PRIORITIZED GEOGRAPHICAL ROUTING IN SENSOR NETWORKS

By

Sachin J Mujumdar

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

May, 2004

Nashville, Tennessee

Approved: Date:

_____ _____

_____ _____

To my loved ones,

My Parents, my Sister and my Wife

ACKNOWLEDGEMENTS

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA), under the Networked Embedded Software Technology (NEST) program.

I would like to thank my graduate advisor, Dr. Gabor Karsai, for helping me focus on my research, and for answering an endless stream of questions. I am also very grateful to Dr. Akos Ledeczi, my research advisor, who motivated me to implement this algorithm. His constant pushing and words of encouragement when I was flagging have gone a long way in making this thesis a reality.

Dr. Mikos Maroti, Dr. Gyula Simon, Peter Volgyesi and other members of the NEST project at ISIS have also been very helpful in answering my doubts. Thank you for always being willing to spare your time to resolve my queries and look for silly bugs in my code.

It would be unthinkable of me not thank my parents for always encouraging me to pursue my beliefs, my ideas and my dreams. Their unquestioning support lent me the courage to pursue this difficult journey and focus on my goals. My sister, Siddhi, with her faithful and loyal support has been instrumental in my endeavors. I would also like to thank my parents-in-law for their blessings. Lastly, but not in the least, I wish to thank my wife and sweetheart, Sujata, for believing in me and motivating me. She provided endless hours of company and cups of coffee, and was remarkable in putting up with my tantrums. Her threats were, in no small measure, responsible for me overcoming my occasional bouts of lethargy.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF ABBREVIATIONS

AGR	Advanced Geographic Routing
AODV	Ad-hoc On-demand Distance Vector
ARPANET	Advanced Research Projects Agency NETwork
BGP	Border Gateway Protocol
BGR	Basic Geographic Routing
CDMA	Code Division Multiple Access
CSMA	Carrier Sense Multiple Access
DFRF	Directed Flood Routing Framework
DSR	Dynamic Source Routing
FS	Forwarding Set
GG	Gabriel Graph
GPSR	Greedy Perimeter Stateless Routing
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LAR	Location Aided Routing
LEACH	Low Energy Adaptive Clustering Hierarchy
LMP	Last Mile Processing
MAC	Medium Access Control
MANET	Mobile Ad-hoc NETwork
MIT	Massachusetts Institute of Technology
MGR	Modified Geographic Routing

NFL	Neighborhood Feedback Loop
NS	Neighbor Set
OSPF	Open Shortest Path First
PDA	Personal Digital Assistant
PGR	Prioritized Geographic Routing
RIP	Routing Information Protocol
RNG	Relative Neighborhood Graph
RRC	Relay Ratio Controller
SNGF	Stateless Non-deterministic Geographic Forwarding
TDMA	Time Division Multiple Access
TinyOS	Tiny Operating System
TOSSIM	TinyOS Simulator
VLSI	Very Large Scale Integration
WSN	Wireless Sensor Networks

CHAPTER I

INTRODUCTION

> *"The number of transistors found on a single integrated chip has doubled over the past four years. The amount of transistors that may be fitted in a single chip, will double every two years"*
>
> *-- Gordon Moore, 1965*

Moore's Law has resulted in the rise of computing devices that are extremely small, powerful and energy-efficient. Particularly, this has resulted in the market being inundated by cheap, compact cell-phones, Personal Digital Assistants (PDAs) and Pocket PCs. The major thrust of these devices is towards integrating communications and processing power on a single device. With the advances in VLSI technology, entire systems are now being fitted on small devices. An important consequence has been the research in the development of devices that can not only communicate amongst themselves or process data, but also sense their surroundings and actuate them. A new set of computing devices that are energy-aware, tiny and which possess communicating, actuating, sensing and processing capabilities are being developed.

With rapid developments such tiny, self-sufficient nodes, Embedded Systems have made a huge leap in the field of distributed computing. Complex networks, consisting of hundreds thousands and potentially millions of cheap computing devices, will eventually be deployed to monitor, sense and actuate their environment. Typically, such devices have a specific functionality and are incapable of doing anything else. They communicate using the radio medium, and hence are also called *Wireless* Devices.

Since the main functionality of these devices is to sense, these devices are called Wireless Sensor Nodes, and their networks are referred to as Wireless Sensor Networks (WSN). Each node communicates with the other nodes and / or to one or more specific powerful nodes (called *base station*) using the radio communication. It is often not possible to transmit a message directly (in a single hop) to a node that is far away. Transmitting a message from one node to another is known as *routing*. Routing a message to its correct recipient is a complex task involving various nodes (*multi-hop*).

Ad-hoc Networks

"*Ad hoc*" is a Latin word meaning unplanned, makeshift or temporary. Though connoting negativity, *ad hoc*-ism in networks is used to describe a large-class of easy-to-deploy networking systems with dynamic topologies.

Murphy et al. [1] define an "Ad hoc Network" to be "transitory associations of mobile nodes which do not depend upon any fixed support infrastructure…can be visualized as a constantly changing graph. Connection and disconnection are controlled by the distance among nodes, and their willingness to collaborate in the formation of cohesive, albeit transitory community". Ad hoc networks are not dependent on any conventional supporting infrastructure such as continuous connectivity, unlimited/large bandwidth, static configuration and topology, and reliable power supply; they are formed by the mere presence of nodes. Communication between nodes is dependent on the distance between them. Such communication may either be direct communication, or communication based on relaying of messages by willing intermediate nodes.

The *ad hoc* nature of the network is primarily due to the dynamism of the constituent nodes. A fresh arrangement of the nodes in the network is formed, simply by

adding/removing a node to/from the network. Furthermore, a mere change in location of one or more of the constituent nodes results in probabilistic connections and a rearrangement of the topology. Connectivity between two nodes is continuously changing due to constant movement of nodes and radio characteristics like noise and interference. The temporal connectivity graph in an ad-hoc network, i.e. the links therein, may be expressed in terms of probability of the presence of the above factors. Hence the connections are probabilistic in nature. It may be noted here that the term “fresh arrangement” denotes a network with a different topology or connectivity graph. New nodes appear in the network either due to movement or their willingness to participate in the network. Similarly, nodes disappear due to movement out of the area, or due to failure. The willingness of nodes to collaborate or participate in the network is a decision made by a network technology-dependent implementation at the network level [4].

A necessary characteristic of *ad hoc* networks is their ability to self-organize. The nodes need to be self-aware such that a random deployment of nodes will eventually lead to an *ad hoc* network being formed. The nodes have no information about the network initially. Each new node will start by recognizing its *neighbors* – nodes that are willing to communicate with the node in question. However, it may be noted that nodes need not know their neighbors to effectively route messages in the network. The adaptivity of such self-organizing networks is covered in further detail in [3]. The network should be capable of allowing a new node to join it, or allow current nodes to leave. Moreover, this should be done in a decentralized manner, in the absence of any controller or administrator.

The *ad hoc* network space may be classified based on the mobility of its constituent nodes [2]. *Ad hoc* networks consisting of nodes that are able to move are termed as *Mobile Ad hoc NETworks* (MANET). Such networks usually consist of PDAs, laptops, cellular phones and other hand-held devices. Due to the ability of the constituent nodes to move around in and out of the network, the topology changes constantly. Protocols and applications based on a MANET need to be designed keeping in view this changing topology.

Ad hoc networks consisting of nodes that are spread out over a geographical area and are immobile are classified as *Smart Sensor Networks* [54] [55] [2]. Such networks are usually application-specific, configured to perform a specific task, and consist of resource-constrained nodes fitted with sensors and/or actuators.

Wireless Sensor Networks

A wireless *ad hoc sensor* network, or WSN, is a collection of autonomous, self-sufficient nodes spread over a geographic area that communicate with each other over a radio network [2]. A WSN usually is a dense deployment of these nodes. The nodes are typically fitted with sensors to sense the environment. They can also be fitted with actuators that perform some mechanical action based on the inputs received. These nodes are usually distributed over an area in an ad hoc manner. The radio network thus formed is a graph, connecting the various nodes in an impromptu and decentralized manner. An example of a mechanical action is the use of feedback control to reduce the turbulence caused by the airflow across surface of the wing of an aircraft.

Such complex networks mentioned above have commercial as well as military significance. The typical applications may be as varied as:

- *Pursuer-Evader, Target Tracking:* This involves a mobile target that is being tracked by a number of sensor nodes deployed in the area of the target's movements [8].
- *Habitat Monitoring*: Sensitive environmental zones, biospheres and wildlife habitats may be monitored and studied non-intrusively and non-disruptively by using WSN. Nodes are dispersed in a particular environment with various sensors like light, temperature etc. Data collected from various nodes is used to monitor the particular environment and detect any changes. Applications may be developed to monitor, predict and minimize damage from any changes to the normal system. Habitat Monitoring using sensor networks was first suggested in [12]. The University of California at Berkeley's initiative non-intrusively monitors the microclimates at Great Duck Island, Maine, which is the habitat of Leach's Storm Petrel [9]. University of Hawaii's PODS project [13] uses WSN to investigate the growth patterns of a certain species of plants is another application in Habitat Monitoring.
- *Forecasting:* WSN may be used to monitor the environment, structures etc. and predict trends related to weather, pollution, floods, earthquakes, bush-fires or structural damages to buildings [14].
- *Kindergarten:* An interaction-based instruction method to educate children in kindergarten has been proposed [15], which makes use of WSN. It is envisioned that this process will replace traditional stimulus-response based methods.
- *Smart Home/Office:* WSN along-with actuators can be used to constantly monitor each individual's preferences for humidity, temperature and other environmental conditions. Homes and Offices can also be equipped with WSN-based Burglar Alarm systems that sense movements in prohibited places / times and take appropriate

action. MIT has developed a Smart-Room project [10] to adapt computers to better understand human intents and feelings [56].

- *Aircraft Industry:* Aircraft interior noise is the result of the engine as well as the turbulent boundary layer. The turbulent boundary layer noise enters the cabin through the fuselage [16]. Such noise could be reduced by using a WSN deployed over the surface of the fuselage. The sensors would sense the noise being developed, and with the aid of an actuator would produce vibrations to cancel out the noise.
- *Shooter Localization:* Various sensor nodes are placed in an *ad hoc* manner over a given area. When a shot is fired, the nodes sense the *shockwave* and *muzzle-blast* generated by the shot. Nodes that sense this data send it back to the base-station. Calculations are performed on data collected from various nodes to calculate the origin of the shot. Counter-Sniper, Battleground Monitoring Systems and Urban-Warfare are archetypal examples for application of such a system. Vanderbilt University has developed a Shooter-Localization system for detecting enemy snipers [11].

It may, however, be noted that Sensor Networks, though a subclass of *ad hoc* networks, differ from them in certain ways. Akayildiz et al [5] give a list of such differences:

- The number of nodes in a sensor network may be much higher than in a typical *ad hoc* network,
- Sensor networks have a dense topology,
- The frequency of node failures in a sensor network is much higher,

- The topology of a sensor network changes frequently due to link and node failures, and to lesser extent due to re-deployment,
- Sensor nodes are resource constrained in terms of memory, processing power and energy,
- Sensor Networks are designed with an end-goal in mind, like gathering data based on particular events, while a simple *ad hoc* network's primary goal is just communication [2].

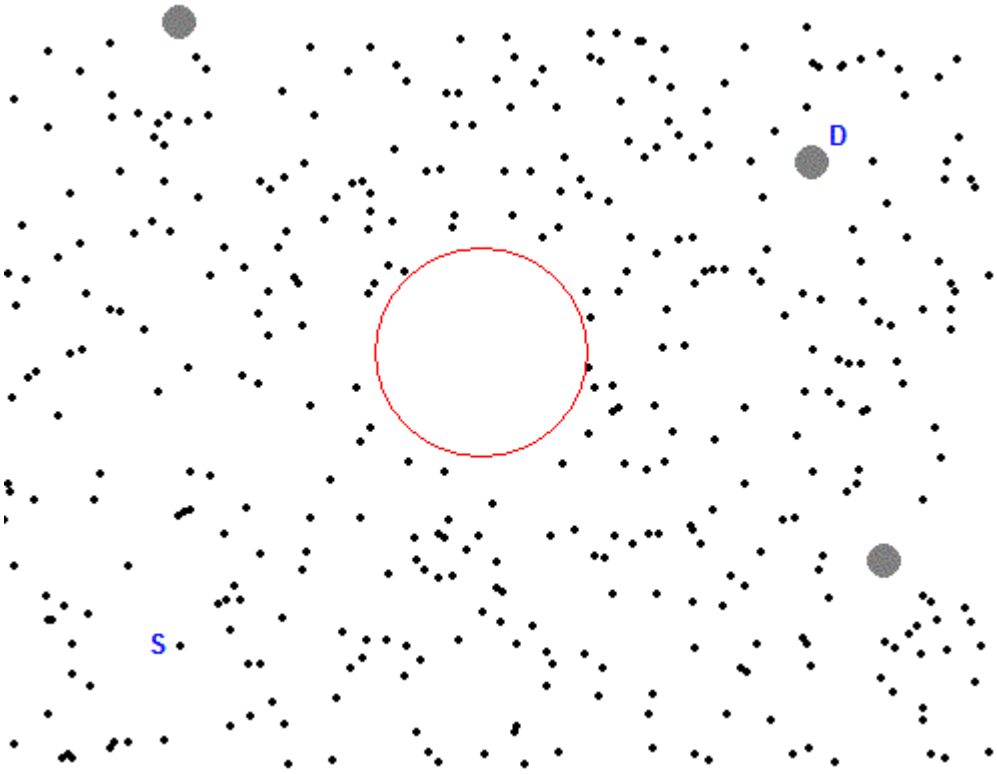

Figure 1 A Typical Sensor Network

Due to the restricted computing and processing capabilities of the sensor node, most sensor network applications use the services of one or more *base stations*. The

sensor nodes collect the data and forward it to the base station. The base station is a special node that has more computing power. Once the data is routed back to the base station, it processes the data.

Figure 1 shows the layout of a typical sensor network. The small dots represent the sensor nodes collecting data. The big dots are the base-stations with higher computing power. Data is routed back to these base-stations possibly using a “multi-hop infrastructure-less architecture” [5]. The circle denotes an area where no nodes are present. This is termed as a void in the network.

Features of WSN

This section lists some of the salient features of a WSN. One of the most important features is the hardware, namely the node itself. A node is a resource-constrained device capable of radio communication, sensing and limited data-processing. It is optionally also capable of actuating the environment. It is low on processing power, energy as well as memory. A sensor node is usually composed of four components: a Processing Unit, a Power Unit, one or more Sensing Units and/or Actuating Units, and a Transceiver. The Processing Unit is typically an 8-16 bit, 1-24 MHz microcontroller with 1kB – 4MB onboard memory. These figures vary within different families of microcontrollers, and with different vendors. The Power Unit usually consists of one or more batteries, providing 3V - 4.5V, generally with a capacity ranging between 1700mAh – 2700mAH. The node can be fitted with various sensors for acoustic, photo, temperature, pressure etc based applications. Each node may also optionally be fitted with an interface for plugging-in an actuator for performing any mechanical actions on a application-specific basis. Figure 2 shows the structure of a sensor node.

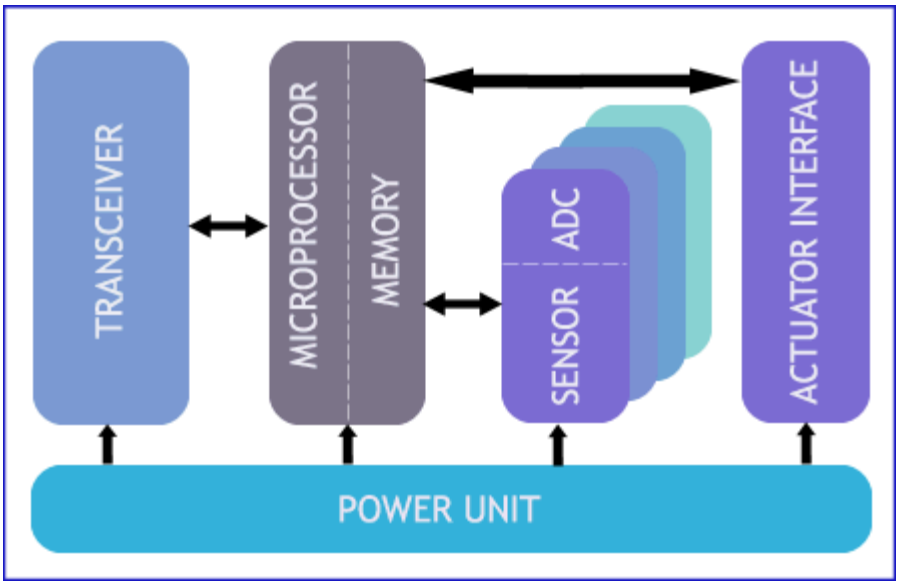

Figure 2 Structure of a Sensor Node

Network Topology is an important aspect of the sensor networks. The lifecycle of a sensor network may be represented in three phases with respect to the topology and its maintenance [5]. During the *Deployment* phase, the nodes are dropped into their positions in an *ad hoc* manner. The nodes need to self-organize into a communicating network. The *Post-Deployment* phase topology maintenance consists of topology changes induced due to the failure of the nodes, failure of radio links, or arrival of some mobile obstacles. The *Re-Deployment* phase deals with the deployment of nodes to replace failed nodes. In each of the three phases, a sensor network should be capable of seamlessly organizing itself to stream data to the base-station.

Sensor Networks are highly sensitive to energy usage. They may, probably, be deployed in inhospitable or hostile environments, where it may not be possible to refresh energy sources. Hence, energy consumption is a major issue, and energy-aware protocols / applications are desirable. Energy consumption is observed at three stages in a node [3]

– communication, sensing and processing. Optimizing the three processes will lead to a reduction in the energy consumed.

Message Routing

Nodes in a network communicate with each other via the transmission of messages. A tiny network, with a few nodes lying relatively close to each other, may be able to establish complete end-to-end communication. This would mean that every node is capable of directly communicating with every other node, without any aid from other nodes. This would lead to a fully-connected topology, i.e. the resulting graph has vertices (nodes) that connected to every other vertex. However, establishing such communication within large networks, consisting of hundreds or thousands of nodes flung far apart, is not possible. Furthermore, many parameters like interference, noise, dispersion, available bandwidth, asymmetry of links and constantly changing signal strength, may make complete connectivity unachievable even in tiny networks. Nodes that can send or receive messages in a single hop are termed as neighbors. A *hop* depicts a direct link of communication between the two nodes. To send messages to nodes that are farther away, the nodes communicate by propagating a message in the network using a commonly-approved protocol, known as the *Message Routing Protocol.*

Message Routing is the process of determining the path that a message will take in the network to travel from the source to reach its destination. Typically, each node in the network observes dual roles: that of a *host* and a *router*. A host is the originator or the final receiver of a message, while a router relays a received message onto a path that will eventually lead to the intended destination. Message Routing is a widely researched subject both within [34][35] and outside [28][29][30][31][32][33] the ambit of Sensor

Networks. The idea behind routing is simple: send a message from one node to the other node, using intermediate nodes as the relaying terminals.

