the network lifetime and the delivery rate high, the upper bounds of the parameter α can be used.

7.4 Performance Comparison of DAR, HC, and MT

We tested the MT and HC protocols in the same environment. Figures 7.03 and 7.04 summarize the performance comparison of the three protocols. Compared to the MT protocol, the DAR protocol increases the network lifetime significantly. The MT protocol always has the highest data delivery rate at the cost of network lifetime. The HC protocol has a pretty high network lifetime by creating network topologies with low hop counts. The DAR protocol maintains a high data delivery rate, which is very close to that of MT, and it also has a high network lifetime.

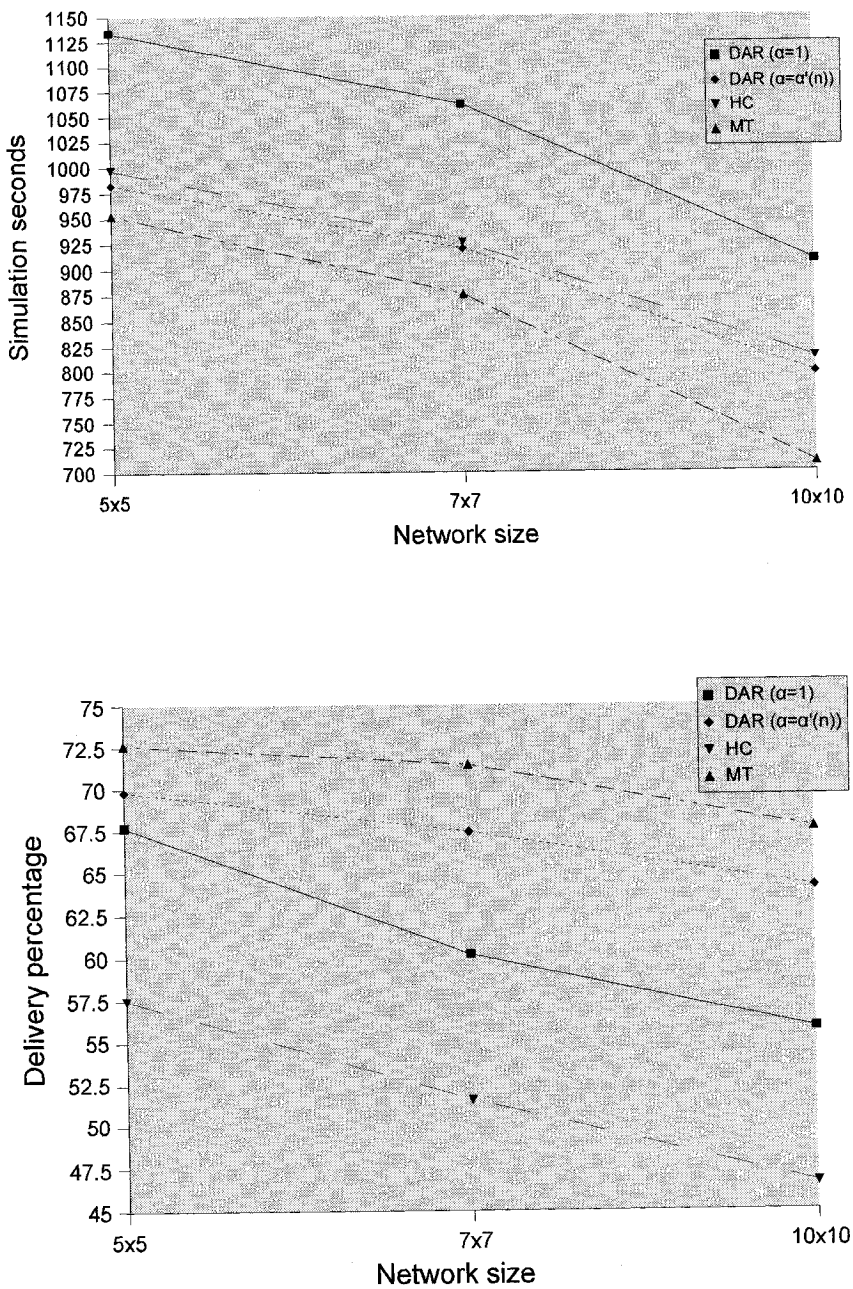


Figure 7.03: Performance comparison of DAR (β=0.8), HC, and MT

Figure 7.04 illustrates average hop counts of the networks created by the three protocols. Note that the figures of HC and DAR(α=1) are almost identical and thus their lines overlapped.

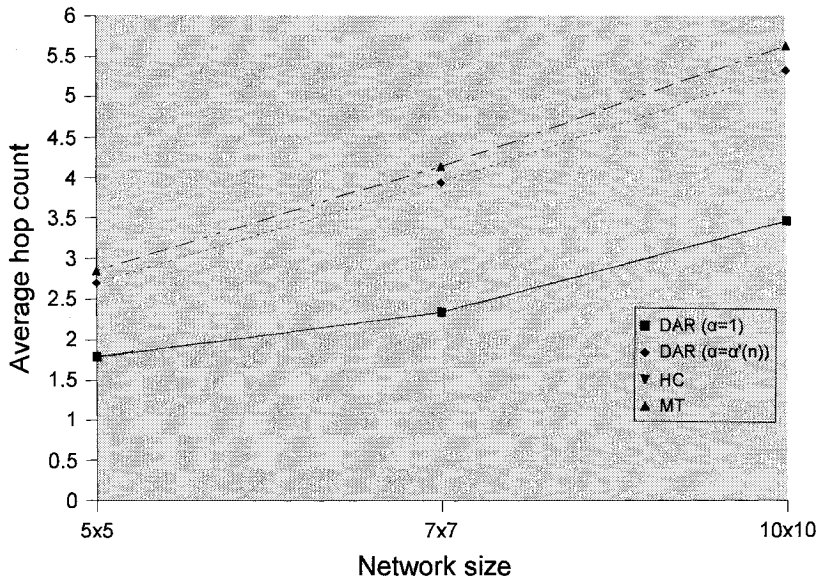


Figure 7.04: Average hop counts of DAR (β=0.8), HC, and MT

7.5 Analysis of the Test Results

From the results of tuning parameters evaluation and performance comparison of the three protocols, we can see that the characteristic of HC protocol is creating topologies with shortest routing paths and the characteristic of MT protocol is creating topologies with highest routing quality. The DAR protocol sits between them and has the advantage of prolonging network lifetime significantly. On average, the hop count difference between HC and MT protocols is about 1 in the 5x5 network and about 2 in the 10x10 network. For the DAR protocol, increasing the tuning parameter α makes the protocol closer to the

MT protocol. The reason that there exist upper bounds for α is because when α is large enough, related to the specific radio model employed in the simulation, all parent candidates with higher hop counts and lower value of loss rate are chosen as parents already. So, increasing α further has no impact any more. As stated in chapter 3, $\alpha > \frac{h}{(\beta * l)}$ is the switching point. For 5x5 size network, when h=1, α=7, and β=0.8, we get l>0.178. That means all parent candidates with a 17.8% higher link quality will be selected as parent nodes even if their hop counts are larger. The same principle also applies to the 7x7 and 10x10 size networks, just the thresholds of α are larger (9 and 13 respectively) since the hop count difference between HC and MT protocols are larger in those bigger networks.

