Geographic Grid Routing for Wireless Sensor Networks

by

Jeffrey Christopher Hornsberger
B.Sc., University of Victoria, 2002

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Supervisor: Dr. G.C. Shoja

ABSTRACT

High resolution data collection using low-cost wireless sensor networks has recently become feasible due to advances in electronics and wireless networking technologies. Unique factors such as large network size, particular traffic patterns and severe power limitations necessitate targeted research in the area of sensor network routing.

Geographic Grid Routing (GGR) is described in detail in this thesis. GGR aims to provide robust task dissemination and data collection from large sensor networks using geographic routing to reduce stored state information and used energy. In this way the useful lifetime of the network is prolonged. The work builds on a previously developed routing protocol called Two-Tier Data Dissemination (TTDD). Our work is differentiated by the use of multiple paths, a more efficient and realistic data collection model, and more realistic environmental assumptions. Realistic experiments are used to evaluate the performance of GGR.

This thesis investigates the correctness, performance and applicability of the GGR protocol. The protocol is validated using state-of-the-art model checking software and the advantages of GGR over TTDD are shown through mathematical modeling. The GGR protocol is implemented using the latest network simulation software with our own extensions that result in a realistic sensor network model. Performance tests were conducted at the University of Victoria's Research Computing Facility. The results show GGR to be a highly scalable, versatile and robust solution.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

DN Dissemination Node. Dissemination Nodes are sensors that make routing decisions and may perform data aggregation. A Dissemination Node is the nearest known sensor to a given Dissemination Point since it cannot be guaranteed that a sensor will exist at the precise location of each Dissemination Point.

DP Dissemination Point. Dissemination Points are exact locations to which traffic should be directed. A Dissemination Point exists at each cross point of the grid formed by dissemination of a TASK message through the network (see Section 4.4.1).

GGR Geographic Grid Routing, the routing protocol presented in this thesis.

IN Intermediate Node. An Intermediate Node is any node that is not the SNP Sender or Destination of a packet.

SN Sensing Node. A Sensing Node is any node within a region of interest. Sensing Nodes gather data according to assigned tasks.

SNP Sensor Network Protocol, the network layer protocol used with GGR (see Section 4.2).

Acknowledgement

I thank my parents, Wilma and Barry, and my brother, Whit for their unwavering support and encouragement. One could not have a closer, more loving family. I appreciate you always being there for me.

There are always periods of frustration during a long project such as this. Friends help give perspective and put a smile on your face during those times. Thank you to Ross, Heather, Ben, Pete and the Iron Dragons team. You made these two years some of my best.

I also gratefully acknowledge the financial support provided by the Natural Science and Engineering Research Council of Canada (NSERC), New Media Innovation Centre (NewMIC) and the University of Victoria's Faculty of Engineering. This work would not have happened without their endorsement.

Thank you to the PANDA (Parallel, Networking and Distributed Applications) Research Group. Dr. Kui Wu and Steve Shelford in particular gave me much of their time. Your criticisms and suggestions helped immensely.

Significant effort was made by some staff members at the University of Victoria Research Computing Facility. Many thanks to Caedmon Somers and Drew Leske. There would be no performance evaluation chapter in this thesis without your help!

Finally, for my supervisor, Dr. Shoja, I am deeply grateful. You helped show me the way, but allowed me to choose the path. Thank you for encouraging creativity. Your dedication to your students is unsurpassed and does not go unnoticed. This could not have happened without you.

Chapter 1

Introduction

1.1 Motivation

Although computers play an important role in our lives today, they remain manual machines requiring active manipulation. Computers of the future however, will be a ubiquitous part of our surroundings. These computers will require access to forms of input other than manual entry. The alternative input will come from sensors that provide the computer with detailed information about its environment. Using this information, the computer will make intelligent decisions without human intervention. In time, these sensors will help realize pervasive computing applications such as the smart home or office in which the thermostat, lighting, stereo and other appliances are controlled automatically.

Even in the not so distant future large arrays of sensors will be used to gather information for a wide range of applications benefiting science, business and the military. Scientific research can use networked sensors to collect data from inhospitable territory. Sensors could be deployed in a forest to track animal migration patterns or near fault lines to detect seismic activity. In business sensors can be used to monitor industrial systems and detect wear and tear before it becomes problematic, or track inventory as it moves from a manufacturing plant through warehouses and on to retail outlets. Finally, networked sensors are of great interest to the military for detecting enemy movements or dangerous conditions during chemical warfare.

Although some wired networks of sensors will be required and have in fact been de-

ployed for uses such as Victoria's own NEPTUNE project [1], deployment of wired sensors will not be feasible in many applications for both economic and practical reasons. In these cases the sensors must be completely autonomous, using radio communication and be either battery powered or self-powering (via solar power or other energy scavenging mechanisms).

A wireless network of inexpensive sensors has many benefits for data collection. Sensors could be deployed in nearly any situation and begin communicating information immediately. The large number of sensors provides high resolution information. When properly designed, sensors in the network can be seamlessly replaced, allowing the information gathering to continue for extended periods of time. In some cases, even when wired sensors are possible, it may be more attractive economically to use wireless sensors rather than incurring the cost of wiring the region during the deployment stage. Further, some applications may not be able to afford long deployment times in which case wireless sensors could be quickly scattered throughout the environment and left to gather data.

1.2 Description

In order to gather high resolution information from the area of interest, sensors are densely deployed in large numbers of perhaps several thousand. The large number of sensors also provides a high degree of redundancy in the network. Redundancy is required to compensate for high failure rates of cheap sensors, the potential for sensors being destroyed in a hostile region, depletion of power resources and the unreliability of the wireless medium. The overabundance of sensors does not guarantee that sufficiently high resolution information will always be available and able to reach the interested party. No such guarantees can be made in such an unpredictable domain. However, through careful network design we can assert that data of the desired quality will *probably* be available for a certain period of time after deployment. It is this time period, the useful lifetime of the network, that we attempt to prolong in sensor network design. For example, a data collection application

may require a certain level of data resolution for a period of time. The likelihood of our network satisfying these requirements can be increased by deploying sensors more densely than what our data collection requires and using routing protocols that are designed for energy efficiency and resilience to failure.

The sensors are assumed to be placed randomly since strategic placement may not be possible in more inhospitable environments. In some cases sensors may simply be dropped en masse from an airplane flying over the region of interest. Once on the ground the sensors communicate to form a sophisticated and efficient communication network. In order to further extend the useful lifetime of the network, sensors may need to be added at a future time to replace some sensors that have ceased to operate. These new sensors will again be placed randomly and must integrate seamlessly with the existing network without manual configuration.

Communication within the sensor field must facilitate both request spreading from a monitoring station within the network to the sensing nodes, and data aggregation from the sensing nodes to the monitoring centre [2]. Request spreading communicates a request for information to nodes in the network. The request could include parameters such as the time period during which the requested information is desired, the geographic region to which the request applies or specifics as to the type of information that is of interest. Data aggregation refers to the transmission of collected data from the sensors to interested monitoring stations. Data may be transmitted by a sensor whenever its readings match a previously received request.

Providing further details in the description of a general wireless sensor network requires presentation of some alternative features. Application requirements dictate the features most appropriate for a particular network. First, the sensor devices may be mobile. In the future, sensor mobility could be required in applications such as tracking weather patterns in the atmosphere using lightweight sensors. Sensor mobility could also allow mostly static networks to reform into a more energy efficient topology. Second, monitoring stations may be mobile within the network and can vary in number. In a battlefield application there

would likely be multiple, mobile soldiers submitting requests for information to the sensor network. However, in a network of devices used to monitor crop conditions in a field there may only be a single monitoring station and it would probably have a static location.

1.3 Current Research

Current research in the area of sensor networks revolves around many distinct sub-problems being pursued in parallel. In this section we will present a few of the most relevant problems.

The first such problem is known as area coverage. As previously stated, sensors are densely deployed throughout the area of interest for redundancy. However, not all deployed sensors are needed at all times. Energy can be saved (and the network lifetime thus prolonged) by putting some of the sensing devices into a power-saving sleep mode. The coverage problem addresses which sensors should be put to sleep while maintaining the full capability of the network. The coverage problem has been addressed by many including Jean Carle and David Simplot-Ryl [2] and Seaphan Meguerdichian [3].

A second problem known as localization deals with how a device in a sensor field can determine its own geographic location. Geographic location is required by most sensing applications to give meaning to collected information. For example, we would like to know where in the area of interest the temperature measures over forty-five degrees Celsius, not just that we got a temperature reading of over forty-five degrees somewhere in our broad sensor field. Localization has been addressed by many, including Savarese, *et al.* [4].

Lastly is the problem of moving messages through the sensor field. This is known as the routing problem. Choosing the most efficient way to handle network traffic is essential in preserving the network for as long as possible. The routing problem is a multifaceted one. Ian Akyildiz offers a good summary of the issues in his survey paper from 2002 [5].

1.4 Outline

The following work describes a new solution to the routing problem for a network of stationary, wireless sensors. The research is based on an idea first published by Fan Ye, *et al.* [6]. However, our scheme is differentiated by a number of features, including the ability to gather data from specific regions of the sensor field, a more robust and energy efficient forwarding structure, and the ability to operate in a real-world environment. Such a setting is characterized by high sensor and communication link failure rates. Our experiments accurately model sensor network applications, the communication medium and the possibility of sensor failures. Thus, our performance evaluations are based on realistic simulations of a wireless sensor network.

The remainder of the thesis is organized as follows. Chapter 2 describes some of the previous work that leads up to the routing problem and critiques existing solutions to the sensor network routing problem. Chapter 3 further details the problem space, sets forth the assumptions that are made and provides an overview of our solution. The chapter also explains our methods for verification and evaluation. The protocol is then specified in detail in Chapter 4. A formal analysis of the protocol appears in Chapter 5. Chapter 6 describes the simulation and implementation of the protocol. Performance evaluation results are given in Chapter 7. Finally the thesis is concluded in Chapter 8 with a summary of major contributions and possible directions for future work.

Chapter 2

Background

Interest in wireless networks began in the 1970s with the DARPA packet radio networks. The popularity of wireless networks has increased dramatically since that time, particularly within the past ten years. These networks now allow users to roam through a metropolitan area without losing connectivity. Recent advances in various areas of technology will allow us to realize large deployments of sensors communicating without wires and capable of gathering high resolution information from an area of interest. However, in order to understand how this is all possible, we must begin with the basics. There are two types of wireless networks – infrastructure based and infrastructure-less or ad hoc.

The first type of wireless network, known as an infrastructure based network, has fixed and wired gateways known as base stations or access points. Devices in this type of network communicate only through the nearest base station. In cellular networks a hand-off occurs when a user moves out of range of one base station and within range of another.

The second type of wireless network is the infrastructure-less network. These networks have no fixed infrastructure of any kind. Instead, all nodes in the network function as routers to cooperatively discover paths and move data to destinations in the network. This activity, known as "multi-hop" forwarding, allows users beyond direct wireless transmission range to communicate.

Ad hoc networks have served as an interesting area of study from an academic point of view, but have garnered little attention from the industrial sector until recently. Ad hoc networks present an ever-changing topology in which information from a source must

make its way toward a destination. The dynamic nature of these networks forms a difficult problem with no clear solution to satisfy all possible requirements. The complexity of this problem has captivated researchers since even before wireless communication networks emerged [7]. Some of the imagined applications for ad hoc networking include communication on a battlefield or other region struck by disaster, network gaming, content distribution and distributed conferencing and collaboration. Unfortunately, these applications have rather limited mass market appeal since much of our world is (or can be) equipped with base stations or access points to provide an infrastructure based network. Infrastructure based networks are generally much more efficient because bandwidth is a major constraint in an ad hoc network. The available bandwidth in an ad hoc network is inversely proportional to the number of nodes in the network when all are attempting to transmit [8] because as the number of nodes increases, each node must use a greater proportion of its bandwidth to forward traffic for other nodes. Fortunately for those of us interested in researching ad hoc networks there is a broad area of application for which an infrastructure based networking solution is not appropriate. Data acquisition from remote areas requires the use disconnected sensors that function together to gather high resolution information and communicate the information to a point where it can be analyzed and used. It is this area of application, known as sensor networks, that has grabbed the attention of a number of corporations for commercialization of ad hoc networking technologies [9].

A sensor network is essentially a wireless ad hoc network with some specific characteristics. The limited energy resources of the sensing devices make high-power, long-range transmissions impractical. Low-power transmissions coupled with multi-hop forwarding techniques must be used for moving information in a wireless sensor network. Given this situation, the study of routing in sensor networks begins with a look at routing in general wireless ad hoc networks. After describing the problem of routing in ad hoc networks we will present the peculiarities that must be addressed by a routing solution for sensor networks. Finally, some existing solutions to routing in sensor networks will be explained and their deficiencies exposed.

2.1 Routing in Ad Hoc Networks

The dynamic topology of a wireless ad hoc network changes the way routing mechanisms operate. In a mobile ad hoc network used for Internet access or voice communications the topology changes as mobile users move within the area covered by the network, or choose to connect and disconnect from the network. This is in stark contrast to wired networks where topology changes are unlikely, typically occurring only when a highly reliable and dedicated router malfunctions. Wired networks use table-driven routing protocols that attempt to maintain consistent path information at each router. That is, the routing strategy in wired networks is proactive. However this method has been largely dismissed as inefficient for ad hoc networks due to the amount of signaling overhead required to maintain updated routing tables in a dynamic network. An alternative strategy known as source-initiated on-demand or reactive routing has been adopted for mobile ad hoc networks. A good survey of routing techniques for mobile ad hoc networks is given in a paper by Royer and Toh [10].

Reactive routing builds routes as they are needed rather than attempting to maintain routes indefinitely. Route discovery is initiated only when a route to a destination is required. An established route is maintained until no longer required or until a link in the path becomes unusable. In general, a source requiring a route to a destination will broadcast a route request message. Confirmation of the route is sent back to the source when a route has been found. During route maintenance the source is informed of any errors occurring along the route and route discovery may be re-initiated. Demand driven routing eliminates the wasteful overhead required by table-driven protocols for maintaining unneeded routes in a changing environment.

On-demand routing is akin to a connection-oriented service where parameters on the desired route can be specified in a manner similar to methods used in ATM or RSVP. The nature of the devices and applications used in ad hoc networks gives rise to specific needs such as Quality of Service (QoS) support and power-aware routing. The on-demand route discovery schemes used in ad hoc networks can easily accommodate the addition of

parameters to the route request. These parameters can specify thresholds used to discover energy-efficient or lightly-loaded routes. Not only is reactive routing more appropriate for ad hoc networks with dynamic topologies, it can help make better use of limited resources in the network.

2.2 Routing in Sensor Networks

Establishment of an end-to-end path in a sensor network is not unlike the strategies used in general ad hoc networks. However, those schemes are not well adapted to the specific characteristics of the devices, networks and traffic flows present in sensor networks. Sensing devices are assumed to be quite unreliable with potential faults resulting from the unreliability of the wireless medium, depletion of the power resource or an external force in a hostile area of operation. Routing protocols must be resilient to these types of failure. Sensor networks also have an order of magnitude more nodes than most mobile ad hoc networks. Routing strategies must be scalable to at least a few thousand nodes to be practical for use in a sensor network.

In Chapter 1 we stated our focus to be on fields of static sensors. Section 2.1 described the ever-changing topology as the rationale for reactive routing techniques in ad hoc networks. The likelihood of temporary or complete node failure in sensor networks makes the topology dynamic even though the nodes themselves are stationary. Further, the energy required to maintain changing routes that may not be needed would not be efficient for a sensor network. Thus demand driven routing protocols are used in static sensor networks.

An opportunity for efficiency gains in sensor networks is *in-network* data aggregation, also known as data fusion. The monitoring station may not require the fine granularity of data offered by the sensor field. In this case intermediate nodes may collect data from a few different sources and forward only the summarized data toward the monitoring station. Monitoring stations are also known as *sinks*. The topic is discussed in two papers by Wendi Heinzelman, *et al.* [11, 12]. Data aggregation can provide significant bandwidth and energy

savings.

As discussed in Chapter 1, sensing devices require location knowledge for data collection applications. The location information can also be leveraged by the network layer for use in routing decisions. By simply moving data geographically closer to the destination we can greatly reduce the amount of stored routing information. A problem with geographic routing is how to route around holes in the sensor field. What happens when a packet reaches a node that is not the destination, but is the closest of its neighbours to the destination? The problem with horseshoe shaped holes is depicted in Figure 2.1. In most cases, sensors that are identified purely by geographic location are easily replaceable without greatly impacting the rest of the network.

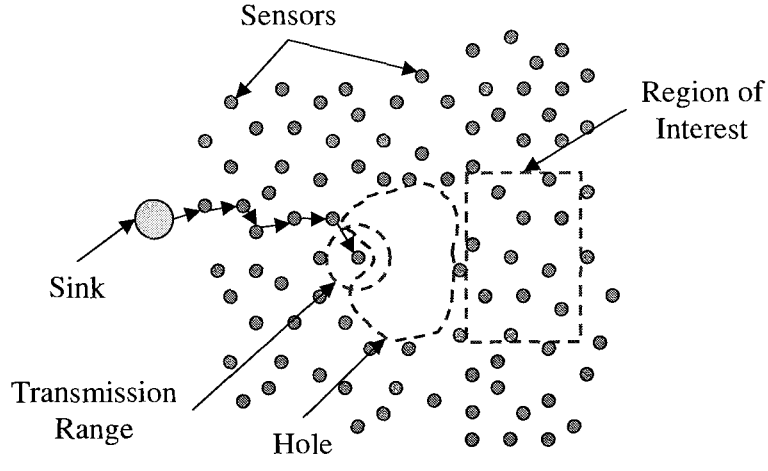


Figure 2.1: The problem with simple geographic routing when horseshoe shaped holes are encountered.

Energy consumption is a concern in any wireless network where devices operate on battery power, however it is a particularly scarce resource in sensor networks. Sensing devices cannot be equipped with large batteries or solar panels due to size constraints. The small amount of power that is available must be stretched as thinly as possible to maximize sensor replacement cycles. Operating costs of the sensor field are reduced when sensor replacement occurs less often. As described in the previous section, reactive routing

schemes can be used to find optimal routes for prolonging the useful lifetime of the network as a whole. Ian Akyildiz describes the following four basic strategies for finding energy efficient paths [5].

Maximum Power Available: Sum the remaining battery power of nodes along the route. The route with the most total remaining battery power is chosen.

Minimum Hop Count: The route comprised of the smallest number of nodes is used.

Minimum Energy: Select the route requiring the minimum energy to transmit the data packets. Note that this strategy reduces to Minimum Hop Count in a homogeneous network of sensors with fixed transmission power.

Maximum Minimum Power Available Node: Compare the nodes with the minimum remaining battery power from each route. Choose the path with the greatest such node.

The number and mobility of sinks is another characteristic of the sensor field. A sole sink presents a single point of failure for the entire network. Using multiple, coordinated sinks provides redundancy against such a scenario. Assuming multi-hop forwarding, the sensors nearest to a sink will have the highest forwarding load since they are the link through which all communication for the sink must pass. When sinks are mobile this load is spread throughout the network.

Thus far, the requirements of a sensor network can be met by general approaches to routing in ad hoc networks. It is the unique traffic flows present in sensor networks that are really the distinguishing factor and present an opportunity for more efficient strategies. As described in Chapter 1, data requests are distributed to the sensors by monitoring centres. When observation data becomes available, it must be routed back to the sink through sensors. Each task from a sink is directed toward a *region of interest* within the sensor field or the entire network (see Figure 2.2). In this scenario we have a one-to-many communication model with a sink collecting data from a group of sensors. Multicast protocols have been widely studied with respect to ad hoc networks, however the task request, data response exchange is particular to sensor networks and opens the door to efficiency gains through

specialization of the protocols.

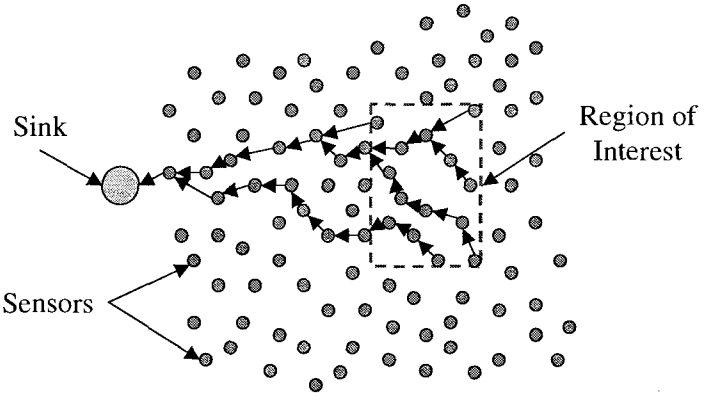


Figure 2.2: Tasks from sinks are assigned to sensors within the region of interest. Data captured by sensors is then relayed back to the sink.

By understanding the traffic flows present in the sensor networks, we can design optimized routing protocols. Jean Carle and David Simplot-Ryl categorize sensor network applications generally as being event-driven or demand driven [2]. We go one step further to categorize the traffic flows.

Triggered Data: The sink is informed whenever a particular phenomena is observed in the region of interest.

Tracking Data: The sink is given the current position of a phenomena as it moves through the sensor field.

Periodic Data: The sink periodically receives observation data from a tasked sensor.

On-demand Data: Current sensor readings are immediately sent when the task is received at the sensors.

Triggered and tracking data flows correspond to the event-driven applications referred to by Carle and Simplot-Ryl. Periodic and on-demand data flows result from demand-driven applications. We assert that all data transmissions are in response to some previously received request from a sink. Sensors must be tasked even for event-driven applications.

The particular characteristics of the sensor network routing problem require focused work resulting in more advantageous solutions. Limited energy resources coupled with the necessity for operation over extended periods of time call for highly optimized protocols, potentially with the loss of generality. By identifying common traffic patterns in sensor networks we can design a routing solution that is general enough for use in varied data acquisition applications, but is highly optimized compared to general ad hoc networking answers.

2.3 Previous Work

The following sections describe current routing protocols for sensor networks. The survey paper on sensor networks by Ian Akyildiz [5] contains a good overview of the state of the art in sensor network routing protocols. Given the requirements of a sensor network described in the previous section we follow the description of each protocol with a critical analysis.