Routing can be categorized as follows:

- *Centralized vs. Distributed:* In centralized routing, the route that a message is supposed to take is calculated by the source, and is embedded into the message. Intermediate nodes just check this route and forward the message to the next node on the route. Conversely, in distributed routing, each node calculates the next node on the route based on the routing protocol. The message consists of only the actual data as the payload, with minimal routing overhead.
- *Static vs. Dynamic:* Static Routing provides the means for explicitly defining the next node from any intermediate node, for a particular destination. This means that every node has an entry for each destination node in a table stating the next node to be chosen in case a message arrives for that destination. Dynamic Routing chooses the next node on the route from multiple nodes based on various criteria like network load and density.
- *Flat vs. Hierarchical:* The entire network is treated as a flat topology in Flat Routing. In Hierarchical Routing, the network topology is assumed to be hierarchical in nature. Groups of nodes form a cluster. Clusters are aggregated to form a higher-level cluster, and so on till the entire network topology is defined. Routing is carried out based on communications between these clusters.
- *State vs. Stateless:* In case of Distributed Routing, each node calculates the next node on the route. For doing this, it may require to store some information regarding its neighbors and/or the message itself. The node maintains expensive routing tables to

keep track of what route to follow for a particular destination. Over a period of time, the nodes identify the entire topology of the network. Routing protocols that store such information are known to follow the State Routing whereas those protocols which do not store any information provide Stateless Routing. While state protocols are expensive in terms of memory, stateless protocols appear to be expensive in terms of time. A good compromise would be to keep track of just enough network information that would enable a node to calculate the next hop correctly without doing any resource-consuming calculations. By correctly, we mean that the next hop chosen should confirm to some routing protocol, and not be chosen randomly.

Message Routing in Sensor Networks

Routing in sensor networks presents a particularly challenging problem due to the intrinsic nature of the medium used to route the messages. Sensor Networks generally communicate via the radio. The use of radio as a medium introduces various problems such as:

- Wireless Link Quality – Radio communication is not bidirectional. Nor does it guarantee a uniform degradation of the link quality as the distance between any two end-points increases. In fact, the quality of a link is transient; it can change drastically over time and distance.
- Noise – The radio medium is beset with problems of noise signals emanating due to the radio antenna, thermal activity, other radio sources etc. This leads to corrupt radio messages.
- Collision – The radio channel is a shared medium. Multiple nodes trying to use the same channel result in collision of messages. A good routing protocol should take

into account message loss due to collisions and adapt itself to deliver the message, overcoming the collisions.

- Multipath – Reflection of radio signals from terrestrial objects and other impedances result in the radio signal being transmitted via multiple paths. This results in interference with other signals, and adds to the noise.
- Fading – Radio signals are transmitted with certain signal strength. Greater signal strength results in a bigger ‘radius’ of the communication. This implies that signals transmitted at a higher strength will be heard by nodes that are farther. However, due to obstacles, energy is absorbed leading to an attenuation of the signal strength. The radio channel is subject to change in conditions that changes the attenuation. This is known as fading.
- Restricted Bandwidth – Bandwidth available to a WSN application is limited. Thus routing protocols should maintain minimum overhead to transfer as much data as possible using the limited bandwidth. This is important in ensuring fast processing of the data.

Problem Description

Imagine an Early Warning Seismic Network that is deployed in a seismically sensitive zone to gather data emitting from the geologically volatile area. While the major concern of such a system would be to sense major upheavals in the zone that may signify an impending seismic activity, the network may also be used to collect data at otherwise "normal" times. This data may provide useful information regarding the geological activities that are generally evolving all the time. The major needs of such a system would be high fidelity (ability to give early warnings) and minimum false alarms.

Routing would play a major part in such a system. The nodes would be constantly pushing "normal" data towards the base-stations. Such data may be classified as regular, low-priority data. Loss of a few occasional packets/messages would not hamper the Early Warning ability of the system. However, when the nodes sense a major change in the data that they are sensing, they need to ensure that such data messages are not lost. Hence, they need to accord a high priority to such messages before dispatching them towards the base-stations.

Such scenarios wherein data being routed may change in importance over time necessitate the evolution of advanced routing solutions that accord varying degrees of priorities to the messages, and ensure the delivery of more important messages with a higher fidelity than that for less important messages.

Limited In-Network Data Processing is an important ingredient of many WSN applications. Self-Localization is a process wherein nodes localize their positions and self-calibrate, generally based on acoustic signals [50][18][19][20][21]. Another instance of in-network data processing is the vibration control in the fuselage of the aircraft. Sensors would sense the amount of vibrations in a particular area, which would then be transmitted to nodes in surrounding areas. The nodes would gather data, process it and send signals to the actuators to take corrective action based on the feedback of vibrations from the surroundings. A routing protocol providing end-to-end solution between any two nodes is essential.

Problem Statement

Hence, we argue for the need of a routing protocol that supports transmission of messages from any node to any other node and treats messages based on priority. A

routing is said to be connected when it is possible to route messages from any node in the network to any other node.

There is a need for a Routing protocol to support message routing in WSN. Such a protocol would support any-node-to-any-node message routing as well as provide facilities to treat message with different priorities differently. I propose to develop a location aware routing protocol that utilizes the positions of the nodes to calculate the route. Location awareness has its own set of unique problems. Nodes need to know their position as well as the position of their neighbors. With constant changes in the radio link strength, it is extremely difficult to keep an updated record of all the neighbors. Furthermore, a location-based routing protocol calculates distances between the nodes. This is extremely expensive in terms of CPU and energy. One of the challenges would be to adapt common algorithms for calculating distance, sorting etc to work well on resource-constrained nodes.

The routing should be robust and insensitive to the changes in topology. It should adapt itself quickly to the connectivity changes. This dynamism precludes the use of static data tables stored locally on the nodes.

CHAPTER II

LITERATURE REVIEW

Survey of Prominent Routing Algorithms

Routing has been a widely and intensely studied subject since the time when networks came into existence. One of the earliest forms of routing is the Hierarchical Alternate Routing algorithm [27] used in static routing of messages for the telephone lines. The first network of computers, ARPANET, heralded the beginning of internet. ARPANET used software based on the Bellman-Ford distance vector algorithms, which also forms the basis for the RIP (Routing Information Protocol) [28][31], one of the earliest and widely used routing protocols. Development of bigger networks and inter-connected labs, offices etc led to the development of more complex routing protocols like OSPF (Open Shortest Path First) [30], BGP (Border Gateway Protocol) [29] etc. With the advent of Wireless Communication and the use of ether as a communication medium of transferring data, newer routing protocols had to be developed that would work with the unique characteristics displayed by the radio medium. Some of the widely used routing protocols in this genre are AODV [32] and DSR [33]. However, with the technology taking the next big-leap into the area of WSN, a newer crop of routing algorithms has been developed that is suitable for the resource-constrained devices in terms of memory, computational power and energy usage. Any algorithm that may be chosen for routing should possess characteristics that suit the medium and platform that it is being used for. Furthermore, it should not only be robust, but also support a range of desired features that make the routing as generic, reusable and flexible as possible.

Dynamic **S**ource **R**outing (DSR)

Dynamic Source Routing (DSR) [33] is one of the most popular routing algorithms for wireless networks. It is a *Source Routing* protocol, which implies that the sender of the message builds the entire route at the source itself, and then forwards the message to the node in the route. Each node that receives the message checks whether it is the destination. If it is not the destination, it just forwards the message to the node next in the route that is embedded in the message header.

The DSR algorithm has two main operations: *Route Discovery* and *Route Maintenance*. Whenever a node wants to transmit a message to another node, it looks up its *Routing Tables* to check whether it has a route to that node. If it has the route, then the message is embedded with the pre-calculated route and forwarded to the first node in the route. If there is no such route in the routing table, then the node initiates a Route Discovery. The host node initiating the route discovery broadcasts the *route request*. This request, among other things, contains an ID and a route record. This record contains the actual route that has been accumulated till that time. On receipt of the route request, each node can (1) discard the request (since it has already processed it sometime back), or (2) append its own address to the route record and forward it, or (3) return a copy of the route in a *route reply* (since it is the destination for which the route request was initiated).

If the destination node has a route to the source node in its routing table, it will send the route reply using that route. If there is no record of the source in the routing table of that node, it initiates a route discovery for the source-node, and piggybacks the route reply onto that route request. Once a node gets the route to the required node, it

enters this route in its routing table, and maintains a link to this route till it continues communication with the destination node.

Wireless networks, as explained in Chapter I, allow for the network nodes to be mobile. Moreover, the links between two hosts are asymmetric in nature. Consequently, links between two nodes change continually, leading to a lot of routes existing in the various caches of the nodes being invalidated / broken. So long as a route is in use, the Route Maintenance procedure monitors the operation of the route and detects any routing errors encountered on the route. Errors are detected on the basis of availability of the hop-by-hop acknowledgements. Whenever a message is passed from one node to the other, the transmitting node expects either an implicit or an explicit acknowledgement from the receiver at the next hop. If the acknowledgement wait-period times out, the route is assumed to be broken. The transmitting node that detects the break in the route then sends back a *route error* message to the source of the message. The node sending back the route error message should have a route back to the source; else, it would either initiate a route discovery, and then forward the error message when it has the entire route, or piggyback the error message on top of the route discovery message. On receipt of the route error message, the route is invalidated after the node that encountered the message.

One of the earliest, and very popular, wireless routing algorithms, DSR does not use periodic broadcasts of a node's position to let other nodes know of its position. Thus, it doesn't have any need for neighborhood tables. However, it does make a heavy use of routing tables to store routes to nodes with whom the source node is communicating. Also, it makes use of IP addresses as host addresses, and is generally suitable for devices with network interface cards. The Internet Protocol (IP) [57] is a data-oriented protocol

used by source and destination nodes for communicating data over packet-switched networks. It is thus unsuitable for WSN devices that are limited memory and energy to employ this algorithm. However, DSR has inspired many current WSN algorithms and is acknowledged to be a very good routing protocol.

Ad-hoc **O**n-Demand **D**istance **V**ector (AODV) Routing

Ad-hoc On-Demand Distance Vector (AODV) Routing [32] is another popular routing algorithm for wireless ad hoc networks that operates using distance-vector routing mechanisms. It is quite similar to the DSR [33] in that it too uses the concepts of Path Discovery and Maintenance. However, AODV builds routes between nodes on-demand i.e. only as needed.

AODV does not depend on network-wide periodic advertisements of identification messages to other nodes in the network. It periodically sends "HELLO" messages in the local context of the system, to build up a set of neighbors. It then uses these neighbors in routing.

Whenever any node needs to send a message to some node that is not its neighbor, the source node initiates a *Path Discovery*, by sending a *Route Request* (RREQ) message to its neighbors. This is somewhat akin to the procedure followed by DSR. Nodes receiving the RREQ update their information about the source. They also set up a backward link to the source in their routing tables. Each RREQ contains the source node's address (IP address) and a Broadcast ID that uniquely identifies it. It also has a current sequence number that determines the freshness of the message. Thus, a message number with a higher sequence number is considered to be *fresher* or more recent than

that with a lower sequence number. The RREQ also contains a hop count variable that keeps track of the number of hops from the source.

On receipt of the RREQ, the node checks whether it has already received the same RREQ earlier. If it has received the same RREQ earlier, it drops the RREQ. Otherwise, if it is an intermediate node without any record of a route to the final destination, the node increases the hop count and rebroadcasts the RREQ to its neighbors. If the node is the final destination, or an intermediate node that knows the route to the final destination, it sends back the *Route Reply* (RREP). This RREP is sent back via the same route traversing which the node had received the message from the source. As the RREP propagates back to the source node, the intermediate nodes setup forward pointers to the actual destination.

When the source node receives the RREP, it checks whether it has an entry for the route. If it did not have any entry in its routing table, the node creates a new entry in the routing table. Otherwise it checks the sequence number of the RREP. If the RREP arrives with the same sequence number as in its tables but with a smaller hop count, or a greater sequence number (indicating fresher route), it updates its routing table and starts using this better route. Once an entry for the new route has been created in the table, the node can start communication with the destination. Every time a node receives subsequent RREPs, it updates its routing table information, and only forwards those that are fresher or contain a smaller hop count. Each routing table entry contains information for the destination, the next node, number of hops to the destination, sequence number for that destination, active neighbors for the route and expiration time of the table entry. The expiration time frame is reset every time the source routes a packet to the destination.

AODV considers two ways a route may be broken. In the first case, the source may move from its position, in which case, it may simply initiate RREQ again. In case the destination or intermediate node die or move, some node would receive a message that it cannot forward to the next node. This node would then check its routing table and find out all the routes that use the failed node for the next route. It then marks all these routes as invalid, and sends out RERR to the source nodes of all the routes. On receipt of the RERR, each node would invalidate its routes that contain entries to the failed node denoted in the RERR. It would then propagate the RERR down onwards as earlier.

Though AODV is a robust algorithm and works very well for the Wireless networks, it is unsuitable for the senor networks as it uses memory consuming routing tables. It also assumes the presence of symmetric links in the medium, and disregards any pair of nodes that don't establish a symmetric link. The algorithm is memory intensive and is meant for devices with complex network interfaces, as DSR.

Location **A**ided **R**outing (LAR)

Location Aided Routing (LAR) [25] provides location based routing using restrained / directed flooding. This was one of the earlier location-based routing protocols, and used few ideas from the DSR protocol described above.

The first phase of this routing is the Route Discovery using flooding. Whenever a node needs to find a route to another node, it initiates the route discovery like in DSR. The requesting node sends a route request to all its neighbors. On receipt of this route request, the neighbors check whether the route request is meant for them. If not, they broadcast the route request again to all their neighbors only once, discarding any more route requests for the same combination of sender and receiver. At every hop, a node is

added to the route that is contained in the route request. Once a route request reaches the destination node, it responds by sending a route reply. This route reply follows a path obtained by reversing the path contained in the route request.

LAR assumes that the mobile nodes are constantly moving and that a node's location at two different times will most probably be different. The *Expected Zone* of a receiving node from the viewpoint of a sending node is the zone in which the receiver is expected to be present based on the prior information of the receiver's location and its velocity of movement. If no such information is available, then the expected zone may potentially be the entire ad hoc network, leading to the algorithm being reduced to flooding. The *Request Zone* is the zone in which a forwarding node must lie. An intermediate node may only forward a route request if it is within the request zone.

The main thrust of LAR is the methodology used to determine whether a node is in the Request Zone or not. Two schemas may be used. One schema assigns the request zone to be a rectangle with its sides being parallel / perpendicular to the X-Y axes. This rectangle is cornered at one side by the sending node. The other corner of the rectangle is formed by the intersection of the tangents to the Expected Zone (usually a circle) of the destination. In case of the sender being located within the expected zone, the request zone is specified to be a rectangle enclosing the expected zone.

The request zone is not specified explicitly in Scheme Two as was done in Scheme One. Instead, the route request contains two parameters. One is the distance of the sender from the last known position of the destination. The last known co-ordinates of the destination are also specified in the route message. On receipt of the message, an intermediate node will calculate its own distance from the destination. If this distance is

less than the distance contained in the message, and it is at least some specific distance away from the previous hop's node, the node will accept the route request, else it will drop it.

The LAR is another protocol that is designed for the wireless networks in general, but does not account for the unique and stringent characteristics of the sensor network. Thus expensive routing tables need to be maintained. This protocol is similar to DSR in operation but differs in the aspect of route building. However, an added disadvantage of this protocol is that route is found out using flooding. This gives an *O(n)* complexity to each route discovery. However, each node receives the same route request from each of its neighbors, making it process *O(n)* requests for propagation. These control messages are too many. The LAR protocol chooses a route that is of the smallest length. However, on receipt of multiple routes of same length resulting from the route request, LAR is unclear about which route to accept and store.

Low **E**nergy **A**daptive **C**lustering **H**ierarchy (LEACH)

Most sensor network applications are related to data collection. The job of the sensor nodes is to monotonously collect and route data to one or more central processing nodes or *base-stations*. LEACH [34] has been designed for WSN running such applications. The basic premise of LEACH is that the control nodes in the WSN are situated far from the area of the deployment of nodes. It also presumes all nodes to be energy-constrained and homogeneous. The assumption of homogeneity aids in the energy calculations. Sensor networks generally contain redundant or supplementary data. Such data needs to be aggregated into a single chunk of useful and meaningful data that can

then be sent over to the base station. This is known as *data aggregation* or *data fusion*. LEACH provides features for data aggregation too.

LEACH is a hierarchical routing algorithm. It assembles the nodes into groups or clusters. Each of these groups has a cluster-head. LEACH has four phases of operation: *Advertisement, Cluster Setup, Schedule Creation* and *Data Transmission*.

In the advertisement phase, each node decides whether it can become a cluster head or not for the current round. This is based on the percentage of cluster heads desired in the WSN as compared to the population of the WSN. Based on this percentage, *P*, a node becomes eligible for being a cluster-head if it has not been a cluster-head in at least $1/P$ rounds in the past. A threshold value is calculated for each node using the current round number, *r*, and *P*. This threshold value is given by the equation:

$$T(n) = \begin{cases} \dfrac{P}{1 - P * (r \% 1/P)}, n \in \textit{Set of nodes not cluster heads in last } 1/P \textit{ rounds} \\ 0 \end{cases}$$

A random number between 0 and 1 is generated. If the generated number is less than the threshold for that node, the node becomes a cluster-head. Each node that is elected to become a cluster-head advertises itself using the Media Access Control (MAC) [59][58] based Carrier Sense Multiple Access (CSMA) [60] protocol and transmitting at the same energy. Once the advertisement is done, each non-cluster-head node decides what cluster to join based on the received signal strength from the cluster-head advertisement. The assumption here is that links are not only symmetric but also consume the same amount of energy for transmission either way. The cluster-head which was heard with the largest signal strength would obviously be the closest to the node, and hence would be chosen. This process is repeated periodically, so that every node in the

WSN is given a chance to become a cluster head, and all nodes can equally share the responsibility of message transmission. This also distributes the energy-usage of transmission, ensuring a longer life for all nodes.

In the Cluster Setup phase, all nodes transmit their membership requests to the chosen cluster-heads. This information is stored by the cluster-heads for the next phase.