7.6 Impact of Multiple Base Stations

To understand the impact to the network performance by adding more base stations, we run the DAR protocol with 2 (nodes 0 and 65 as bases) and 3 (nodes 0, 37, and 74 as bases) base stations in the 10x10 network. The configuration of parameters are β=0.8 and α=13.

Figure 7.05 summarizes the performance difference when different numbers of base stations are deployed. The results show that adding base stations can greatly improve network performance. The greatest performance increase occurs when the second base station is added to the network. Adding the third base station has less effect than adding the second one.

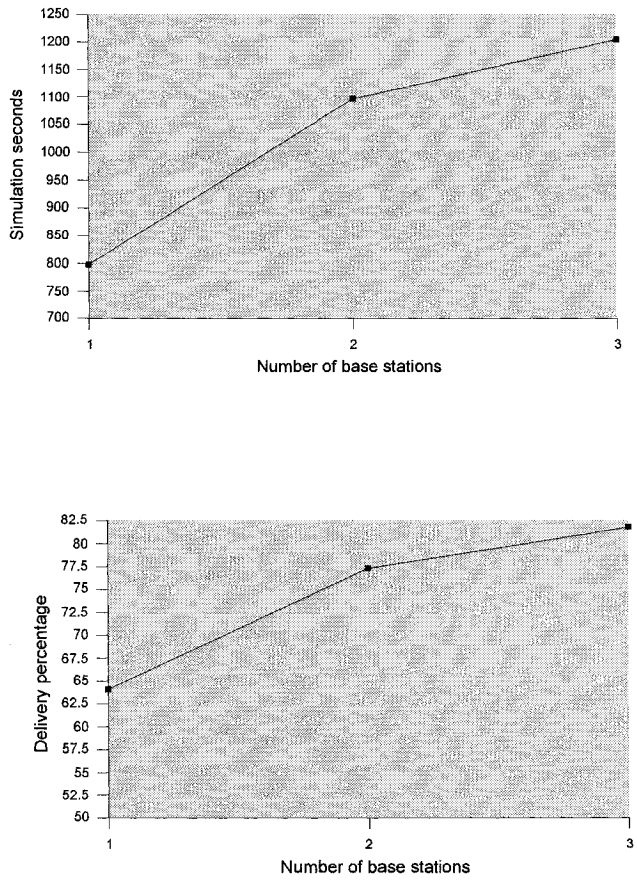


Figure 7.05: Performance of DAR with 1, 2, and 3 base stations

7.7 Summary

We compared the performance between the networks created by the DAR, HC, and MT protocols. The results demonstrate that DAR creates networks with longer lifetime and high delivery rate. Also the impact on network performance of adding base stations is studied. The experiments show that adding base stations is an effective way to improve network performance.

8 Conclusion and Future Work

In this thesis, we surveyed the existing algorithms and protocols available for wireless sensor networks and we proposed a new algorithm to estimate routing cost and a new protocol to construct anycast topology proactively in wireless sensor networks. Performance of the algorithm was extensively evaluated and the result demonstrates that it can prolong network lifetime significantly and at the same time maintain a high data delivery rate. The improvement in these two performance metrics is very desirable in wireless sensor networks. The protocol was also implemented and tested on TelosB motes. As part of our contribution to the wireless sensor network research community, we made all of our work available to other researchers by publishing our code on an open source Subversion hosting server. Thus, other researchers can reuse the code to develop new algorithms and protocols.

Many challenges still remain before wireless sensor networks can be widely deployed in all the possible application areas. Making progress in wireless sensor networks is a surprisingly difficult job. This field involves many research topics including micro-electronics, wireless communication, embedded operating systems, programming languages, network protocols, and system integration. The topic of network protocols is our main interest but not the only one. In this thesis, we studied topology formation and multi-hop routing algorithms and made a contribution toward establishing a simple and yet dynamic protocol suitable for data-gathering type wireless sensor networks. We proposed the DAR protocol and demonstrated that it works on TinyOS based real

hardware. TinyOS is an embedded operating system, tiny in memory consumption. Programming with it involves nesC, Java, and Python programming languages and requires a thorough understanding of areas in both computer science and electrical engineering.

Working with large numbers of sensor nodes is a difficult and labor-intensive work. Programming the motes and then placing them in desired locations is time consuming. A worthy direction of future work is to install the motes in desired locations in the lab permanently and power them from the Tmote Connects gateways' USB ports. Each Tmote Connect device can connect two motes. Then all the gateways can be connected to the department network and programming motes can be done remotely through the gateways without physically reaching the motes. Another two advantages of this installation scheme are: 1) Batteries are no longer needed since motes are powered through USB ports. 2) Motes can be shared more efficiently and easily among different research groups. Furthermore, a complete sensor information system can be set up similar to Motelab [43].

In this research, we did not measure and compare the performance of various protocols on real hardware because of the limitation on the available time. Conducting this kind of measurement and comparison is a worthwhile future work though it is difficult and very time consuming. Only through that sort of field experience can we understand sensor networks more deeply.

Another possible future work is to study the performance impact with different placement of base stations. In our current design, base stations are assigned according to their node identity numbers and there is one base station is placed in the corner. If base stations are placed strategically, it could increase the network performance as well.

At the time of this thesis is writing, a new version of TinyOS, TinyOS 2.0, has just been released. TinyOS 2.0 is a significant step up from TinyOS 1.1, which is the one we used in this thesis. It has a better internal structure and an improved programming interface. It would be interesting to port the DAR protocol to TinyOS 2.0 in the future.

In conclusion, wireless sensor networks are still a new and challenging research area. In the near future, we will see more and more development in both the hardware and software sides to make the wireless sensor network an ubiquitous reality.

Bibliography

[1] TinyOS: http://www.tinyos.net

[2] MoteIV: http://www.moteiv.com

[3] SourceForge: http://www.sf.net

[4] Subverion: http://subversion.tigris.org

[5] Cygwin: http://www.cygwin.com

[6] NS-2: http://www.isi.edu/nsnam/ns/

[7] Chipcon: http://www.chipcon.com

[8] TI Semiconductors: http://www.ti.com

[9] Zigbee: http://www.zigbee.org

[10] Yingqi Xu Lee and W.C. Lee. *Exploring spatial correlation for link quality estimation in wireless sensor networks*. Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM'06), pages 10-19, March 2006.

[11] Niwat Thepvilojanapong. *HAR: Hierarchy-Based Anycast Routing Protocol forWireless Sensor Networks*. Proceedings of the IEEE Symposium on Applications and the Internet (SAINT 05), pages 204-212, 2005.