2.3.1 Flooding

Flooding is a well-known routing method that has been covered at length by a number of articles [5, 7, 11]. Nodes in the network forward packets to all neighbours. Each packet contains a *Time to Live* field which is the maximum number of hops the packet is permitted to travel through the network. Duplicate messages and those whose Time to Live have expired are not retransmitted.

Flooding is a brute force method for message transmission that makes no attempt at efficient resource utilization. Every node will transmit the message once. Flooding also leads to a problem called *implosion*. As depicted in Figure 2.3, implosion occurs when the same packet is received from multiple neighbours at the same time. The problem can cause congestion at the receiving node. Despite being wasteful of energy and causing congestion, flooding is a perfect solution with respect to fault tolerance. As long as the network is

not partitioned a packet is guaranteed to be delivered by flooding. For this reason many protocols rely on flooding in some part of their operation.

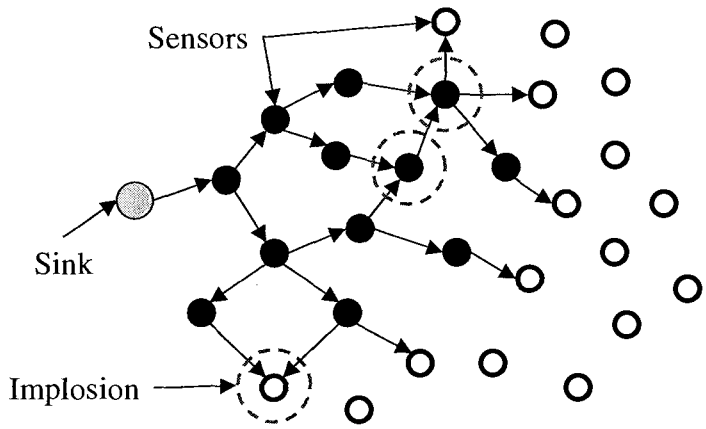


Figure 2.3: A sink floods a query across a sensor network. The implosion problem is highlighted in a number of places.

2.3.2 Gossiping

Gossiping is only slightly more sophisticated than flooding for message routing and was developed long before the recent interest in sensor networks. Gossiping is discussed primarily in an article by Sandra Hedetniemi [7]. Nodes using a gossiping mechanism select a single neighbour at random as the next hop. Forwarding stops when the destination is reached. Gossiping is more advanced than flooding because it requires neighbour knowledge and a random selection function.

Gossiping has a number of problems for routing in sensor networks. The protocol is not effective in providing the multicast communication that is often required in sensor networks. Similar to flooding, gossiping also has no regard for the energy resources in the network. Additionally, gossiping may take a long time to deliver a packet in a large sensor network. In the worst case the packet will be delivered to all sensors in the network, one at a time.

2.3.3 Small Minimum-Energy Communication Network

The Small Minimum-Energy Communication Network (SMECN) protocol was developed by Li Li and Joseph Halpern [13]. SMECN is in fact a topology control algorithm rather than a routing protocol. Variable transmission power is assumed, meaning a node can increase or decrease its transmission range. This characteristic can be used to save power when required transmission ranges are short. The purpose of SMECN is to determine an efficient transmission power level for each node, with an actual routing protocol used on the topology discovered by SMECN.

The algorithm begins with all sensors transmitting at maximum power. At this level the network is as connected as possible, but will also use the most energy. The protocol then follows a discovery period where transmission powers are reduced and high energy links removed. When this period has completed, each node in the network has a specific energy level at which to transmit. The stripped topology includes the minimum energy path between any two nodes.

SMECN does not guarantee multiple paths between any two nodes. Only the minimum energy path between any two nodes is sure to remain. If a node on the minimum energy path fails, some nodes may become disconnected. Thus, the topology will have to be reformed as sensors fail, a costly task.

2.3.4 Sensor Protocols for Information via Negotiation

The Sensor Protocols for Information via Negotiation (SPIN) take a different approach to network layer routing. The ideas behind SPIN were presented at the 1999 MobiCom conference [11]. The philosophy is to remove the need for routing protocols by saturating the network with observations reported by all sensors. SPIN provides an energy efficient algorithm for this data dissemination.

When a device in the network receives new information, either from its own sensors or from a neighbour, it advertises that information to each neighbour. The advertisement

contains only meta-data describing the observation. Each neighbour may in turn request the advertised data if it does not already have a copy. Finally, the actual observations are transmitted to each requesting neighbour. In this way, all data are propagated throughout the entire network.

SPIN makes a number of implicit assumptions about the sensor field scenario. It is presupposed that all observation data in the sensor network are of interest. If this is not the case, critical resources will be wasted by disseminating unwanted information. Also, the amount of data to be gathered from the network must be very small. Otherwise the network bandwidth and device storage capacity will be quickly exhausted.

2.3.5 Sequential Assignment Routing

Sequential Assignment Routing (SAR) is the routing protocol given as part of the suite of protocols for sensor networks put forth by Katayoun Sohrabi, *et al.* [14]. Protocols from the link layer through the transport layer are proposed in the work.

Multiple tree structures are constructed across the network by the protocol. Each tree is rooted at a neighbour of the sink. Neighbours of the sink flood the network with a tree creation message, shown in Figure 2.4. The message is pushed away from the sink by forwarding to nodes at successively greater hop distances. Upon completion of the tree building process, most sensors will belong to multiple trees and thus have multiple paths back to the sink. Sensors near the sink will generally have their energy depleted at a higher rate. The multiple trees allow load balancing among these nodes. A path is selected based on available energy and QoS metrics.

Sequential Assignment Routing is a robust approach for sensor networks due to the multi-path scheme used. When sensor failure occurs, a local recovery scheme is used to avoid rebuilding the entire tree. Like many other protocols, SAR requires neighbour knowledge. This type of information relies on bidirectional links. Bidirectional links are provided by sophisticated MAC layer protocols that use two-way exchanges to ensure frame reception. Such features are often not practical for use in sensor networks because of the

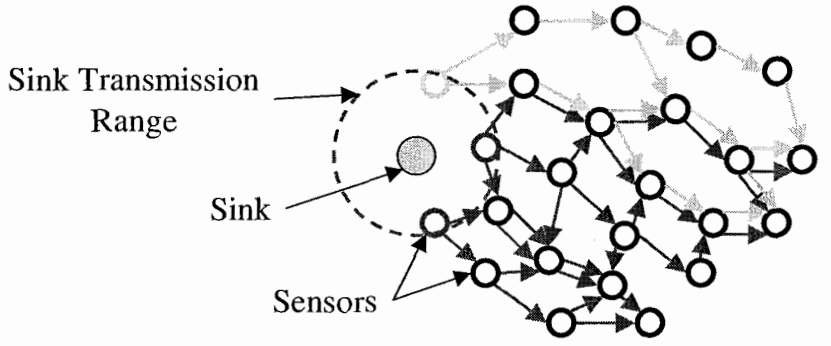


Figure 2.4: Multiple trees based at each neighbour of the sink are formed by SAR.

increased overhead.

2.3.6 Low-Energy Adaptive Clustering Hierarchy

Low-Energy Adaptive Clustering Hierarchy (LEACH) is a hierarchical routing protocol developed at MIT [12]. Like the SAR protocol discussed in Section 2.3.5, this work addresses the issue of sensors near the sink depleting their power resources more quickly than other sensors. As depicted in Figure 2.5, LEACH forms a hierarchical topology in which clusters of sensors communicate with the sink through cluster heads.

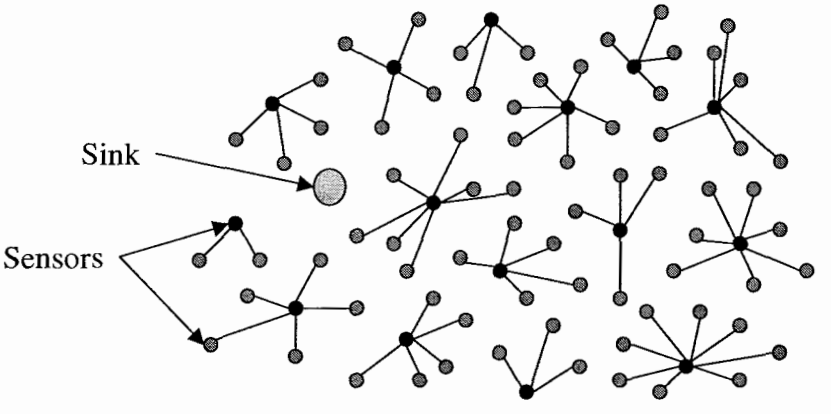


Figure 2.5: Routing hierarchy formed by LEACH.

LEACH forms a hierarchical topology that balances power usage across sensors over

time. The protocol begins with some members of the network randomly nominating themselves as cluster heads. A sensor becomes a cluster head with a certain probability related to its remaining energy level. Other nodes in the network join the cluster head that will require the minimum transmission power. Sensors send observed data to the cluster head. The cluster head then transmits data directly to the sink using a high power transmission. New cluster heads are selected periodically so that the high power job of cluster head is distributed among all members of the sensor field.

The work implies the ability of any sensor to communicate directly with the sink using the maximum transmission power. Although this assumption is convenient for the load balancing problem, it is not realistic. The radios that have been proposed for use on sensing devices have a range of five to ten meters, far less than the imagined diameter of a sensor field.

2.3.7 Directed Diffusion

Directed Diffusion provides a unique model for data collection in sensor networks. Presented at the MobiCom conference in Boston in 2000 [15], Directed Diffusion is a flexible and adaptive protocol that is able to take advantage of characteristics that may exist in the network. Directed Diffusion can also dynamically optimize the trade-off between energy consumption and fault tolerance.

The algorithm uses a fairly typical exchange of sink requests and observation data. Sink request dissemination sets up gradients in the network that draw data toward the sink. If members of the network possess location knowledge, geographic routing methods may be used to direct the request toward the region of interest. Flooding is used when optimizations are not possible. Data requests are cached and may be merged with other compatible requests. When a new observation is either received from a neighbour or generated locally, the request cache is checked for a current gradient. If one is found, the data is transmitted back to the sink along the stored path. Observation data is also cached and is used for loop detection and intelligent data fusion operations.

A unique feature of Directed Diffusion is the ability to dynamically tune the level of fault tolerance. Individual paths can be either reinforced or allowed to expire based on energy or delay metrics. Using this mechanism the energy efficiency and fault tolerance needs of the network can be optimized. Unfortunately, this property requires costly periodic request retransmission.

2.3.8 Geographical and Energy Aware Routing

Geographical and Energy Aware Routing (GEAR) is described in a UCLA technical paper [16]. GEAR uses location and power availability knowledge to efficiently route queries to the target region. This protocol only attempts to solve the request spreading part of the problem. Data aggregation is apparently left as future work.

GEAR uses a tunable cost function in selection of the next hop for each packet. The cost function is based on the neighbour's distance to the target region and its remaining energy level. Although convergence may take some time, the hole problem described in Section 2.2 is handled. Nodes near a hole will eventually have an increased cost and present resistance to packets headed toward the hole.

The protocol specifies a new method for disseminating a packet within a rectangular target region, called Recursive Geographic Forwarding. The region is recursively split into four sub-regions. The packet is sent to each of those regions using the GEAR protocol. The Recursive Geographic Forwarding mechanism is shown in Figure 2.6.

A significant problem arises when either the network density is low or the transmission range is small compared to the target region size. Non-termination can occur under either of these conditions if there are no live sensors left in the target region. Routing will continually circle the target region under the assumption that a hole has been encountered.

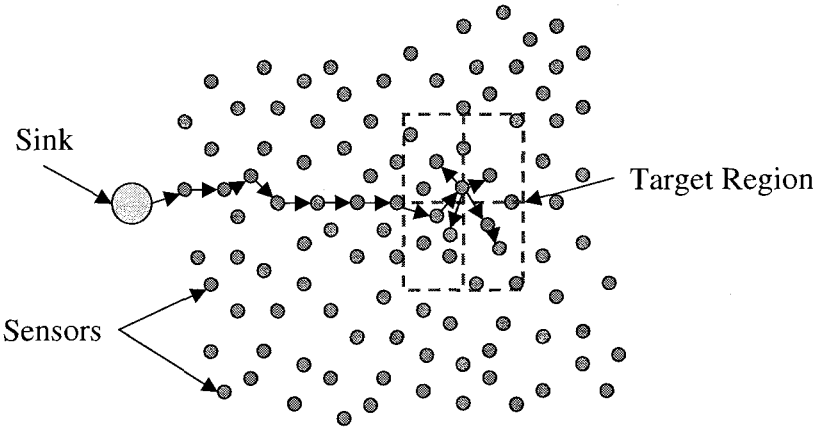


Figure 2.6: Query forwarding using GEAR.

2.3.9 GRAdient Broadcast

The GRAdient Broadcast (GRAB) system is presented in another UCLA technical report [17]. The algorithm proposes an integrated system of density control and packet forwarding, however only the packet forwarding ideas will be explained here. A controllable mesh is used to achieve a multi-path forwarding solution.

GRAB begins by building a *cost field* using an advertisement message flooded by the sink. The flood establishes paths to the sink after which each sensor knows its minimum cost path to the sink. Only the minimum cost path to the sink is stored. No neighbour or path information is required.

The redundancy in the multi-path mesh can be varied by the data source. The source assigns each message a certain *credit*. An intermediate node will only forward the message if the consumed cost plus the stored cost at this node is less than or equal to the message credit. If the credit given at the source is equal to the stored cost of the minimum cost path, the message will only follow the minimum cost path. By giving extra credit, the message can follow multiple redundant paths.

The cost field must be refreshed to account for changes in the network. Cost field refreshing requires another flooding by the sink. The operation is initiated when the sink

detects major changes in success ratio or traveled hop count.

GRAB effectively solves both the area coverage and data aggregation problems at the same time. The data collection portion of the system is able to balance energy consumption with fault tolerance. The cost field setup and maintenance is an expensive operation, requiring a network-wide broadcast. Fortunately it is triggered by changing network conditions rather than a blind periodic timer.

2.3.10 Two-Tier Data Dissemination

A novel form of dynamic hierarchical routing has been proposed by Fan Ye, *et al.* [6]. The Two-Tier Data Dissemination (TTDD) protocol creates a virtual grid structure on which data is delivered. The protocol assumes the availability of location knowledge and multiple, *mobile* sinks. Sinks may be mobile when they are contained in hand-held devices operated by a person walking through the sensor field. Sink mobility requires tracking their location to ensure an uninterrupted flow of data from the sources.

TTDD operation is initiated by the data source through advertisements similar to those used by SPIN, described in Section 2.3.4. The source forms a virtual grid structure over the entire network and becomes the first crossing point of the grid (called a *dissemination node*). The advertisement is then sent to each adjacent dissemination node. Geographic routing is used to forward packets between dissemination nodes. Each neighbour in turn forwards the advertisement to its adjacent crossing points and so on until the grid covers the entire sensor field. Since the grid is based at the source, different sources will use different dissemination nodes. A TTDD grid is shown in Figure 2.7.

Following the advertisement phase, data can be requested and sent using the grid structure. Data requests are flooded by sinks within an area the size of a cell. When a query reaches a dissemination node for matching data, the query is sent toward the source using the reverse path of the advertisement. Finally, data is returned to the sink through the grid using the reverse path of the query.

Routing information represents a soft-state and eventually expires. Therefore both data

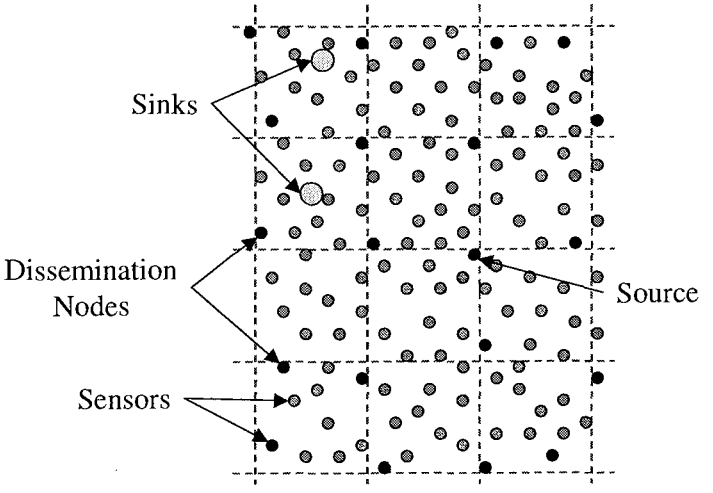


Figure 2.7: Grid formed using TTDD source initiated advertisement.

advertisements and requests must be periodically retransmitted. Stored sink locations must expire because the sinks are mobile. As a sink moves, queries are retransmitted to find new dissemination nodes.

TTDD provides a number of positive solutions to difficult problems. Provided no partitions exist, data announcement messages will reach the entire network without full-scale flooding. The use of distinct dissemination nodes by each source achieves load balancing and improves robustness and scalability because each node will hold state for no more than a few sources at a time.

Unfortunately there are a number of significant drawbacks to the TTDD solution. First, the periodic signaling represents significant overhead in maintaining the grid structure. Second, failure of the single selected source could result in all data being lost from a small region of interest. Third, replication of routing information in sensors near to dissemination nodes is proposed as a way of dealing with node failure. The communication required for the replication only adds to the overhead. Fourth, TTDD does not provide a mechanism for requesting data from the sensor field. This limits the type of applications that can use TTDD to those involving triggered or event-driven data collection. Finally, the grid used by TTDD is static. If a single data source were to continuously transmit data to the sink,

the network would eventually partition.

2.4 Advancing the Current State of the Art

All of the routing solutions proposed to date for sensor networks have shortcomings of one form or another. In the chapters that follow we describe a new routing and addressing scheme for sensor networks called Geographic Grid Routing (GGR). The algorithm builds upon the grid construction concept used by Two-Tier Data Dissemination (TTDD) developed by Ye, *et al.* [6] and draws ideas from other work on both sensor networks and mobile ad hoc networks. Our work aims to address the shortcomings of previous work in the following ways.

- Construct a more robust grid that provides multiple paths from data source to data sink.
- Use sink-initiated rather than source-initiated grid construction allowing for greater data fusion opportunities.
- Maximize the useful lifetime of the sensor field by preferring energy efficient paths.
- Better distribute the data forwarding workload.
- Allow for greater flexibility in data acquisition through sensor *tasking*. The idea of tasking sensors is used by Directed Diffusion.
- Reduce state information to reduce storage requirements and transmissions, thereby improving scalability and performance.
- Permit operation in a network where bidirectional links may not exist.

The preceding sections have given a broad background to the problems of request spreading and data aggregation in sensor networks. The basics of routing in ad hoc networks were explained with highlights on the similarities to sensor networks. We then covered the issues that make traffic management in sensor networks a distinct problem. Next we discussed previous solutions and exposed some of their shortcomings. In the end we

gave a preview of how those weak points will be handled in our own solution. The following chapter will give a more detailed description of the application domain for our work with our basic assumptions.

Chapter 3

Problem Statement, Proposed Solution and Methodology

The general context and scope of the project were given in the preceding chapters where sensor networks were introduced and previous work was described. The deficiencies of prior solutions as they relate to our problem space were also exposed. The end of Chapter 2 gave a brief glimpse of how we intend to improve upon the latest ideas for routing in sensor networks. In particular the issues of data fusion, network lifetime, load balancing, application flexibility and network scalability will be addressed. Our own efforts stem from recent work in the field and are based on realistic assertions about the operating environment.

In this chapter we define our study in greater detail and explain our approach to the issues that remain unsolved by previous efforts. Further specification of the problem domain will be provided through scope clarification, a set of assumptions and a concise problem statement. The premises having been set forth, we expand on how our algorithm overcomes the hurdles to sensor network routing. The final section of the chapter lays out the process by which we show improvement over the past endeavours.

3.1 Problem Definition

3.1.1 Scope and Assumptions

The work detailed in this thesis focuses on request dissemination and data collection from a large sensor network. It is clear from Chapter 2 that sensor networks can vary greatly in the characteristics of the network and the features offered. In this section we state the features and characteristics assumed by our work.

Sensor nodes possess the following capabilities and limitations:

- Production costs are low and physical size is small. Otherwise deployment of a large sensor network in a range of environments would not be feasible.
- The power supply is restricted by sensor size.
- Energy consumption levels can be monitored and reported.
- Processing power and memory capacity are limited by cost constraints.
- Short-range radios are used owing to power limitations. Long-range communication is accomplished through multi-hop routing.
- Radio transmission power, and thus communication range, is static.
- High failure rates are expected due to environmental conditions and depletion of power resources.
- Physical location in the sensor field is fixed and known.

Sink nodes can be described by the following functionality:

- Physical location in the sensor field is fixed and known.
- Energy resources are unlimited.

The networking environment is assumed to conform to the listed assertions:

- The sensor field is represented as a two-dimensional area.
- Multiple sinks are deployed throughout the area of interest.

- Traffic flows only between groups of sensors and a sink. No sensor-to-sensor or sink-to-sink communication is required.
- Simple MAC layer protocols are used for energy efficiency and do not guarantee bidirectional links.
- Sensors are identified only by geographic position.
- Sinks are identified by an address that is unique among all sinks in the network.

A few aspects of routing are extraneous to our specific problem. These concerns are left as future work. First, real-time data collection may well be required by many applications of the future and support at the routing layer will likely be required. Second, reliable end-to-end communication may be required for dissemination of network management directives or for some data collection types. Those issues should be handled at the Transport Layer. Finally, Quality of Service (QoS) in sensor networks relates to the resolution of the information gathered from the sensor network. Solving the QoS problem is mostly dependent on the coverage problem discussed in Section 1.3. Data fusion also affects the quality of the information received at a monitoring station. Data fusion is application dependent and is the responsibility of the application layer.

3.1.2 Statement of the Problem

The constraints on the sensors and unique traffic characteristics of the network necessitate targeted research in the field of sensor networks. Much can be learned from previous work in energy efficient and multicast routing in ad hoc networks, however these general approaches fail to address the needs of sensor networks with respect to scalability, fault tolerance and device constraints. The communication of task instructions from sinks to sensors and the retrieval of data corresponding to those tasks from the sensors continues to be an area of open research. The communication must be efficient, particularly in the face of failure, and must maintain a high level of coverage in the sensor network. The goals of energy efficiency and load balancing are viewed as orthogonal because one can only be

optimally reached at the expense of the other. Thus satisfying both of these goals creates a challenging problem.