The Schedule creation phase involves the creation of a Time Division Multiple Access (TDMA) [61] schedule. The cluster-head creates a TDMA schedule for interacting with every node in its cluster. This schedule allots a fixed time slice to every cluster-node during which it can establish an active communication channel with the cluster-head. The nodes can either switch themselves off or perform other duties when they are not slotted to communicate with the cluster-head. It then sends this schedule information to the cluster members informing them of their time-slot during which they may transmit their data to the cluster-head.

The final phase involves Data Transmission. At the end of every TDMA cycle, the cluster-head collects all the data that has been gathered. It then compresses all the data using some application-specific signal processing algorithm. This composite data is now transmitted to the far-away base station using a high-energy signal.

Communication between the nodes in each cluster is based on a TDMA schedule with CDMA [62] protocol. However, having the same CDMA codes for all clusters would lead to collisions / interference with messages from other clusters. Such interference is avoided by the use of different CDMA codes.

Each cluster-head is rotated periodically. This ensures that the load of high-energy transmission to the base station is distributed fairly among all nodes. Since the cluster

membership is decided on the basis of signal strength of the message received from the cluster-head, communication within a cluster leads to low energy consumption. This algorithm may be extended to multiple levels of hierarchical clusters.

However, the most obvious constraint of this routing is its specific usage for applications involving data aggregation only. Though no expensive routing tables need to be maintained, the actual routing is done over only two hops. Nodes in most WSNs, especially those deployed in hostile environments, are not distributed uniformly. Thus, it may not be possible for a cluster head to transmit to a base station even using higher energy to increase the signal strength of the transmission. There is no support for routing from one node to any other. Furthermore, the all the phases considered together implicitly impose the constraint of symmetric links being present in the WSN. This constraint cannot be satisfied very easily. The algorithm also does not talk regarding scenarios involving nodes that do not chose to join any clusters due to a plethora of reasons; the node may not be able to hear any cluster-heads, the node may send the message to join a cluster later than the deadline etc. Thus, the LEACH routing protocol is not suitable for WSN applications whose primary responsibilities involve duties other than data collection and aggregation.

SPEED

John Stankovic et al. proposed the SPEED protocol [35] that utilizes geographic routing and provides three different services for routing messages: Regular Unicast, Area Multicast and Area Anycast. Regular Unicast involves sending a message to a single node in the network. Area Multicast provides for transmitting messages to a particular geographical area in the network. All the nodes in the area will receive the messages.

Area Anycast offers the choice of sending a message to any node in a particular geographic area. This means that the application does not care about which particular node receives the message so long as the message is delivered to any node in the area.

SPEED claims to provide a QoS routing with “Soft Real-Time” guarantees. Its major components are *Neighborhood Beacon Exchange, Stateless Non-deterministic Geographic Forwarding, Neighborhood Feedback Loop, Backpressure Rerouting* and *Last Mile Processing*.

Neighborhood Beacon Exchange involves periodic broadcast of the neighborhood beacons containing the node ID and the position / location. Each Neighbor Table entry contains the NeighborID, its location, a SendToDelay (calculated as round-trip delay for a message to the neighbor) and the time of the entry’s existence in the neighbor table. The last entry basically denotes the lifetime of the neighbor.

The protocol distinguishes its set of neighbors into a Neighbor Set (NS), which are inside the radio range of the node, and a Forwarding Set (FS), which is a subset of Neighbor Set, consisting of nodes that are closer to the destination. The Stateless Non-deterministic Geographic Forwarding (SNGF) is employed to forward messages to only those nodes that are present in the FS of the node. Nodes in FS are subdivided into two sets; those that have a *relay speed* (defined as Distance to destination / hop delay) higher than some desired speed, called setpoint, and those that have a relay speed less than the setpoint. SNGF forwards the message to nodes that have higher relay speed. If there are no such nodes, then a relay ratio is calculated and fed to the Neighborhood Feedback Loop (NFL). This loop essentially activates a Relay Ratio Controller (RRC) in case there are no nodes in the Forwarding Set, FS, which have a higher relay speed than the

setpoint. The RRC takes into account the miss ratios (percentage of misses wherein the neighbor receives a packet that is below the relay speed) of the neighboring nodes and calculates the relay ratio such that the NFL can converge the miss ratios of the neighbors to zero.

SPEED uses MAC layer feedback to calculate congestion of traffic on a node's forwarding nodes. Heavy traffic leads to a reduction in the relay speed affecting the real-time guarantees. Once a node senses the congestion around a forwarding set of neighbors, it re-routes the packets through another set of forwarding neighbors, thus partially reducing the traffic around the congested nodes. Eventually all the neighbors of the congested nodes would route packets around them, thus reducing the congestion and increasing the relay speed. In case of congestion around all the forwarding nodes of a node, say *x*, 'x' drops a few packets. 'x' also sends a *back-pressure* beacon which details the node id, the delay to send the messages and the direction of the delay (based on the ultimate destination of the message). This enables nodes, that are forwarding messages to 'x' to subsequently deliver to the destination 'd', to reroute the messages to 'd' via some other node. Figure 3 explains this with an example. In this case, the node 'x' corresponds to node 5, while destination, d, is the node 13.

Node 2 delivers message to node 13 via nodes 3, 5 and one of 7, 9, or 10. However, there is a temporary outage of connection between nodes 7, 9, and 10 on one side, and node 13 on the other. Consequently, there is a delay in forwarding messages to node 13. Node 5 senses the delay and sends a back-pressure message to node 3, which then starts routing messages intended for node 13 via nodes 4, 11 and 13. It may be noted that node 5 still continues to participate in forwarding messages from node 4 to node 6.

The SPEED protocol avoids the voids in the network using a similar mechanism as Backpressure Routing.

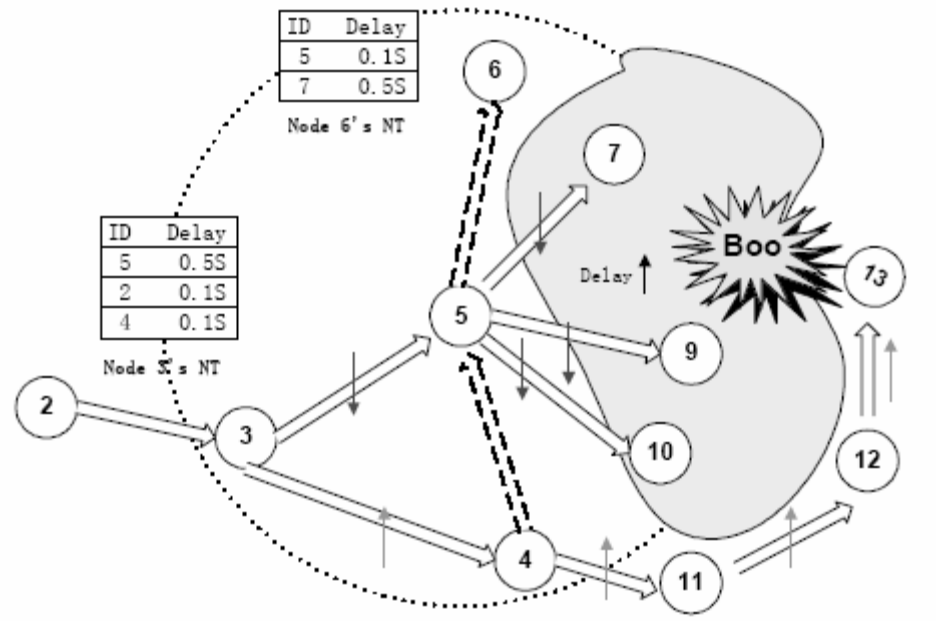

Figure 3 Backpressure Re-Routing [35]

SPEED employs the Last Mile Processing (LMP) to transmit messages in the Area Multicast and Area Anycast services. The area in these services is generally defined by a sphere. In case of an anycast, any node inside this sphere that receives the message will store the message immediately. Multicast service forces nodes in the area to broadcast the message to other nodes within the boundary of the area, while keeping a copy of the message. The LMP is also used for Unicast wherein the message is broadcast but only the destination node may save a copy of the message.

The SPEED protocol is another Greedy geographic forwarding protocol designed to avoid the pitfalls of general greedy forwarding of messages. It provides real-time guarantees to delivery of messages. However, the algorithm is dependent on a rich MAC

layer in the NFL and Backpressure rerouting phases. MAC layers conforming to IEEE 802.11 specifications as required in SPEED may not available in most sensors. SPEED has been implemented on the Berkeley motes [36] with a code size of 6,036 bytes. However, it does not offer any service to distinguish messages based on their importance.

Greedy **P**erimeter **S**tateless **R**outing (GPSR)

The GPSR [22][23] proposed by Karp and Kung is one of the earlier geographical routing protocols. The GPSR adopts a greedy forwarding strategy to route messages. It makes uses of a neighborhood beacon that sends a node's ID and its position. However, instead of sending this beacon periodically and add to the network congestion, GPSR piggybacks the neighborhood beacon on every message that is ever sent or forwarded by the node.

Every node in GPSR has a neighborhood table of its own. Whenever a message needs to be sent, the GPSR tries to find a node that is closer to the destination than itself and forwards the message to that node. However, this method fails for topologies that do not have a uniform distribution of nodes, or contain voids. Hence, the GPSR adapts to this situation by introducing the concept of Perimeter Routing utilizing the *right-hand graph traversal* rule. Whenever a message is received by the node, x, from a node, z, it forwards the message to another neighboring node, y, such that the edge x – y is the first sequential edge traversed counterclockwise from the edge z-x. The authors claim that this approach in conjunction with a heuristic of no-crossing that is used is able to find routes 99.5 % of the times. However, the no-crossing heuristic that is used to force the nodes to follow the right-hand rule may also result in the disconnection of the network.

The amended GPSR thus takes into account the planarization of the network graph. The GPSR converts the network graph into either a Relative Neighborhood Graph (RNG) or a Gabriel Graph (GG). Every time a new neighbor is detected, or the network topology is changed, the graph is re-planarized. Whenever a node has a message to be delivered, it starts its transmission in the greedy mode. If the node can transmit the message to a node that is closer to the destination than itself, it will do so. However, if there is no such closer node, then the GPSR will enter the perimeter node, and transmit the message to the next node using a simple planar-mode graph traversal. On entering the perimeter mode, the GPSR saves the address of the node, say *l*, which was the transition point between the greedy mode and the perimeter mode. Every time the message reaches a new node, the GPSR checks the message to verify whether the current node is closer to the destination than *l*. If it is, the GPSR immediately transfers to the greedy mode, and resumes greedy routing.

Every packet transmitted in GPSR has a fixed number of retransmits. This information is given to the node by the MAC layer that is required to be compliant to the IEEE 802.11 standard. Mostly, the MAC layers are merely simplified versions of the 802.11 standard. This may render the GPSR protocol unusable in its default form. The GPSR does not elucidate more on the action taken in case a message is unable to be transmitted even in perimeter mode. Finally GPSR disallows the use of periodic broadcast of the neighborhood beacons, and piggybacks these beacons on the messages sent by each node. This leads to an increase in the message size by 12 bytes, which is a lot in case of resource constrained nodes that typically have a message body of ~30-50 bytes. Moreover, the overhead introduced in planarizing the graph every time the

topology changes may render the algorithm unusable in scenarios involving a highly dynamic topology / link interface between nodes.

Conclusions from the Survey

Various routing protocols are available. Most of the routing protocols that were studied were either unsuitable for use in WSN, or had issues with implementation on highly resource-constrained nodes. DSR and AODV are arguably the most popular of all routing protocols. However, due to their extensive use of routing tables and source routing, they render themselves inappropriate for routing in WSN. LAR introduces the concept of routing using geographical positions of nodes. It also demonstrates the use of smart flooding to build routes. Nevertheless, it is still a source routing protocol. LEACH is the first routing protocol for WSN that was studied. However, LEACH may be unsuitable for use in most environments that do not involve base-station routing or data aggregation. GPSR and SPEED are the most promising, though both make use of the IEEE 802.11 MAC layer that may not be supported in most devices. However, none of these protocols support a prioritized mode for delivering messages. They do not provide any service for distinguishing the importance of messages in the network.

Developmental Tools

Development of algorithms for WSNs is a complex task. From conceptualizing the algorithm to implementing it and finally testing it is a long, tough and arduous undertaking. A valuable asset in this development is a process that allows one to test the rudimentary ideas before actually implementing the algorithm on the target platform. However, once the implementation is achieved, it needs to be thoroughly tried and tested

before it can be rapidly deployed on WSNs in real-world scenarios. Testing an algorithm for a WSN or a WSN application would involve use of hundreds, if not thousands, of nodes deployed over a vast area. A simulation environment that aids such a deployment and testing of the algorithm needs to be used. Some of the popular simulation environments for sensor networks are MATLAB [39], SensorSim [37][42], which is an extension of ns-2 [41], and GloMoSim [38]. This section reviews the tools and environments that were used in the development and analysis of PGR. We chose to use Prowler [40], a Discrete-Event simulator for WSN, which runs under MATLAB.

MATLAB and Prowler

MATLAB

MATLAB, developed by Mathworks [39], was originally developed by Dr. Cleve Moler to provide functionality for handling complex matrix libraries. MATLAB is a language for technical computing. The basic data element in MATLAB is a matrix that does not require any pre-dimensioning. The language provides a user-friendly environment that integrates computing, programming and visualization. It is typically used in environments requiring computation, analysis, modeling, simulation, application development, rapid prototyping, visualization, and algorithm development among others.

MATLAB comes equipped with a wide variety of toolboxes developed for specific domains like real-time systems, signal processing, control systems, fuzzy logic, neural systems, simulation, state-charts etc. Each of these toolboxes can be used separately or in conjunction with others.

The MATLAB system consists of five parts:

- The MATLAB language: This contains functions and other programming features that let the user perform command-line operations based on a matrix as its data element.
- The Graphics System: This handles the graphical interface part of the application development. It provides 2D and 3D visualization, image processing and GUI support.
- The MATLAB API: This provides an interface for integrating C and FORTRAN programs with MATLAB.
- The MATLAB IDE: This includes an editor, a debugger, a command-line, a workspace environment etc.

Prowler

"Prowler is a probabilistic wireless network simulator capable of simulating wireless distributed systems, from the application to the physical communication layer" - [40]. Prowler allows an application developer to simulate a WSN application using two different radio models. It simulates the radio transmission, propagation and reception involved in ad-hoc radio networks. Prowler is also able to successfully simulate collisions of messages and the operation of the MAC layer.

A Prowler application consists of three basic files that contain the topology, animation and the actual application information. Users can also provide information or help file, as well as a parameter file for specifying the application-specific parameters. The main application file contains the application-specific code. The topology file specifies the various topology structures that an application uses. This file contains code

to generate a fixed or random topology as required by the user. The animation file specifies how the Prowler GUI would react to the events generated by the application.

Prowler provides applications with various events during the simulation. It lets the user application react to the initialization of the application, sending and receiving of packets, collisions, and the end of application. Prowler also provides multiple timers. However, these are single-firing timers, and need to be set to fire again after a user-defined interval. Pre-defined actions that an application may use include Send_Packet for sending messages and Set_Clock for starting the clock. An application may be simulated from Prowler by registering it with Prowler. The user can specify the number of nodes that need to be simulated, and each node is installed with an image of the application.

Prowler has been developed as a simulator specifically targeting the TinyOS platform for sensor networks. Nevertheless, it can also be used as a generic simulator for applications using WSN. One drawback of Prowler is that it is a homogeneous simulation environment. It cannot simulate different applications concurrently in the same environment.

TinyOS, nesC and TOSSIM

TinyOS

TinyOS is a component-based operating system for the sensor networks. It consists of a tiny scheduler and a graph of components [45]. Each component is a collection of "*command handlers*, *event handlers*, an encapsulated fixed-size *frame*, and a bundle of *threads*". A typical TinyOS application is a collection of components. Each component uses zero or more other components, and is in turn used by multiple components. There is a single top-level component in the hierarchy with a zero

cardinality of being used by any other component. These components interact with each other using interfaces.

TinyOS uses a static memory allocation model to allocate memory blocks. This manifests as a reduction in execution time savings due to access of variables at statically compiled locations instead of dynamic pointer access. The overhead incurred due to dynamic allocation of memory is also eliminated, thus ensuring that the memory footprint of a component is known at compile time.

Commands are functions that implemented by low level components and are provided for use to the higher level components. They are non-blocking requests made to the components providing them. Commands should return a value to its caller to indicate the status of the call.

Event handlers are used to deal with hardware events. Whenever any hardware interrupt like timer, counter, etc. occur, a chain of event handlers is invoked that propagates up the hierarchy of the component structure. It may also propagate downwards using the commands. To prevent cycles, commands cannot signal events.

Tasks or threads are functions that provide the primary functionality in a component. They are posted from within a command or an event. They may signal higher-level events, call lower-level commands and post other tasks within the component. These tasks are atomic with respect to other tasks, though they may be pre-empted by an event. Tasks are scheduled using a simple FIFO scheduler. A Task neither takes any parameters as its arguments nor does it return any value.

TinyOS uses the Active Message communication model used widely in parallel, distributed systems [46]. Each active message contains a user level handler that needs to

be invoked on its arrival at a node. It contains a data payload as well. Both are embedded as arguments in the Active Messages.

nesC

nesC [44], an extension of the C language [47], has been designed to code applications in TinyOS. nesC has been designed to directly support TinyOS' event-based concurrency model and need for static memory allocation.

nesC offers a component-based design solution for programming applications for TinyOS. *Components* interact with other components via *Interfaces*. An interface is a manifestation of the behavior, or part thereof, exhibited by a component. Interfaces can either be provided or used by components. An interface, quite like a header file, contains information regarding commands provided and events used. A component providing an interface needs to implement the functionality of the commands contained therein. Similarly, a component using the interface, while calling the commands implemented elsewhere, needs to provide implementation for any events defined by the interface. Thus, Interfaces are bi-directional. They have a set of functions that are implemented by a component that provides the interface, as well as a set of functions that are implemented by the component that uses the interface. The provided interfaces represent the functionality offered by a component to its user, whereas the used interfaces correspond to the functionality required by a component.

nesC segregates the component-space into two: *Configurations* and *Modules*. Modules contain the application code, implementing one or more interfaces used / provided by the component. They are usually named as *ComponentName*M, the trailing 'M' denoting that the component is a Module. Alternately, Configurations, identified by

the trailing ‘C’ in *ComponentName*C, contain the logic to wire all the other components together. This wiring logic connects the interfaces provided by components to interfaces used by other components. Every nesC application is contains a single top-level configuration.