[12] Y. Thomas Hou. *Optimal Base Station Selection for Anycast Routing in Wireless Sensor Networks.* IEEE Transactions on Vehicular Technology, volume 55, issue 3, pages 813 – 821, May 2006.

[13] Congzhou Zhou. *Localized Topology Generation Mechanisms for Wireless Sensor Networks.* IEEE Global Telecommunications Conference (GLOBECOM '03), volume 3, pages 1269-1273, December 2003.

[14] Satoshi Doi. *Design, Implementation and Evaluation of Routing Protocols for IPv6 Anycast Communication.* Proceedings of the IEEE Nineteenth International Conference on Advanced Information Networking and Applications (AINA'05), pages 833-838, 2005.

[15] J. Elson. *EmStar: An Environment for Developing Wireless Embedded Systems Software.* Technical Report 9, Center for Embedded Networked Sensing, University of California at Los Angeles, March 2003.

[16] Jianliang Zheng. *A Comprehensive Performance Study of IEEE 802.15.4.* Technical Report, Samsung Advanced Institute of Technology, 2004. Available at http://ees2cy.engr.ccny.cuny.edu/zheng/pub/file/wpan_press.pdf

[17] Tiago Camilo. *IPv6 in Wireless Sensor Networks, a New Challenge*. Research report of Centre for Informatics and Systems of the University of Coimbra, Portugal, 2005. Available at http://www.cisuc.uc.pt/view_pub.php?id_p=972

[18] Chenyang Lu. *RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor Networks*. Proceedings of Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, pages 55-66, 2002.

[19] Paolo Santi. *Topology Control in Wireless Ad Hoc and Sensor Networks.* ACM Computing Surveys, volume 37, issue 2, pages 164-194, June 2005.

[20] I.F. Akyildiz. *Wireless sensor networks: a survey*. IEEE Communication Magazine, volume 40, issue 8, pages 102-114, 2002.

[21] Chalermek Intanagonwiwat. *The Sink-based Anycast Routing Protocol for Ad Hoc Wireless Sensor Networks.* Technical Report 95-698, Computer Science Department, University of Southern California, 2004. Available at http://www.usc.edu/dept/cs/technical_reports.html.

[22] Joseph Polastre. *Telos: Enabling Ultra-Low Power Wireless Research*. Proceedings of the Fourth IEEE International Symposium on Information Processing in Sensor Networks (IPSN 2005), pages 364-369, 2005.

[23] Jason Lester Hill. *System Architecture for Wireless Sensor Networks.* Ph.D Thesis, Department of Computer Science, University of California at Berkeley, 2003. Available at http://www.jlhlabs.com/jhill_cs/jhill_thesis.pdf

[24] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. *System architecture directions for networked sensors*. Proceedings of the Ninth ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pages 93-104, 2000.

[25] Philip Levis. *TinyOS and NesC Programming manual.* Available at http://www.tinyos.net

[26] David Curren. *A Survey of Simulation in Sensor Networks.* Research Report, Department of Computer Science, State University of New York at Binghamton, 2004. Available at http://www.cs.binghamton.edu/~kang/teaching/cs580s/david.pdf

[27] Anish Arora and Emre Ertin. *Kansei: A High-Fidelity Sensing Testbed.* IEEE Internet Computing, volume 10, issue 2, pages 35-47. April 2006.

[28] Ian F. Akyildiz. *Wireless sensor and actor networks: research challenges*. Ad Hoc Networks Journal, Elsevier, volume 2, issue 4, pages 351-367, October 2004.

[29] Holger Karl and Andreas Willig. *A short survey of wireless sensor networks.* Technical Report, Telecommunication Networks Group, Technische Universitt Berlin, 2003. Available at http://www.tkn.tu-berlin.de/publications/papers/

[30] Alexander K. *Deterministic boundary recognition and topology extraction for large sensor networks.* Proceedings of the Seventeenth annual ACM-SIAM Symposium on Discrete Algorithm (SODA 06), pages 1000–1009, January 2006.

[31] A. L. Murphy. *An Exercise in Formal Reasoning about Mobile Communications.* Proceedings of the Ninth IEEE International Workshop on Software Specifications and Design, pages 25-33, April 1998.

[32] RFID WiKi: http://en.wikipedia.org/wiki/Radio_Frequency_Identification

[33] IETF 6lowPan work group: http://tools.ietf.org/wg/6lowpan/

[34] The Greatduck Island project: http://www.greatduckisland.net/index.php

[35] Charles Perkins and Elizabeth Royer. *Ad hoc On-Demand Distance Vector Routing.* Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications, pages 90-100, 1999

[36] Sachin J Mujumdar. *Prioritized Geographical Routing in Sensor Networks*. Master thesis, Department of Electrical Engineering, Vanderbilt University, Tennessee, 2004. Available at http://www.isis.vanderbilt.edu/publications/archive

[37] Alec Woo, Terence Tong, and David Culler. *Taming the underlying Challenges of Reliable Multihop Routing in Sensor Networks*. Proceedings of the First International Conference on Embedded Networked Sensor Systems (SenSys'03), pages 14-27, 2003.

[38] Scott Weber. *A Survey of Anycast in IPv6 Networks*. IEEE Communications Magazine, volume 42, issue 1, pages 127-132, January 2004.

[39] Can Basaran and Sebnem Baydere. *Research Integration: Platform Survey. Critical evaluation of platforms commonly used in embedded wisents research*. Technical Report, Embedded WiSeNts consortium, June 2006. Available at http://teachware.distlab.dk/58/

[40] Contiki Operating System: http://www.sics.se/contiki/

[41] Philip Levis. *TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications*. Proceedings of the First International Conference on Embedded Networked Sensor Systems (Sensys03), pages 126–137, 2003.

[42] TinyOS Eclipse plugin: http://www.dcg.ethz.ch/~rschuler/

[43] Motelab at Harvard University: http://motelab.eecs.harvard.edu/

[44] Bor-rong Chen. *Lessons Learned from Implementing Ad-Hoc Multicast Routing in Sensor Networks*. Technical Report TR-22-05, Division of Engineering and Applied Sciences, Harvard University, November 2005. Available at http://www.eecs.harvard.edu/~brchen/papers/tinyadmr-techrept05.pdf

[45] Werner Allen. *Deploying a Wireless Sensor Network on an Active Volcano.* IEEE Internet Computing, volume 10, issue 2, pages 18-25, March 2006.

[46] IEEE802.15.4 Task Group: http://www.ieee802.org/15/pub/TG4.html

[47] G. Tan. *Self-organizing Bluetooth Scatternets*. Master's thesis, Department of Computer Science and Engineering, Massachusetts Institute of Technology, 2002. Available at http://nms.lcs.mit.edu/papers/tan-ms-thesis.pdf

[48] Tython user manual: http://www.tinyos.net/tinyos-1.x/doc/tython/manual.html

[49] Chalermek Intanagonwiwat, Ramesh Govindan and Deborah Estrin. *Directed Diffusion for Wireless Sensor Networking.* Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00), pages 56-67, August 2000.