The specific problem to be addressed is the provision of robust task dissemination and flexible collection of high resolution data from a large network of sensors such that the useful lifetime of the network is prolonged. This problem is generally known as the routing problem in wireless sensor networks. The stated assumptions and scope provide a clearly defined environment in which our solution is valid and most effective.

3.2 Solution Overview

Geographic Grid Routing (GGR) is a hierarchical protocol for disseminating tasks in a sensor network and retrieving the corresponding data. Only a small subset of nodes maintain routing information and direct traffic. Other nodes simply forward traffic according to the rules of simple geographic forwarding. Simple geographic forwarding dictates that a received packet is retransmitted if the receiver is geographically closer to the destination than the sender.

Due to the homogeneity of the devices, the multi-hop nature of the communication and the task specific design of the sensing devices, nodes in a sensor network take on different roles over time. The idea of role assignment in sensor networks is discussed in a paper by Manish Bhardwaj, *et al.* [18]. The GGR routing protocol defines three roles for devices. A sensor may take on more than one role at a time. Those nodes that make intelligent routing decisions are known as Dissemination Nodes (DNs). When receiving a packet for which it is not the specified destination, a sensor acts as an Intermediate Node (IN). INs only perform simple geographic forwarding operations. Finally, sensors that exist within the specified region of interest of a data request take on the role of Sensing Node (SN). The roles in a GGR network are depicted in Figure 3.1.

The Dissemination Nodes in a GGR network form a virtual grid rooted at a sink. Each sink forms its own grid through which messages are routed. The grid is formed and re-

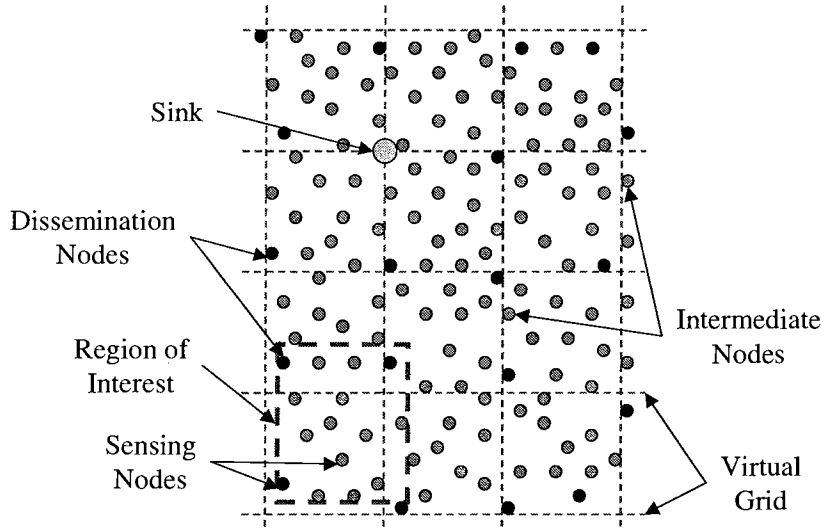


Figure 3.1: Roles in a GGR network.

freshed during the task dissemination process. Thus the two-dimensional sensor field is divided into a grid of cells with a DN at each cross-point of the grid. The cross-points are equally spaced such that they are not within direct transmission range. DNs direct packets toward the next cross-point on their way to their destination. Simple geographic forwarding is used to forward messages between DNs. The virtual grid changes over time to spread the extra load associated with the DN role. The distance between grid cross-points changes in response to network conditions.

Dissemination of data request messages has a dual purpose in a GGR network; grid creation and task assignment. Dissemination Nodes store the most energy efficient upstream links toward the sink as they forward data request messages and thus form the grid. The grid provides multiple paths between the region of interest and the sink, however the path with minimum used energy is favoured for communication of data back to the sink.

3.2.1 Qualitative Analysis and Comparison

In this section we provide some qualitative commentary on the proposed solution and comparison to the TTDD routing algorithm, described in Section 2.3.10. The idea of a grid

structure for routing was first used in TTDD. The many possible paths provided by the grid structure solve the problem of routing around holes in the sensor field. The problem with holes is explained in Section 2.2.

The requirement for location knowledge in GGR operation does not limit the application of our solution. It is our view that location knowledge is required for any sensing application in order to make use of the data gathered by the network. In most applications it is not useful to know the data gathered from a sensor field unless the position from which it was collected is known. In this context our need for location knowledge is not an extra requirement being placed on the sensor network. Instead the GGR solution simply makes use of information already available at the application layer.

The accuracy of various locationing systems cannot be guaranteed. Such errors in the reported location of a device could adversely affect the GGR protocol. The proposed routing scheme would be negatively affected if the reported location of a device can change over time. Since geographic location is used to address sensors, a changing location would be equivalent to changing the address of the sensor. Such a situation would be handled by the protocol in the same manner as a sensor failure and would hurt the performance to some degree. Consistent errors in the position reporting would negatively affect the geographic routing mechanism, however these errors are assumed to be relatively small such that geographic routing would continue to be effective.

The GGR algorithm uses a number of techniques to achieve its goals of scalability, robustness and energy efficiency. *Diversity injection* contributes to the reliability and energy efficiency of the scheme. Diversity injection is described by Pearlman and Haas [19] and is used to select preferable routes by discovering multiple paths from a source to a destination. Scalability is achieved through the hierarchical design which uses only local knowledge in routing decisions. The hierarchical design means just a small group of nodes have to maintain routing information for a particular sink. By the same token the hierarchical design contributes to the energy efficiency because a subset of the devices in the network perform routing operations.

The task dissemination used by GGR is a major advantage over TTDD for two reasons. First, sensor tasking allows for great flexibility in the type of tasks that can be handled by the network. TTDD does not provide a mechanism for sensor tasking, instead it assumes all sensors have already been tasked for event-driven data collection. Other types of data collection, such as periodic, will be important in many sensor network applications. The mechanism could also be used for distributing network management directives. TTDD does not support any form of sink initiated communication to sensors. Second, the grid used by GGR changes randomly over time to spread the role of DN over many sensors. The grid used by TTDD is static.

The sink initiated approach taken by GGR provides much greater opportunity for data fusion than what exists in TTDD. Since each source in TTDD creates a grid, data from a region of interest may travel back to the sink using multiple independent grids. Data aggregation cannot occur between the different grids. GGR provides the opportunity for data aggregation at Dissemination Nodes that are shared by many data sources.

An important feature of GGR is that absolutely no communication occurs unless it is either initiated by the submission of a new request or a data transmission from sensors. Our event-driven grid reconstruction technique discovers new paths only when triggered by changing network conditions.

3.3 Methodology

The previous sections of this chapter defined the problem space and requirements, and gave a brief overview of our solution. The process by which the effectiveness of our algorithm will be shown is now explained. Three techniques will be used in this endeavour; formal verification, mathematical analysis and performance evaluation.

3.3.1 Formal Verification and Analysis

Prior to showing the results of simulations that model real-world conditions, we use a couple of techniques to demonstrate the validity of the Geographic Grid Routing protocol. First, we use a model checker to verify the correctness of the protocol. Second, a mathematical analysis is given to show that our algorithm is a theoretical improvement over previous work.

Modeling involves constructing a prototype of the system under study. Correctness requirements of the proposed system are then formally verified through the use of a model checker. The model checker is able to prove the logical consistency of the system according to correctness properties defined by the user. Exhaustive state-space searching is used to verify the simplified model.

The SPIN model checker was chosen for verification of Geographic Grid Routing. SPIN (Simple PROMELA Interpreter) verifies models specified in the PROMELA (PROcess MEta LAnguage) language. The PROMELA language and SPIN system are designed for modeling and verification of concurrent and distributed systems. The SPIN software is freely available from the Internet [20] and is explained in detail in Gerard Holzmann's book [21]. A verification by the model checker is used to verify safety properties of our system and check for invalid end-states.

Our mathematical analysis explores the theoretical communication overhead and state complexity. Communication overhead indicates energy efficiency because it measures the number of transmissions required for a given operation in the sensor network. Scalability is shown through our state complexity calculations. A lower state complexity indicates that a protocol makes more efficient use of network resources and will be better able to function in larger networks or with larger regions of interest.

3.3.2 Performance Evaluation

Having verified the logical consistency of our design with the SPIN model checking system, the performance of the protocol must be shown. Unfortunately the deployment of a full sensor network is, at this point, beyond our means. However, there are a number of quality simulation packages available. The GloMoSim (Global Mobile information systems Simulator) [22] tool provides a full featured and realistic simulation environment for wireless communication systems.

The GloMoSim simulation package was chosen for its ability to model large scale networks and its realistic radio model. The transmissions in GloMoSim may suffer from interference even when nodes are within transmission range. Under these circumstances we can measure the performance of our system in a real-world environment where unidirectional links may exist. The ability to simulate large networks is critical for simulation of sensor networks where scalability is a primary concern.

A full implementation of Geographic Grid Routing is given within the GloMoSim framework for measurement of various system properties. Metrics such as data delivery efficiency, task targeting efficiency, energy consumption and delay are studied as network size, sensor failure rate and the data collection scenario are varied.

The computing resources required of our simulation are high and necessitate the use of a supercomputer. GloMoSim requires significant processing power and memory. In order to collect our detailed performance data in a timely manner, we used the Mercury Linux cluster at the University of Victoria's Research Computing Facility [23]. This is an eighty-four node cluster where each node has either 1.5 GB or 2.5 GB of memory and either dual 2.4 GHz or 2.8 GHz Intel Xeon processors.

This chapter has provided a stepping stone to the rest of the thesis. A concise definition of the problem to be addressed was provided. A brief overview of our solution was given in the subsequent section. The last section was a description of the process by which enhancements over the current state of the art will be shown. The following chapters explain the GGR protocol, our analysis and performance evaluation in greater detail.

Chapter 4

Protocol Design

The preceding sections have provided motivation and background for the routing protocol described in this chapter. The problem has been clearly identified and an overview of the proposed solution has been given. The following sections specify the Geographic Grid Routing protocol for wireless sensor networks.

The GGR routing protocol is a hierarchical protocol for disseminating tasks in a sensor network and communicating the corresponding data to sinks. The protocol architecture of the GGR system is depicted in Figure 4.1. The majority of our algorithm functions at the routing sublayer of the network layer. Three messages are defined at this sublayer in support of the required functionality. TASK type messages are used for grid construction and data requests. The actual gathering of observation data uses DATA and ACK type messages.

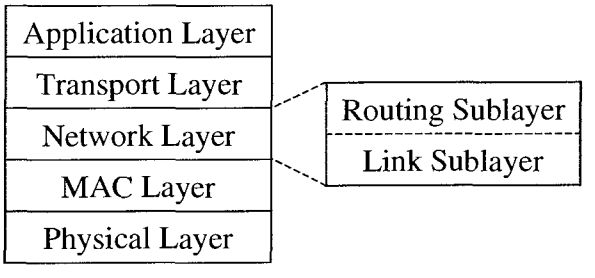


Figure 4.1: GGR network architecture.

The message types used by GGR are specified in detail as they are introduced throughout the chapter. The fields of each message are given a data type. The data type can be one

of those listed in Table 4.1 or a compound type. In addition, some fields may be marked with type APP meaning the data type is application dependent. Finally, some fields are actually lists of the named type. These fields are marked with the LIST modifier. Some field types may need to be adjusted to satisfy application requirements.

Table 4.1: Basic type definitions.

Type	Definition
BOOL	Single bit, either 1 or 0.
INT8	Signed 8-bit value.
UINT8	Unsigned 8-bit value.
INT16	Signed 16-bit value.
UINT16	Unsigned 16-bit value.
INT32	Signed 32-bit value.
UINT32	Unsigned 32-bit value.
FLOAT	32-bit floating point value.

As stated in Section 3.2, the GGR protocol defines various roles for sensors in the network. Each role in the GGR network corresponds to responsibilities at a layer from Figure 4.1. Sensing Node (SN) operations occur at the application layer. The functions of the Dissemination Node (DN) happen at the routing sublayer of the network layer. Intermediate Node (IN) activities are performed at the link sublayer of the network layer.

The DN, SN and IN roles that are assigned to sensors represent their state in the network with respect to a particular sink. Roles assigned by other sinks are independent. Whether or not a sensor has been assigned the SN role embodies its application layer status. The SN role is taken on when the sensor is located inside the named target region of a TASK message. Application layer state is independent of the network layer state. At the network layer, a node is either in the DN or IN state, as depicted in Figure 4.2. Sensors begin in the IN state for all sinks by default and must be promoted to the DN state. This in no way implies that state information must be stored for each sink in the network by each

sensor. The advantage of using geographic routing is that the IN role requires no state information. DN assignment occurs via reception of a TASK message where the sensor is the closest known node to the intended grid cross-point. Note that a sink always has the role of Dissemination Node and never has the role of Sensing Node.

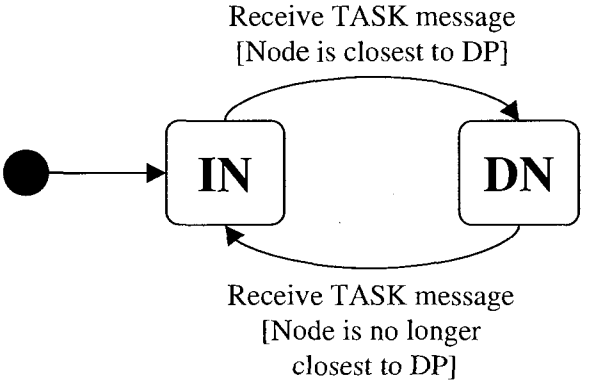


Figure 4.2: High level state transitions of the network layer.

4.1 Sequence Number Usage

There are two distinct and unrelated sequence numbers used at the two sublayers of the GGR system. A SNP sequence number is maintained at the link sublayer and is incremented after each transmission by a Dissemination Node. The SNP sequence number is not incremented when simple geographic forwarding is performed by an Intermediate Node. A GGR sequence number is maintained by each sink and is used at the routing sublayer. The GGR sequence number is incremented after the sink sends a new TASK message to all necessary recipients.

Maintaining and comparing sequence numbers requires some type casting. Sequence numbers are to be treated as unsigned values when they are incremented so that rollover occurs correctly. However, comparing sequence numbers must be done using signed arithmetic. A received message is considered new if its sequence number minus the currently

stored sequence number results in a positive number. Otherwise the message must be discarded.

4.2 Link Sublayer Operation

All the messages are prefixed with a Sensor Network Protocol (SNP) header at the link sublayer. The format of the SNP header is provided in Table 4.3. The compound type POSITION is defined in Table 4.2. The format of the POSITION fields assume a locationing technology such as GPS where X and Y coordinates are given in terms of a floating point latitude and longitude. Other schemes may be used and the format of the POSITION fields changed accordingly.

Table 4.2: Compound type POSITION definition.

Field	Type	Description
X	FLOAT	X coordinate on the two-dimensional plane.
Y	FLOAT	Y coordinate on the two-dimensional plane.

Table 4.3: SNP Header Definition.

Field	Type	Description
Destination	POSITION	Next DN.
Sender	POSITION	Sender of this message (last DN).
Last_Hop	POSITION	Last IN to handle the packet.
Seq_Num	INT16	Sender's sequence number.
Type	UINT8	Message type. Must be one of TASK, DATA or ACK.

Activities at the link sublayer are common to all nodes regardless of the type of message being sent or the role of the sensor. Referring to Table 4.3, the Destination and Sender fields

are never modified by the link sublayer. These fields are specified at the routing sublayer by Dissemination Nodes. The Sender field always gives the position of the last DN to forward the message. The Last_Hop field of the SNP header is always set to the local position at the link sublayer prior to forwarding the packet. The Seq_Num field is only set to the next SNP sequence number when the message originates from higher layers (i.e. a message being sent by a DN). A message that is simply being forwarded closer to the Destination is not modified other than the Last_Hop field. The Type and Payload fields are for use by the routing sublayer.

4.3 Grid Construction

GGR uses a forwarding grid to route messages back to the sink. The parts of the grid are shown in Figure 4.3. The grid is established during the task assignment phase. Cross-points of the grid are called Dissemination Points (DPs). The size of the cells is determined by the sink such that DPs are not within direct transmission range. The sink's position is the first DP. Knowing its own position and the size of each cell, the sink is able to send a TASK message to each adjacent Dissemination Point in the grid. Simple geographic forwarding is used to reach these locations. Upon receiving the TASK message, the closest known node to each DP promotes itself to become a Dissemination Node (DN). Next, the DN forwards the TASK message to each of *its* adjacent DPs, except the point from which the TASK message was received. These action are repeated as the TASK message propagates and Dissemination Nodes are chosen throughout the sensor field. At the same time, the DN records the sender as an upstream link in its routing table, creating the forwarding grid. The grid provides multiple paths between the region of interest and the sink. However, as will be described later, special procedures are used to maintain loop freedom and to favour the least costly path for communication of data back to the sink.

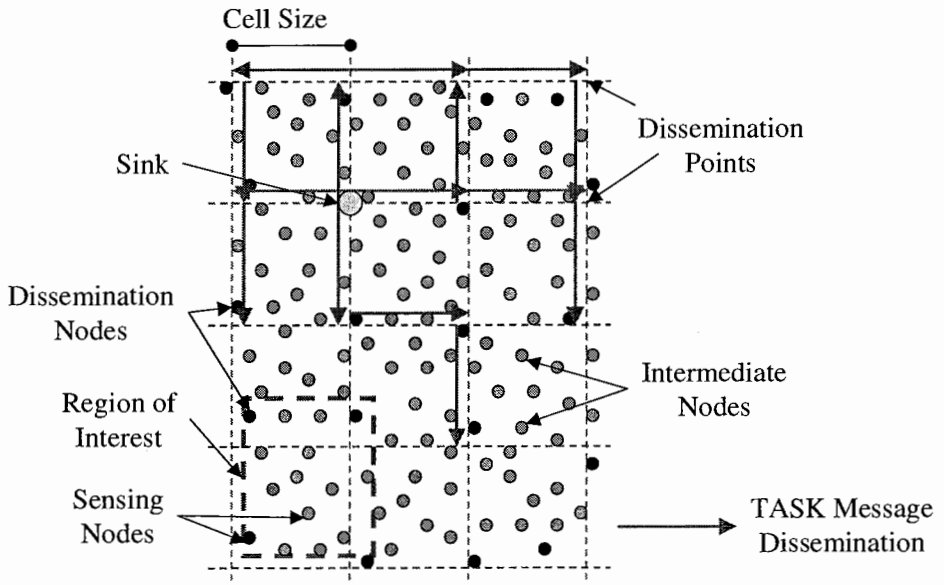


Figure 4.3: Grid construction and roles in a GGR network.

4.4 Sink Operation

4.4.1 Task Assignment

Sensors in the network must be assigned a task before data can be gathered. A sink initiates task assignment by propagating a TASK message. The message is sent to the four nearest DPs to initiate construction of the grid. Simple geographic routing is used to route a message between sources and destinations that are not within direct transmission range, as shown in Figure 4.4. Sending to multiple locations with a single message using geographic routing adds unwanted complexity so the TASK message is sent four times. In each message the Destination field of the SNP packet is set to the location of the target DP. Adjacent DP locations are calculated as the cell size added to the current sink location in each direction. A new cell size is randomly selected from the range MIN_CELL_SIZE to MAX_CELL_SIZE whenever the grid is rebuilt. The Type field of the SNP header is set to TASK and the Payload is the contents of the TASK message.

The TASK message is used to carry data requests from sinks to sensors. The fields

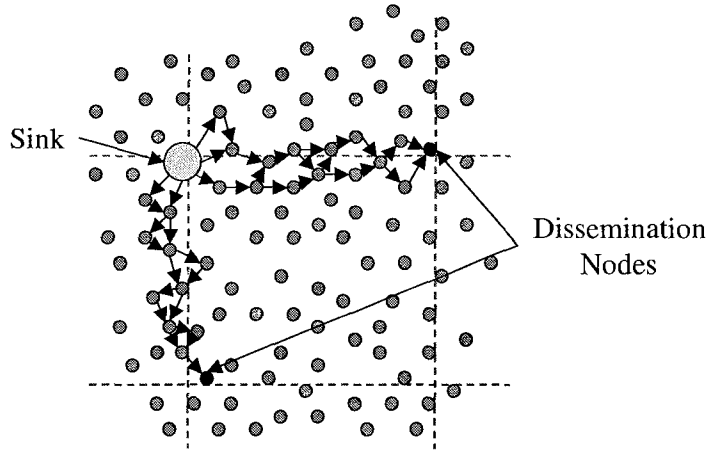


Figure 4.4: Task sending by a sink using simple geographic routing to reach DNs.

of the TASK message are defined in Table 4.6. Two compound types are also introduced in Tables 4.4 and 4.5. The REGION type simply defines a rectangular area as a geocast target. Fields of type TASKDEF describe a single task and all request information for the application layer. The type of the Cell_Size field of the TASK message and the X_Offset and Y_Offset fields from the REGION type definition are given as FLOAT for consistency with the previous assumption of a GPS type locationing system. The units in such a system are degrees of latitude and longitude and require floating point values to precisely address regions of the sensor field.

Table 4.4: Compound type REGION definition.

Field	**Type**	**Description**
P1	POSITION	A corner point of the region.
P2	POSITION	Corner point opposite P1.

The fields of the TASK message should be set by the sink in the following way. The Sink_Id is set to the unique identifier for the sink. Seq_Num is set to the next GGR sequence number for the sink, which is incremented when a new TASK message is propagated through the network. The randomly chosen cell size is written to the Cell_Size field. The

Table 4.5: Compound type TASKDEF definition.

Field	Type	Description
Target	REGION	Target region for this task.
Description	APP	Application specific task description.

Flood field is set to zero. The Rebuild field is set to zero unless the message was triggered by the event-driven reconstruction subsystem described in Section 4.4.2. Used_Energy is also set to zero since the sink is assumed to have infinite energy reserves. Hop_Cnt is set to zero by the sink because it is the first hop. Num is set to the number of tasks contained in the Tasks field.