TinyOS offers concurrency through tasks and events. nesC can determine the race condition related issues at compile-time. “The concurrency model of nesC is based on run-to-completion tasks and interrupt-handlers, which may interrupt tasks and each other. The nesC compiler signals the potential data races caused by the interrupt handlers” - [48].

TOSSIM

TOSSIM or Nido [43][49] is a discrete-event simulator designed for simulating nesC applications written for TinyOS. TOSSIM uses the nesC code written for the TinyOS application and compiles it for the PC platform.

TOSSIM simulates the behavior and operations of up to a thousand nodes running TinyOS applications. It generates discrete-event simulations directly from the TinyOS structure by replacing a few low-level components. This results in the translation of hardware interrupts into discrete simulator events. The entire TinyOS node-network is modeled as a directed graph. The nodes form the vertices of this graph, while an edge exists between two nodes if the source of the edge can transmit to its destination. Each edge has a bit-error rate value associated with it, which signifies the fidelity of the radio link between the two nodes. A bit-error rate of zero (0) signifies perfect transmission conditions, while a value nearing one (1) denotes a very low probability of message transmission due to high noise / interference.

The start of a simulation results in the instantiation of a user-specified number of nodes. These nodes are booted in a staggered sequence to prevent artificial or involuntary synchronization amongst the nodes. This is a result of TOSSIM keeping a global time. Also, to achieve maximum fidelity with the TinyOS applications running on the mote, TOSSIM operates at the granularity of the mote instruction clock cycle.

Since TOSSIM simulates the TinyOS application on a PC platform, a mechanism needs to be provided that would simulate the hardware interrupts. This is achieved through the use of a simulator-event queue. A simulator event, different from a TinyOS event, invokes the interrupt handler of the hardware abstraction components used in the TOSSIM executable. Each simulator event is associated with a specific mote. When the simulator event is executed, a global state is et depicting the currently running mote. Once control transfers out of the hardware abstraction layer, TOSSIM executes the TinyOS code as it would be executed on a real mote platform. A few of the hardware components abstracted by TOSSIM are the clock, the ADC, the EEPROM, several components in the radio stack etc.

TOSSIM also provides facilities to use various degrees of radio fidelity from perfect transmission to a lossy communication model. Different radio model provide the users with the ability to test the applications at different levels of complexities.

TOSSIM provides a framework for simulating the radio network as well as the nodes. It also provides mechanisms through which the simulated nodes can communicate with external world, by use of Java-based applications. A Java-based graphical interface for TOSSIM, TinyViz, has also been developed recently. It provides visualization and control capabilities to simulations via the use of various plugins.

Summary

Various simulation tools, frameworks and integrated environments exist to test algorithms and applications for WSNs. A typical application / algorithm development life-cycle starts from the conceptual stage, and runs through the feasibility study, implementation, testing, and finally, deployment stages. It is unviable to investigate the feasibility of an algorithm by testing it on a native platform. Hence a simulation environment that allows testing of the general concepts of the algorithms is desired. We use the MATLAB-based Prowler for these purposes. Once an algorithm has been investigated and implemented, a testing environment is desired to check the correctness of the implementation. Since it is impractical to test such an implementation on a real WSN due to sheer logistics involved, a simulation environment that provides features closer to the native environment is favorable. We use TOSSIM towards this purpose.

CHAPTER III

PRIORITIZED GEOGRAPHICAL ROUTING

This chapter describes the priority-based, location-aware routing algorithm named Prioritized Geographical Routing (PGR). The first section describes a beaconing component that enables the nodes in the network to learn about their neighbors. This beacon also allows the nodes to continuously update their record of their neighbors and adapt themselves to the changing topology of the ad-hoc network. Later, the basic algorithm for greedy geographic forwarding is described. Topologies that lead to the failure of this greedy scheme are identified. Therefore, there is a need for developing an alternative mechanism to ensure the delivery of appreciably important messages to their intended recipient, while allowing less important messages to be lost. "*Failure*" indicates the loss of a message in the network due to poor connectivity, noise, interference, network-holes or change in topology. A message should be dropped if and only if no path is found to the destination. Any other cause for the loss of a message should be construed to be failure.

Protocol for Neighborhood Detection and Maintenance

Most routing algorithms depend on the knowledge of the nodes in their vicinity, *viz.* their neighbors. The PGR also needs information about its neighbors to successfully forward messages towards their intended recipients. This is achieved by a simple algorithm wherein each node periodically broadcasts a message requesting for neighbors to identify themselves. Identification is accomplished by replying to the requesting node

with their position coordinates. We model the position as a triple with *x*, *y* and *z* coordinates, thus enabling the protocol, and by extension routing, in a real-world three dimensional (*3-D*) system. This is necessary to accurately deploy the routing in real systems.

A simplistic neighborhood beaconing protocol may involve a node broadcasting its own position. Nodes receiving this broadcast would add the broadcasting node and its position to their neighborhood. As discussed in Chapter I, however, one of the intrinsic characteristics of radio networks is asymmetry. To recap, a node that can send a message to another node may not necessarily receive a message from that node and *vice-versa*. Thus, a neighborhood detection protocol that relies only on sending a node's own position, or advertising self, may not yield a neighborhood that will actually have nodes that can hear the node sending a message. This leaves the neighborhood in an incorrect state, due to the presence of nodes that cannot be reached.

For reasons explained above, a smart neighborhood beaconing protocol is needed. This protocol should ensure that any node that is added to the neighborhood is reachable. The reachability requirement is realized by the use of implicit symmetry constraints embedded in the protocol. Each node broadcasts an advertisement message, called *ADV_MSG*, at regular intervals, requesting information from its neighbors. The information consists of the node's position and optionally, the node's ID. Many WSN applications are more concerned with the location from where the information arrives, rather than the ID of the node sending the information. In such applications, storing a node's ID is unnecessary. Nonetheless, the node's ID may be required on platforms whose underlying infrastructure does not provide support for directing a message to a

location. On receipt of such a message, the receiving node checks to see whether it knows its own position. Though location awareness is resolved in WSNs, it may happen that a node may have been unable to determine its position. Nodes of this kind are not allowed to join the routing. A reply is issued only if node has knowledge regarding its position in the network. The reply message is a unicast to the requesting node. It may happen that a set of nodes do not desire to participate in the routing. Such nodes may choose to refuse to reply to the ADV_MSG.

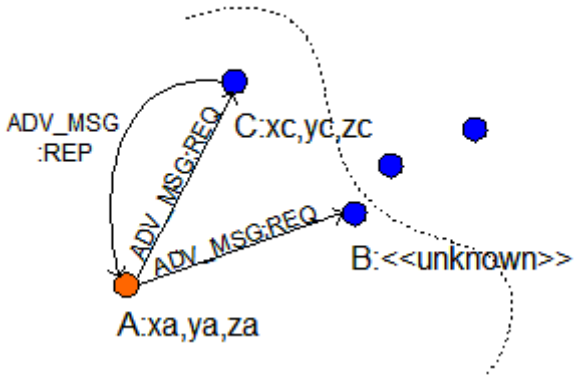

Figure 4 Broadcasting of ADV_MSG Requests

When a node receives a reply to its earlier ADV_MSG, it adds the replying node to its neighborhood. A successful entry into the neighborhood results in the neighboring node being assigned an initial weight, *W*. This weight is treated as an *age-factor* that is periodically reduced. Once the weight reduces to zero, the neighbor is perceived to be dead, out-of-range or incommunicado and is dropped from the neighborhood. ADV_MSG replies from a node already present in its neighborhood are discarded. Once a node is entered in the neighborhood, its age is decreased gradually. Thus, the algorithm stipulates that every node be dropped from the neighborhood after its lifetime is over.

Figure 4 shows the working of the protocol, detailing the process of advertising. Node A broadcasts the ADV_MSG. Nodes B and C hear the request. However, only Node C replies with its position. Node B cannot reply due to it being unaware of its position.

However, an obvious drawback to this schema is the protocol's inability to correctly reflect the current situation. Consider nodes 'A' and 'B' that are currently in the neighborhood of node 'X'. Once entries corresponding to 'A' and 'B' are made in the neighborhood of 'X', they would be aged periodically. Meanwhile, if 'A' dies or moves away, it still has the same weight relative to B's weight. This leads to the presence of outdated information in X's neighborhood. In applications using a fixed-size neighborhood, as most WSN applications are, this may also prevent legitimate and more current neighbors from joining the neighborhood.

Another shortcoming of this simplistic neighborhood beaconing protocol is related to a node's faithfulness. A node that is entered in the neighborhood table may refuse to forward messages relayed to it. Moreover, it may regularly reply to the ADV_MSG, thus ensuring its continued presence in the neighborhood.

A Revised Neighborhood Protocol

Problems highlighted in the previous section may be addressed by improving the simple beaconing protocol. As mentioned earlier, nodes periodically request for and receive positions from any neighboring node *via* the ADV_MSG. On receipt of the ADV_MSG from a neighbor, the node may check for it's presence in its neighborhood. Whereas the simple protocol accepted replies only from new nodes (nodes not currently present in the neighborhood), the new protocol processes replies from all nodes. If a node

already exists in the neighborhood, its current weight is increased by a scalar, thus lengthening its lifetime.

Another technique of reflecting the current topology more accurately allows the node to snoop the network, listening for routing messages, called *GEOROUTE_MSG*. Every time the sensor node relays or sends a message to its neighbors, it snoops on its neighbors to listen whether they are successful in forwarding the message. On recording a forwarded message, the node rewards the neighbor by increasing the weight. Figure 5 demonstrates this technique. Node A has relayed a message to Node C and is overhearing the network messages. It hears Node C forwarding the message to some node X, and increases the weight of Node C.

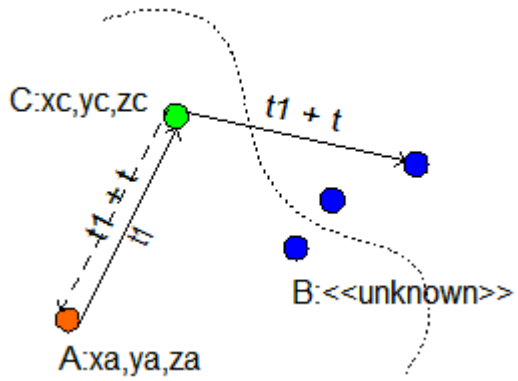

Figure 5 Network Snooping

The initial weight assigned to a neighbor as well as the subsequent increments may be pre-determined; or, they may be calculated at run-time depending on the density of the network or the load of messages passing through the node. For e.g., if a node is a part of a dense network, some nodes will be in a relatively small region. Thus, a node may age faster and be removed from the neighborhood. In this case, the initial weight and

increments may be small, so that the weight becomes zero as soon as possible. Conversely, in regions having a less dense population, each active neighbor is a precious resource. Hence, care must be taken to ensure that its lifetime in the neighborhood is as long as possible. Similarly, a node involved in relaying a high number of messages is utilizing more energy. It should not be forced to participate in further relaying. Thus, smaller weight and increment values may be assigned to it. The run-time determination of the neighbor's weight and its subsequent value in the neighborhood makes the protocol highly reactive to the current dynamics of the WSN. Pre-calculated scalars, based on empirical data, have been used to configure the neighborhood in the implementation.

The Neighborhood is a component that is used by the routing component. It operates in two phases; the first phase sets up the neighborhood graph for the node, whereas the second phase updates this graph continually. This ensures that any changes in the topology of the network are detected and eventually reflected in the neighborhood of the node.

```
Neighbor Structure:
struct Neighbor{
      unsigned integer NodeID;
      Position NodePosition;
      unsigned integer NodeWeight;
}

NeighborTable
list of Neighbor;
```

Figure 6 Neighbor Table

Figure 7 and Figure 8 below detail the algorithm for building and maintaining the neighborhood. Each entry in the neighbor table / neighborhood is a structure consisting of three elements as shown by the pseudo-code in Figure 6. It may, however, be noted that

applications not making use of a Node ID may choose not to use the corresponding field in the neighbor table.

```
BuildNeighborhood
Input:      NodeID, Position, TimerFrequency, ADV_MSG
Output:     NeighborTablePointer
Internal:   NeighborTable, InitialWeight, Increment, Decrement
Algorithm:
Neighborhood.Build(NodeID, Position)
{     Set Timer to every TimerFrequency secs;
      ADV_MSG.mode := REQ;
      ADV_MSG.node := NodeID;
      ADV_MSG.position := Position;
      return NeighborTablePointer;
}

Neighborhood.Timer.fired()
{     ADV_MSG.destination     := ALL;
      Neighborhood.Send(ADV_MSG);
      forAll Neighbors, n, in NeighborTable
      {     n.NodeWeight -= Decrement;
      }
}

Neighborhood.Send(ADV_MSG)
{     if(ADV_MSG.position != NULL)
            Send ADV_MSG to ADV_MSG.destination;
}

Neighborhood.Receive(ADV_MSG)
{     if(ADV_MSG.mode == REQ)
      {     ADV_MSG.mode            := REP;
            ADV_MSG.destination     := ADV_MSG.node;
            ADV_MSG.node            := NodeID;
            ADV_MSG.position        := Position;
            Neighborhood.Send();
      } elseif (ADV_MSG.mode == REP)
      {     if (ADV_MSG.node is present in NeighborTable)
            {     Get Neighbor, n, in NeighborTable;
                  n.NodePosition    := ADV_MSG.position;
                  n.Nodeweight += Increment;
            }
            elseif (NeighborTable is not full)
            {     Neighbor n.NodeID := ADV_MSG.node;
                  n.NodePosition    := ADV_MSG.position;
                  n.NodeWeight      := InitialWeight;
                  NeighborTable.insert(n);
            }
      }
}
```

Figure 7 Building the Neighborhood

The routing component initializes the Neighborhood component at the start. The Neighborhood component initiates the building of the neighborhood using the Neighborhood.Build(…), where it sets a timer to periodically broadcast the advertisement messages. The Neighbor.Receive(…) is an event that is fired whenever a ADV_MSG is received. It processes the message and either updates the Neighbor Table or replies to the ADV_MSG request. Messages are sent using the Neighborhood.Send(…) command.

```
MaintainNeighborhood
Input:      NeighborTablePointer
Internal:   RelayIncrement, RelayDecrement

Neighborhood.Maintain()
{     Send message, m, to neighbor, N;
      if ( N acknowledges m)
            NeighborTablePtr->N.NodeWeight += RelayIncrement;
      else
            NeighborTablePtr->N.NodeWeight += RelayDecrement;
}
```

Figure 8 Maintaining the Neighborhood

The Neighborhood Maintenance is performed by snooping over messages in the network. Whenever the routing component transmits a message, it waits for an acknowledgement. On receipt of the acknowledgement, the routing component invokes the Neighhborhood.Maintain(…) command to update the weight of the node to which the message was relayed. It can be seen that neighborhood maintenance is carried out at two levels; the first consists of a reply to periodic requests for node positions, while the second allows the routing component to interact with the neighborhood component

Basic Geographical Routing

This section presents the greedy geographical routing that was also the basic component in GPSR [22][23]. It is based on the assumption that the node knows the geographical position of the destination node. This approach to routing involves relaying the message to one of its neighbors that is geographically closest to the destination node of all the neighbors, and is geographically closer to the destination than the node itself. This approach attempts to find a short path to the destination, in terms of either distance or the number of hops. This is based on the geographical distances between the nodes in the graph (network).

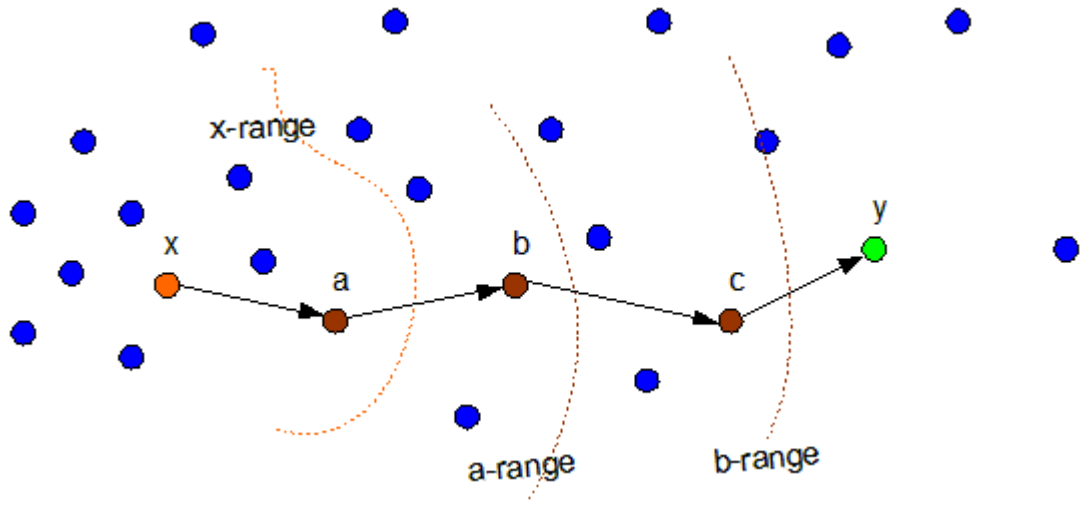

Figure 9 Sample Route in Basic Geographical Routing

A node that requires sending a message acquires the address of the destination. The address is in the form 3-D coordinates of the physical location of the destination. The coordinate system is presumed to be determined by the application. After preparing the message, it calculates the Euclidean distance from self to the destination. Next, it calculates the Euclidean distance from each of its neighbors to the destination. The

greedy approach tries to always shorten the distance to be traveled to the destination to the maximum possible extent. Therefore, the node considers only those neighbors that are closer to the destination than itself. The sending node then chooses the node closest to the destination and relays the message onto the neighbor.

```
BasicGeographicRouting
Input:      MSG, NeighborTablePtr, Counter
Internal:   Position, EuclideanDistance(a,b), Sort(…)

BGR.Send(MSG, Counter)
{     if(MSG.destination != ALL)
      {     dist := EuclideanDistance(MSG.destination, Position);
            Neighbor *n := NeighborTablePtr->head;
            i := 0;
            while(n && n->next != NULL)
            {     dists[i] := EuclideanDistance( n->NodePostion, …
                              MSG.destination);
                  n = n->next;
            }
            sort_dist = Sort(dists, ASCENDING);
            if( sort_dist[Counter] > dist )
                  exit;
            else
                  MSG.destination := node corresponding to
                        sort_dist[Counter];
      }
      Send MSG;         // Uses the OS-provided API
}

BGR.Receive(MSG)
{     if( MSG.destination == myself)
      {     MSG.destination := ALL;
            MSG.mode := ACK;
      }
      BGR.Send(MSG, 0);
}
```

Figure 10 Basic Geographic Routing

A node receiving a message may either be the final destination, or it may be one of the intermediate nodes on the route to the destination. On receipt of a message, a node checks whether it is the final destination. If it is, it broadcasts an acknowledgement to signal the receipt of the message to the last relaying node. The other nodes in the vicinity

ignore this signal. If the node is an intermediate hop to the message being relayed, the node will calculate the next hop of the message in the manner described above.