[50] David Braginsky and Deborah Estrin. *Rumor routing algorthim for sensor networks.* Proceedings of the First ACM international workshop on Wireless Sensor Networks and Applications, pages 22–31, 2002.

[51] Stuart Kurkowski. *MANET Simulation Studies: The Incredibles.* Mobile Computing and Communications Review, volume 9, issue 4, pages 50-61, October 2005.

[52] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. *Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks.* Technical Report CSD-TR 02-0013, University of California at Los Angeles, February 2002.

[53] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. *Large-scale Network Discovery: Design Tradeoffs in Wireless Sensor Systems.* Proceedings of the Symposium on Operating Systems Principles (SOSP 2001), 2001.

[54] Crossbow Technology: http://www.xbow.com

Appendix

This appendix gives detailed step by step instructions of how to set up the TinyOS development environment and then build/run the DAR protocol on the TOSSIM simulator and the TelosB sensor motes. It assumes the host computer is running the Windows XP operating system. Linux host can also be used, but the installation steps are fragmented and the process involves many manual tweaks.

The complete simulation results of evaluating the tuning parameters are also attached in this appendix.

I. Install TinyOS

The version of TinyOS used here is a special version of TinyOS 1.1.15 called Boomerang version 2.0.4 distributed by Moteiv. It bundled TinyOS 1.1.15 with some Moteiv utilities for TelosB motes. It also contains fixes to some installation issues found in the original TinyOS 1.1.15.

Download Boomerang version 2.0.4 from http://www.moteiv.com and install it. This will install everything we need including Cygwin and TinyOS 1.1.15 environment. The installed directories are:

C:\cygwin – Cygwin environment

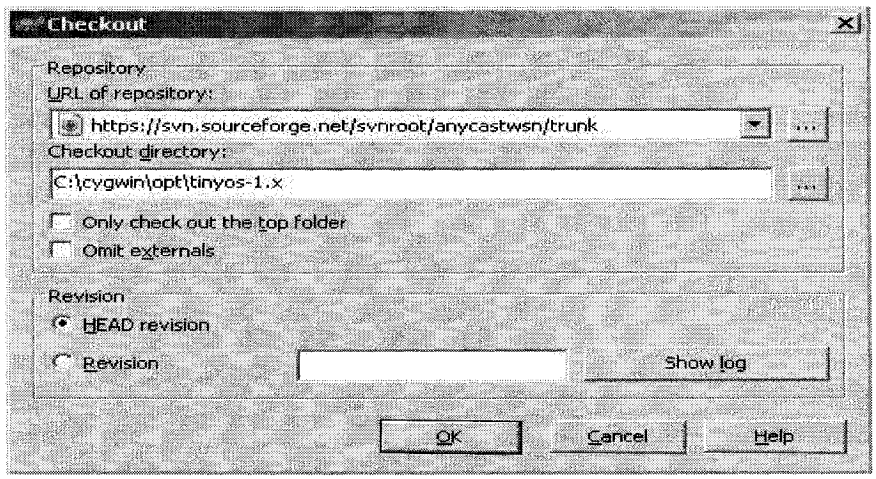
C:\cygwin\opt\msp430 – Utilities for TI MSP430 based TelosB mote

C:\cygwin\opt\tinyos-1.x – TinyOS 1.1.15

II. Install Subversion and retrieve DAR source code

The DAR source code is hosted on SourceForge and version controlled under Subversion. To retrieve the DAR code, a Subversion client is needed. Download the Subversion client TortoiseSVN from *http://tortoisesvn.tigris.org/* and install it.

After TortoiseSVN is installed, right click the folder *C:\cygwin\opt\tinyos-1.x* and select "SVN Checkout". Then, fill out the "URL of repository" field with "https*://svn.sourceforge.net/svnroot/anycastwsn/trunk*" as illustrated by the following diagram:



After the Checkout, the following directories/files should have been created:

C:\cygwin\opt\tinyos-1.x\tos\lib\Dar – DAR protocol source code, it became part of TinyOS's library

- *DarMultiHop.h* : Header file of DAR protocol

- *DarRouter.nc* : DAR protocol configuration
- *DarSelectM.nc* : DAR protocol module internal implementation
- *MultiHopEngineM.nc* : DAR protocol's common interface implementation

C:\cygwin\opt\tinyos-1.x\tos\interfaces – TinyOS's interface files are all in this directory

- *DarRouteSelect.nc* : Interface definition for DAR protocol

C:\cygwin\opt\tinyos-1.x\apps\DarApp – A demo application using DAR protocol to send messages

- *DarApp.h* : Message header definition
- *DarAppCmd.h* : Command message header definition
- *DarApp.nc* : Application configuration
- *DarAppM.nc* : Application module implementation
- *Makefile* : Makefile for building the application
- *Lossy7722* : Lossy radio signal model for 7x7 nodes with distance 22 feet. This file is created by using TinyOS tool "*LossyBuilder*". The command line is "*java net.tinyos.sim.LossyBuilder –d 7 7 –s 22 -o lossy7722*".

C:\cygwin\opt\tinyos-1.x\tools\java\net\tinyos\sim\plugins – TOSSIM simulator plug-in directory

- *DarTopoPlugin.java*: A Java plug-in to display network topology created by DAR protocol

The Java plug-in for DAR has to be compiled and added to the original TOSSIM plug-in program. To do that, add a line "plugins/DarTopoPlugin.class" to the file

C:\cygwin\opt\tinyos-1.x\tools\java\net\tinyos\sim\plugins\plugins.list and enter the command "*make*" in the *sim* directory.

III. DAR demo application in TOSSIM

To run the DAR demo application in TOSSIM simulator, follow the following steps:

a. Build the DAR demo application

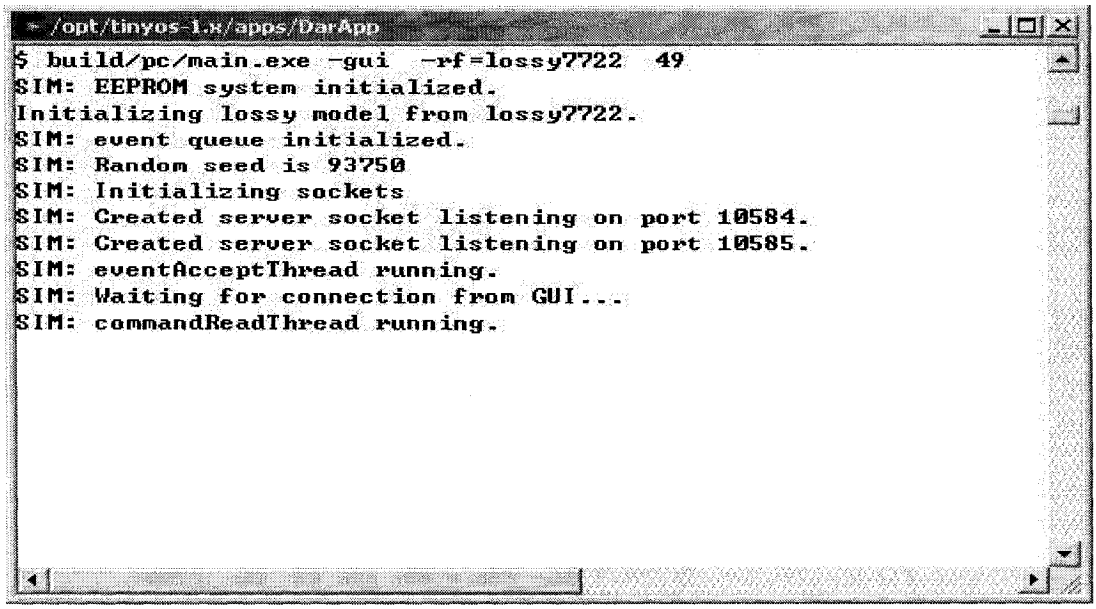
- Open a Cygwin window
- Cd to */opt/tinyos-1.x/apps/DarApp*
- Type "*make pc*"