Each entry in the Tasks field contains a single task description. The definition of the region of interest for the task is stored in the Target field. The Description field is set to the task description given by the application layer.

4.4.2 Event-Driven Grid Reconstruction

In order to spread the energy consumption load associated with the DN role, the sink must eventually build a new grid and assign the DN role to other sensors. GGR does this in an event-driven fashion similar to the scheme used by GRAB [17] and described in Section 2.3.9. In addition to its primary purpose of moving data from sensors to sinks, the DATA message is also used to trigger grid reconstruction by the sink. The fields of the DATA message are defined in Table 4.7. The Used_Energy and Hop_Cnt fields of each DATA message allow the sink to compute the mean energy use at DNs traversed by the DATA packet. The sink stores a running average of the mean energy consumption at Dissemination Nodes using the last GRID_REBUILD_WINDOW DATA messages received. Grid reconstruction takes place when this running average has increased by GRID_REBUILD_THRESHOLD percentage points since the grid was built.

A new TASK message is sent when it is determined that the grid needs to be rebuilt.

Table 4.6: TASK Message Definition.

Field	Type	Description
Sink_Id	UINT16	Assigning sink identifier (unique in this network).
Seq_Num	INT16	Sink's GGR sequence number.
Cell_Size	FLOAT	Size of cells in the grid being constructed.
Flood	BOOL	Set when inside the region of interest to indicate that flooding rather than grid construction should be used to propagate the message.
Rebuild	BOOL	Set when the purpose of this message is to rebuild the grid. The message is not actually delivered to sensor applications.
Used_Energy	UINT32	Additive used energy percentage at visited DNs.
Hop_Cnt	UINT8	Number of DNs visited.
Num	UINT8	Number of tasks in this message.
Tasks	TASKDEF LIST	List of Num task definitions.

The message contains all active tasks and has the Rebuild field set. The message is then sent as previously described, but using a new random cell size. In this way a new grid using different DNs is created. Figure 4.5 shows the impact of the changed cell size on DN selection. The need to balance the load of the DN role was discussed by Ye, *et al.* [6]. If no active tasks exist, no message is sent by the sink.

4.5 Intermediate Node Operation

An Intermediate Node (IN) is any node that is not the SNP Destination of a packet. INs perform simple geographic forwarding between DNs. No state information is required for geographic forwarding. All nodes are capable of performing IN functions.

Table 4.7: DATA Message Definition.

Field	Type	Description
Source	REGION	Source region from which this data originated.
Sink_Id	UINT16	Destination sink (unique in this network).
Seq_Num	INT16	Corresponding TASK GGR sequence number from sink.
Used_Energy	UINT32	Additive used energy percentage at traversed DNs.
Hop_Cnt	UINT8	Number of DNs traversed.
Data	APP	Application specific data.

When a packet is received, the SNP Sender and Seq_Num fields are checked to see if the packet is new. If the IN is within transmission range of the Destination, the message is passed to the routing sublayer for processing. The Last_Hop field is replaced with the INs own position and the packet is retransmitted if the node is closer to the Destination than the Last_Hop, but not within transmission range. Otherwise the packet is dropped.

4.6 Dissemination Node Operation

Dissemination Nodes (DNs) fill a critical role in the sensor network. All routing operations are performed by DNs. Routes formed by nodes with relatively low remaining energy resources will be avoided in favour of higher energy routes. In this way low power nodes are saved for the more critical sensing tasks.

Each DN stores a table of information used in routing decisions. The routing table format is shown in Table 4.8. The Seq_Num field stores the most recent GGR sequence number received from the sink. Upstream is a list of upstream Dissemination Nodes that can be used to reach the sink. In each Upstream entry, the Used_Energy field is the total used power of the path to the sink and Hop_Cnt is the number of hops to the sink along the given route. Other fields have the same meaning as identically named fields of the TASK

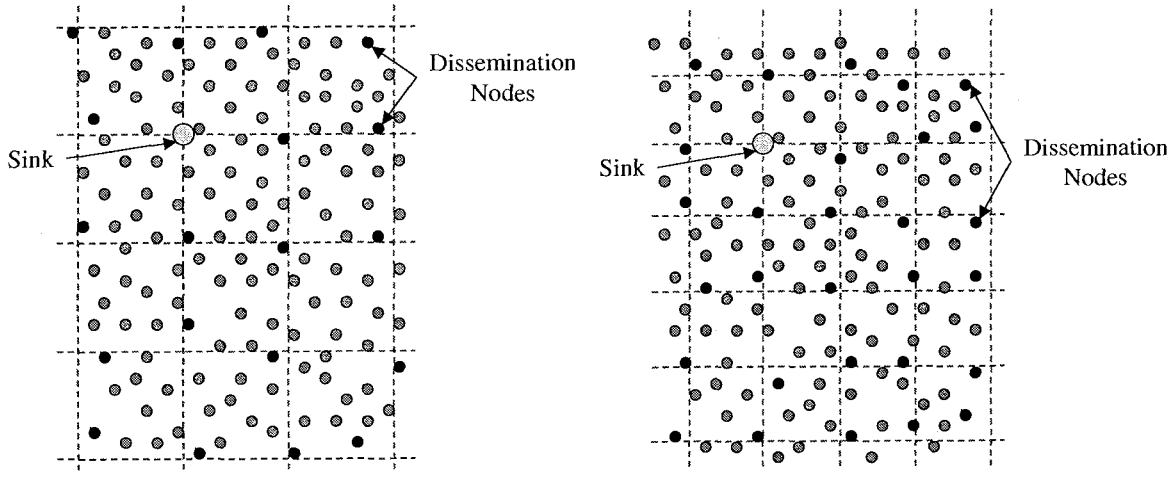


Figure 4.5: Different Dissemination Nodes are chosen by varying the cell size when re-building the grid. The same network is shown in both (a) and (b). Each is overlaid with a grid using different cell sizes to show the Dissemination Nodes chosen in each case.

message. It is possible that a sensor could become a DN for more than one sink. When such a scenario arises there will be multiple entries in the routing database. There is no procedure for removing entries from the routing database because this would require extra transmissions. Stale entries are detected with GGR sequence numbers and therefore are not problematic. The number of entries in the routing table is limited by the number of sinks in the network which has been assumed small.

4.6.1 Task Dissemination and Role Assignment

The handling of TASK messages is by far the most complex operation in the GGR system. TASK messages cause new upstream links to be added to the routing table. Each path has a cost calculated using Equation 4.1 where C is the path cost, E is the used energy along said path, H is the number of hops in the route and w is a weighting factor between zero and one. w allows network designers to give higher priority to either the used energy or the

Table 4.8: GGR routing table format.

Field	Type
Sink_Id	UINT16
Seq_Num	INT16
DN	BOOL
Upstream	UPSTREAM LIST

(a) Routing Table Format

Field	Type
Position	POSITION
Used_Energy	UINT32
Hop_Cnt	UINT8
Strikes	UINT8
Backed_Off	BOOL
Dead	BOOL

(b) UPSTREAM Field Format

hop count when paths are selected. The cost of a link is the cost of the best path available to the sink from the upstream DN. A new Upstream entry is added to the routing table only if the cost offered by the upstream DN is less than the lowest cost that can be offered to it. If a better offer can be made to an upstream DN for which a link exists in the routing table, then the Upstream entry is removed. In other words, no two DNs will ever have each other as Upstream entries in their respective routing tables.

$$C = (w \times E) + ((1 - w) \times H) \tag{4.1}$$

A received TASK message is processed as described by the pseudocode in Figure 4.6. Line 18 shows how message flooding continues within the area of interest when the Flood bit of the TASK message is set. Otherwise, the receiving node must first determine whether it is the closest node to the stated Dissemination Point at line 28. This action minimizes the number of nodes that take on the role of Dissemination Node.

Once it is known that this node is a DN or is inside the region of interest, a routing entry must exist for the sink referenced in the TASK message. A new routing table entry is created at line 49 of Figure 4.6 if none exists or the message is new. Old messages are considered stale and are ignored. The cost of the advertised link is examined at line 58

```
/* Global Variables */
RoutingTable rt;
Position myPos;
bool closerNodeFound;

/*
 * Process: handleTaskMsg
 * Description: Handles reception of a TASK message. The routing table
 *  is updated and the message is propagated as required.
 */
proctype handleTaskMsg(SNPHeader snpHeader, TASKMsg taskMsg)
{
  /* Local variables. */
  TASKMsg newTaskMsg;
  int oldMinCost;

  if
  :: (taskMsg.flood) ->
    if
    :: (/* This node is inside the region defined by
         * taskMsg.tasks[0].target. */) ->
      /* Continue flooding taskMsg inside region of interest. */
    :: else ->
      /* Route taskMsg geographically closer to snpHeader.destination. */
      goto handleTaskMsg_end
    fi
  :: else ->
    /* We have to figure out whether we're the closest node to the
     * Dissemination Point. */
    if
    :: (distance(myPos, snpHeader.destination) <
        distance(snpHeader.lastHop, snpHeader.destination)) ->
      /* Route taskMsg geographically closer to snpHeader.destination. */
      closerNodeFound = false;
      sleep(DN_DELAY);
      if
      :: (closerNodeFound) -> goto handleTaskMsg_end
      :: else -> skip
      fi
    :: else -> closerNodeFound = true
    fi
  fi;

  /* Check if this is a new, current or old message. */
  if
  :: ( (!rtEntryExists(taskMsg.sinkId)) ||
       (taskMsg.seqNum > rt[taskMsg.sinkId].seqNum) ) ->
    /* New message. */
    /* Create a new routing table entry for taskMsg.sinkId. At this point the
     * entry has no upstream links. Any old routing table entry for the sink is
     * replaced. */
  :: (taskMsg.seqNum < rt[taskMsg.sinkId].seqNum) ->
    /* Old message. */
    goto handleTaskMsg_end
  :: else -> skip /* Current message. */
  fi;

  /* Find out current minimum cost upstream link for this sink. */
  oldMinCost = COST(getMinCostLink(taskMsg.sinkId));

  /* Update the cost of the message sender in the routing table. */
  /* Create a new upstream link for taskMsg.sinkId in the routing table using
   * snpHeader.sender. */

  /* Add my own cost to the taskMsg before further propagating it. */
  newTaskMsg = taskMsg;
  newTaskMsg.usedEnergy = getMinCostLink(taskMsg.sinkId).usedEnergy +
```

Figure 4.6: PROMELA specification of actions taken during TASK message reception.

```
      getUsedEnergy();
    newTaskMsg.hopCnt = getMinCostLink(taskMsg.sinkId).hopCnt + 1;

    /* If I am a DN or this is not a flood TASK message then remove any upstream
     * links that have a greater or equal cost to what I am now offering. */
    if
    :: (!taskMsg.flood) || (rt[taskMsg.sinkId].DN) ->
      int i = 0;
      do
      :: (i < rt[taskMsg.sinkId].upstream.length) ->
        if
        :: ( COST(rt[taskMsg.sinkId].upstream[i]) >= COST(newTaskMsg) ) ->
          /* Remove rt[taskMsg.sinkId].upstream[i]. */
        :: else -> i++
        fi
      :: else -> break
      od
    :: else -> skip
    fi;

    /* If I was previously a DN, then only propagate the TASK message if I can
     * now make a better offer. Otherwise, only propagate the message if this is
     * the first non-flood TASK message I've received. */
    if
    :: (rt[taskMsg.sinkId].DN)
      if
      :: ( COST(getMinCostLink(taskMsg.sinkId)) < oldMinCost ) ->
        /* Send taskMsg to each adjacent DP on the grid except the sender of this
         * TASK message. Next, for each task definition in taskMsg.tasks where
         * part of taskMsg.tasks[i].target is inside an adjacent grid cell, flood
         * a TASK message containing only that task definition to the part of the
         * region of interest contained in the adjacent grid cell. Use geographic
         * forwarding to reach the sub-region if necessary. */
      :: else -> skip
      fi
    :: (!taskMsg.flood) ->
      /* Send taskMsg to each adjacent DP on the grid except the sender of this
       * TASK message. Next, for each task definition in taskMsg.tasks where
       * part of taskMsg.tasks[i].target is inside an adjacent grid cell, flood
       * a TASK message containing only that task definition to the part of the
       * region of interest contained in the adjacent grid cell. Use geographic
       * forwarding to reach the sub-region if necessary. */
    :: else -> skip
    fi;

    /* Update node state. */
    rt[taskMsg.sinkId].DN |= (!taskMsg.flood);

handleTaskMsg_end:
  skip
}
```

Figure 4.6: Continued

to determine whether a link to the sender should be created or updated. Only a link that would provide a lower cost path to the sink is added. In this way DNs have multiple low energy paths available to the sink. The node's own cost metrics are added to the information carried by the TASK message at line 65. TASK messages essentially advertise paths toward the sink. The cost of such a path is derived from the used energy of DNs on the path and the number of hops. In order to maintain loop freedom, DNs must remove links associated with paths that have a higher cost than what the DN itself is able to offer. Pruning these high energy paths is shown at line 71 of Figure 4.6.

The final action required in processing a TASK message is to forward the message on to adjacent Dissemination Points. The decision on whether or not to forward the TASK message begins at line 88 of Figure 4.6. The TASK message is propagated only if it offers a better cost than what this node had previously offered or if the message is new. The actual TASK propagation is a lot like the activity of a sink when sending a TASK message. Those procedures are detailed in Section 4.4.1. The TASK message is first forwarded to each adjacent DP except that from which it was received. The locations of adjacent Dissemination Points are calculated by adding and subtracting the Cell_Size field from the received SNP Destination. The new TASK message is not sent if the calculated Destination field is less than the transmission distance away from the Sender of the received TASK message.

4.6.2 Data Forwarding

Having established the reverse paths to the sink through Dissemination Nodes, data forwarding can occur. Data forwarding involves reception and forwarding of DATA messages, acknowledgment of those messages and reception of acknowledgments for messages sent. DATA message forwarding and acknowledgment is described by the pseudocode in Figure 4.7.

Upon receiving a DATA packet, the local routing table is checked for upstream links to the corresponding sink. This action is shown at line 19 of Figure 4.7. If no such link

```
/* Global Variables */
RoutingTable rt;
Buffer dataBuffer;
Timer ackTimer;

/*
 * Process: handleDataMsg
 * Description: Handles reception of a DATA message. The DATA message is
 *  forwarded on the cheapest upstream link. An ACK message is sent to the
 *  last hop.
 */
proctype handleDataMsg(SNPHeader snpHeader, DATAMsg dataMsg)
{
  /* Local variables. */
  ACKMsg ackMsg;

  /* Make sure I have an active upstream link for this sink. */
  if
  :: ((!rtEntryExists(dataMsg.sinkId)) &&
      (rt[dataMsg.sinkId].upstream.length != 0) &&
      (/* Not all upstream links for dataMsg.sinkId are dead. */)) ->
    /* See if there is any space in the dataBuffer. */
    if
    :: (!isFull(dataBuffer)) ->
      /* Fill in the energy metrics for the ACK message. */
      ackMsg.usedEnergy = getMinCostLink(dataMsg.sinkId).usedEnergy;
      ackMsg.hopCnt = getMinCostLink(dataMsg.sinkId).hopCnt;

      /* Buffer DATA message. */
      bufferAdd(dataMsg, dataBuffer);
      if
      :: (dataBuffer.length < BACKOFF_THRESHOLD) -> ackMsg.ackType = NORMAL
      :: else -> ackMsg.ackType = BACKOFF
      fi;

      /* Send ACK message. */
      send(ackMsg, snpHeader.sender);

      /* Send DATA message. */
      /* Update energy metrics of data message. */
      dataMsg.usedEnergy += getUsedEnergy();
      dataMsg.hopCnt++;
      do
      :: (/* All upstream links for dataMsg.sinkId are backedOff. */) ->
        sleep(BACKOFF_TIME);
        break;
      :: else -> skip
      od;
      send(dataMsg, getMinCostLink(dataMsg.sinkId).position);
      startTimer(ackTimer, ACK_TIME)
    :: else ->
      /* Send ACK message. */
      ackMsg.ackType = FULL;
      send(ackMsg, snpHeader.sender)
    fi
  :: else -> /* No active upstream links. */
    /* Send ACK message. */
    ackMsg.usedEnergy = 0;
    ackMsg.hopCnt = 0;
    ackMsg.ackType = REMOVE;
    send(ackMsg, snpHeader.sender)
  fi
}
```

Figure 4.7: PROMELA specification of actions taken following DATA message reception.

exists, the sender is directed not to send any further DATA messages to this node at line 57. Otherwise, the node checks if it has enough space to buffer the DATA message. Data messages need to be buffered until they are acknowledged in case retransmission is necessary. An acknowledgment informing the sender that the DATA message could not be buffered and requesting that further DATA messages not be sent for BACKOFF_TIME ms is sent at line 52 if no buffer space is available. At line 25 the DATA message is accepted for forwarding. An ACK message is created with updated cost metrics for this DN. The contents of the ACK message is given in Table 4.9. Piggybacking cost information on ACK messages refreshes routing table information so that more accurate decisions are made in choosing minimum energy paths. The type of ACK message to be sent is chosen at line 31. If there are BACKOFF_THRESHOLD or more messages buffered, the sender is asked to refrain from sending further messages for BACKOFF_TIME ms. The BACKOFF acknowledgment type provides a simple flow control mechanism.

Table 4.9: ACK Message Definition.

Field	**Type**	**Description**
Ack_Type	UINT8	Type of acknowledgment. Must be one of NORMAL, BACKOFF, FULL or REMOVE.
Sink_Id	UINT16	Sink for which the acknowledged DATA message was destined (unique in this network).
Ack_Seq_Num	INT16	SNP Sequence number of the received DATA message.
Used_Energy	UINT32	Additive used energy percentage at upstream DNs.
Hop_Cnt	UINT8	Number of DNs visited.

Once the DATA message has been acknowledged it can be forwarded closer to the sink. The node's own cost metrics are added to the DATA message at line 40 of Figure 4.7. At line 44 the node checks if all upstream links for this sink are currently being backed off. Sending of the DATA message is delayed for BACKOFF_TIME ms in that case. Otherwise the message is sent on the lowest cost active upstream link and the acknowledgment timer

is started for ACK_TIME ms. An active link is one that is not being backed off. DNs may also perform data fusion operations as discussed in Section 2.2. Aggregation of data is entirely the concern of the application layer. A plug-in architecture is suggested to allow the routing software to call application layer data aggregation functions.

At the sender, either an ACK will be received or the aforementioned ACK_TIME ms timer will expire. The procedure followed by the receiver of an ACK type message is given in Figure 4.8. At line 24 the upstream link is marked as *dead* if the acknowledgment type is REMOVE. Otherwise the upstream link's cost metrics are updated from the ACK message. A back-off timer is started if the acknowledgment type is either BACKOFF or FULL. No DATA messages will be sent to the upstream DN until this timer expires. Finally, starting at line 42, the acknowledgment timer is stopped and the acknowledged DATA message is either removed from the buffer or retransmitted depending on the acknowledgment type. If the DATA message is retransmitted it may be sent to a different upstream DN.

A series of operations are performed when the acknowledgment timer expires before an ACK message is received. These operations are shown in Figure 4.9. Starting at line 21, the strike count of the intended destination of the corresponding DATA message is incremented in the routing table. When this count reaches MAX_STRIKES the link is marked as dead and will not be used again for forwarding DATA messages. Routing loops are possible if the link is removed rather than being marked as dead. Finally the corresponding DATA message is retransmitted at line 34.

4.7 Task Dissemination Within the Region of Interest

Upon reaching a region of interest, the TASK message must be disseminated to all sensors in the defined area. A different type of routing is employed when the TASK message reaches a DN whose surrounding cells contain part of the Target. Flooding is the simplest and most effective way to distribute the TASK message to all nodes within the target region. Flooding was described in Section 2.3.1.

```
/* Global Variables */
RoutingTable rt;
Buffer dataBuffer;
Timer ackTimer;
Timer backoffTimer;

/*
 * Process: handleAckMsg
 * Description: Handles reception of an ACK message.
 */
proctype handleAckMsg(SNPHeader snpHeader, ACKMsg ackMsg)
{
  /* Local Variables */
  UpstreamEntry upstream;
  RoutingTableEntry entry;

  /* Get the upstream routing table entry for the sender. */
  entry = getRoutingEntry(rt, ackMsg.sinkId);
  if
  :: (upstreamEntryExists(entry, snpHeader.sender)) ->
    upstream = getUpstreamEntry(entry, snpHeader.sender);

    if
    :: (ackMsg.ackType == REMOVE) -> upstream.dead = true
    :: else ->
      upstream.usedEnergy = ackMsg.usedEnergy;
      upstream.hopCnt = ackMsg.hopCnt;
      upstream.strikes = 0;
      upstream.dead = false;

      if
      ::((!upstream.backedOff) &&
         ((ackMsg.ackType == BACKOFF) || (ackMsg.ackType == FULL))) ->
        upstream.backedOff = true;
        startTimer(backoffTimer, BACKOFF_TIME)
      :: else -> skip
      fi
    fi
  :: else -> skip
  fi;

  /* Stop the ackTimer for the acknowledged message and handle the
   * dataBuffer. */
  stopTimer(ackTimer);
  if
  :: ((ackMsg.ackType == NORMAL) || (ackMsg.ackType == BACKOFF)) ->
    /* Remove corresponding DATA message from dataBuffer. */
  :: ((ackMsg.ackType == FULL) || (ackMsg.ackType == REMOVE)) ->
    /* Send buffered DATA message again using the same procedures as
     * handleDataMsg(...). */
  fi
}
```

Figure 4.8: PROMELA specification of actions taken following ACK message reception.

```
/* Global Variables */
RoutingTable rt;
Buffer dataBuffer;
Timer ackTimer;
Timer backoffTimer;

/*
 * Process: handleAckTimeout
 * Description: Handles expiration of the ackTimer.
 */
proctype handleAckTimeout()
{
  /* Local Variables */
  UpstreamEntry upstream;
  RoutingTableEntry entry;

  /* Get the upstream entry for the intended destination of the DATA message. */
  entry = getRoutingEntry(rt, ackMsg.sinkId);
  upstream = getUpstreamEntry(/* Intended destination of DATA message. */);

  /* Mark a strike against the intended destination. */
  if
  :: (!upstream.dead) ->
    upstream.strikes++;
    /* Mark the upstream link as dead if it strikes out. */
    if
    :: (upstream.strikes == MAX_STRIKES) ->
      upstream.dead = true
    :: else -> skip
    fi
  :: else -> skip
  fi;

  /* Send corresponding buffered DATA message again using the same procedures as
   * handleDataMsg(...). */
}
```

Figure 4.9: PROMELA specification of actions taken following an acknowledgment time-out.