A sample topology is shown in Figure 9. Nodes *x* and *y* are the "sender" and "receiver" respectively. Node *x* sends the message to node, *a*, which is the closest of its neighbors to the destination node. On receiving the message, *a* calculates *b* to be the next closest neighbor and relays the message to *b*. Node *b* in turn relays the message to *c*, which in turn relays it to the intended recipient of the message, *y*.

The Basic Geographic Routing (BGR) does not use any data structures stored locally on a node apart from the Neighbor Table. Thus, no information is stored locally. An algorithm for this basic geographical routing is shown in Figure 10.

The sending component does not differentiate between the source of the message and an intermediate node on its route. The receiving component needs to handle to two different types of messages; one that says that the node is the destination, and the other that specifies the node to be an intermediate node for relaying the message. Both messages are handled in exactly the same way, without any form of distinction.

Problem Topology

A void in the network is an area where there are no nodes that can participate in communication. A void may be present in the system at deployment itself. For e.g., sensor nodes are placed in an area which also consists of a water body. A void may also be created due to noise, interference or merely due to low-energy on the node, once the network has been deployed and is in-use. It is obvious that voids in the network will lead to the failure of the basic geographical routing. Figure 1, in Chapter I, shows an instance of where a node marked 'S' will not be able to route the message to the destination

marked ‘D’ due to the presence of a void in the network, along the route. This void is shown by the circle. Moreover, given the unstable nature of links in radio communications, even a densely deployed network over time will create topologies unsuitable for the routing described above.

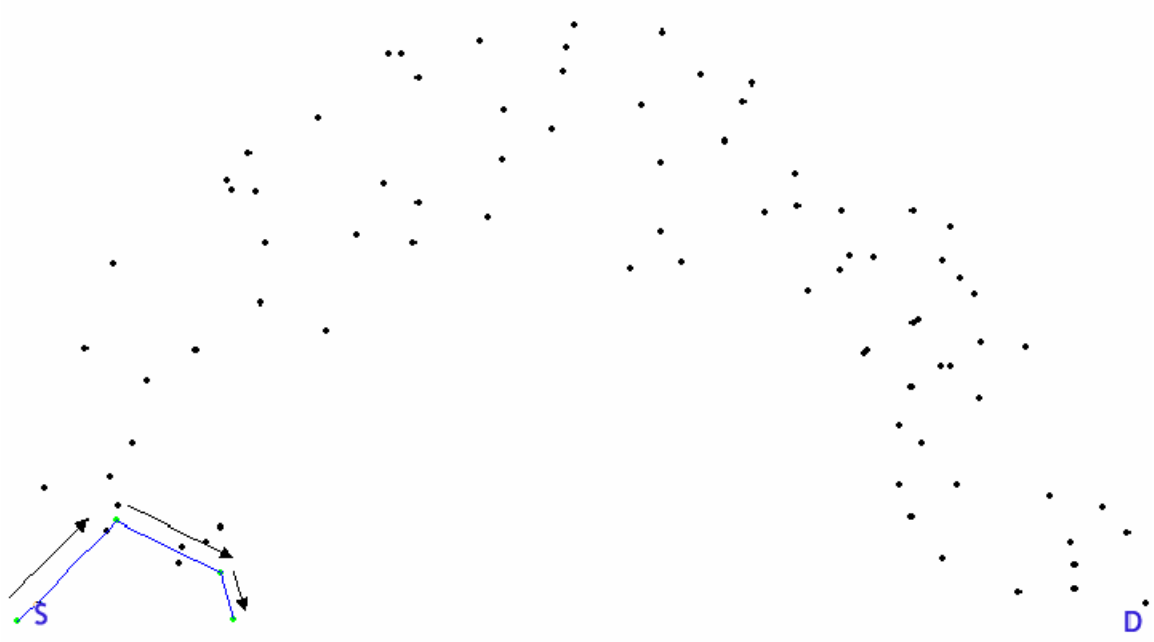

Figure 11 Example of Topology leading to failure of BGR

The general premise on which the above algorithm operates is that a message can only be relayed in the forward direction, i.e. the distance to the destination can only be shortened. However, this policy fails in situations covered by some generic topologies. In this section, we introduce topologies that motivated us to find a solution to robustly deliver high-priority messages.

We present an example network topology that results in the failure of the basic geographic routing. It may be noted that though the topology presented below is in a two-dimensional frame, the problems it illustrates can be extended to three-dimensional

layouts as well. The example topology, presented in Figure 11, is in the form of a curve. This layout generally describes scenarios wherein there is a path to the destination, but the path does not always shorten the distance to the destination. In this situation, the sender and the receiver were positioned at the two extreme ends of the curve. The basic algorithm tries route the message towards the destination, always reducing the remaining distance. But, eventually, it encounters a void, leading to the failure of the algorithm. Depending on the depth of the curve and the distance between the two ends, the message travels till a point from where the message needs to traverse to a node such that the distance to the destination is not the shortest possible distance. However, a path does exist to the destination whose length is much greater than the distance between the source and the destination. The basic algorithm is incapable of handling such topologies.

Modified Geographic Routing

The previous section demonstrated the need for a uniform layout to find an increasingly shortest path, based on the basic geographic algorithm. In the absence of a uniform layout, finding such a path to the destination invariably failed. We define the uniform layout as a random deployment of nodes distributed uniformly over an area; i.e. the density of nodes in any given area is close or approximately same to the density in any other area. Sometimes, it may be more beneficial to take a temporary detour and go farther away from the destination in order to eventually approach and reach it [17]. This problem is encountered when, due to the loss of communication or presence of geographical constraints, voids are created in the radio network. Thus, in the process of relaying the message, it is possible to reach a node that is located on the periphery of these blanks spaces. In such cases, it is generally impossible to relay a message across

onto the opposite bank of nodes, unless the message is relayed with higher power consumption. The distance to which a message can be transmitted, or the range of the node, increases monotonously with the power consumed in transmitting that message [7]. Thus, increasing the power utilized to transmit a message leads to an increasing its signal strength, and eventually the distance at which it can be received. However, using more power in energy-constrained environments is inadvisable as well as detrimental. Regardless of the extra power consumption, a successful message transmission is not guaranteed in situations where the blank space (void) is large enough to thwart a communication.

Hence, it is desirable to have a solution that can handle the topologies covered previously. One way of doing it is using *Perimeter Routing* as in GPSR [22][23]. We propose two incremental solutions that fit-in with our need for prioritized routing, Modified Geographic Routing (MGR) and Advanced Geographic Routing (AGR). The AGR is built incrementally on top of the MGR, which itself is based on the BGR.

The Basic Geographic Routing tries to find neighbors closest to the destination at each node. When the next hop on the route is not available, due to a variety of reasons discussed earlier, the routing fails and the message is discarded / lost. A more robust approach involves the use of *snooping* as covered in the Neighborhood Building section. Whenever a message needs to be relayed, the algorithm proceeds in the same way as earlier. The node calculates its own distance from the destination node. It then calculates the distances of all its neighbors from the destination. Using the earlier condition, it chooses the neighbor that is closest to the destination, and closer to the destination than itself. It then relays the message to the neighboring node.

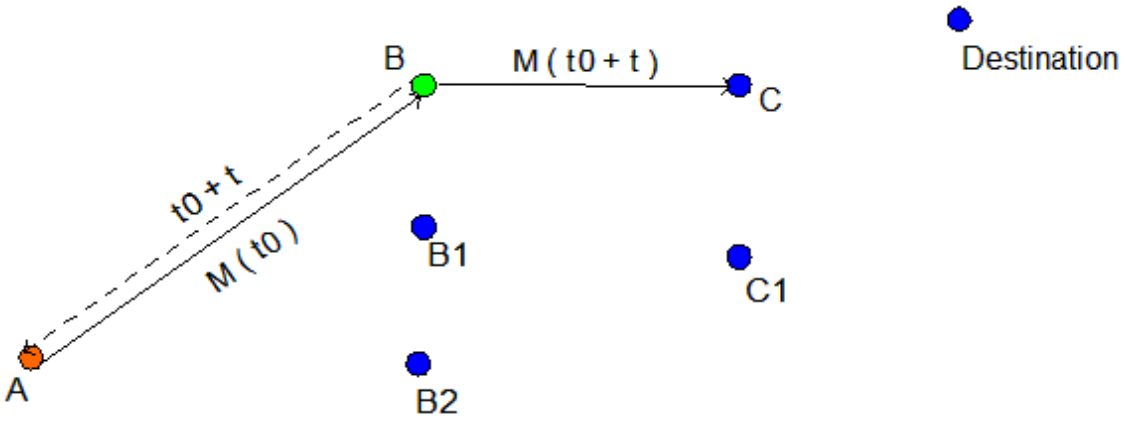

Figure 12 Implicit Acknowledgement in MGR

However, instead of discarding the message as done earlier, the node stores a reference to the message in the "OutMessages" table. This table is a list of messages that have been forwarded and awaiting acknowledgements. It then starts snooping the network, and waits for the relayed message to be acknowledged. The acknowledgement is either explicit or implicit. The receiving node may be the intended recipient (final destination) in which case the acknowledgement will be an explicit one, represented by a broadcast announcing the arrival of the message. Otherwise, if the receiving node was just another intermediate hop intended to relay the message, the acknowledgement will be implicit in nature. The node in its snooping mode will wait to overhear the further transmission of the message by the next node. This implicitly requires the radio communication to be symmetric. However, lack of symmetry merely leads to the spawning of another copy of the message, as described later.

Figure 12 shows an example for implicit acknowledgement. If node A sends a message with Message ID M to node B at time 't_0', the node A will wait to receive either an explicit acknowledgement from B, or an implicit one. The implicit acknowledgement

will be in the form of an overheard transmission of message M by node B to some other node, say node C, at time '$t_0 + t$'.

```
RelayedMessage Structure:
struct RelayedMessage{
      integer (or string) MessageID;
      integer BestNeighborCount;
      enumeration MessageMode;
}


RelayedMessages Table:
array 1..X of RelayedMessage;
```

Figure 13 "Relayed Messages" Table

As discussed above, the transmitting node stores a reference to the message and awaits an acknowledgement after transmitting the message. Once it receives an acknowledgement, the transmitting node discards the message. However, it stores the unique message ID and the "BestNeighborCount" in a "Relayed Messages" Table [Figure 13]. The BestNeighborCount is used to keep track of the next-best neighbor available to forward the message. This table also contains a field for "Message Mode" which is discussed in detail later.

The node discards the message from the OutMessages table once an acknowledgement is received. If, within a certain time-frame, no acknowledgement is received, the message is presumed to be lost. In such a case, the transmitting node, A in the example in Figure 12, chooses the next best neighbor from its neighborhood that satisfies the same constraints *viz.* the neighbor is closer to the destination than all other neighbors, and closer to the destination than the sender itself. The transmitting node then relays the message to the newly chosen hop / neighbor. This process is repeated till the node either receives an acknowledgement, or runs out of neighbors that are closer to the

destination than itself. If the node does not have any more forwarding neighbors, it will send a Failure Message back to node from whom it received the message. The failure message is nothing but the original message that was received by the node sending the failure message, with a failure-to-forward flag attached. On receipt of a failure message, the node looks up its RelayedMessages table and gets the last best neighbor. It then tries to find next-best neighbor, relative to the last best neighbor, as described above.

As long as the node is trying to relay the message to its next-best neighbors, it will store the MessageMode in the RelayedMessages table to “Advance”. On failing to forward it to any of its next-best neighbors, the node sends the failure message. At this point, it marks the message with a “Fail” mode.

Since the WSN nodes are resource-constrained, the entries in the Relayed Messages table need to have some lifetime defined after which they are discarded. The MGR removes entries from this fixed-size table either when they are timed out or on a successful delivery of a message that is fresher than the entry in RelayedMessages table. This is determined by the MessageID, which is a combination of the NodeID and a sequence number. If the RelayedMessages table is full, and there is no entry that may be replaced, the oldest entry in the table is removed.

This enhancement to the algorithm ensures that nodes do not drop packets till they have searched for as many forwarding paths as possible, while trying to always move forward. This ensures that the resulting path, though not being formed by choosing the closest neighbor-to-destination possible, is still the most direct route towards the destination. It is obvious here that there are no loop-backs. At no point in time is the message relayed to a node that is farther to the destination. Thus, the only way a message

can be farther to the destination than it was earlier in time is by the transmission of failure messages. No node receiving a failure message will relay the message farther back unless it is a failure message. However, it is not guaranteed that the message wont fork on its way to the destination. Due to the dynamicity of the connectivity, asymmetrical links may crop up in the radio network after the neighborhood has been built. This invalidates the implicit symmetry requirement between a node and its neighbor. Hence, it is probable that a message that is relayed further by a neighbor is not "heard" by the transmitter. In such a scenario, the transmitter may fork another copy of the message along another route, leading to the presence of multiple copies in the network. Nodes that are further down the route and receiving multiple copies may resend the same message. Possibilities of such occurrences are mitigated by use of unique Message IDs. However, such possibilities may not be entirely prevented. If the Message ID has timed out or been removed from Relayed Messages table, the node has no way of knowing whether it has already relayed the message earlier. Moreover, a destination node may still end up with multiple copies of the message if the two forked routes are entirely disjoint.

In the example shown in Figure 12, A transmits M to B at t_0. Say, B does not acknowledge the receipt of the message. A then finds the next-best neighbor B_1 and relays the message to it. In case of even B_1 not acknowledging, A transmits M to B_2, which is the next-best neighbor. B_2 then may transmit to C_1. If A cannot hear any acknowledgements, it tries to find the next-best neighbors. If it cannot find any, A sends a failure message back to the node from which it received the message M.

```
ModifiedGeographicRouting
Input:      MSG, NeighborTablePtr
Internal:   Position, EuclideanDistance(a,b), Sort(…),cnt
```

```
MGR.Send(MSG, cnt)
{     if(MSG.destination != ALL)
      {
            dist := EuclideanDistance(MSG.destination, Position);
            Neighbor *n := NeighborTablePtr->head;
            i := 0;
            while(n && n->next != NULL)
            {     dists[i] := EuclideanDistance( n->NodePostion, …
                              MSG.destination);
                  n = n->next;
            }
            sort_dist = Sort(dists, ASCENDING);
            MSG.destination := node corresponding to
                        sort_dist[cnt];
            do
            {     Send MSG;         // Uses the OS-provided API
                  OutMessages.Insert(MSG); //ignore if present
                  RelayedMessage r.MessageID    := MSG.messageID;
                  r.BestNeighborCount           := cnt;
                  r.MessageMode                 := MSG.Mode;
                  RelayedMessagesTable.UpdateInsert(r);
                  cnt                           := cnt + 1;
            }while (ACK not received || No Forwarding Neighbors);

            if(ACK received)
            {     OutMessages.Delete(MSG);
            }
            else        // Sends Failure Message
            {     MSG.destination         := MSG.last_transmitter;
                  MSG.Mode                := FAIL;
                  Send MSG;         // Uses the OS-provided API
            }
      }
      else  // Message ACK by destination
            Send MSG;
}


MGR.Receive(MSG)
{     cnt   := 0;
      mode  := NORMAL;
      if( MSG.destination == myself)
      {     MSG.destination := ALL;
            MSG.mode := ACK;
      }
      elseif( MSG.Mode == FAIL )
      {     RelayedMessage r  :=
                        RelayedMessagesTable.Retrieve(MSG.MessageID);
            cnt               := r.BestNeighborCount;
            MSG.mode          := NORMAL;
      }
      MGR.Send(MSG, cnt);
}
```

Figure 14 Modified Geographic Routing

The MGR is given by the algorithm in Figure 14. This approach will easily overcome such network voids as those demonstrated by Figure 1 and Figure 11.

Problem Topology

However, there would still be situations wherein even the MGR fails. An example topology with which this failure was most frequent is the one shown in Figure 15. This layout has two or more paths to the destination. However, one of the paths, generally the shortest path, leads to a void. The nodes on this path do not have any other immediate neighbors that can route to the destination. Any protocol, including the BGR and MGR presented above, that takes this path and does not provide any facility for backtracking is doomed to fail. The deceptive nature of the topology was devised to comprehensively test a routing algorithm's ability to deliver a message. The layout is deceptive in the sense that it leads the algorithm to believe that there is a short path to the destination. However, this proves to be deceptive and the routing encounters a void. In this topology, even backtracking may not work. As MGR always tries to relay the message to a node that is closer to the destination, messages may need to backtrack till the source. However, a node that is closer to the destination than the current node may still not be found. Moreover, a failure message sent back may be lost, leading to a failure in backtracking.