This will build the application for the "*pc*" target, which can run in TOSSIM simulator. The generated executable file is "*main.exe*" and it is placed in a newly created directory "*build\pc*".

b. Run the DAR demo application

- Open a Cygwin window
- Cd to */opt/tinyos-1.x/apps/DarApp*
- Type "*export DBG=sim*". This will set up the demo application to only print out debugging information related to simulation. Otherwise, the debugging messages may be too much to be read easily.
- Type "*build/pc/main.exe –gui –rf=lossy7722 49*" to run the application using the radio model specified in the file "*lossy7722*" with total of 49 motes.

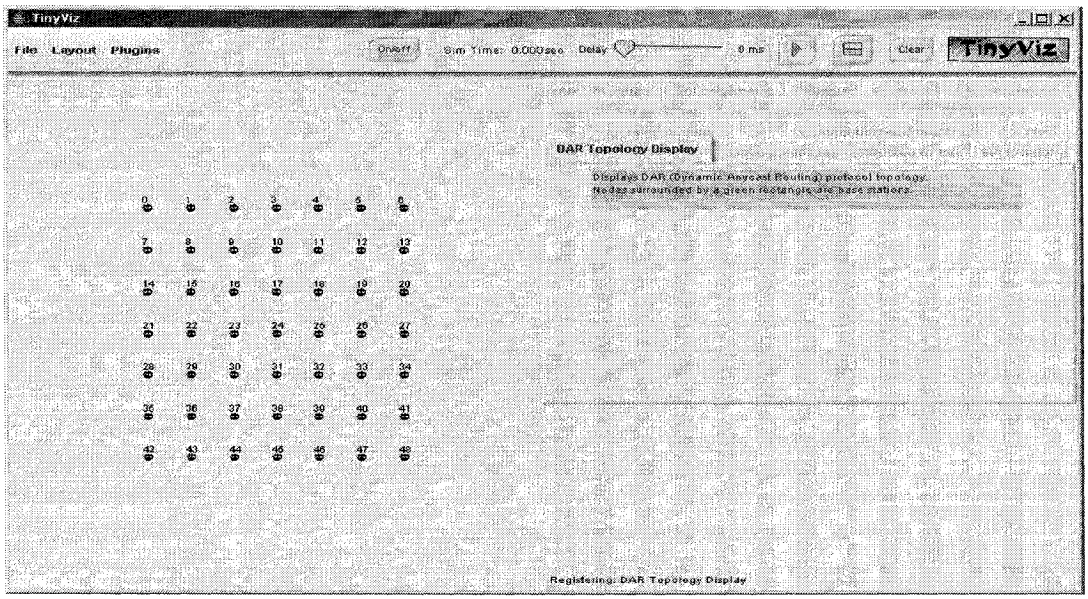
Now, the application will try to connect to TOSSIM GUI simulator as illustrated in the following diagram:



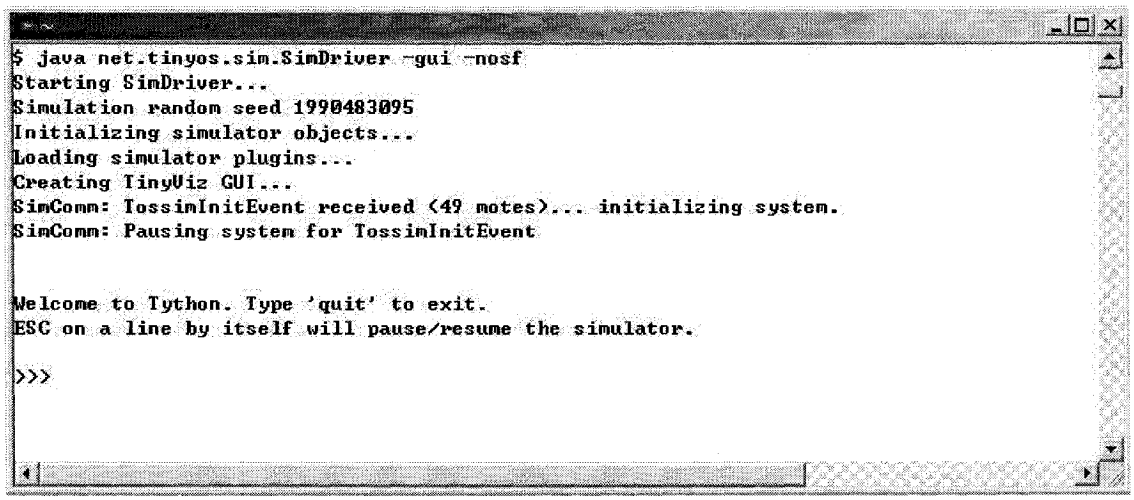
Since we have not started TOSSIM yet, it will wait there until we do the following to start TOSSIM GUI:

- Open a new Cygwin window
- Type "*java net.tinyos.sim.SimDriver -gui -nosf*". This will start the TOSSIM with a graphic interface called TinyViz. The "*-nosf*" option tells TOSSIM that there is no Serial Forwarder program running so that it must only communicate with local application, which is our DAR demo application.

Now the TOSSIM is launched, we can click the green start button to start the simulation.