As defined in Section 4.4.1, a region is specified as a rectangular area. The region of interest is divided into sub-regions by the virtual grid structure. A single DN is only concerned with those sub-regions that are contained in surrounding cells. Sensors within those sub-regions may use the DN as the first hop when sending observation data to the sink.

An extra transmission is sometimes required by DNs further than those described in Section 4.4.1 and 4.6.1 for each task for which some part of the target region is contained within a surrounding cell. In this case the Flood field is set to one. The new TASK message contains only the tasks destined for the target region. One such message is sent to each adjacent cell that contains a part of the target region. In each such message the Target is set to the sub-region contained within the destination cell. Flooding starts immediately from the DN when the DN is located inside the target region. Task dissemination by a DN near to a target region is shown in Figure 4.10.

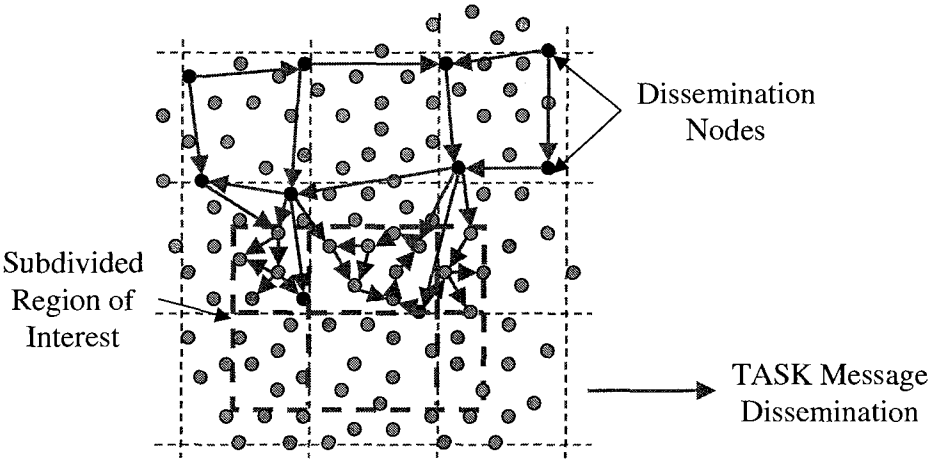


Figure 4.10: The TASK message is sent directly to sub-regions of the target when it is less than one cell size from the DP. The sub-regions are defined by the existing grid. The message is then flooded within the sub-region.

A simple trick is needed to flood data in a network where geographic routing is employed. The Destination field of the SNP header is set to the position of the sending node.

Otherwise the packet will not be accepted for processing by higher layers in those nodes that are not within transmission range of the SNP Destination.

4.8 Sensing Node Operation

Any node that is included in the Target region of a received TASK message with the Flood bit set takes on the role of Sensing Node (SN). Forwarding of these flood messages proceeds as explained in Section 4.7.

Received TASK messages are processed as defined in Section 4.6.1. All offered upstream links are accepted by SNs that are not DNs because these nodes cannot form routing loops. TASK messages with the Rebuild flag set are not passed to the application layer. DATA messages originate from the SN application layer as defined by the TASK Description, but are held at the network layer for a random period of time up to MAX_SEND_DELAY ms. The delay helps to spread the DATA message congestion from the region of interest in time. The sending of DATA messages by SNs involves the same DATA, ACK exchange described for forwarding by DNs in Section 4.6.2.

4.9 Configuration Parameters

A number of configuration parameters were defined in preceding sections. Those values are summarized in Table 4.10 along with factors that should be considered in their setting.

Table 4.10. *System Configuration Parameters.*

Parameter	Description	Determinants
MIN_CELL_SIZE	Minimum cell size that can be selected by a sink.	Sensor density, transmission range and network area. See Section 4.4.1.

Parameter	Description	Determinants
MAX_CELL_SIZE	Maximum cell size that can be selected by a sink.	Sensor density, transmission range and network area. See Section 4.4.1.
GRID_REBUILD_WINDOW	Number of samples used by the sink in deciding when to build a new grid.	High values will cause slower reaction to depletion of energy resources in the network. Low values could cause grid rebuilding to be triggered by a sporadic situation. See Section 4.4.2.
GRID_REBUILD_THRESHOLD	Percentage value used by the sink in determining whether to build a new grid.	High values will result in higher variance of energy levels among sensors. Lower values will use more energy due to the higher frequency grid reconstruction. See Section 4.4.2.
w	Weighting parameter used in the path cost function given in Equation 4.1. $0 \leq w \leq 1$	Higher values give a greater importance to total path used energy while low values put a higher priority on the number of hops in the route. See Section 4.6.1.

Parameter	Description	Determinants
DN_DELAY	Delay used by DNs to see if another DN nearer to the DP can be discovered.	Depends on message transmission time and sensor density. See Section 4.6.1.
ACK_TIME	Time used by DNs to wait for an ACK message after sending a DATA message.	Depends on message transmission time, MAX_CELL_SIZE and sensor density. See Section 4.6.2.
MAX_STRIKES	An upstream link is marked as dead after this many consecutive DATA messages go unacknowledged.	Setting of the MAX_STRIKES parameter depends on the efficiency of the MAC layer protocol. Higher values should be used for an unreliable MAC layer. See Section 4.6.2.
BACKOFF_TIME	DNs back-off from sending on an upstream link for this amount of time after a BACKOFF or FULL type ACK is received.	Depends on the size of the data buffers and the threshold at which a BACKOFF type ACK is triggered. See Section 4.6.2.

Parameter	Description	Determinants
MAX_SEND_DELAY	DATA messages passed from the application layer are held at the network layer for a random period up to this value. The action spreads DATA message congestion in the region of interest over time.	This value depends on the MIN_CELL_SIZE and MAX_CELL_SIZE parameters as those dictate the size of the cells from which data will be gathered. See Section 4.8.

Chapter 5

Analysis

Formal analysis of the GGR protocol is given in this chapter. Correctness properties of the protocol are proven using the SPIN model checker. The second part of the chapter provides a formal analysis and comparison of GGR and the TTDD protocol.

5.1 Protocol Verification

5.1.1 Overview of SPIN

Any system is deemed correct if it meets its design requirements. However, showing a distributed system to be correct is extremely difficult due to the high level of complexity. Testing cannot show the absence of bugs and an attempt at a mathematical proof of a concurrent system quickly becomes overwhelmingly difficult for any mathematician. Implementations of these intricate systems cannot be completely verified because the state space and computational power required far exceed what any machine can currently offer. Engineering projects often use models to work through complex design problems. Properties of distributed systems can also be verified mechanically by constructing abstract models.

Model checking is the technique used to exhaustively verify simplified models. The models appear as small programs written in specialized programming languages. The modeling language provides facilities for abstract specification and definition of requirements properties. A model checker is a tool for exhaustively checking the requirements under

all possible executions of the model. The model checker can easily find design problems related to deadlock, live-lock and constraint violations that would be virtually impossible to uncover in the full implementation. It is up to the designer to abstract a simple, but sufficient model of the system such that the proof is accurate and can be completed in a reasonable amount of time on the available computing resources.

SPIN (Simple PROMELA INterpreter) is a powerful model checking tool that has been widely used for verification of communication systems. PROMELA (PROcess MEta LAnguage) is the model specification language used by SPIN. SPIN was mainly developed at Bell Labs in the eighties and nineties. In 2002 the ACM (Association for Computing Machinery) recognized SPIN with the prestigious *Software System Award.* Today the SPIN software is freely available from the Web [20] and active development continues. During this time SPIN has amassed a broad user base in both academia and industry. The SPIN system is explained in greater detail in Gerard Holzmann's book [21].

SPIN has two modes of operation, simulation and verification. Simulation mode is used for debugging purposes. A single execution of the model is explored in simulation mode. A graphical interface shows message sequence charts and finite automata representations to aid in model debugging. SPIN's verification mode explores the entire state space of the modeled system, checking correctness criteria throughout. The graphical interface includes tools for development of the correctness properties. A few verification options allow tuning for available computing resources with memory, speed and proof completeness as possible trade-offs.

5.1.2 GGR Model

The SPIN model checker was used to prove a number of important characteristics of the GGR protocol. Our abstracted model focuses on the grid construction and task dissemination portions of the protocol such that the state space remains manageable for the verifier. The model provides an unambiguous specification of these portions of the GGR protocol. The PROMELA model is given in Appendix A.1.

Our model covers the following parts of the protocol. TASK message propagation is modeled because the robustness of the grid construction procedure is the most complex part of the algorithm. It is also important to prove the correctness of the route discovery part of our scheme. The model includes ACK message timeouts and REMOVE type acknowledgments as these are the events that can disrupt established routes. DATA messages only affect route priorities, but cannot establish or remove paths from the network. For this reason DATA messages are not included in the model.

Assertions embedded in the PROMELA code were used to verify that the following properties of the GGR algorithm hold under all possible conditions:

- Loop freedom.
- Internal routing state cannot be corrupted.
- Correct sequence number use.
- Various consistency properties remain stable.
- No transaction can result in an unexpected end-state.

Although every effort has been made to abstract the model such that it defines the smallest sufficient model of the actual system, the state space remains large. The use of a powerful computer was required in order to exhaustively verify the GGR protocol. The Minerva supercomputer at the University of Victoria Research Computing Facility [23] provided the required resources. Minerva allows up to eight gigabytes of memory for each process.

The output of the exhaustive verification on Minerva is given in Figure 5.1. Lines 1 through 15 of the output show the progression of the model checker during the verification. The type of verification is shown as a full state-space search for assertion violations and invalid end states on lines 19 through 23. The output then shows information about the state machine used to represent our model and how it was handled by the verifier on lines 25 through 31. Most importantly, line 25 shows that the verifier has found zero errors. It can also be observed on line 26 that the finite state machine has over fifteen million states.

The section of output on lines 33 to 39 provides the memory usage statistics. Line 39 shows that the verification required roughly 2.7 GB of memory. The messages on lines 41 through 49 show that some states in our model cannot be reached by any sequence of events. These are not error messages. Our verification required approximately twenty minutes on the Minerva supercomputer.

5.2 Overhead Analysis

An analysis of the communication overhead and state complexity of GGR is given in this section. GGR is compared to the TTDD protocol in which GGR has its roots and which is presented in Fan Ye's paper [6]. The efficiency and scalability of the protocols are compared. In an effort to keep the analysis simple and easy to follow we focus on the worst-case overhead for each protocol. We also attempt to do an analysis similar to that done by Fan Ye in his paper [6] for comparison purposes.

5.2.1 Model and Notations

Our analysis is based on a sensor network of N devices, covering an area A. The sensor field is assumed to be square in order to simplify the analysis. The nodes are uniformly distributed in this area with a constant density d. k sinks are distributed through the sensor field. One data packet is received from each sensor in a region of interest. As previously mentioned, both GGR and TTDD use a grid mechanism for efficient communication. Each grid cell has an area of α^2 and contains $n = d \cdot \alpha^2$ sensor nodes. The total area for regions of interest in the network is R. Our worst-case analysis assumes that all sinks are interested in data from all regions of interest in the network.

TTDD suggests a scheme whereby data from sources within close proximity to one another would elect a single sensor to act as the data source for the group. Ye's paper [6] does not give specific details on this portion of the protocol, but the idea would decrease the state complexity and communication overhead by nearly a constant factor. We ignore this

```
Depth=     726 States=   1e+06 Transitions= 4.61814e+06 Memory= 307.581
Depth=     726 States=   2e+06 Transitions= 1.14834e+07 Memory= 480.637
Depth=     726 States=   3e+06 Transitions= 1.68289e+07 Memory= 653.591
Depth=     726 States=   4e+06 Transitions= 2.25637e+07 Memory= 826.749
Depth=     726 States=   5e+06 Transitions= 2.73744e+07 Memory= 999.600
Depth=     726 States=   6e+06 Transitions= 3.32824e+07 Memory= 1172.554
Depth=     726 States=   7e+06 Transitions= 3.86489e+07 Memory= 1345.303
Depth=     726 States=   8e+06 Transitions= 4.4228e+07 Memory= 1518.154
Depth=     726 States=   9e+06 Transitions= 4.88154e+07 Memory= 1691.312
Depth=     726 States=   1e+07 Transitions= 5.44683e+07 Memory= 1864.266
Depth=     726 States= 1.1e+07 Transitions= 6.03452e+07 Memory= 2037.219
Depth=     726 States= 1.2e+07 Transitions= 6.48754e+07 Memory= 2210.071
Depth=     726 States= 1.3e+07 Transitions= 6.92044e+07 Memory= 2382.922
Depth=     726 States= 1.4e+07 Transitions= 7.40321e+07 Memory= 2555.671
Depth=     726 States= 1.5e+07 Transitions= 7.86025e+07 Memory= 2728.522
(Spin Version 4.1.2 -- 21 February 2004)
  + Partial Order Reduction

Full statespace search for:
  never claim           - (not selected)
  assertion violations  +
  cycle checks        - (disabled by -DSAFETY)
  invalid end states  +

State-vector 172 byte, depth reached 726, errors: 0
1.50417e+07 states, stored
6.37423e+07 states, matched
7.87839e+07 transitions (= stored+matched)
9.88261e+08 atomic steps
hash conflicts: 1.48575e+07 (resolved)
(max size 2^24 states)

Stats on memory usage (in Megabytes):
2767.669  equivalent memory usage for states (stored*(State-vector + overhead))
2603.587  actual memory usage for states (compression: 94.07%)
  State-vector as stored = 161 byte + 12 byte overhead
134.218 memory used for hash table (-w24)
0.039   memory used for DFS stack (-m800)
2735.690  total actual memory usage

unreached in proctype injector
  (0 of 30 states)
unreached in proctype environment
  (10 of 106 states)
unreached in proctype sensor
  (37 of 388 states)
unreached in proctype :init:
  line 661, "pan_in", state 83, "-end-"
  (12 of 83 states)
```

Figure 5.1: Output of the SPIN model checker for an exhaustive verification of the GGR routing protocol.

feature of TTDD in our analysis due to the lack of detail and the fact that we are primarily interested in the scaling of the protocols rather than constant factors.

The analysis follows a pattern of developing mathematical expressions of overhead and then plotting those expressions for comparison purposes. The graphs are based on a typical sensor network scenario where $k = 3$, $N = 1000$, $R = 1$ km^2, $A = 10$ km^2, $d = 100$ nodes/km^2 and $\alpha = 0.6$ km. N and R are varied to show the scalability of GGR and TTDD. The area, A, is varied automatically with the number of nodes, N, in order to maintain the density, d. An appropriate value for the density was determined through experimentation using our simulation environment to be described in Chapter 6.

5.2.2 Communication Overhead

The communication overhead analysis presented in this section studies the number of transmissions required for the two primary aspects of protocol operation, task dissemination and data collection. Expressions for the overhead of each protocol are developed, followed by comparison based on our example network.

Communication of task messages in GGR requires grid construction followed by flooding within the region of interest. The flooding overhead is proportional to the number of sensors in the region of interest and is calculated as $N\frac{R}{A}$. There are N/n cells in the grid. A square sensor field is assumed so there are $\sqrt{N/n}$ cells along each edge of the network. Every cell has a Dissemination Node at each corner, so there are $\sqrt{N/n}+1$ DNs along each edge of the network and $(\sqrt{N/n}+1)^2$ Dissemination Nodes in total. Assuming the simple case, each DN makes three transmissions during grid construction. Thus construction of a grid requires $3 \cdot (\sqrt{N/n}+1)^2$ transmissions. These actions are performed by all sinks in the worst case resulting in task dissemination overhead of:

$$k \cdot (N\frac{R}{A} + 3 \cdot (\sqrt{N/n}+1)^2) \tag{5.1}$$

TTDD includes no facilities for *mission updates*, as task dissemination is referred to

by Ye, *et al.* [6]. The only way to disseminate a new task to nodes in the network is to flood the entire network. It is assumed that only one sink submits a new task to the network so the mission update overhead for TTDD is proportional to the number of sensors in the network, N.

Figure 5.2 shows comparisons between GGR and TTDD for task dissemination overhead. In Figure 5.2(b) it can be seen that GGR is more efficient when the region of interest is small relative to the area of the sensor field, but less efficient as the size of the region of interest grows. The TTDD overhead is constant in Figure 5.2(b) because it only depends on the number of nodes in the network and not on the size of the region of interest. The GGR overhead is predominantly lower in Figure 5.2(a) because it does not have to flood the entire network.

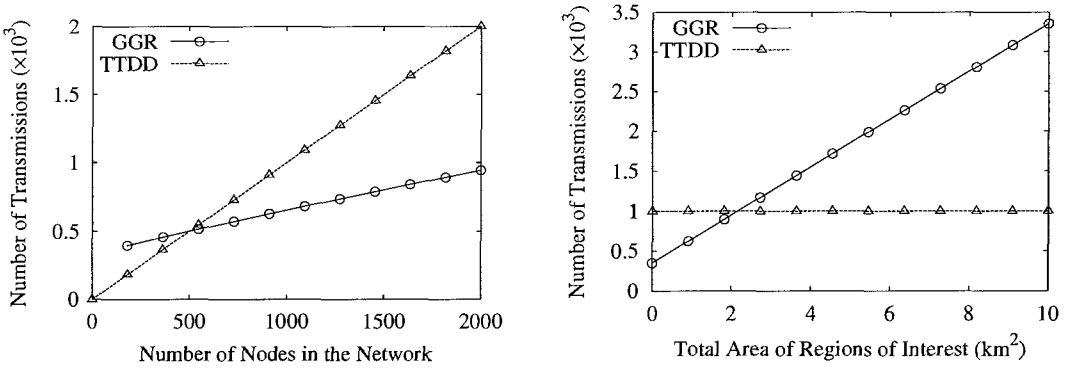


Figure 5.2: Comparison of communication overhead during task dissemination in GGR and TTDD networks. The number of transmissions are calculated (a) as the network size increases, and (b) as the area of regions of interest in the network is expanded.

Data transmissions use the grid for communication in both GGR and TTDD. Our analysis disregards the possibility of data aggregation in order to simplify the calculations. There are roughly $\sqrt{N}$ nodes on each side of the sensor field. Communication of a message in the network along the straight-line path requires at most $c\sqrt{N}$ transmissions, where

$0 < c \leq \sqrt{2}$. The number of transmissions needed to send a message along a grid path is increased by a further factor of $\sqrt{2}$ in the worst case. Thus the number of transmissions to send a message along a grid path is bounded by $\sqrt{2N}c$. In GGR a grid communication is required for each data message to each sink. There are $N\frac{R}{A}$ data transmissions from data sources. The resulting overhead expression is:

$$\sqrt{2N}cN\frac{R}{A}k \tag{5.2}$$

TTDD data transmission requires grid construction for each data source, each requiring $3 \cdot (\sqrt{N/n} + 1)^2$ transmissions. The protocol then uses local flooding at each sink within an area the size of a cell for each data source. Finally, two grid communications are made, one for a sink query and the second for the data itself. These grid communications require $2 \cdot \sqrt{2N}c$ transmissions for each sink/source pair. The overhead for TTDD data communications is:

$$N\frac{R}{A}(3 \cdot (\sqrt{N/n} + 1)^2 + k \cdot (n + 2 \cdot \sqrt{2N}c)) \tag{5.3}$$

The overhead incurred by the two protocols during the data communication part of the transaction is shown in Figure 5.3. GGR incurs less overhead from this operation because it has already constructed a grid and has already recorded interest from the various sinks. TTDD, on the other hand, must construct a grid for each data source, receive interest queries from sinks and finally send data to the sinks.

Following from Equations 5.1 and 5.2, the expression for the total communication overhead incurred by GGR is:

$$k \cdot (N\frac{R}{A} + 3 \cdot (\sqrt{N/n} + 1)^2) + \sqrt{2N}cN\frac{R}{A}k \tag{5.4}$$

Similarly, the total communication overhead for TTDD is derived from the task dissemination overhead, N, and the data transmission overhead given in Equation 5.3. The resulting equation is given as:

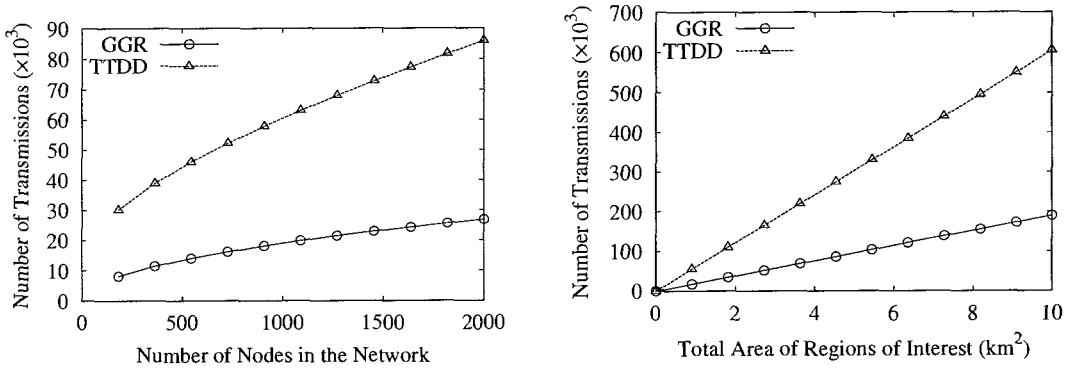


Figure 5.3: Communication overhead during data communication is compared for GGR and TTDD. Network size is varied in (a). The total area of regions of interest in the sensor field is varied in (b).