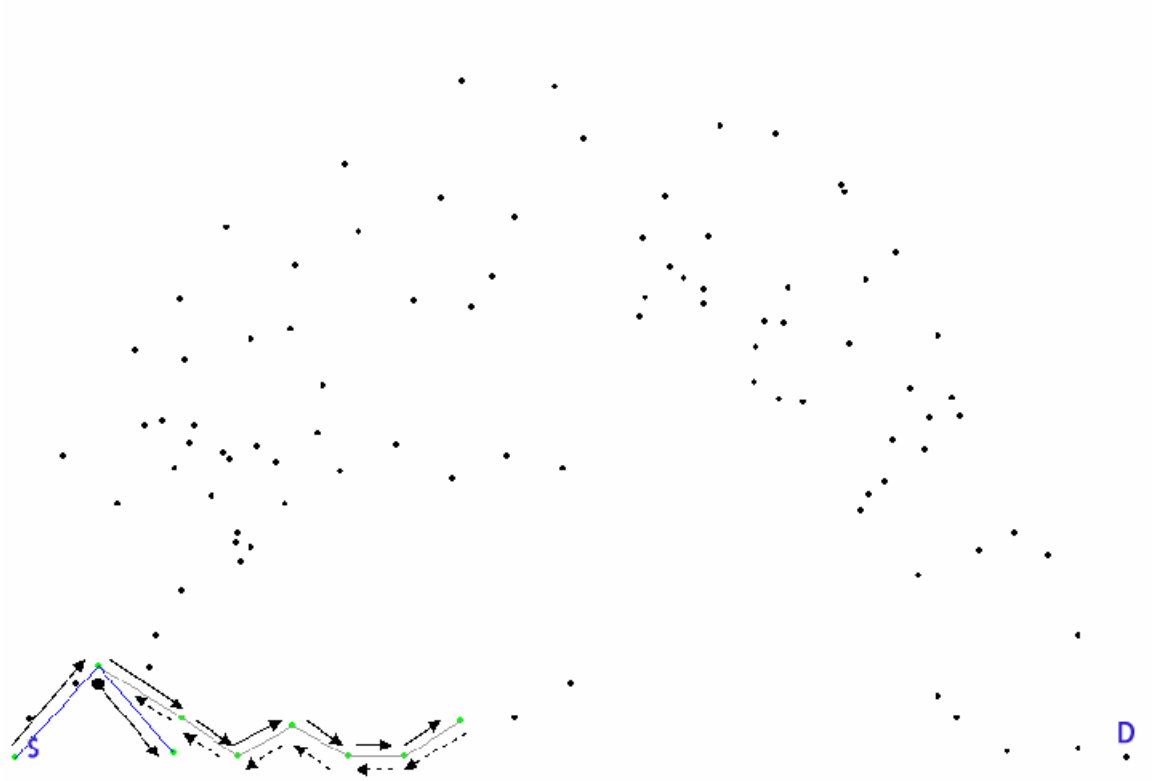

Figure 15 Example of Topology leading to potential failure of MGR

MGR's success in this topology depends on the nodes near the periphery of the void. The successive failure messages will get the message back to the node that did have alternate path. If the nodes along the periphery of the void are closer to the destination than the node currently with the message, the MGR will succeed. However, if there is no such node, then MGR may backtrack till the source and fail. This behavior may be alleviated by sending a message backwards, i.e. to a neighbor that is farthest from the destination as well as self. Figure 15 shows an example, wherein the message traverses the deceptive route. This is shown by the line-arrows. Lack of forwarding neighbors causes successive nodes on the deceptive route to send back failure messages. These are shown by the dotted arrows. The second node (immediately after 'S') sends the message to another node. This node sends a failure message which is lost.

Advanced Geographic Routing

The MGR tries to relay the message to nodes that are closer to the destination than itself. In doing this, it chooses nodes that are progressively farther than the destination i.e. it chooses neighboring nodes between itself and the destination, such that first choice is the closest neighbor, and the subsequent choices, if any, farther from the destination than the earlier choices. Yet, choosing these nodes would result in a path that leads the message to a node that is closer to the destination than any of the nodes traversed on the route (discounting the nodes that have resulted in failure messages and backtracking). However, the routing tends to fail when it is unable to send a failure message. This may occur due to either the node trying to send the failure message being the source, or the node being unable to hear the implicit acknowledgement. In such a scenario, it is necessary to travel backwards, i.e. to a node that is farther than the current node to the destination. The layout in Figure 15 is a very good example of a layout where even the Modified Geographical Routing will eventually fail. However, this failure cannot be predicted with surety. It is merely an assessment based on simulations, observed due to the probabilistic connections in the network.

High priority messages need to reach the intended recipient. They may contain data that is very important in the context of the WSN application deployed. Hence, we introduce the concept of *temporary backward traversal* to the Modified Routing Algorithm, and name the resultant algorithm as the Advanced Geographical Routing.

This approach involves waiting for the acknowledgements of the relayed messages. Once a message is relayed to the node on the next hop, the transmitter waits to hear an acknowledgement as explained in Modified Geographic Routing. The algorithm

proceeds as in MGR till a failure message is sent. If the transmitting node does not receive any acknowledgement to this failure message, it tries to find nodes that are farther to the destination than itself. It chooses the neighbor that is farthest to the destination than itself. The message is then relayed to the chosen node which then tries to forward the message using the same algorithm. This approach tries to ensure that the new set of neighbors that the next-hop node will consider to relay the message will be significantly different from the current set of neighbors. We adopt this policy as the transmitting node would have already considered all the nodes that it can forward a message to. Relaying a message to a neighbor which is just farther than itself to the destination, in effect very close to the transmitter, would result in a new set of neighbors that is almost equal to the current set of neighbors. This would lead to an unnecessary overhead of relaying the message to the same nodes that had been unable to forward the message earlier. An approach of transmitting to the farthest possible node from self leads to a set of new neighbors that is noticeably different from the current set. But, the difference in the set of neighbors depends on the location of the node with respect the line joining the transmitter and the destination. If the node is closer to this line, then the neighbor set may not be much different. However, if the node is further away from the line, the set of neighbors differ by about 45%.

The AGR makes use of the RelayedMessages table, Figure 13, to keep track of the mode of the message. Once the "Fail" flag is set in the MessageMode, it waits for an acknowledgement to the failure message. If it does not receive the acknowledgement, it changes the mode to "Backward" and then transmits a message to the farthest neighbor. If a node exhausts all possible neighbors to whom it can relay a message, it marks the

message with a "Permanent Failure". Any incoming message with this MessageID is not processed further, but dropped. Figure 16 shows the detailed algorithm for the Advanced Geographical Routing.

```
AdvancedGeographicRouting
Input:      MSG, NeighborTablePtr, Mode
Internal:   Position, EuclideanDistance(a,b), Sort(…),cnt

AGR.Send(MSG, cnt, Mode)
{     if(MSG.destination != ALL)
      {
            dist := EuclideanDistance(MSG.destination, Position);
            Neighbor *n := NeighborTablePtr->head;
            i := 0;
            while(n && n->next != NULL)
            {     dists[i] := EuclideanDistance( n->NodePostion, …
                              MSG.destination);
                  n = n->next;
            }
            if(Mode == ADV)
                  sort_dist = Sort(dists, ASCENDING);
            else
                  sort_dist = Sort(dists, DESCENDING);

            MSG.destination := node corresponding to
                        sort_dist[cnt];
            do
            {     Send MSG;         // Uses the OS-provided API
                  OutMessages.Insert(MSG); //ignore if present
                  RelayedMessage r.MessageID    := MSG.messageID;
                  r.BestNeighborCount           := cnt;
                  r.MessageMode                 := MSG.Mode;
                  RelayedMessagesTable.UpdateInsert(r);
                  cnt                           := cnt + 1;
            }while (ACK not received || No Forwarding Neighbors);

            if(ACK received)
            {     OutMessages.Delete(MSG);
            }
            else        // Sends Failure Message
            {     MSG.destination   := MSG.last_transmitter;
                  if (MODE == ADV)
                  {     MSG.Mode          := FAIL;
                        Send MSG;   // Uses the OS-provided API
                        if (NO ACK received)
                        {     MSG.destination := node corresponding to
                                    sort_dist[last_entry];
                              post task to call AGR.Send(MSG, 0, BACK);
                        }
                  }
                  else  // Failed to send a Backward Message
                  {     RelayedMessage r :=
                        RelayedMessagesTable.Retrieve(MSG.MessageID);
                        r.MessageMode := PERMA_FAIL;
                  }
            }
      }
```

```
      else  // Message ACK by destination
            Send MSG;
}



AGR.Receive(MSG)
{     cnt   := 0;
      mode  := ADV;
      if( MSG.destination == myself)
      {     MSG.destination := ALL;
            MSG.mode := ACK;
      }
      elseif( MSG.Mode == FAIL )
      {     RelayedMessage r  :=
                        RelayedMessagesTable.Retrieve(MSG.MessageID);
            cnt               := r.BestNeighborCount;
            MSG.mode          := NORMAL;
      }
      AGR.Send(MSG, cnt, mode);
}
```

Figure 16 Advanced Geographic Routing

This algorithm differs from the earlier ones in that it does not always try to find a path that reduces the distance to the destination. The advanced geographical routing works in two phases. In the first phase, it is exactly the same as the modified routing presented above. The second phase starts only when there is a failure in delivering the message during the first phase, which may be due to asymmetric links formed over time. This phase involves choosing a route which results in a temporary diversion. This diversion leads to a hop in the path that lengthens the distance to the destination. This lengthening of the distance is so chosen to increase the distance to the maximum possible extent. The transmitting node finds the farthest node to the destination from its neighborhood and relays the message to that neighbor.

Prioritized Geographical Routing

Using the three techniques / variations described above, we present a priority based any-node-to-any-node geographical routing algorithm call the Prioritized Geographical Routing (PGR).

PGR supports three priority levels – Low, Medium and High. PGR is a configurable routing algorithm that enables transmission of messages from any node in the ad-hoc network to any other node, provided the geographic position of the destination is known.

Whenever a node needs to transmit a message to some other node in the network, it invokes the PGR. It gets the destination's position, and then based on the application, decides the priority of the message from any of Low, Medium and High. Based on the priority chosen, PGR uses the appropriate routing from the three techniques described above. For sending a message with *low* priority, PGR uses the *Basic Geographical Routing*. This ensures that low priority messages do not clog up the network, trying to reach their intended destination, and tie-up the resources on the intermediate nodes. Whenever there is a need to send a message with a higher priority, but it is not entirely necessary for the delivery of the message, the nodes may send the message with a *Medium* priority. In this case, PGR uses the *Modified Geographic Routing*. However, as demonstrated in Chapter I, there would be cases wherein messages need to reach the intended recipient if at all a path exist from the originating node to the destination node. Such a constraint calls for the use of the *Advanced Geographic Routing* which the PGR deploys for messages with a priority level of *High*.

The PGR flags the messages using two types: Normal and Failure, as necessitated by the three geographic routing techniques covered yet. A *normal* message, as the name suggests, is a message that carries an important payload and is relayed from a node to another. This message is intended to carry the data to the destination. A *failure* message is one that is sent by a node saying that it could not relay the message ahead. This message is received by the node that had earlier forwarded the message to sender of the failure message.

```
Message

Structure:
struct PGR_Message{
      integer (or string) MessageID;
      boolean MessageType;
      Position (or integer) NextNode;
      Position LastNode;
      Position Destination;
      enumeration Priority;
      integer data[];
}
```

Figure 17 Message Structure

The structure of the messages used in PGR is given by Figure 17. MessageID is a uniquely generated ID based on the source of the message. MessageType is a Boolean value specifying whether a message is in “Normal” mode or “Failure” mode. The NextNode contains the node address to which a message is being relayed. The LastNode field is used to keep track of the node from whom the message arrived. Destination, as the name suggests, is the final destination of the message. Priority is an enumerated field indicating the low, medium or high priority of the message.

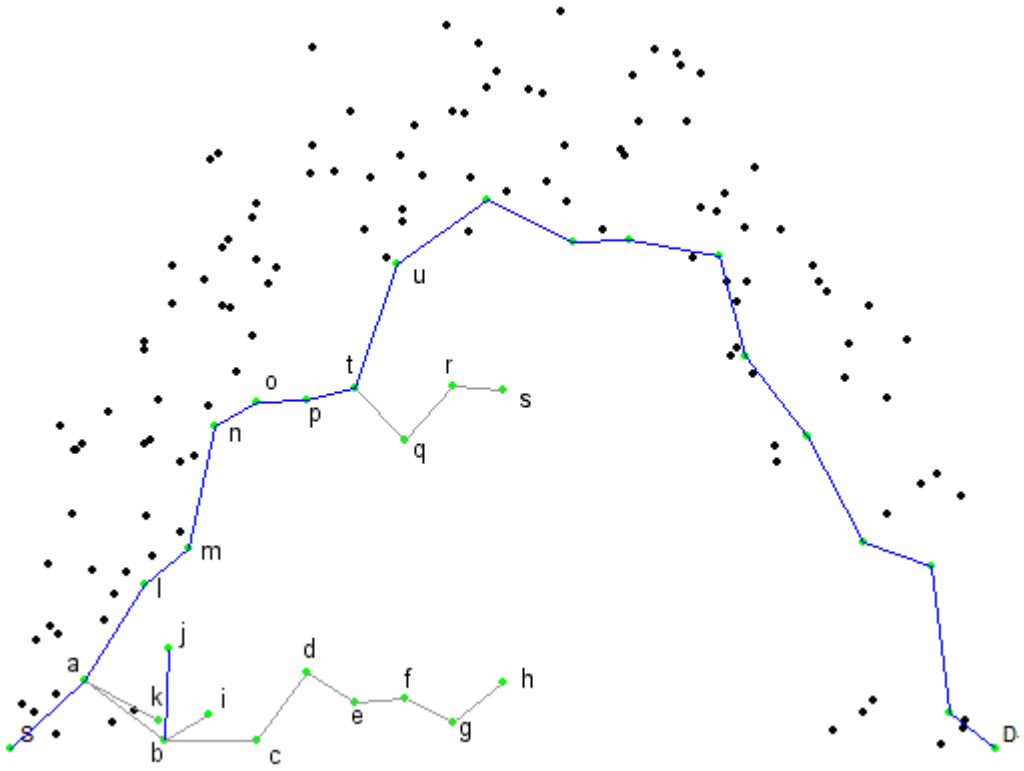

Figure 18 Message Transmission using PGR

Figure 18 shows an example of a successful transmission. This example shows the delivery of the message using a High Priority scheme in a layout that is deceptive in nature. It further describes the PGR in detail. Here, the sender / message-originator is the node labeled "**S**", while the node labeled "**D**" is the intended recipient / end destination. The example walks through the working of PGR, using each of the three priority levels.

In the first case, the message was sent with a Low priority. The path that was followed by the message was: **S** → a → b → c → d → e → f → g → h. Once the message reaches node "h", it is unable to find any forwarding nodes. Hence, it drops the message.

Now, consider the case of the message being transmitted with a Medium priority. In this case, the message follows the same path as earlier, and reaches node “h”. On being unable to find any forwarding neighbors, it sends a failure message back to node “g”. Node “g” tries to forward it using alternate paths, and fails again. So, it sends a failure message back to node “f”. This pattern is followed till the failure messages reach back to node “b”. Disregard node “j” for the moment. Node “b” then forwards the message to the next-best node “i”, which again sends a failure message back to node “b”. Note that, “i” will be unable to forward the message to the other nodes “c”, “d” etc as they have already failed for this particular destination, and would not accept the message for transmission. “b” sends a failure message to “a”, which finally sends a successful relay to node “l”. The message is relayed till node “s”, wherein it is again blocked from progressing further towards the destination. Nodes “s”, “r”, “q” and “t” all send back failure messages to “r”, “q”, “t” and “p” respectively. However, “p” is unable to receive the failure message from “t”, leading to the failing of the routing.

The third case pertains to the message transmission with priority set to high. In this scenario, the message is routed like in the earlier two priority levels. However, when “b” tries to send a failure message to “a”, it does not receive any acknowledgement; implicit or explicit. As a result, “b” relays the message using the Advanced Geographical Routing, to its neighboring node “j” that is farther than itself from the destination “D”. However, “j” is unable to relay it further. A similar situation arises later in the route, when “t” sends a failure message back to “p”. Not receiving any acknowledgment from “p”, “t” finds its farthest neighbor, “u”, from “D” and sends the message to “u”. “u” is then able to successfully relay the message further until it reaches “D”.

The PGR is given in Figure 19. It uses the three components – BGR, MGR and AGR – to route messages using the priority specified.

```
PrioritizedGeographicRouting
Input:      MSG, Destination, Priority, NeighborTable

Algorithm:
PGR.Send(MSG)
{     MSG.Priority      := priority;
      switch (Priority)
      {     case LOW:         Use BGR;
            case MEDIUM:      Use MGR;
            case HIGH:        Use AGR;
      }
}

PGR.Receive(MSG)
{     if( MSG.destination == myself)
      {     MSG.destination := ALL;
            MSG.mode := ACK;
            Send MSG;
      }
      elseif( MSG.Mode == FAIL )
      {     RelayedMessage r  :=
                        RelayedMessagesTable.Retrieve(MSG.MessageID);
            cnt               := r.BestNeighborCount;
            MSG.mode          := NORMAL;
            if(MSG.Priority == HIGH)
                  AGR.Send(MSG, cnt, mode);
            else
                  MGR.Send(MSG, cnt);
      }

}
```

Figure 19 Prioritized Geographic Routing

CHAPTER IV

ANALYSIS, EVALUATION AND EXPERIMENTATION

The PGR provides a priority-based facility to transmit messages and an 'any-node-to-any-node' solution for WSN applications requiring in-network data processing. This chapter provides details of the algorithms implementation, the analysis and results of experiments.

Implementation

A WSN consists of anywhere between hundreds to thousands of nodes. Usually, these nodes are situated in a relatively small area, leading to a conceivably dense population of nodes. This may in turn lead to each node having neighbors on the order of ten (10). However, the nodes are resource-constrained devices with low memory. Hence, it is imperative to have a restriction on the number of neighbors that a node can have. Depending on the WSN application, a restriction may also be placed regarding the proximity of the neighbors to the node itself. For instance, an application collecting data and routing it back to a base-station may have a restriction that each node should have neighbors that are least a fixed distance away from node. Conversely, an in-network data processing application may restrict the neighbors to be at most a fixed distance away.

Nonetheless, irrespective of such trivial restrictions, the neighbor table should be optimized to hold a fixed number of nodes. A good parameter for deciding the size is density of deployment of the WSN. If the density is known beforehand, the fixed size of the neighborhood table is easily calculable. Based on simulation statistics, a neighbor

table of size equaling approximately 0.05% - 0.1% of the density (number of nodes / km^2) seems to yield a fairly satisfactory neighborhood.

The PGR is essentially greedy in finding a path to the destination. The essence of the entire process is to continually calculate Euclidean distances between two nodes. This process is repeated at every node, and is directly proportional to the number of neighbors a node has. However, the square function is monotonic and hence, the calculation of euclidean distances by square root is unnecessary. The node can cache the sorted list of its neighbors that satisfy the closeness constraint till it receives an acknowledgement of the message transmission. Calculation of Euclidean distances involves multiplications and square root, a fairly high CPU-consuming process. Hence, it is crucial to use an alternate way to perform these calculations. A fairly accurate method for calculating the integer square root by shift-operations is shown in [51] by D. J. Evans for advanced calculations involving gaming applications. This results in an integer square root with less than 5% error, and is sufficient for calculations for finding the next relay node.

Resolving the neighbor closest to the destination entails keeping track of the smallest distance to the destination and the corresponding node. This can be implemented easily using simple algorithms. However, these calculations need to be done every time an acknowledgement is not received and the node needs to forward the message again to the next-best neighbor. Similar calculations are also triggered by the arrival of the failure message. This may lead to a lot of time being devoted to finding the next neighbor. This scheme is suitable for applications where messages are processed periodically and occasionally.