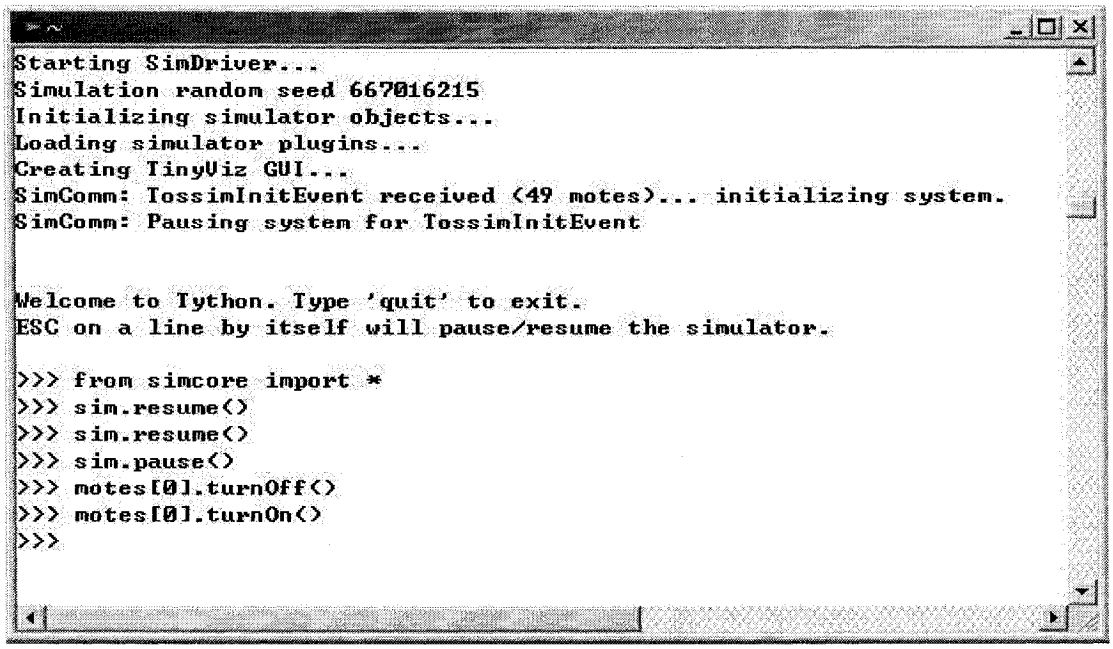


After TOSSIM is started, the Cygwin window used to launch TOSSIM will provide a Tython (TinyOS Python) interface to accept interactive Python scripts from the user.



From this Tython interface, user can stop/resume the simulation, turn on/off nodes. The turn on/off nodes feature is used to simulate motes joining/leaving the wireless sensor network dynamically in real life situation.

- *>>> from simcore import ** – To control TOSSIM execution from Tython, the "*simcore*" module has to be imported to Tython
- *>>> sim.resume()* – Resume the TOSSIM simulation process
- *>>> sim.pause()* – Pause the TOSSIM simulation process
- *>>> motes[0].turnOff()* – Turn off mote 0
- *>>> motes[0].turnOn()* – Turn on mote 0



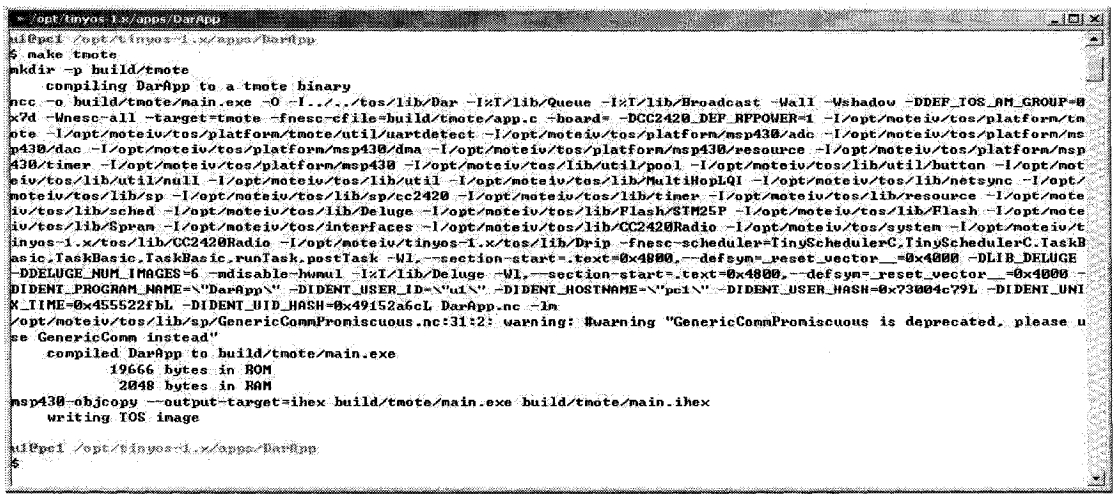
IV. DAR demo application on motes hardware

Running the application on the real hardware motes involves many more steps compared to running it in a simulator. This section outlines the detailed steps.

a. Build the DAR application for motes

- Open a Cygwin window
- Cd to */opt/tinyos-1.x/apps/DarApp*

- Type "*make tmote*"

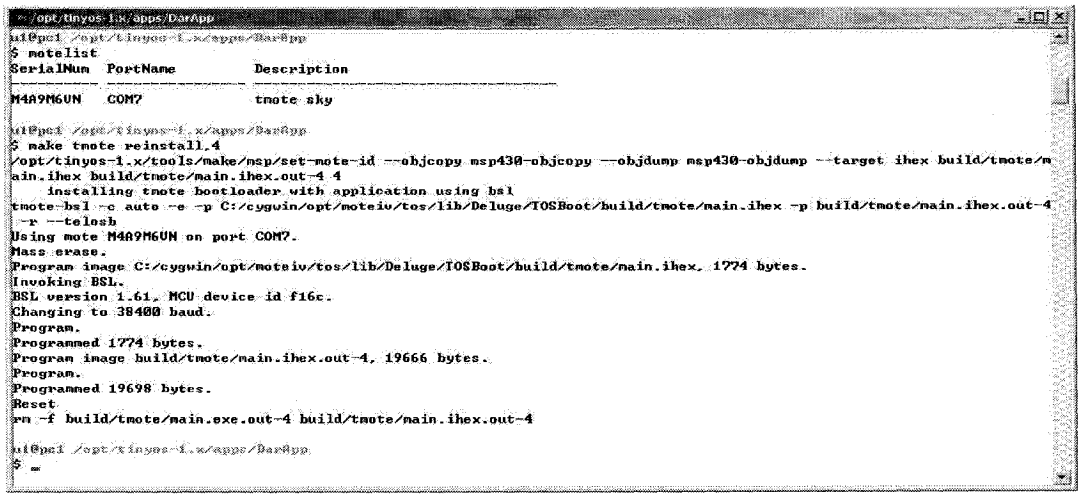


This will build the application for the tmote sky (TelosB) target, which can run in Tmote Sky and TelosB motes. The generated target binary image file is "main.ihex" and it is placed in a new created directory "build\tmote\".

b. Load flash image to motes

Connect a Tmote Sky mote to a free USB port on the computer, the mote will be treated by Windows Virtual Com Port driver as a serial port device (the COM port number is automatically assigned). To verify that the mote is recognized by the computer, enter the command "motelist". The mote's serial number should be printed out.