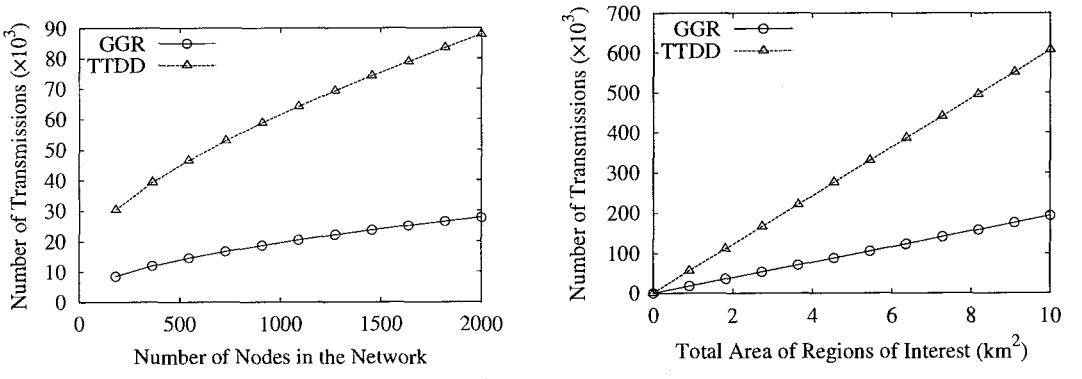
$$N + N\frac{R}{A}(3 \cdot (\sqrt{N/n} + 1)^2 + k \cdot (n + 2 \cdot \sqrt{2Nc})) \tag{5.5}$$

The dominant factor in both Equations 5.4 and 5.5 is by far the number of transmissions required during data communication. As a result, the plots in Figure 5.4 strongly resemble those from Figure 5.3. Also, GGR shows less total overhead than TTDD because it has more efficient data communication.

5.2.3 State Complexity

In this section the state complexity of GGR and TTDD is analyzed. The goal is to show the scalability of each protocol. Thus, we are only concerned with the main factors that contribute to the growth of routing information. Routing information is stored in Dissemination Nodes and data sources (sensors within the region of interest). Expressions for the state complexity of each protocol are developed followed by comparison based on our example network.

A GGR Dissemination Node stores state information for each Sink for which it is a DN.



(a) (b)

Figure 5.4: Total communication overhead incurred by GGR and TTDD. Again, (a) varies the network size. Region of interest area is varied in (b).

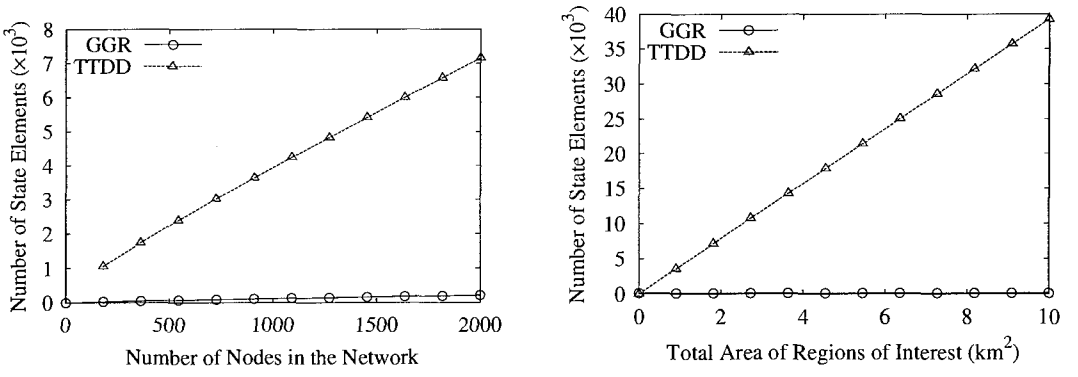
Based on the previous calculation of $(\sqrt{N/n}+1)^2$ Dissemination Nodes for each sink in the network, we calculate the state information related to DNs in GGR as:

$$k \cdot (\sqrt{N/n}+1)^2 \tag{5.6}$$

Similarly, TTDD uses $(\sqrt{N/n}+1)^2$ Dissemination Nodes for each grid. However, TTDD creates an independent grid for each data source. Again, as previously noted, there are $N\frac{R}{A}$ sensors within the region of interest. Thus, the number of Dissemination Nodes in a sensor field running TTDD is:

$$N\frac{R}{A} \cdot (\sqrt{N/n}+1)^2 \tag{5.7}$$

The contribution toward total state complexity by Dissemination Nodes is shown in Figure 5.5. The routing information stored by GGR is insignificant compared to that of TTDD and is barely visible on the plots. The reason for this is that as the network grows larger in Figure 5.5(a), DNs are added to all grids. However, TTDD uses far more virtual grids than does GGR. TTDD creates more DNs as the number of data sources increases. This is the reason for the linearly increasing state complexity of TTDD in Figure 5.5(b).



(a) (b)

Figure 5.5: State complexity stored by Dissemination Nodes in GGR and TTDD networks. The number of state elements shown (a) as the network size increases and (b) with expanding regions of interest.

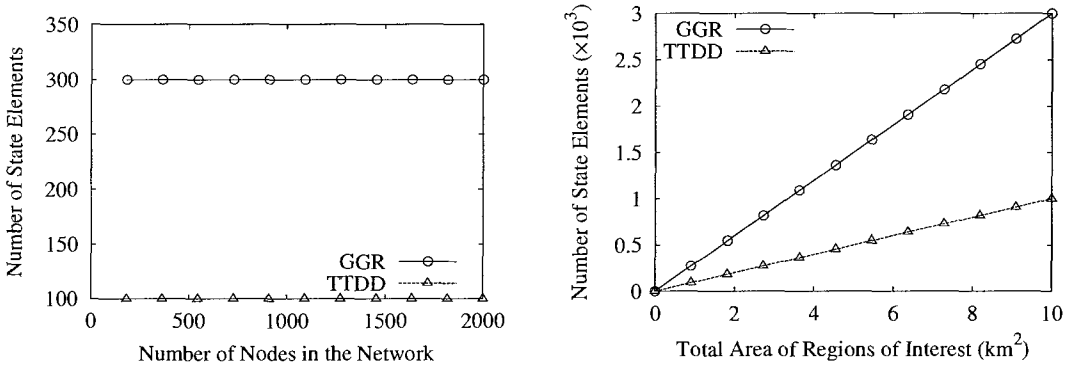
The other significant source of state complexity for GGR and TTDD is that used by data sources. As shown above, there are $N\frac{R}{A}$ data sources in the network. GGR data sources store state information related to each sink that has shown interest. Under the worst-case model this results in the following expression:

$$k \cdot N\frac{R}{A} \tag{5.8}$$

TTDD data sources do not store information about each sink so the complexity is less by a factor of k:

$$N\frac{R}{A} \tag{5.9}$$

Figure 5.6 compares Equations 5.8 and 5.9. Neither of the protocols is affected by the changing network size as this change does not affect the number of data sources or the number of sinks. The two protocols react to the changing R value in a similar fashion, but GGR stores k times as much information because it stores information about each sink whereas TTDD does not.



(a) (b)

Figure 5.6: Amount of state information stored by data sources. The number of nodes is increased in (a). The area covered by regions of interest is shown in (b).

Based on Equations 5.6 and 5.8, the expression for the total state complexity of GGR is:

$$k \cdot (\sqrt{N/n} + 1)^2 + k \cdot N\frac{R}{A} \tag{5.10}$$

In the same way, TTDD's total state complexity is developed from Equations 5.7 and 5.9 as:

$$N\frac{R}{A} \cdot (\sqrt{N/n} + 1)^2 + N\frac{R}{A} \tag{5.11}$$

Figure 5.7 plots Equations 5.10 and 5.11. The state information stored by Dissemination Nodes is the largest contributing factor to the total state complexity for both GGR and TTDD. GGR shows reduced total state complexity because it stores far less information in Dissemination Nodes. The reason for this is that GGR creates a grid for each sink rather than for each data source.

Overall GGR is a much more scalable protocol than TTDD and has significantly less overhead. Both the communication overhead and state complexity are greatly reduced using the grid per sink approach of GGR rather than a grid per data source as is done by

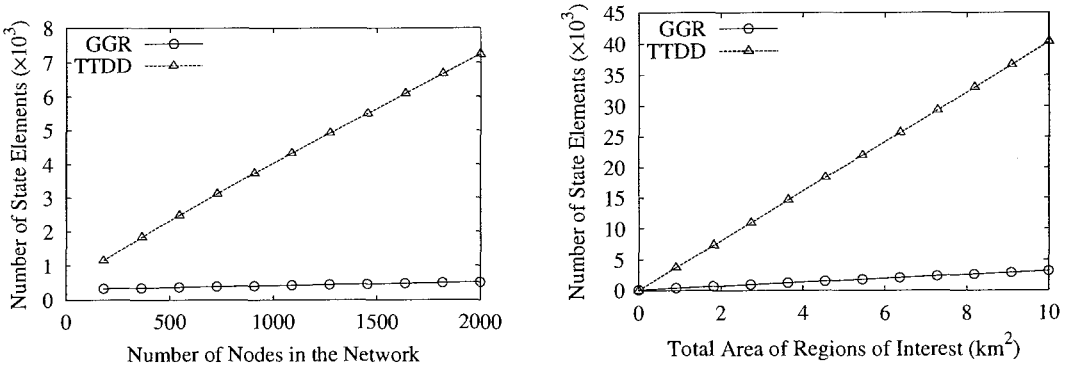


Figure 5.7: Total state complexity of GGR and TTDD. Network size is varied in (a). The region of interest area is expanded in (b).

TTDD. These theoretical results predict that GGR will be more energy efficient than TTDD because it requires fewer transmissions to carry out the same data collection. The results also indicate better robustness for GGR since less state information is stored and will thus be lost in the event of failure.

Chapter 6

Prototyping and Simulation

6.1 Simulation Environment

6.1.1 Selection

There are a number of network simulation packages available to the academic researcher, each with its strengths and weaknesses. Our selection was based on the following factors:

- Documentation and technical support.
- Ease of use.
- Simulation realism.
- Popularity within research community.
- Simulation scalability.

Documentation, technical support and ease of use were important factors to ensure short development time. The platform should also model reality as closely as possible. Popularity within the research community was a factor both for community support and so that future work can extend our implementation. Finally, a major factor was scalability. The ability to simulate large networks is key to accurately evaluating the performance of a sensor network.

We considered the NS-2 [24], OPNET [25] and GloMoSim [22] network simulators for our performance evaluation. NS-2 is a popular network simulator within the academic community with an established support network. NS-2 was designed for wired networks

with wireless support added later by the Monarch Research Group [26]. As a result NS-2 has become quite complicated. In addition, NS-2 does not accurately simulate wireless communication because it does not take radio interference into account. OPNET is a quality simulation package, but does not have the following within the research community that NS-2 has acquired.

After careful consideration, GloMoSim was chosen for its simplicity, realism and scalability. GloMoSim is the Global Mobile Information Systems Simulation Library for simulation of wireless and wired communication networks. The project is actively developed by the Parallel Computing Laboratory at the University of California at Los Angeles. The simulator is based on the discrete-event simulation environment provided by the Parsec project [27], also created at UCLA's Parallel Computing Laboratory. The environment is based on the C programming language and can run on many different hardware architectures and operating systems. GloMoSim is the newest of the three platforms considered and has quickly become popular among researchers. A couple of tutorials are available online [28, 29], but their focus is on setting up a simulation using the provided modules rather than extending GloMoSim with new protocols. The GloMoSim mailing list and investigation of the source code is currently the best way to familiarize oneself with how to extend GloMoSim. The tool has a clean design that makes it easy to understand despite the lack of information. GloMoSim was designed from scratch for wireless network simulation and accurately models radio signal propagation. Finally, GloMoSim was built with scalability as a primary objective. Large sensor networks can be modeled using the GloMoSim tool.

6.1.2 Operation

GloMoSim is a discrete event simulator. Simulation execution is driven by events in the system under study. These events come in the form of messages either sent locally or to another node in the simulated network. A message sent to a remote node generally represents a communication while a message sent locally signals the expiration of a timer. In this way the simulation proceeds through a series of message receive and timeout events.

The user can easily define new message types.

GloMoSim uses a global configuration file from which system settings such as protocol selection, network parameters and radio specifications are read. Application traffic is also read from a separate configuration file, which essentially drives the whole simulation. These configuration files are read at startup. Simulation continues until all traffic has passed or a configured maximum simulation time has expired.

Adding new protocols to GloMoSim is quite straightforward for anyone with C programming experience. Although GloMoSim is built using the Parsec programming language and compiled with the Parsec compiler, the actual code written by the researcher is entirely C. No knowledge of Parsec is required for development in GloMoSim. GloMoSim has a layered architecture to which new protocols can be added without impacting other layers. The included modules at each layer are given in Table 6.1. The simplest way to add a new protocol is to copy and replace one of the included protocols at the same layer.

Table 6.1: GloMoSim layers and included modules.

Layer	**Modules**
Mobility	Random waypoint, Random drunken, Trace based
Radio Propagation	Two ray, Free space
Radio Model	Noise accumulating, No noise
Packet Reception Model	SNR bounded, BER based with BPSK/QPSK modulation
Data Link (MAC)	CSMA, IEEE 802.11, MACA
Network (Routing)	IP with AODV, Bellman-Ford, DSR, Fisheye, LAR scheme 1, ODMRP, WRP
Transport	TCP, UDP
Application	CBR, FTP, HTTP, Telnet

A graphical visualization tool is included with the GloMoSim distribution. The tool is written in Java and can be run on any of the supported platforms. The visualization tool shows all nodes in the network and their interactions with varying levels of detail adjustable

by the user. It is of great use in debugging and understanding simulations.

6.2 Implementation Design

The layered architecture offered by GloMoSim fits most communication protocol implementations including the GGR protocol stack depicted in Figure 4.1. A logical view of the system implementation extends this model and is given in Figure 6.1, which shows the sink and sensor applications, the two sublayers of the network layer and the Energy Management Plane. The MAC and physical layers are provided by the base GloMoSim distribution, but were modified slightly to work with the Energy Management Plane and the SNP sublayer. The SNP sublayer performs geographic routing operations as described in Section 4.2 and the GGR sublayer is charged with the higher level routing explained in the rest of Chapter 4. The Energy Management Plane and the application layer are described further in Sections 6.2.1 and 6.2.3.

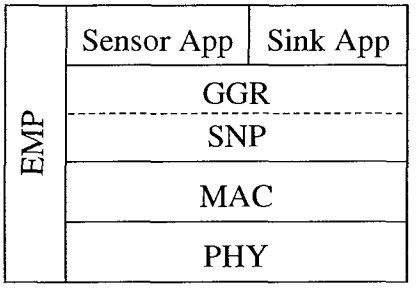


Figure 6.1: GGR implementation design.

A number of factors made the implementation of GGR in the GloMoSim framework somewhat unique. First, the only network layer protocol included with the GloMoSim distribution is IP. The SNP sublayer from the GGR protocol stack replaces IP at the network layer. Unfortunately the separation of concerns between the IP module and other GloMoSim modules was not what it might have been and modification of the MAC layer was necessary for inclusion of the SNP module. Second, GGR does not include a transport

layer. Fortunately the layered GloMoSim architecture allows our applications to simply make calls directly to the GGR sublayer, bypassing the transport layer completely. Finally, as it is an important factor in real-world operation, sensor failure can be modeled through our implementation and affects the application and network layers. Our failure model is described in Section 6.2.2 and works through the sensor application.

6.2.1 Energy Model

The GloMoSim distribution includes only a partially complete energy model. Our model works by measuring the time spent transmitting at the radio layer during the simulation. The radio is assumed to be receiving at other times. Values for the radio characteristics were taken from actual wireless sensing devices currently available from Crossbow Technology Inc. [30]. We were able to model the Crossbow MPR400CB processor and radio platform with information taken from the Crossbow MICA2 data sheet [31]. The exact details of parameters used for the simulation are given in Table 6.2.

Table 6.2: Simulation radio and energy characteristics.

Radio Frequency	916 MHz
Data Rate	19200 bps
RF Transmit Power	0 dBm
Receive Sensitivity	-98 dBm
Receive Threshold	-80 dBm
Propagation Pathloss	Two-ray
Battery	2×1.5 V AA batteries (8400 mWh)
Current Draw	
Receive	18 mA
Transmit	35 mA

The energy consumption functions added to the radio layer represent the Energy Man-

agement Plane of the logical design given in Figure 6.1. Modules at any layer may call these functions to get energy usage information. In our implementation the SNP layer uses the EMP functions to disable sensors that have exhausted their energy source. The Energy Management Plane also provides energy usage statistics to the user.

6.2.2 Failure Model

As described in Section 2.2, routing protocols in sensor networks must be resilient to failure. Our failure model assigns a failure time to a sensor through the GloMoSim application configuration file. The actual implementation of the failure model exists in the sensor application which informs the SNP layer when the node has failed. The SNP layer does not pass traffic in a failed node. Under our model a failed sensor does not participate in the network for the remainder of the simulation. There are no transient faults. A sensor also ceases to participate in the network when it has depleted its battery power. Sinks cannot fail under any circumstances in our simulations.

6.2.3 Applications

The applications provided as part of the GloMoSim distribution are not adequate for modeling a sensor network. As explained in Section 2.2, sensor networks exhibit traffic characteristics unlike existing Internet traffic. We developed two applications to test our proposed routing algorithm.

The sink application is able to assign multiple tasks to multiple regions of interest in each simulation run. The tasks are read from the GloMoSim application configuration file. Tasks can specify demand-driven, periodic or triggered data collection. The sink monitors incoming DATA packets and uses the event-driven grid reconstruction described in Section 4.4.2 to balance the forwarding load. The sink application also provides one side of the statistics gathering capability for delivery efficiency and delay metrics.

The sensor application receives tasks from the sink and generates DATA packets ac-

cording to those tasks. The sensor application represents the other side of the delivery efficiency and delay statistics gathering. As a result, the actual data sent to the sink includes various pieces of debugging and statistical information. The failure model also resides in the sensor application module. The application configuration file can specify a time during the simulation at which a sensor will fail.

Our implementation represents a realistic sensor network simulation platform. A lot of time has been spent on testing and modeling aspects of the network that would impact our algorithm in a real-world deployment. Therefore we have a high degree of confidence in the performance results presented in the following chapter.

Chapter 7

Performance Evaluation

The type of simulations that were needed to properly evaluate our GGR implementation had both high processing and memory requirements. Some simulation runs needed more than twenty-four hours of processing time on a 2.8 GHz Intel Xeon processor and used more than 2 GB of memory. Seventeen different scenarios were selected for evaluation of GGR. Each of these scenarios was then executed twenty times. Twenty executions was determined to provide sufficiently meaningful statistical data because the variance in the results of a number of the scenarios did not decrease with more executions. Clearly our computing needs were quite high for these 340 executions of the GloMoSim simulator.

The required computing resources were met by the Mercury Linux cluster at the University of Victoria's Research Computing Facility [23]. Mercury is an eighty-four node cluster. Fifty-four nodes in the cluster use dual 2.4 GHz Intel Xeon processors and have approximately 1.5 GB of memory. The other thirty nodes use dual 2.8 GHz Intel Xeon processors and have 2.5 GB of main memory. The results below could not have been gathered without this facility.

7.1 Evaluation Scenarios

As previously mentioned, seventeen scenarios were chosen for our performance evaluation. These scenarios represent five dimensions in which simulation parameters are varied to show different aspects of GGR performance. Only what were perceived as the most im-

portant parameters were varied due the computing resources required for the experiments. Our five experimental variables are summarized in Table 7.1.

Table 7.1: Experimental Variables.

Variable	Description
Number of Nodes	The number of nodes in the network is varied between 300 and 600. As with the overhead analysis presented in Section 5.2, the sensor field area is adjusted to maintain a constant average density, which was determined through experimentation. The sensor field area ranges from roughly 2.8 km^2 to 5.6 km^2. The sensor field is again chosen to be square in shape to avoid scenarios of narrow sensor fields. GGR is not appropriate for such cases because an effective grid cannot be constructed.
Number of Sinks	The number of sinks in the network is varied from one to four.
Number of Tasks	The number of tasks issued by a single sink is varied from one to four.
Task Type	The type of data collection requested by the sink is either demand, periodic or triggered.
Failure Rate	The failure rate parameter is varied between 0% and 30%. For example, a failure rate of 15% indicates that 15% of the sensor nodes will fail at random over the course of a simulation.

The base scenario consists of three-hundred nodes, one sink, one task, demand traffic and 0% failure. The parameters are varied in each dimension, one at a time. Each scenario is run twenty times to gather meaningful statistical results. On each simulation run there are many simulation parameters that are selected randomly. These random parameters are: sink and sensor locations, task submission time by sinks, task duration and reporting period if applicable, and task region of interest sizes and locations. The wide range and impact of these parameters can cause large variance in the collected data. This being so, we base our

analysis on general trends in the results only. Each simulation run lasts for 156 hours of simulation time.

Although the node density is not an independent variable in our study, the density does decrease as a result of increasing failure rates. The performance data related to changing failure rates can therefore be used to infer the performance impact of decreasing node density.

7.2 Performance Metrics

Measurements are taken for six different metrics during each simulation run. These measurements are described in Table 7.2 and allow us to evaluate a number of different performance aspects of the routing protocol.

7.3 Results

The performance results presented in this section show averages over all simulations for each evaluation scenario. The value returned from a single simulation run is in turn an average of all the values gathered from that run. Thus the data points represent a mean of averages.

7.3.1 Delivery Efficiency

Delivery efficiency was measured separately for TASK messages and DATA messages. The tasking efficiency of the network is described by the plots in Figure 7.1. The average percentage of sensors that successfully receive a TASK message is plotted on the Y-axis. The number of nodes in the network is increased in plot (a). The impact of increasing numbers of sinks is shown in graph (b). Graph (c) plots query delivery efficiency against the number of tasks issued by a sink. The type of tasks issued is varied in plot (d). Finally, the sensor failure rate is changed in graph (e).

Table 7.2: Performance Metrics.