An alternative is to cache the nodes in a sorted list. On the non-arrival of an acknowledgement, the node needs to merely lookup the BestNeighborCount in the Relayed Messages Table, and pick up the next best neighbor from this list. Once an implicit acknowledgement is received, this sorted list may be destroyed. In this case, the sorting operation is required only twice, once when the message arrives in the normal mode, and once when the message arrives back in failure mode. The problem with this scheme is that space proportional to the neighbor table needs to be allotted for every message or at the very least every destination, being currently processed by the node. This leads to an exponential rise in the amount of space required based on the number of messages being processed simultaneously by a node. This scheme is suitable for applications that do not generate a lot of traffic at once. It may also be used in applications wherein the primary task of the nodes is routing of data to the base-station.

In case sorting needs to be performed for the above schema, various sorting algorithms may be used. Conventional sorting algorithms like Quick Sort, Merge Sort etc provide a performance complexity of O (n*(log n)). However, their performance for sorting a few numbers is not satisfactory. In fact, Bubble Sort, with a complexity of O (n^2), is better-suited for this. The implementation of PGR uses Comb Sort [52], a variation of Bubble Sort that is an improvement over Bubble Sort without sacrificing Bubble Sort's simplicity.

PGR Message Overhead

The structure of a PGR message is given in Figure 17 . Position contains three fields, each of 16 bits. The MessageID is a 23 bit field. It can be combined with the 1-bit field of MessageType, leading to a consumption of 24-bits. The Priority field is 2 bits in

length. A WSN operating system would provide facility for transmitting a message to a particular location. Thus, the overhead of actual routing is restricted to the MessageID, MessageType, LastNode, Destination and Priority. On the TinyOS platform, PGR requires 11 bytes in a packet length of 29 bytes. The underlying message transmission in TinyOS is restricted to either a broadcast or a node ID. Thus, we are able to avail the LastNode and NextNode fields as Node IDs (2 bytes) instead of a Position (6 bytes).

Comparison via Simulation

The PGR is compared with the Directed Flood Routing Framework (DFRF) [53]. The simulation environment used for the comparison is the MATLAB-based Prowler [40]. Prowler is a discrete-event based simulation tool, developed for simulating the performance characteristics of WSN algorithms. The Prowler has been covered in detail in Chapter II. We use the Directed Flood Routing Framework as a basis for comparison as it also results in an assessment of two different approaches towards routing in wireless networks in general, and a WSN in particular. Furthermore, the working of DFRF had already been validated using simulations in Prowler. Thus, a valuable set of data was readily available for comparisons.

The DFRF revolves around a flood-routing engine for WSNs. The engine communicates with the radio stack provided by the underlying OS. Applications desiring to use the DFRF register with the engine. The engine also registers the routing policy to use. This routing policy is also based on flooding. The DFRF engine handles the actual transmission and reception of the messages. The policy manages the pool of messages to be transmitted / relayed. It makes decisions regarding the importance of the message, the lifetime, the direction/angle of its transmission etc and provides the engine with the order

in which the messages need to be transmitted. Thus, the DFRF achieves separation of the policy of routing the messages from the actual transmission details. The routing policy can be described using a simple state machine.

One of the policies that may be used in DFRF is the Gradient Convergecast policy. This policy involves the base-station or the root node sending a message (s) to all the nodes in the network. Nodes receiving the message calculate the gradient with respect to the root node in terms of the number of hops it takes to reach root node. Whenever a message is received, it will only be relayed forward if the gradient of the recipient is lesser than that of the transmitter. Other examples of policies, outside the scope of our analysis, are the Broadcast policy and the Spanning Tree policies.

This section compares the PGR to the DFRF with a gradient convergecast policy. Routing in sensor networks can primarily be of two kinds: any node to a base station or any node to any other node. Most of the WSN applications use the any-node-to-base-station routing. Hence, a comparison of PGR with the DFRF is also tailored towards evaluating this aspect of routing.

Simulation Environment

The wireless-connectivity graph of nodes in a WSN changes frequently. Nodes that are participating in routing messages between two nodes may cease doing so. Demise of a small number of nodes that are concentrated in a tiny area of operation will generally not affect the routing of messages between two nodes. However, if a significant number of nodes are unable to participate in routing, message transfer in the network may be jeopardized. It is assumed that all nodes are obligated to participate in routing messages. A good metric for evaluating the robustness of a routing protocol is its adaptivity to such

voids in the network. These voids may be created in the network during its deployment; or they may materialize due to noise, loss of power etc. A routing protocol should not only be able to transfer messages across voids that exist on deployment, but also adapt to handle voids created due to the dynamicity of the network. We adopt three topologies for this evaluation.

The *Uniform* topology is a randomly generated network of nodes that are uniformly distributed across the field. Most of the envisaged WSN deployments generally lead to a uniform deployment. Hence, this can be treated as a default layout to test the efficacy and performance of the routing protocols. The *Void* topology is again a static deployment of uniformly distributed nodes. However, a void is created at deployment such that most routing messages would need to clear this void. This topology tests the routing protocol's ability to traverse the voids. The final topology is a combination of the uniform and void topologies. This topology is deployed as uniform distribution of the nodes. However, over a period of time, a few nodes are killed to artificially create a void in the network. Hence, it is named as the *Dynamic Void* topology. This scheme predominantly tests the adaptivity of the routing protocol to the changing network connectivity.

Though topologies play a prominent part in the functioning of message-routing protocols, a routing protocol cannot be judged based solely on the virtue of its adaptivity to various topologies. The wireless medium is a shared medium. Any transmitted message – addressed to all nodes in the vicinity, or multiple nodes, or specifically addressed to a single node – is actually heard by all the nodes. The lower-level MAC layer decides whether the message needs to be processed or not, and subsequently, either

ignores / drops the message or passes it on to higher-level components in the network stack. Thus, each node that hears any message needs to spend some amount of time and energy to perform this minimum processing. Moreover, in a policy such as broadcast, most nodes would receive the message and be expected to process the message and relay it further. Transmission of a message also costs energy. Thus, it is important to reduce the number of messages, rather copies of a message, in the network. Hence, the *Number of Transmissions* resulting from a single message is an important benchmark for assessing the routing protocol.

Multi-hop message routing is the traversal of a message across multiple nodes from the source node on its way to the destination node. Messages need to traverse across a network in bounded time. A delayed message is useless most of the time; it also clogs up the network carrying useless or outdated information. Hence, measuring the *Traversal Time* to reach a destination is a good benchmarking metric.

Usually, messages are routed via multiple routes. This behavior may have been incorporated intentionally in the routing protocol. Broadcast-based protocols will evidently route multiple copies of the message. Oftentimes, nodes that do not hear a message being relayed, or some sort of acknowledgement, spawn off another copy in the network. Such copies of the same message, apart from increasing the processing load on the nodes, hog the network bandwidth. We study this facet of the performance of the routing protocols using a timing parameter, *Message Lifetime*. This measures the amount of time lapsed since the transmission of the message by the source node, till the last copy of the message is delivered to the node, i.e. the time during which the message is alive in the network. Another important parameter that determines the quality of a routing

protocol is its ability to deliver the messages. This ability is quantified by the *Success Rate* of message transmissions. Success Rate is defined as the ratio of number of messages delivered successfully to the total number of messages actually transmitted.

The PGR and the DFRF framework with a convergecast policy are compared using the above-presented four benchmarking metrics. Both protocols are benchmarked against each of the metrics, for all the three topologies described above.

Finally, a study is performed to analyze the performance of the routing under different network load conditions. A routing may perform fairly robustly in low-load conditions involving transmission of a single message. Thus, a good evaluation should also consider the performance of routing under varying degrees of network traffic. This requirement is evaluated by using three transmission schemes. The first scheme is a simple routing of a single message. The next scheme involves monotonous / periodic transmission of messages from the same source. The third and final scheme involves transmission of multiple messages in the network at about the same time. Thus, multiple nodes will act as sources and transmit a message for delivery within a very small timeframe.

The experimentation setup consisted of 50, 100 and 150 nodes dispersed in an area depicting 100m x 100m in Prowler. The average radio range of transmission was a radius of 10m. However, the radio model in Prowler was setup to model the transmission range as an imperfect circle.

Results

This section presents and explains the results of the experiments. We first study the effect of control messages overhead. PGR periodically sends control messages for building up the neighborhood.

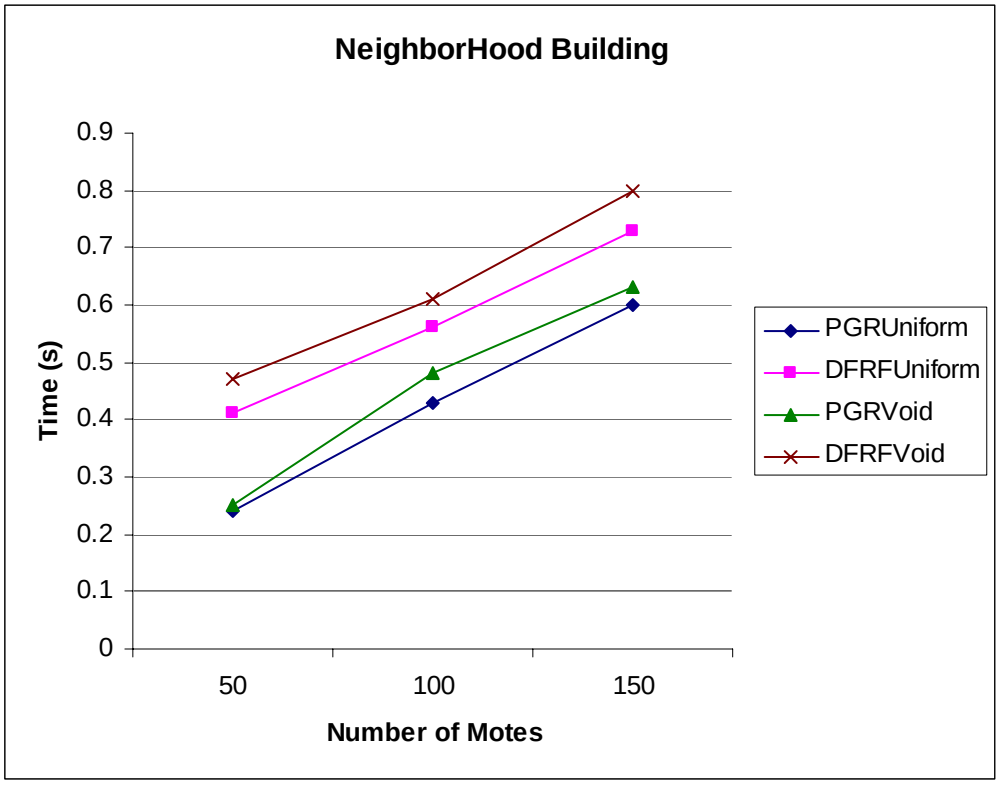

Figure 20 Control Messages Overhead per Single Round

For the experimentation, the PGR periodic advertisement is set for every 5 seconds. On the other hand, DFRF uses control messages only once, at the deployment phase of the WSN application. However, the settling time for these control messages turns out to be quite high. Experiments were conducted on 50, 100 and 150 motes. Two topologies, Uniform and Void, were used. The data is provided in Figure 20. In general, the PGR performed better than the DFRF for one single message in the “knowing the

neighbors” phase. The performance of PGR was observed to be same for both Uniform and Void topologies. PGR neighborhood beacon was indifferent to the variation in topology. The minimal increase in time for 100 motes is an acceptable difference. DFRF, due to transmission of messages to all the nodes via broadcast, spawns more messages leading to a greater time for building the neighborhood, or as in case of DFRF finding the gradient. It was also observed that the number of nodes directly affect the ability of a network to settle down after sending control messages. This time seems to increase proportionally to the number of nodes. This is due to a proportional increase in the number of control messages transmitted by each node added to the network.

However, DFRF builds its gradient by continuously sending the control-message. Greater the number of messages better is the gradient. Thus, the time before a WSN application can start routing data back to the base-station using DFRF is directly proportional to the number of control messages needed. The above figure presents statistics for only one control message, which results in a far from satisfactory gradient. Conversely, in PGR, each node sends the advertisement message periodically, in this case every 5 seconds. However, the WSN application can start routing at the end of first round. Further rounds are merely for adapting to changing topologies. Thus, PGR performs better when there is a need to immediately start routing data after deployment.

Though the PGR is expected to perform reasonably well using a low priority for uniform layouts, the following round of tests were carried out in a high priority mode. This was done to ensure that worst-case behavior may be observed and accounted for in the experimentation results.

The next round of results shows data from the transmission of a single message from a single node. As earlier, the test setup consisted of 50, 100 or 150 motes spread over a simulated area of 100m x 100m. The source transmits a single message and the time to reach the destination is calculated. The experiment is repeated for over 25 messages, and the times are averaged. Only four topologies are used. The dynamic void topology behaves similar to the Uniform topology as the void is not created until after the transmission of the first message is long over.

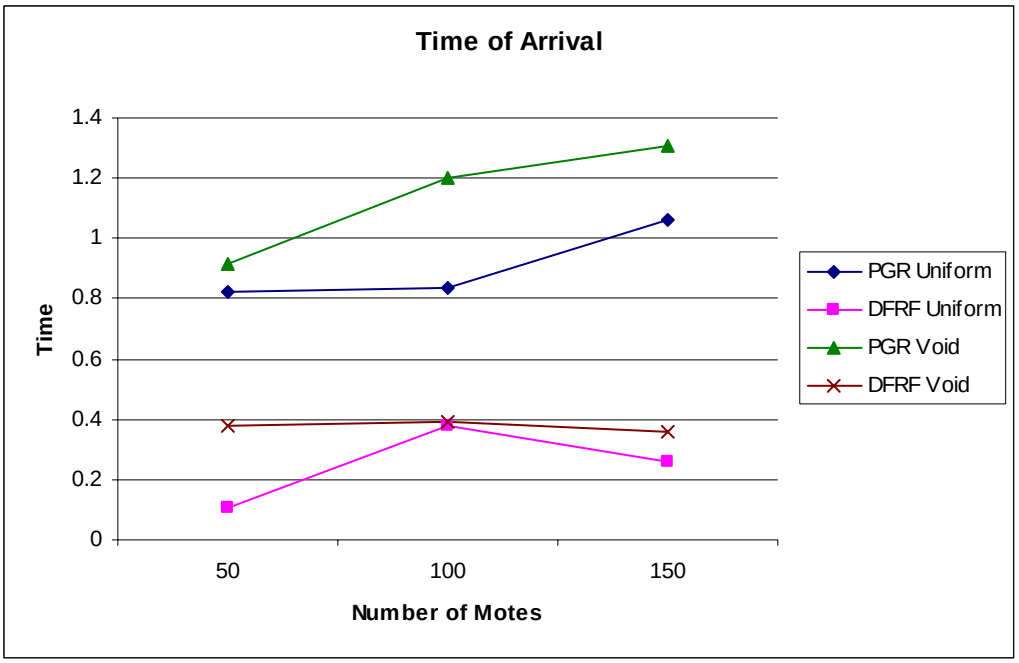

Figure 21 Time of Arrival: Single Source, One Message

The time of arrival of a message gives the time delay from the time the message was transmitted by the source to the time it was delivered at the destination. Figure 21 shows the performance of PGR and DFRF on two topologies. The setup that was used had the source situated in the left-bottom corner of the network, while the destination / base-station is located at the right-top corner. The source and the destination were

separated by an euclidean distance of approximately 87m. This translates to an estimated hop count of about nine (9). In general, both PGR and DFRF take a longer time on topologies with a void in between a source and destination. DFRF takes a longer time due to its inability to adapt itself to the void. PGR finds voids along the route. Nodes that encounter these voids wait and retransmit the message. This waiting time adds up the time required to deliver the message. Due to the availability of pre-calculated gradients, DFRF performs better than PGR, which has to perform calculations at every node, in the delivery-time stakes. PGR's time of arrival is also bumped up due to the waiting-time required in the retransmissions to the next-best neighbors.

Settling Time is the time lapsed from the first transmission till the last transmission of the copy of the same message. This gives an important indication about the lifetime of a message and its use of network resources after it has been delivered. Figure 22 shows the performances of the both the algorithms on the previous setup. PGR takes a significantly less amount of time to settle down after the transmission of the message. Its performance is similar in either topologies, but fares marginally worse in the Void layout as compared to the Uniform layout. An interesting observation is that PGR appears to perform badly in a network of 50 nodes as compared to its performance in networks of 100 or more nodes. This is due to a probable lack of neighbors closer than in itself to the destination. This reflects more on the topology used than PGR's performance. DFRF, due to its broadcast policy, takes a longer time to settle down.

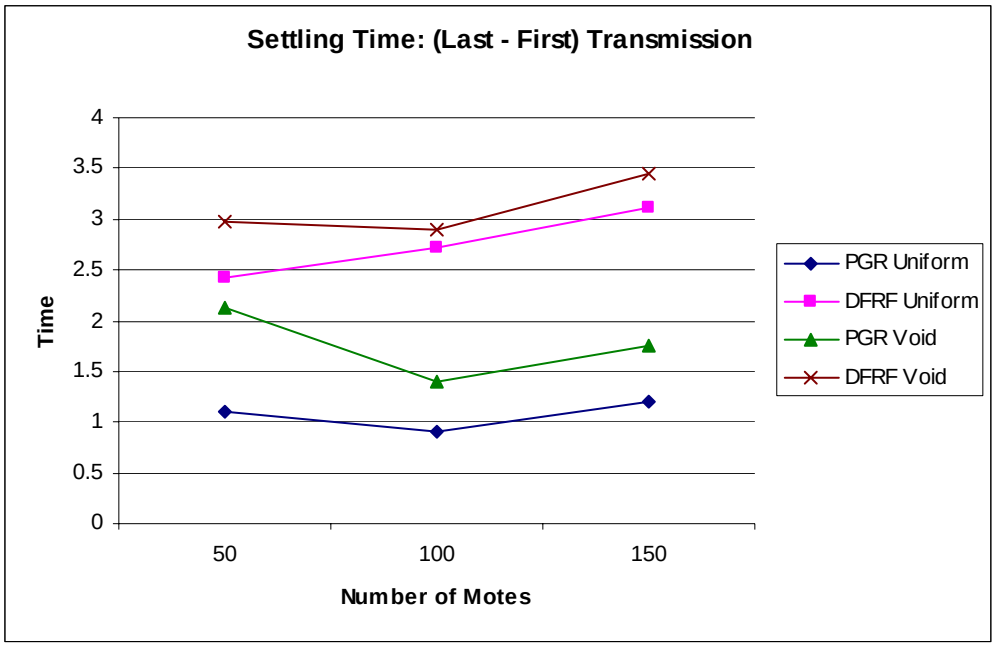

Figure 22 Settling Time: Single Source, One Message

Another parameter that was studied is the Number of Transmissions per message. This parameter symbolizes the total number of hops and attempted hops a message and its copies traveled in the network. Ideally, it should be equivalent to the hops required to follow the most optimum route available. However, this is seldom so. Usually, nodes either broadcast / multicast the message, or resend a message in case of a missing acknowledgement etc. Due to this, multiple copies of the message exist in the node, with each copy in turn potentially spawning more copies. As may be seen in Figure 23, PGR is much better as compared to DFRF. The only time PGR performed worse was in a layout of 50 nodes, wherein it needed to go into its high-priority mode and do some backward-traversals to eventually reach the destination.