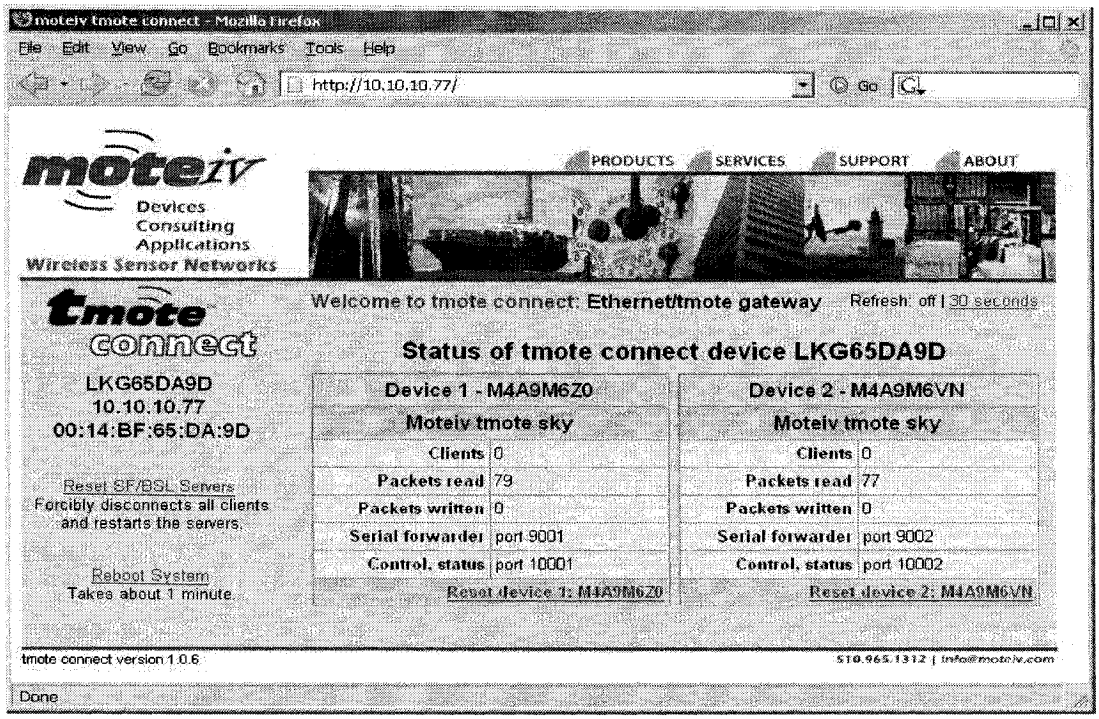
Then, enter the command "make tmote reinstall, x" (where x is the numeric network address to be assigned to the mote) to program the mote. The Bootstrap Loader program that resides in the mote will start to program the on-chip flash upon receiving the above command. In the following example, the attached mote is assigned a network address 4.



Repeat the above step to program all motes with different network addresses. Currently, if a mote's address is a multiple of 20, the DAR protocol treats it as a base station. So, in the following example of trying out the DAR protocol with two base stations, we will program a mote with address 0 to act as a base station, a mote with address 20 to act as the second base station, and a few motes with address from 1 to 10 as normal sensor nodes.

c. Set up Tmote Connect gateway for base stations

Tmote Connect gateway can be used in both DHCP-enabled network and Non-DHCP network. In this example setup, the Tmote Connect gateway is assigned to a fixed IP address 10.10.10.77 by DHCP server. Two motes acting as base stations are connected to the two USB ports of the Tmote Connect gateway. Open a web browser to the gateway to display the gateway and the base station motes's status. The two base stations' data will be available on the gateway's TCP port 9001 and 9002 respectively.



d. Run the DAR demo application on motes

Turn on the sensor motes and place them randomly with a distance about 80cm between each other.

e. Build and run the SurgeTelos Java tool

A java tool called SurgeTelos in TinyOS is used to display the topology of the motes on a PC. The following steps will recompile the tool from the source for Tmote Sky motes:

- Open a Cygwin window
- Cd to */opt/tinyos-1.x/tools/java/net/tinyos/surge/*
- Enter "*make clean*"
- Enter "*SURGE_PLATFORM=telos make*"

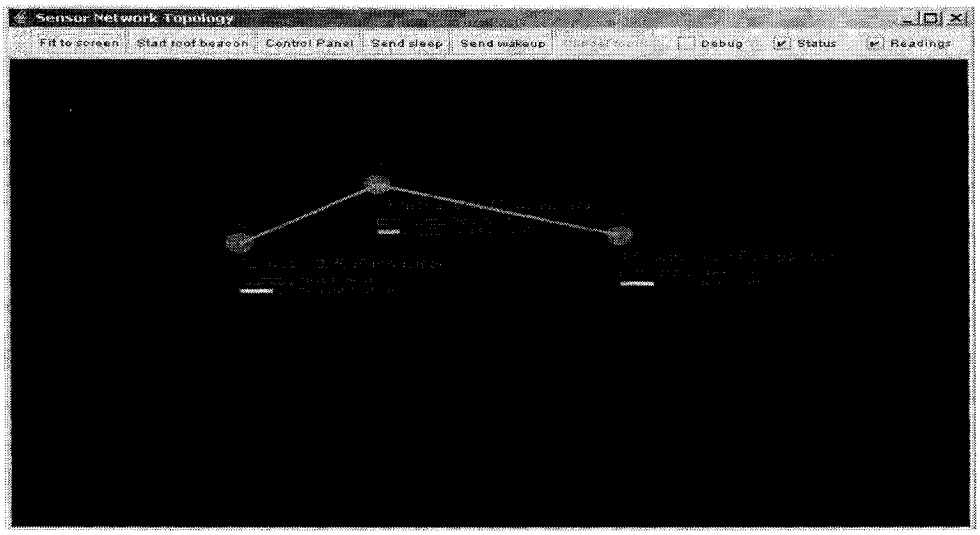
After the Java tool is ready, create two instances of it by running it in two different Cygwin windows to display network topology:

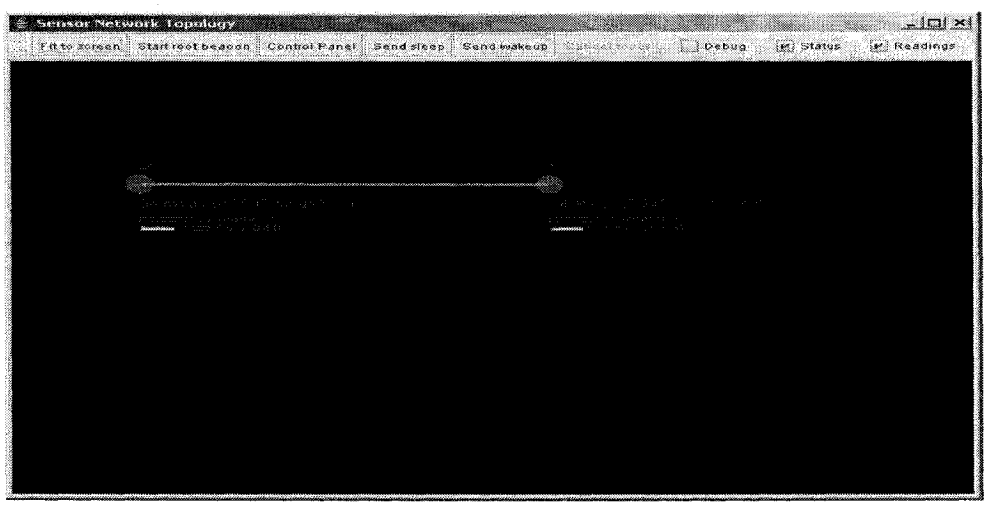
- Open a Cygwin window
- Enter "*MOTECOM=sf@10.10.10.77:9001 java net.tinyos.surge.MainClass 0x7d*"
- Open another Cygwin window
- Enter "*MOTECOM=sf@10.10.10.77:9002 java net.tinyos.surge.MainClass 0x7d*"