Metric	Description
Task Delivery Efficiency	The percentage of sensors within the region of interest that actually receive the TASK message (data request).
Data Delivery Efficiency	The percentage of DATA messages sent by sensors that are received by the sink.
Energy Use	The percentage of battery power used during the course of the simulation. As stated in Section 7.1, each sensor is active for 156 hours during a simulation. From Table 6.2, sensors draw current at 18 mA in receive mode and have 8400 mWh of stored energy in two 1.5 V AA batteries. Given this configuration, the energy use will be at least 50.14% for every sensor in the network.
Energy Use Standard Deviation	The standard deviation of the energy used by nodes during the course of a simulation. This metric measures how well GGR is able to balance the energy consumption load among sensors in the network.
Task Delay	Time from when the sink begins TASK message propagation in the network to the time the TASK message is received at a sensor.
Data Delay	Time from when a DATA message is passed from the sensor application layer to the time it is received by the sink application.

In general, the task delivery efficiencies described by Figure 7.1 are quite stable around the 70% to 85% mark, showing the protocol's scalability in various dimensions, flexibility to different scenarios and robustness under adverse conditions. Increasing the number of nodes, sinks or the failure rate seems to lead to a small degradation in task efficiency. These minor trends are due to the increased distance the TASK message needs to travel in a larger network, the increased interference in the network when there are more sinks and a reduced forwarding capacity in the network as nodes fail. A large drop in task delivery efficiency occurs when there is more than one task issued by the sink. The reason for this reduced

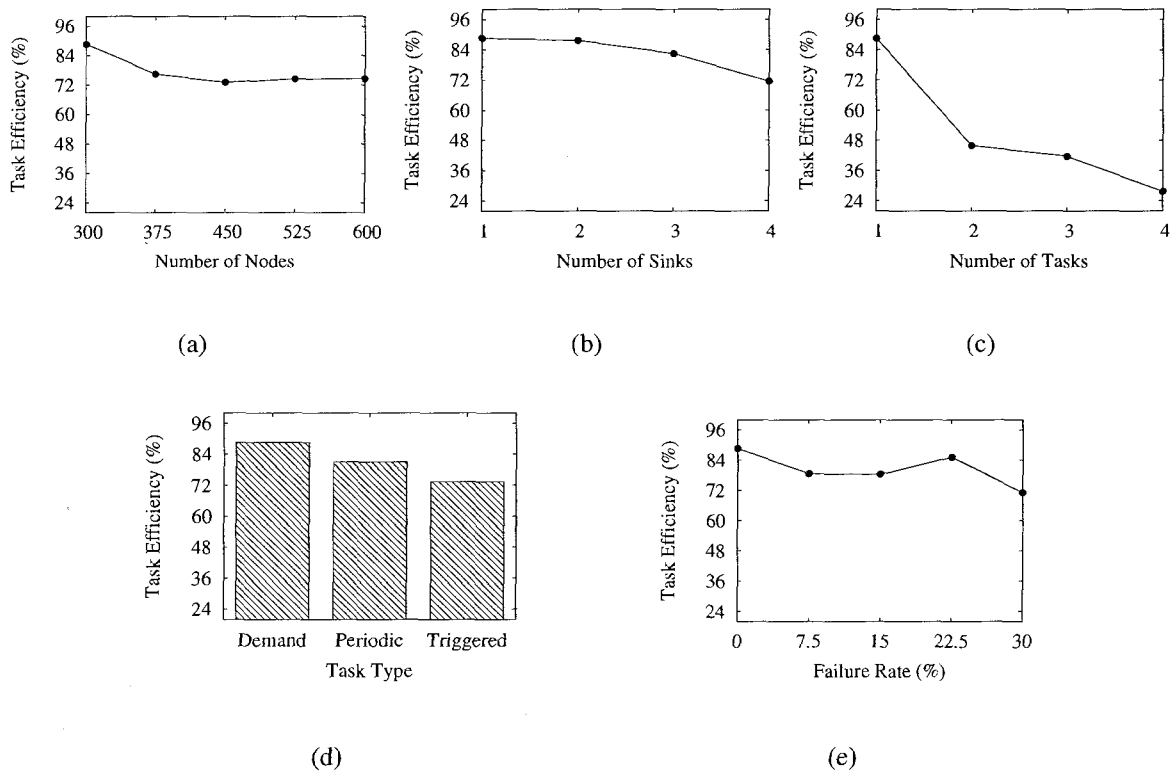


Figure 7.1: GGR TASK message delivery efficiency results.

efficiency can be traced to increased grid traffic after the first task has been issued and data is being collected. The protocol does not offer any sort of reliability for TASK message transmission, instead relying on the redundancy provided by the grid. Unfortunately, if TASK messages are dropped near the sink, before they become widespread on the grid, they may never reach the intended region of interest. Overall, the data shows a high degree of scalability and resilience to failure, with the exception of the increasing number of tasks. The variance in task efficiency between different task types can be attributed to normal statistical variation as the type of task being delivered does not affect the efficiency of that delivery.

The average percentage of DATA messages sent by sensors that were successfully received at the sink is plotted on the Y-axis of the graphs shown in Figure 7.2. Graph (a) plots data efficiency against the number of nodes in the network. The number of sinks is

varied in plot (b). Plot (c) increases the number of tasks issued by the sink. The impact of different task types is shown in graph (d). Finally, plot (e) shows the result of increasing sensor failure rates.

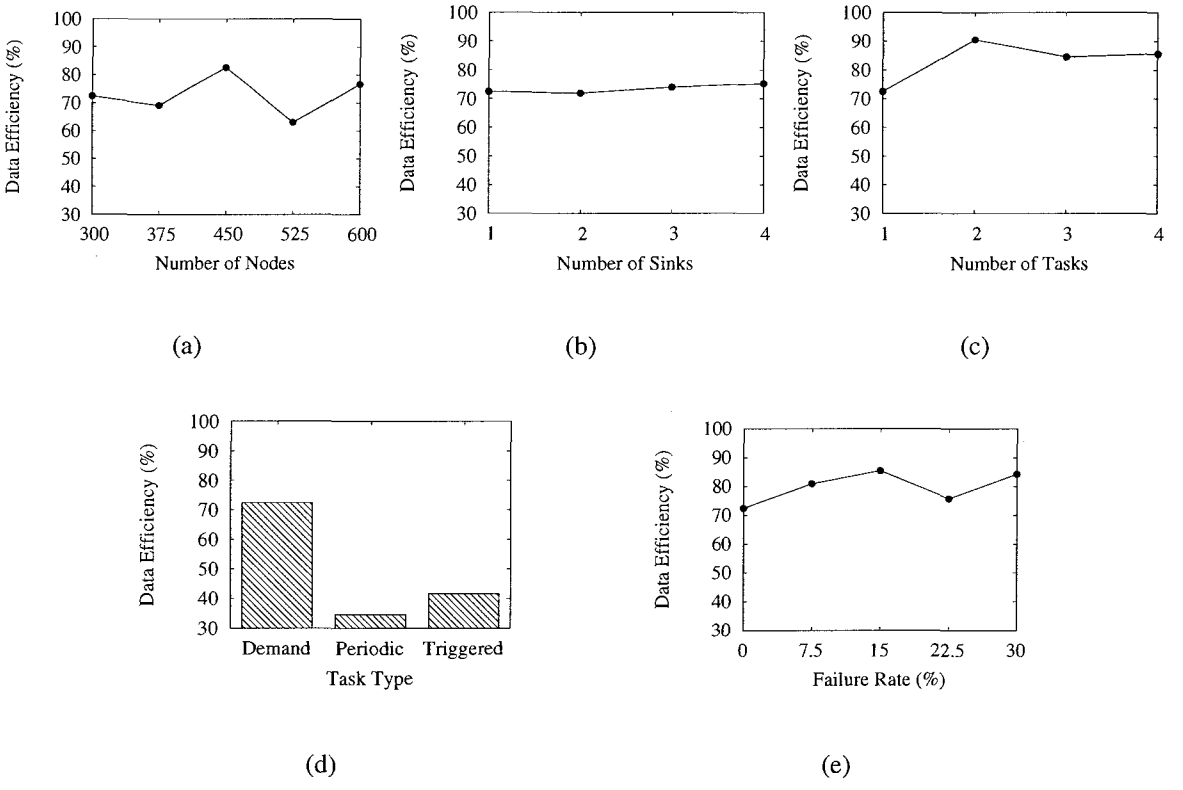


Figure 7.2: GGR DATA message delivery efficiency results.

Data delivery efficiency is affected by the tasking efficiency in that a lower task efficiency means fewer nodes transmitting DATA messages and can often lead to reduced congestion and increased delivery rates. Examples of this effect can be seen by comparing graphs (b), (c) and (e) of Figures 7.2 and 7.1. Increasing failure rate can also have a beneficial effect on the data efficiency because fewer nodes are sending data toward the sink. The variation in the DATA message delivery efficiency seen in graph (a) can be attributed to normal statistical variation since no trend is obvious. The periodic and triggered data traffic types can significantly reduce data efficiency. The periodic type is known to pro-

duce far more data than simple demand traffic. This can lead to increased congestion and dropped DATA messages. Triggered data is extremely random and can, at times, produce large amounts of data, causing the same effect as periodic traffic.

7.3.2 Delay

Like delivery efficiency, TASK message delay was measured separately from DATA message delay. TASK message delay is shown in Figure 7.3 with the average time taken for a TASK message to reach a sensor plotted on the Y-axis. The task delay is plotted against the number of nodes in the network in graph (a). Plot (b) varies the number of sinks. The number of tasks issued by the sink increases in graph (c). Graph (d) shows the influence of different task types. Finally, plot (e) shows the results of different sensor failure rates.

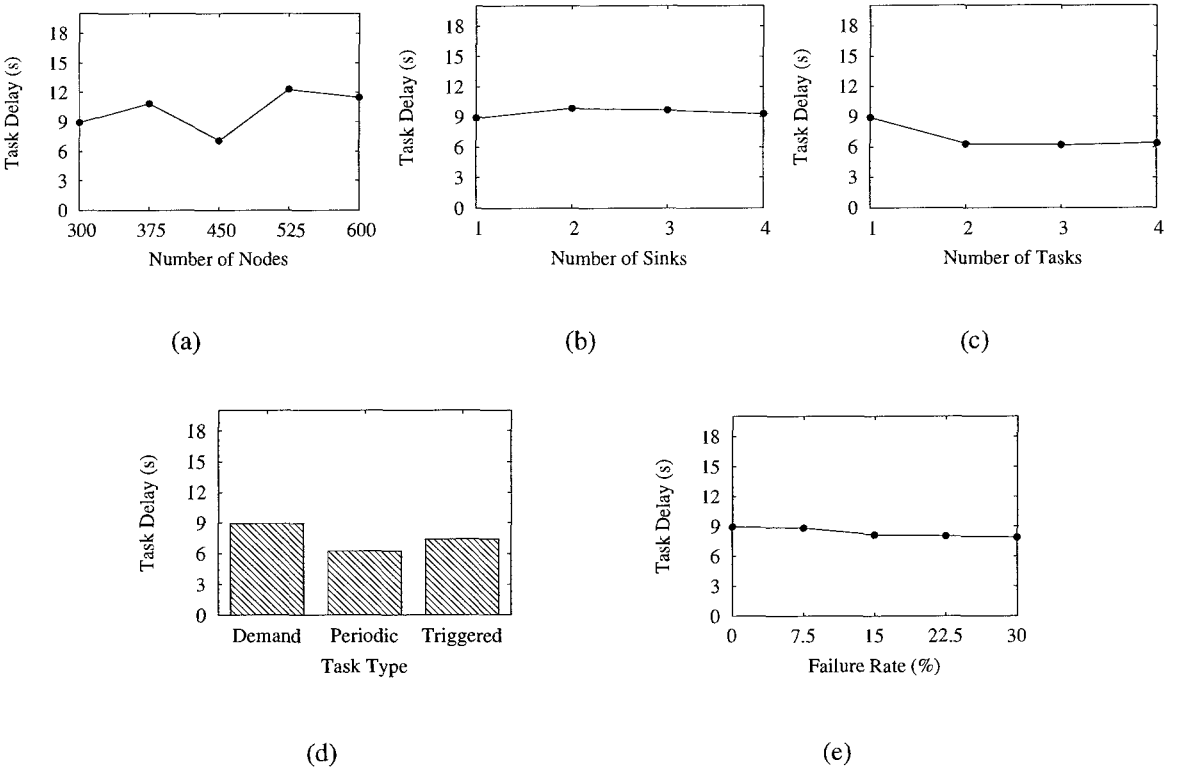


Figure 7.3: GGR TASK message delay results.

The data plots in Figure 7.3 show a high degree of scalability with respect to increasing numbers of nodes, sinks and tasks. Only a slight upward tendency is shown in graph (a) as the size of the sensor field increases. Such a result is to be expected as messages may travel further to reach the region of interest. The slightly decreasing delay exhibited in plot (c) can be attributed to the decreasing task delivery efficiency shown in Figure 7.1(c), which leads to lower network congestion. The effect was previously described in Section 7.3.1, concerning DATA message delivery efficiency results.

Figure 7.4 gives DATA message delay results for the various network scenarios. The average time taken for a DATA message to reach the sink is plotted on the Y-axis. Graph (a) shows how the delay is affected by increasing network size. The number of sinks is incremented in plot (b). The number of tasks issued by the sink increases along the X-axis of graph (c). Graph (d) shows how different task types affect the data delay. Again, the impact of failing sensors on the data delay is shown in plot (e).

DATA message delays are higher than TASK message delays and show greater variation. The reason for this is the randomized algorithm used to spread congestion in the region of interest over time at the network layer. The technique is described in Section 4.8. No general trends are obvious from the plots of Figure 7.4, although triggered data shows somewhat less delay than other traffic types. As stated in Section 7.3.1, triggered data is extremely random. The lower data delay could be due to a few scenarios where little data was generated, causing nearly no congestion.

The TASK and DATA message delay data does not show signs of any significant trends. This stability indicates strong scalability, fault tolerance and versatility properties.

7.3.3 Energy Efficiency

Sensor energy consumption is an important factor because it dictates how long a sensor network can remain functional without node replacement. Figure 7.5 shows average energy consumption for each of the scenarios described in Section 7.1. In addition to the average amount of energy used by sensors, it is important to measure how well GGR is able to

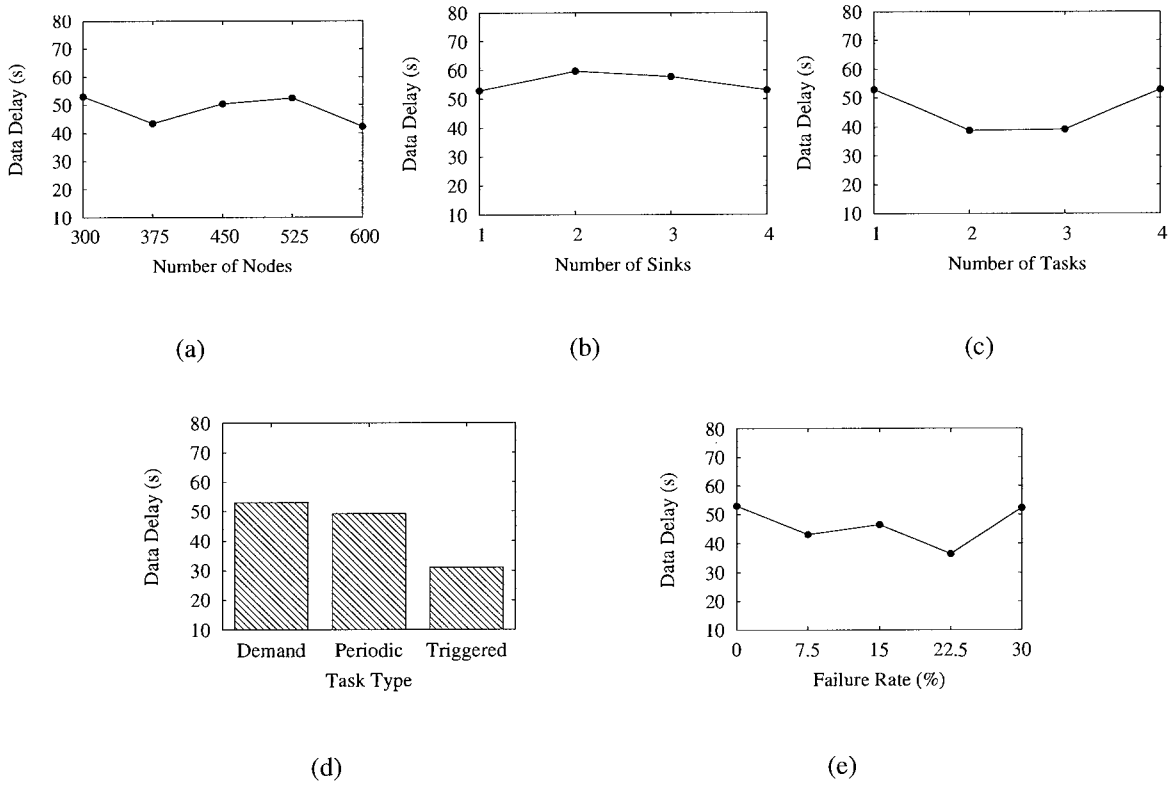


Figure 7.4: GGR DATA message delay results.

balance the energy consumption among nodes in the network. Figure 7.6 shows the average standard deviation in the energy consumption data (i.e. how much the energy use varies among sensors on average). As already mentioned, the minimum energy consumption for a sensor in our simulations is 50.14%.

The average percentage of battery power consumed by a sensor is plotted on the Y-axis in Figure 7.5. Graph (a) plots energy use against the number of nodes in the network. The number of sinks in the network is increased in plot (b). The number of tasks issued by the sink is increased in graph (c). The impact of different task types is shown in graph (d). Finally, plot (e) shows the energy consumption at different failure rates.

The data plots of Figure 7.5 again show GGR to be highly scalable, fault tolerant and energy efficient. Only increasing the number of sinks, and thus the number of independent

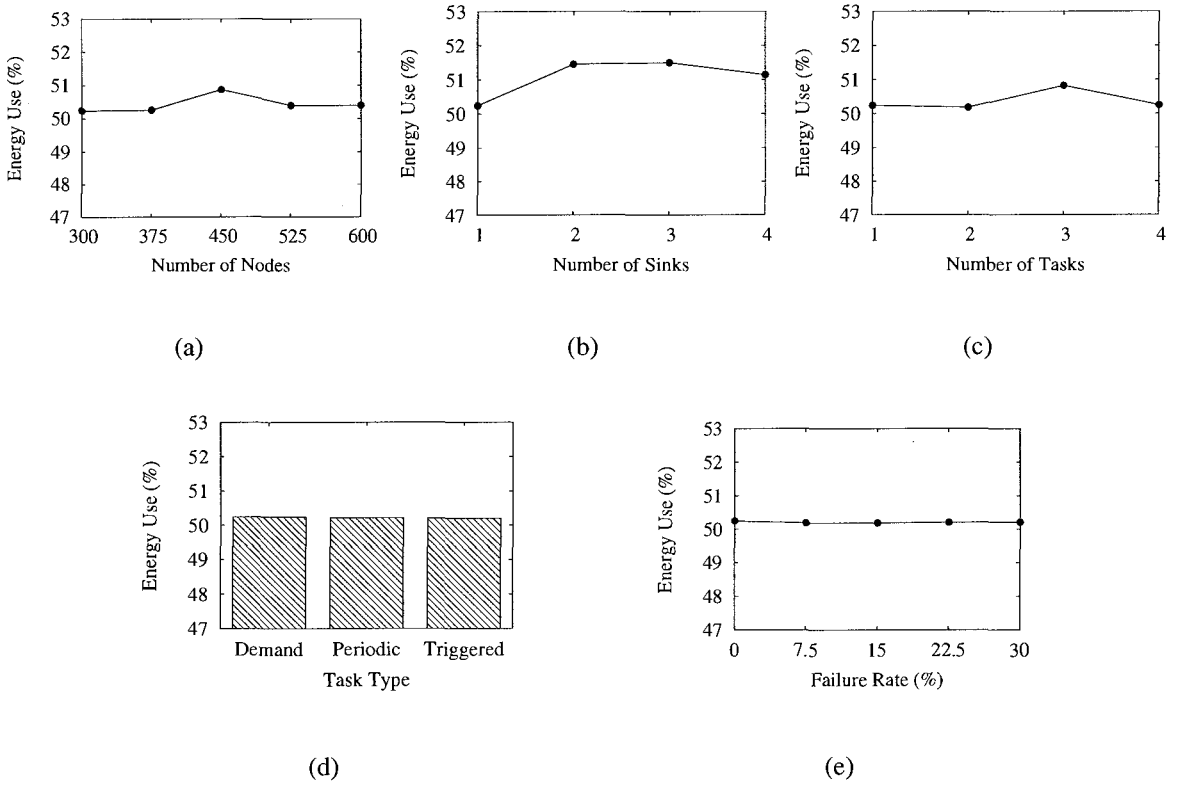


Figure 7.5: GGR energy consumption results.

grids being used, results in a mild trend of increasing energy consumption. Other variables have almost no affect on the average amount of energy consumed because only a small proportion of the sensors are actually used to forward packets.

Figure 7.6 shows the average standard deviation in percentage of battery power consumed by a sensor. The number of nodes are varied in graph (a). Plot (b) shows the impact of increasing the number of sinks in the network. The number of tasks issued is increased in graph (c). The effect of different task types is shown in plot (d). Lastly, energy consumption deviation as the failure rate increases is given in graph (e).