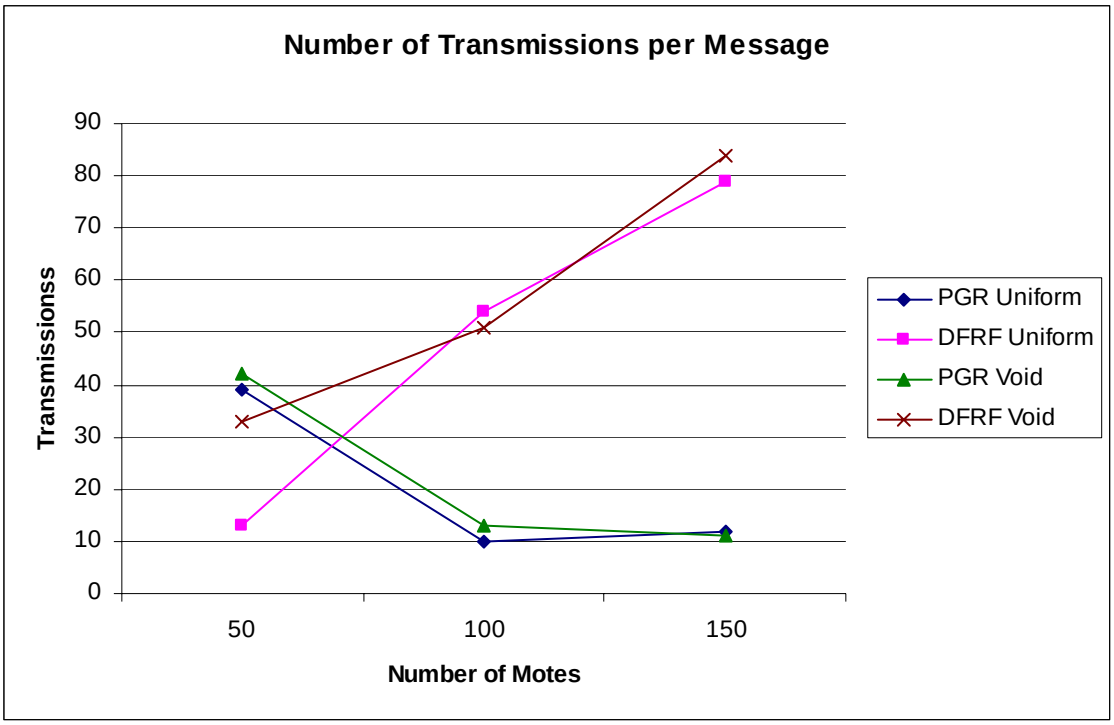

Figure 23 Number of Transmissions: Single Source, One Message

All the experiments resulted in a 100% Success Rate in this scenario.

Further tests were carried out using the same framework as above. However, the message transmission policy was changed. In the first instance, a sequence of messages was transmitted from a single source periodically. Network congestion was studied in terms of the above parameters. The trend shown above continued most of the time. However, significant changes were observed in case of the Dynamic Void topology. Whenever the topology changed and a void was created, it took a significantly longer amount of time for DFRF to deliver the message. Furthermore, it ended up traversing all the nodes in delivering a single message. Consequently the settling time was also observed to be more. PGR, on the other hand, did not exhibit any trouble in adapting itself to the changed topology. Spikes in data were observed for the message being routed

when the void was created artificially. This was due to the back-tracking required to find an alternate path. PGR also failed in delivering the message when the node with the message was killed. The success rates for PGR were observed to be between 90-100%, at an average of 98.37% with a Standard Deviation of 2.756%. The success rates for DFRF were between 95-100%, averaging 98.96% with a Standard Deviation of 1.724%.

CHAPTER V

CONCLUSION AND FUTURE WORK

Conclusion

Advancements in Micro-Electrical Mechanical Systems (MEMS) are leading to development of cheap devices that can be used in various applications. One such area is the Wireless Sensor Network. WSNs are deployed in applications ranging from data collection to in-network data processing and actuation. Routing is an integral and a very important part of such WSNs. There exists a need for a multi-faceted routing protocol that can cater to various highly specialized applications.

PGR or Prioritized Geographical Routing strives to address this need. PGR provides a robust, lightweight routing protocol with the ability to route messages from any node to node. Furthermore, it has the capability to treat messages of differing priorities differently, and adopts varying strategies to ensure message delivery.

A routing protocol based on the position of the nodes in the network is more robust and scaleable than protocols that do not make use of any location-based information. Additionally, it also eliminates the need to keep track of various node IDs in a WSN that may typically run into thousands of nodes. A geographical routing protocol was developed and implemented to provide two services: Any-node-to-Any-node message delivery and Priority-based routing.

The PGR protocol does not store state of a route at any of the nodes. Neither does the PGR store information about the route to be followed for a particular destination, nor does it build complex data structures to describe a route to be taken to reach a particular

destination. This enables the routing of messages from any node in the network to any other node. The route from one node to another is calculated in a distributed manner, taking into account the Euclidean distance between the current node and the final destination.

PGR also provisions for handling messages with differing priorities. It offers three different priority levels that may be used according to the domain of the application. The three different priority levels ensure that the network is not clogged by unimportant messages, while performing an exhaustive search for very important messages.

The PGR protocol was subsequently implemented in nesC for the MICA2 platform using TinyOS. It was used to provide routing in small and medium sized applications. Successful execution on the MICA2 platform as well as simulations in TOSSIM proves that (1) Messages could be sent from any node to any other node successfully, (2) Messages with higher priority had higher probability of being delivered to their intended destination.

The PGR was compared with DFRF framework using Prowler. The tests showed that advantages of the PGR protocol are: (1) In a uniformly distributed network of nodes, the average time for delivery is on an average one-half times more than DFRF, (2) The average settling time of the network due to a message in a uniformly distributed network was twice less than that of DFRF. This was due to the low number of message-copies spawned by PGR, (3) PGR exhibited a better adaptation to the changing topologies as compared to the DFRF. The number of transmissions per message by PGR over the uniform topology, voided topology and a dynamically created void in the topology show an increase of only ~2%. The same scenario leads on an increase of ~6% in the DFRF.

Comparisons with DFRF lead us to believe that, PGR can be successfully deployed in environments requiring flexible topology and immediate adaptation to topology changes. PGR showed a comparable Success Rate with respect to DFRF. PGR also uses less control messages as compared to the DFRF. Thus, its initial settling time, or the network initialization time is better as compared to the DFRF.

A prioritized location-based message routing is a feasible protocol for communication within a sensor network. Furthermore, such an implementation is a good way of reducing the traffic in the network.

Location-based routing protocols have been studied for long [22][23][25]. The major advantage of location-based protocols is their scalability. In a network that has hundreds of nodes constantly joining and leaving, and the link quality varying continuously, a protocol that scales according to the number of nodes in the network is desirable. Results provided in the previous chapter show that PGR easily adapts itself to the change in the number of nodes. Since the routing is based entirely on the geographical position of the nodes and is not affected by any other factor, the PGR scales to a change in the area (graph) boundaries as well. It has been proven that protocols that do not use location based routing are not as scalable [24][26].

Location awareness is resolved in most WSN applications, and the nodes know their position, either relative or absolute. Location-based routing uses localized algorithms and works in a decentralized manner. This makes the protocol light-weight and robust.

Future Work

Several new approaches can be investigated to improve the efficiency and working of the algorithm. To begin with, sensor nodes are highly energy-constrained. The PGR, while reducing the total number of hops in the network and thus the energy required to transmit a message, is a greedy approach that tends to use the same set of forwarding nodes for a route between a given source and destination. Situations wherein a message needs to be continually routed between the same pair of nodes would lead to rapid draining of power from the set of nodes when compared to the neighboring nodes.

The greedy approach also tends to utilize the computing power of the same set of nodes, while keeping their neighbors relatively free of any routing computations and thus any routing overhead incurred. A future research direction may be the study of forwarding a message to not the best neighbor, but any one from a set of neighbors.

The routing protocol described in this thesis is dependent on the presence of symmetric links implicitly. Without an acknowledgement of message receipt from the next node, the current node launches into a message retrieval mode and tries to forward the message to other nodes, leading to the possible existence of multiple messages in the network. A research effort could look into the minimization of such multiple messages by reducing the dependence on implicit acknowledgements.

REFERENCES

[1] A. L. Murphy, G. C. Roman and G. Verghese, "An Exercise in Formal Reasoning about Mobile Communications", *Proceedings of the Ninth international Workshop on Software Specifications and Design*, IEEE Computer Society Technical Council on Software Engineering, IEEE Computer Society, Ise-Shima, Japan, pp.25-33, April 1998.

[2] http://w3.antd.nist.gov/wctg/manet/, Project: Wireless Ad hoc Networks, Advanced Network Technologies Division, National Institute of Standards and Technology, Gaithersburg, Maryland.

[3] Frank Schweitzer and Benno Tilch, "Self-Assembling of Networks in an Agent-Based Model", *Physical Review E*, Volume 66, Article 026113, 2002.

[4] Benoit Garbinato and Philippe Rupp, "From Ad Hoc Networks to Ad Hoc Applications", *7th International Conference on Telecommunications (ConTel)*, Zagreb, Croatia, June 2003.

[5] Ian F Akayildiz, Weilian Su, Yogesh Sankarasubramaniam and Erdal Cayirci, "A Survey on Sensor Networks", *IEEE Communications*, Aug 2002.

[6] Chee-Yee Chong and Srikanta P Kumar, "Sensor Networks: Evolution, Opportunities and Challenges", *Proceedings of the IEEE*, Vol 91, No 8, Aug 2003.

[7] Rex Min and Anantha Chandrakasan, "Energy-Efficient Communication for Ad-hoc Wireless Sensor Networks", *Asilomar Conference on Signals, Systems, and Computers*, Vol. 1, November 2001.

[8] H Yang and B Sikdar, "A Protocol for Tracking Mobile Targets using Sensor Networks", *Proceedings of IEEE Workshop on Sensor Network Protocols and Applications*, 2003.

[9] A Mainwaring, J Polastre, R Szewczyk, D Culler and J Anderson, "Wireless Sensor Networks for Habitat Monitoring", *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, GA, September 2002.

[10] http://www-white.media.mit.edu/vismod/demos/smartroom/, Project: Smart Rooms, Massachusetts Institute of Technology.

[11] G. Simon et al., "Shooter Localization in Urban Environments", *Information Processing in Sensor Networks (Submitted)*, April 2004.

[12] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao., "Habitat Monitoring: Application driver for wireless communications technology", *Proceedings of the ACM SIGCOMM Workshop on Data Communications*, Latin America and the Caribbean, April 2001.

[13] Edoardo Biagioni and Kent Bridges, "The Application of Remote Sensor Technology to assist the recovery of rare and endangered species", *Special issue on Distributed Sensor Networks for the International Journal of High Performance Computing Applications*, Vol. 16, N. 3, August 2002.

[14] http://www.citris.berkeley.edu/applications/disaster_response/smartbuildings.html, Project: Smart Buildings, CITRIS, Berkeley, CA.

[15] M B Srivastava, R R Muntz and M Potkonjak, "Smart Kindergarten: Sensorbased Wireless Networks for Smart Developmental Problem-solving Enviroments, *Mobile Computing and Networking*, pages 132.138, 2001.

[16] K. D. Frampton and R. L. Clark, "Control of Sound Transmission through a Convected Fluid Loaded Plate with Piezoelectric Sensoriactuators," *Journal of Intelligent Materials Systems and Structures*, Vol. 8, No. 8, pp. 686-696, August 1997.

[17] G. G. Finn, "Routing and Addressing Problems in Large Metropolitan-scale Internetworks", *Tech. Rep. ISI/RR-87-180*, Information Sciences Institute, March 1987.

[18] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-less low-cost outdoor localization for very small devices", *IEEE Personal Communication*, vol. 7, pp. 28–34, October 2000.

[19] C. Savarese, J. Rabaey, and J. Beutel, "Locationing in distributed ad-hoc wireless sensor networks," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, Salt Lake City, UT, pp. 2037–2040, May 2001.

[20] A. Savvides, C. C. Han, and M. B. Srivastava, "Dynamic fine-grained localization in ad-hoc wireless sensor networks," *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom) 2001*, Rome, Italy, July 2001.

[21] L. Girod, V. Bychkovskiy, J. Elson, and D. Estrin, "Design locating tiny sensors in time and space: A case study", *Proceedings of the 2002 International Conference on Computer Design (ICCD 2002)*, Sep 2002.

[22] Brad Karp, H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks", *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, p.243-254, Aug 06-11, 2000.

[23] Brad Karp, "Geographic Routing for Wireless Networks", *PhD Dissertation*, Harvard University, Oct 2000.

[24] J. Li, J. Jannotti, D. Decouto, D. Karger, and R. Morris, "A Scalable Location Service for Geographic ad-hoc Routing", *Proceedings of 6th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 120–130, 2000.

[25] Young-Bae Ko and Nitin Vaidya, "Location-aided routing (LAR) in Mobile Ad Hoc Networks", *Proceedings of the 4th annual ACM/IEEE international conference on Mobile Computing and Networking*, p. 66-75, 1998

[26] R. Jain, A. Puri, and R. Sengupta, "Geographical Routing using Partial Information for Wireless ad hoc Networks", *IEEE Personal Communications*, Feb. 2001.

[27] AT & T, "The Switching Plan", *Notes on the Network, Bell Switching Practices*, issue 2, Dec 1980.

[28] C. Hedrick, "Routing Internet Protocol", *Internet RFC 1058*, June 1988.

[29] K. Lougheed and Y. Rekhter, "A Border Gateway Protocol (BGP)", *RFC 1163*, June 1990.

[30] J. T. Moy, "OSPF: Anatomy of an Internet Routing Protocol", *Addison Wesley*, 1999.

[31] G. Malkin, "RIP Version 2, Carrying Additional Information", *Internet RFC 1723*, Nov 1994.

[32] C. E. Perkins and E. M. Royer, "Ad hoc On-demand Distance Vector Routing", *Proceedings of IEEE, WMCSA'99*, New Orleans, LA, Feb 1999.

[33] D. Johnson and D. Maltz, "Dynamic Source Routing in Ad hoc Wireless Networking", *Kluwer Academic Publishers*, 1996.

[34] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, "Energy-Efficient Communictaion Protocol for Wireless Microsensor Networks", *Proceedings of Hawaiian International Conference On System Science*, Jan 2000.

[35] Tian He, J. Stankovic et al., "SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks", *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, Providence, RI, May 2003.

[36] J. Hill et al, "System Architecture Directions for Network Sensors", *ASPLOS 2000*.

[37] S. Park, A. Savvides, and M. B. Srivastava, "SensorSim: ASimulation Framework for Sensor Networks", *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM2000)*, p. 104–111, 2000.

[38] X. Zeng, R. Bagrodia and M. Gerla, "GloMoSim: a Library for the Parallel Simulation of Large-scale Wireless Networks", *Proceedings of PADS*, 1998.

[39] http://www.mathworks.com/, Product: MATLAB.

[40] http://www.isis.vanderbilt.edu/projects/nest/prowler/, Project: Prowler, NEST, Institute for Software Integrated Systems, Vanderbilt University.

[41] http://www.isi.edu/nsnam/ns/, Project: Network Simulator, ISI, University of Southern California.

[42] http://nesl.ee.ucla.edu/projects/sensorsim/, Project: SensorSim, University of California, Los Angeles.

[43] P. Lewis, N. Lee, M. Welsh, D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications", *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003.

[44] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems", *Proceedings of Programming Language Design and Implementation (PLDI)*, June 2003.

[45] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister, "System Architecture Directions for Network Sensors", *ASPLOS 2000*, Cambridge, November 2000.

[46] T. von Eicken, D. E. Culler, S. C. Goldstein, and K.E. Schauser, "Active messages: A Mechanism for Integrated Communication and Computation", *Proceedings of the 19th Annual International Symposium on Computer Architecture*, p. 256-266, Gold Coast, Qld., Australia, May 1992.

[47] B. W. Kernighan and D. M. Ritchie, "The C Programming Language, Second Edition", Book: Prentice Hall, 1988.

[48] D. Gay, P. Lewis, D. Culler, E. Brewer, "nesC 1.1 Language Reference Manual", Manual: UC, Berkeley, May 2003.

[49] P. Lewis, N. Lee, "TOSSIM: A Simulator for TinyOS Networks", Manual: UC Berkeley, Sep 2003.

[50] J. Sallai, G. Balogh, M. Maroti, A. Ledeczi, "Acoustic Ranging in Resource Constrained Sensor Networks", *Technical Report*, ISIS, Vanderbilt University, 2004

[51] Derek J. Evans, "http://www.theteahouse.com.au/gba/files/___yeti_h.html", *Portable GameBoy Advanced 3D Engine*.

[52] Stephen Lacy and Richard Box, "A Fast, Easy Sort", Byte, p.315, April 1991

[53] Miklos Maroti, "The Directed Flood Routing Framework", *Technical Report*, ISIS, Vanderbilt University, 2004

[54] http://w3.antd.nist.gov/wctg/smartsensors/sensornetworks.html, Research: Distributed Detection for Smart Sensor Networks, National Institute of Standards and Technology, Gaithersburg, Maryland.

[55] J. Warrior, "Smart Sensor Networks of the Future", *Sensors Magazine*, Mar 1997.

[56] A. Pentland, “Machine Understanding of Human Action”, *Proceedings of 7th International Forum Frontier of Telecommunication Tech.*, Nov 1995.

[57] J. Postel, “The Internet Protocol”, *IETF RFC 791*, Sep 1981.

[58] Jim Geier, “802.11 MAC Layer Defined”, Tutorial: Wi-Fi Planet, http://www.wi-fiplanet.com/tutorials/article.php/1216351

[59] ANSI, IEEE Std. 802.11, “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, 1999.

[60] http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?CSMA%2fCD, “Definition CSMA / CD”, *FOLDOC*.

[61] http://www.iec.org/online/tutorials/tdma/, “Time Division Multiple Access”, Tutorial: International Engineering Consortium.

[62] Andrew J. Viterbi, “CDMA : Principles of Spread Spectrum Communication”, Book: Prentice Hall, 1st Edition, 1995.