Then, the Cygwin windows should look like:

```
/opt/tinyos-1.x/apps/DarApp
$ MOTECOM=sf@10.10.10.77:9002 java net.tinyos.surge.MainClass 0x7d
Using AM group ID 125 (0x7d)
Starting mote listener...
Creating mainFrame...
Starting DisplayManager thread...
Creating ObjectMaintainer...
Creating LocationAnalyzer...
Creating SensorAnalyzer...
Making MainFrame visible...
Decay thread running...
Ready.
```

And, the topology windows are like those:





Over time, the topology will change accordingly when new motes are discovered or when old motes lose connections to the network.

V. Results of Performance Evaluation

This section gives the complete performance evaluation results of the DAR protocol's tuning parameters α and β in network size of 5x5, 7x7, and 10x10. The value *network lifetime* is in simulation seconds. The value of *delivery rate* is in percentage.

α=1:

Network lifetime			
β	5x5	7x7	10x10
0	1135	1064	913
0.25	1135	1064	912
0.5	1134	1063	913
0.75	1136	1063	910
0.8	1134	1062	908
0.85	1125	1053	902
0.9	1103	1028	887
0.95	1072	977	868
1	999	926	814

Delivery rate			
β	5x5	7x7	10x10
0	57.5	51.6	46.7
0.25	64.3	57.4	51.6
0.5	65	59.6	54.1
0.75	67.5	60.1	55.5
0.8	67.7	60.2	55.8
0.85	67.9	60.4	55.9
0.9	68	60.6	56.1
0.95	68.2	60.7	56.2
1	68.2	60.9	56.3

α=3:

Network lifetime			
β	5x5	7x7	10x10
0	1122	1049	898
0.25	1118	1039	893
0.5	1116	1040	892
0.75	1114	1038	891
0.8	1114	1038	891
0.85	1110	1038	890
0.9	1094	1019	880
0.95	1073	1002	849
1	1033	955	787

Delivery rate			
β	5x5	7x7	10x10
0	58.1	54.5	49.2
0.25	64.8	59.5	53.8
0.5	67.1	60.5	55.9
0.75	67.9	61.3	56.8
0.8	67.9	61.5	56.9
0.85	68	61.5	57
0.9	68.1	61.7	57.1
0.95	68.2	61.8	57.2
1	68.3	62.1	57.3

α=5:

Network lifetime			
β	5x5	7x7	10x10
0	1075	1003	867
0.25	1070	998	862
0.5	1069	997	861
0.75	1068	997	859
0.8	1067	989	859
0.85	1055	984	853
0.9	1035	974	837
0.95	1008	948	822
1	968	905	776

Delivery rate			
β	5x5	7x7	10x10
0	59.2	56.8	52.1
0.25	65.1	61.1	56.1
0.5	67.8	63.6	57.9
0.75	69	65	58.7
0.8	69	65.3	58.8
0.85	69.1	65.3	58.9
0.9	69.2	65.6	59.1
0.95	69.4	65.6	59.2
1	69.4	65.9	59.2

α=7:

Network lifetime			
β	5x5	7x7	10x10
0	992	961	848
0.25	985	956	843
0.5	984	956	843
0.75	984	954	842
0.8	982	954	842
0.85	975	950	839
0.9	969	925	827
0.95	958	904	815
1	920	861	767

Delivery rate			
β	5x5	7x7	10x10
0	60.3	57.3	54.1
0.25	66.8	62.6	58.1
0.5	68.2	65.9	59.1
0.75	69.7	66.5	60.5
0.8	69.8	66.8	60.9
0.85	69.8	66.9	60.9
0.9	70	67.1	61.1
0.95	70.2	67.1	61.1
1	70.2	67.2	61.1

α=9:

Network lifetime			
β	5x5	7x7	10x10
0	992	927	831
0.25	985	923	828
0.5	984	924	828
0.75	982	922	826
0.8	982	920	826
0.85	975	909	825
0.9	969	899	812
0.95	958	892	795
1	920	851	756

Delivery rate			
β	5x5	7x7	10x10
0	60.3	59.1	55.1
0.25	66.8	63.4	58.2
0.5	68.2	65.9	60.4
0.75	69.7	67.1	61.7
0.8	69.8	67.4	61.8
0.85	69.8	67.5	61.8
0.9	70	67.8	61.9
0.95	70.2	67.9	62.1
1	70.2	68	62.1

α=11:

Network lifetime			
β	5x5	7x7	10x10
0	992	927	820
0.25	985	923	814
0.5	984	924	814
0.75	982	922	812
0.8	982	920	812
0.85	975	909	812
0.9	969	899	808
0.95	958	892	801
1	920	851	752

Delivery rate			
β	5x5	7x7	10x10
0	60.3	59.1	56.5
0.25	66.8	63.4	60.2
0.5	68.6	65.9	61.7
0.75	69.7	67.1	63
0.8	69.8	67.4	63.2
0.85	69.8	67.5	63.3
0.9	70	67.8	63.3
0.95	70.2	67.9	63.3
1	70.2	68	63.4

α=13:

Network lifetime			
β	5x5	7x7	10x10
0	992	927	805
0.25	985	923	801
0.5	984	924	799
0.75	982	922	798
0.8	982	920	798
0.85	975	909	794
0.9	969	899	788
0.95	958	892	781
1	920	851	737

Delivery rate			
β	5x5	7x7	10x10
0	60.3	59.1	805
0.25	66.8	63.4	801
0.5	68.6	65.9	799
0.75	69.7	67.1	798
0.8	69.8	67.4	798
0.85	69.8	67.5	794
0.9	70	67.8	788
0.95	70.2	67.9	781
1	70.2	68	737

α=15:

Network lifetime			
β	5x5	7x7	10x10
0	992	927	805
0.25	985	923	801
0.5	984	924	799
0.75	982	922	798
0.8	982	920	798
0.85	975	909	794
0.9	969	899	788
0.95	958	892	781
1	920	851	737

Delivery rate			
β	5x5	7x7	10x10
0	60.3	59.1	805
0.25	66.8	63.4	801
0.5	68.6	65.9	799
0.75	69.7	67.1	798
0.8	69.8	67.4	798
0.85	69.8	67.5	794
0.9	70	67.8	788
0.95	70.2	67.9	781
1	70.2	68	737