The data plots depicted in Figure 7.6 show GGR to be highly energy efficient. The energy use deviation among the sensors in the network is quite low, usually less than one percentage point. Increased deviation can be seen as the size of the sensor field grows

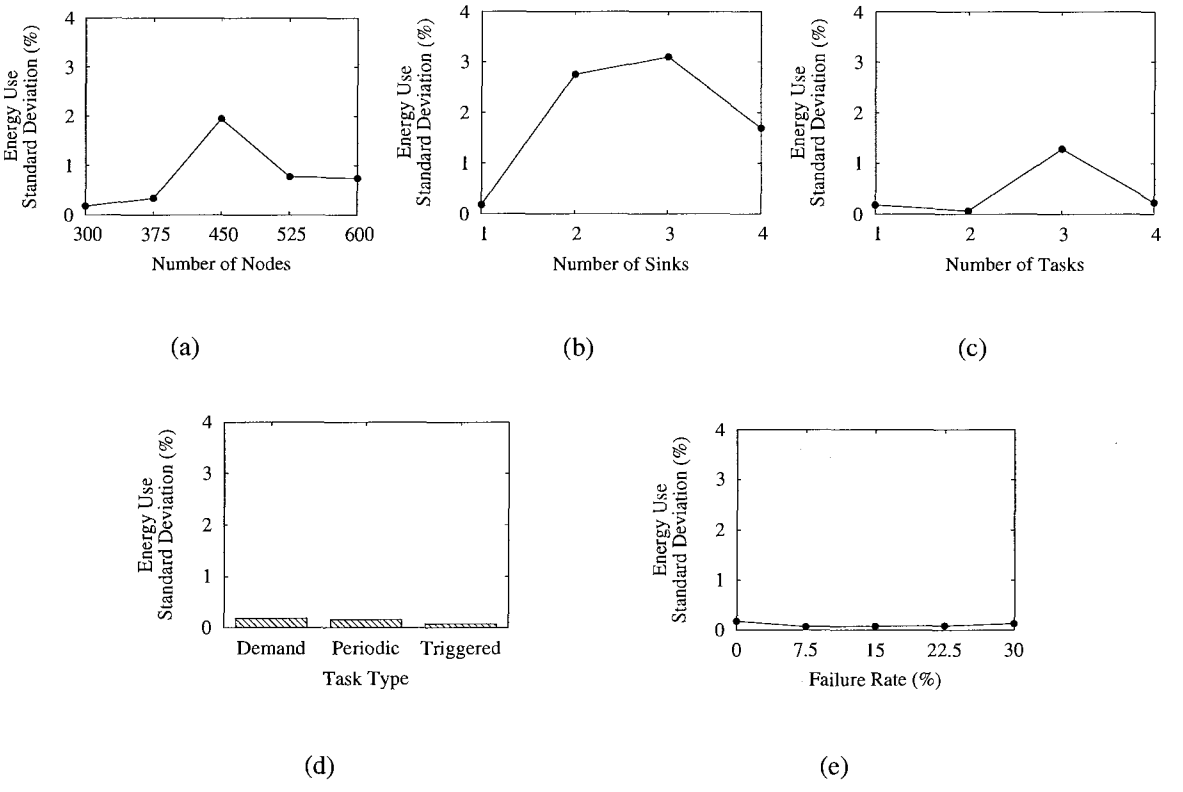


Figure 7.6: Standard deviation of energy consumption among sensors in a GGR network.

larger in Figure 7.6(a). As the network area grows larger the average region of interest size will also increase, causing more data to flow through the network. The increased traffic will put more demand on the Dissemination Nodes and cause them to use more energy. The same effect of increased demand can be seen as the number of tasks issued increases in Figure 7.6(c). The impact of increasing numbers of sinks has an even greater effect, as shown in Figure 7.6(b). When more sinks are operating in the network there will be a greater proportion of the sensors acting as Dissemination Nodes and expending energy in that capacity.

Overall, few of the scenarios modeled had a significant impact on the performance criteria studied. In modeling a sensor field with such a range of variables and under the conditions described in Chapter 6, GGR has proven itself to be efficient, scalable, robust

and versatile for remote sensing applications.

The energy efficiency, scalability and robustness shown by the simulation results support the theoretical analysis given in Chapter 5. The slowly increasing communication overhead shown in Section 5.2.2 is validated by the data presented in Section 7.3.3. Figure 7.5(a) in particular shows the slowly increasing energy consumption as the size of the sensor field increases. The scalability indicated by the theoretical state complexity shown in Section 5.2.3 is supported by the slow rate of increase in all the results presented in this Section where the number of nodes is increased. Finally, the assertion was made in Section 5.2.3 that GGR would show good resilience to failure because of the small amount of state information that is stored. This statement is supported by the limited impact of sensor failure shown in the performance results.

7.4 Comparison to TTDD

Unfortunately the results presented in this Chapter cannot be directly compared to the performance data reported for TTDD in Fan Ye's paper [6]. There are a number of reasons for this. First, their implementation uses IEEE 802.11 at the MAC layer, which is far more reliable than the CSMA MAC layer used in our GGR experiments. Second, the NS-2 simulator [24] is used for the TTDD simulations. As explained in Section 6.1.1, NS-2 uses an unrealistic radio model that does not consider radio interference. For these reasons accurate comparisons between results from our GGR experiments and the published TTDD performance data cannot be made. In order to accurately compare the performance of TTDD and GGR an entirely new implementation of TTDD would be required for the GloMoSim simulator based on our environmental assumptions. The time available for our study simply did not allow for this. However, based on our analysis from Section 5.2, it is fair to say that TTDD would not perform as well if subjected to the interference, increased region of interest size and simple MAC layer protocol that were used for our experiments.

Chapter 8

Conclusions

8.1 Summary

This thesis has proposed a new routing protocol for wireless sensor networks called Geographic Grid Routing (GGR). The work built on a previous idea put forth by Fan Ye, *et al.* [6], but aimed to make improvements in a number of areas. A detailed design, verification and performance analysis of the new protocol were provided in this thesis. High resolution data acquisition applications can benefit from our work in a number of ways including better scalability, versatility, robustness and energy efficiency than previous efforts. Our scheme attempts to balance the competing goals of minimizing sensor energy consumption and maximizing the resilience to various forms of network failure. The primary contributions of our work are summarized below followed by a discussion of areas requiring further research.

8.2 Main Contributions

The GGR routing protocol for wireless sensor networks offers significant improvements over previous efforts, and the TTDD protocol in particular. Our work shows greater scalability in the network size, and in the number of nodes from which data can be gathered simultaneously. Versatility on multiple levels has been demonstrated, including the type and scale of data collection, and the environment in which the protocol can operate. Ap-

plications can request various forms of data collection from regions of arbitrary size. The protocol makes few assumptions about the facilities provided by lower layers. Specifically, GGR can function in a network where bidirectional links are not guaranteed, making it more applicable to many sensor field scenarios than other routing protocols. The same ability to function without bidirectional links leads to the robustness property with respect to link failures. Intermittent or permanent failure of a link between any two nodes has little to no effect on GGR. Further, it has been shown that GGR can withstand significant node failures, and offers greatly improved energy efficiency and load balancing. A final important property of GGR that is present in little of the previous work is the ability to add devices to the sensor field at random with no configuration or communication of any kind. This property simplifies, and reduces the cost of, network maintenance. The particular innovations that contribute to the aforementioned properties are listed below.

- *Robust and Efficient Grid Forwarding*

  Three aspects of the GGR grid construction and usage are marked improvements over the TTDD effort. The first point of enhancement is in the load balancing characteristics of GGR. TTDD uses a static cell size for each virtual grid. As a result, any source that transmits data continuously to a sink will always use the same Dissemination Nodes. GGR spreads the load of Dissemination Node assignment by varying the cell size over time. In changing the cell size used for a grid, different Dissemination Nodes are selected. This point is discussed in Section 4.4.2.

  A second point of improvement is in the use of multiple grid paths by GGR. TTDD uses only a single path between any source/sink pair. This property coupled with the static grid used by TTDD could lead to network partitions. Rather than establishing a specific path, GGR sets up many gradients toward the sink. The robustness of the grid forwarding structure is greatly improved by having multiple paths to the sink available. The establishment of the forwarding gradients without creating routing loops is explained in Section 4.6.1.

  As a result of the multiple paths available, GGR is able to select the most energy ef-

ficient path. The process of selecting preferable routes by discovering multiple paths from a source to a destination is known as diversity injection and was covered by Pearlman and Haas [19]. TTDD does not attempt to select an energy efficient route, it simply chooses the first established route. Our adaptive path selection process improves the useful lifetime of the network by balancing the energy used in forwarding and reacting to changing network conditions. This process is described in Section 4.6.2.

- *Operation on Unidirectional Links*

  Section 3.1.1 lists the possibility of unidirectional links as one of the assumptions for GGR operation. GGR is the only known sensor network routing protocol to make this significant assumption. The assumption distinguishes GGR because it means the protocol can function in wireless networks using simple MAC layer protocols. Schemes used to guarantee the existence of bidirectional links use many more transmissions than are required for unidirectional links. Some ways to provide bidirectional links include a Request-To-Send/Clear-To-Send exchange or periodic probing for neighbour sets. These protocols are not as energy efficient as protocols that only provide a unidirectional link and are not likely to be used in wireless sensor network deployments. TTDD requires bidirectional links for a number of its operations including node failure recovery.

- *Flexible Data Acquisition*

  The sink-initiated grid construction used by GGR is a major improvement over TTDD because it improves both flexibility and scalability. The sink-initiated approach allows for sensor tasking, similar to what was proposed by Intanagonwiwat [15]. Sensor tasking makes possible a wide range of data retrieval applications. Data can be collected from specific areas of the network for varying periods of time using any of the various reporting schemes identified in Section 2.2. In addition, GGR provides an efficient mechanism for delivering these tasks to a specific region of the sensor field. New tasks must be flooded over the entire network when using TTDD. It was

shown in Section 5.2, that our sink-based grid approach is far more scalable for data collection as the region of interest becomes large. Simultaneous data collection from the entire sensor field can be made quite efficient using GGR, a task for which TTDD was simply not designed.

- *No Unneeded Communication*

  An important feature of GGR is that absolutely no communication occurs unless it is either initiated by the creation of a new task or a data transmission from sensors. When no new tasks are being assigned and no data is flowing in the network, no energy will be expended on communication activities. The event-driven grid reconstruction, explained in Section 4.4.2, means new paths are only actively discovered when triggered by information obtained from data packets flowing through the network. TTDD and the vast majority of other sensor network routing protocols require periodic transmissions to maintain network paths.

Although GGR offers many advantages over TTDD, there is a specific data collection scenario for which TTDD was designed and is effective. TTDD may be a better solution than GGR when only triggered type data collection is required and data is to be delivered to multiple sinks from a small number of sensors at a time. This situation presupposes that the sinks have no outside communication channel since the use of such a channel would be more efficient than taxing the sensor network with redundant delivery of the same data to multiple sinks. The source based grid construction used by TTDD is more effective in the described scenario because the number of sources from which data originates is less than the number of sinks to which it is being delivered. An application that fits this scenario is enemy detection in a military battlefield scenario where soldiers act as data sinks and do not have an outside communication channel.

Given GGR's flexibility, the potential areas of application for the protocol are many. Some examples include collection of scientific data from inhospitable environments, structural integrity monitoring in buildings and other man-made structures, and inventory tracking. All of these applications require flexible data collection and the ability to target specific

areas of the sensor field to gather data from multiple sensors simultaneously.

8.3 Future Work

A number of areas of future research became apparent over the course of this project. These topics range from slight modifications to the protocol to work that is related to, but lies outside the scope of, a routing protocol.

Currently, a node will sometimes mark links as dead even when it is possible to reach the target Dissemination Node. This can be due to extremely high congestion on the link. A first approach is to consider changes to the link removal algorithm outlined in Section 4.6.2. It may be worth using a less drastic approach to the scenario of missing acknowledgments so that links are not so easily given up. Another way to alleviate this situation could be to reduce the congestion on a given link. GGR currently only uses a single path at a time even though backup paths are available. Using many paths at once could increase delivery efficiency, but may hurt energy efficiency.

Like GGR, TTDD supports multiple sinks, but unlike GGR it allows those sinks to be mobile within the sensor field. Mobile sinks can help spread the forwarding load incurred by sensors near the sink. GGR could use a similar algorithm to the *trajectory forwarding* used by TTDD. Adding this to the protocol would be trivial and beneficial future work for GGR.

The sink initiated approach taken by GGR provides much greater opportunity for data aggregation than what exists in TTDD. Since each source in TTDD creates a grid, data from different sensors in a region of interest travel back to the sink using multiple independent grids. Data aggregation cannot occur between the different grids. In GGR, data aggregation can happen at Dissemination Nodes along the path to the sink. Future work could look at the performance advantages offered by such data aggregation.

Finally, two aspects of sink operation have been ignored by our work because they represent a higher level view of the network. First, the sink device has been assumed to

have infinite power resources. In most cases this is not a valid assumption. However, the management of energy reserves in sinks is left as future work. Second, multiple sinks help to balance the load across the network and increase the fault tolerance of the network as a whole. In many networks an overseeing entity will be able to control the selection of sinks for task dissemination. Sink selection could be based on energy reserves in the sink, energy reserves of sensors near the sink or proximity to the region of interest. Selection of the appropriate sink is an interesting topic requiring further research.

Bibliography

[1] University of Victoria. (2004, Mar.) NEPTUNE Canada - Network Infrastructure. [Online]. Available: http://www.neptunecanada.ca/network/

[2] J. Carle and D. Simplot-Ryl, "Energy-efficient area monitoring for sensor networks," *Computer*, vol. 37, no. 2, pp. 40–46, Feb. 2004.

[3] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava, "Coverage problems in wireless ad-hoc sensor networks," in *Proceedings Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies.*, ser. INFOCOM, no. 20. Anchorage, AK, USA: IEEE, Apr. 2001, pp. 1380–1387 vol.3.

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

[5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, Aug. 2002.

[6] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A two-tier data dissemination model for large scale wireless sensor networks," in *Proc. of the Eighth ACM International Conference on Mobile Computing and Networking*, ser. MobiCom, no. 8. Atlanta, GA, USA: ACM, Sept. 2002, pp. 585–594.

[7] S. Hedetniemi, S. Hedetniemi, and A. Liestman, "A survey of gossiping and broadcasting in communication networks," *Networks*, vol. 18, pp. 319–349, 1988.

[8] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, Mar. 2000.

[9] Intel Corporation. (2004, Mar.) Intel Research - Exploratory Research - Sensor Networks. [Online]. Available: http://www.intel.com/research/exploratory/wireless sensors.htm

[10] E. M. Royer and C.-K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," *IEEE Personal Communications*, vol. 6, no. 2, pp. 46–55, Apr. 1999.

[11] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information

dissemination in wireless sensor networks," in *Proceedings ACM/IEEE international conference on Mobile computing and networking*, ser. MobiCom, no. 5. ACM Press, 1999, pp. 174–185.

[12] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proc. of the 33rd Annual Hawaii International Conference on System Sciences*, 2000, pp. 3005–3014.

[13] L. Li and J. Halpern, "Minimum-energy mobile wireless networks revisited," in *IEEE International Conference on Communications*, ser. ICC, Helsinki, Finland, June 2001, pp. 278–283 vol.1.

[14] K. Sohrabi, I. Gao, V. Ailawadhi, and G. Pottie, "Protocols for self-organization of a wireless sensor network," *IEEE Personal Communications*, pp. 16–27, Oct. 2000.

[15] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the sixth annual international conference on Mobile computing and networking*, ser. MobiCom, no. 6. ACM Press, 2000, pp. 56–67.

[16] Y. Yu, R. Govindan, and D. Estrin, "Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks," University of California at Los Angeles, Los Angeles, CA, USA, Technical Report UCLA/CSD-TR-01-0023, Aug. 2001.

[17] F. Ye, S. Lu, and L. Zhang, "GRAdient Broadcast: A robust, long-lived large sensor network," University of California at Los Angeles," Technical Report, 2001.

[18] M. Bhardwaj and A. P. Chandrakasan, "Bounding the lifetime of sensor networks via optimal role assignments," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.*, ser. INFOCOM, no. 21. New York, NY, USA: IEEE, June 2002, pp. 1587–1596.

[19] M. R. Pearlman and Z. J. Haas, "Improving the performance of query-based routing protocols through "diversity injection"," in *Wireless Communications and Networking Conference*, ser. WCNC. New Orleans, LA, USA: IEEE, Sept. 1999, pp. 1548–1552 vol.3.

[20] Gerard J. Holzmann. (2004, Apr.) SPIN - Formal Verification. [Online]. Available: http://www.spinroot.com

[21] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual.* Addison-Wesley, 2003.

[22] UCLA Parallel Computing Laboratory. (2001, Feb.) GloMoSim - Global Mobile Information Systems Simulation Library. [Online]. Available: http:

//pcl.cs.ucla.edu/projects/glomosim/

[23] University of Victoria. (2004, July) Research Computing Facility at the University of Victoria. [Online]. Available: http://rcf.uvic.ca

[24] SAMAN Group, CONSER Group, ICIR Group, *et al.* (2004, July) The network simulator - ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns/

[25] OPNET Technologies Inc. (2004) OPNET. [Online]. Available: http://www.opnet.com

[26] Rice Monarch Group. (2000, Nov.) Wireless and mobility extensions to ns-2. [Online]. Available: http://www.monarch.cs.rice.edu/cmu-ns.html

[27] UCLA Parallel Computing Laboratory. (2001, Nov.) Parsec - Parallel Simulation Environment for Complex Systems. [Online]. Available: http://pcl.cs.ucla.edu/projects/parsec/

[28] J. Nuevo, "A comprehensible GloMoSim tutorial," Universite du Quebec, Montreal, Quebec, Canada, Tech. Rep., Mar. 2003.

[29] ——, "A comprehensible GloMoSim tutorial," Universite du Quebec, Umea, Sweden, Tech. Rep., June 2002.

[30] Crossbow Technology Inc. (2004) Smarter Sensors in Silicon. [Online]. Available: http://www.xbow.com

[31] ——, "MICA2 Wireless Measurement System," Product datasheet available from http://www.xbow.com, San Jose, CA, USA, June 2004.

Appendix A

Source Code

A.1 PROMELA Model (ggr.pml)

```
/* Promela verification model for Geographic Grid Routing. */

/* Macro definitions. */
#define NUM_NODES 2         /* Number of sensor nodes in the simulation. */
#define NUM_CHANS 3         /* Should always be one more than NUM_NODES. */
#define NUM_NEIGHBOURS 2    /* Number of neighbours for each node. */
#define MAX_PAYLOAD_SIZE 3 /* Size of the payload in each message. */
#define CHAN_BUFFER_SIZE 4 /* Channel buffer size. */
#define MAX_COST 3          /* Assumption: MAX_COST > NUM_NODES */

/* Definition of the TASK message fields. */
#define FLOOD 0
#define SEQ_NUM 1
#define COST 2

/*
 * Function: validState
 * Description: Examines the routing tables of all nodes and returns true if
 *  the system is in a valid state. False otherwise.
 *  NOTE: This was made a macro rather than an inline function so that it can
 *  be used as an assert(), which inline functions apparently cannot.
 */
#define validState() \
( \
  /* No routing loops. */ \
  ( \
    (routingTables[0].links[0].cost == MAX_COST) || \
    ( \
      (routingTables[1].links[0].cost == MAX_COST) && \
      ( \
        (routingTables[1].links[1].cost == MAX_COST) || \
        (routingTables[0].links[0].cost > routingTables[1].links[1].cost) \
      ) \
    ) \
  ) && \
  ( \
    (routingTables[1].links[0].cost == MAX_COST) || \
    ( \
      (routingTables[0].links[0].cost == MAX_COST) && \
      ( \
        (routingTables[0].links[1].cost == MAX_COST) || \
        (routingTables[1].links[0].cost > routingTables[0].links[1].cost) \
      ) \
    ) \
  ) && \
  /* The internal state of each node is sane. */ \
  (routingTables[0].links[0].cost != 0) && \
  ( \
    (routingTables[0].links[0].cost == MAX_COST) || \
    (routingTables[0].links[1].cost == MAX_COST) || \
    (routingTables[0].links[0].cost == (routingTables[0].links[1].cost)) || \
    (routingTables[0].links[0].cost == (routingTables[0].links[1].cost+1)) || \
    (routingTables[0].links[0].cost == (routingTables[0].links[1].cost-1)) || \
    (!routingTables[0].dNode) \
  ) && \
  (routingTables[1].links[0].cost != 0) && \
  ( \
    (routingTables[1].links[0].cost == MAX_COST) || \
    (routingTables[1].links[1].cost == MAX_COST) || \
    (routingTables[1].links[0].cost == (routingTables[1].links[1].cost)) || \
    (routingTables[1].links[0].cost == (routingTables[1].links[1].cost+1)) || \
    (routingTables[1].links[0].cost == (routingTables[1].links[1].cost-1)) || \
    (!routingTables[1].dNode) \
```

```
  ) \
)


/* Enumerations. */
/* Message types. */
mtype = {TASK, TIMEOUT_REMOVE};


/* Data structure definitions. */
typedef RTEntry
{
  byte id;
  byte cost;
}

typedef RoutingTable
{
  bit seqNum;
  bool dNode;
  RTEntry links[NUM_NEIGHBOURS];
}

typedef Payload
{
  byte data[MAX_PAYLOAD_SIZE];
}


/* Global data structures. */
/* Channels for communicating with sensors. */
chan channels[NUM_CHANS] =
  [CHAN_BUFFER_SIZE] of {mtype, byte, byte, Payload}
/* List of neighbours for each node. */
RoutingTable routingTables[NUM_NODES];


/* Inline function definitions. */
/*
 * Function: sendForged
 * Description: Used to send on an unreliable channel with a faked source.
 */
inline sendForged(type, dest, src, payload)
{
  atomic
  {
    if
    :: true -> channels[dest]!type, dest, src, payload
    :: skip
    fi
  }
}

/*
 * Function: send
 * Description: Used by all sensors to emulate sending on an unreliable
 *  channel.
 */
inline send(type, dest, payload)
{
  sendForged(type, dest, id, payload)
}
```

```
/*
 * Function: recv
 * Description: Used by all sensors to receive message from the channel.
 *  Only used for API consistency with the send function.
 */
inline recv(type, src, payload)
{
  ch?type, eval(id), src, payload
}

/*
 * Function: flood
 * Description: Used by sensors to flood a TASK message.
 */
inline flood(payload, src, counter)
{
  atomic
  {
    assert(payload.data[SEQ_NUM] <= 1);

    counter = 0;
    payload.data[FLOOD] = true;

    do
    :: (counter < NUM_NODES) ->
      if
      :: (counter != id) ->
        sendForged(TASK, counter, src, payload)
      :: else -> skip
      fi;
      counter++
    :: else -> break
    od;

    /* Reset local variables to reduce state space. */
    counter = 0
  }
}

/*
 * Function: propagate
 * Description: Used by sensors to send a TASK message to all of it's
 *  neighbours except the source which is the node from which the message was
 *  received.
 */
inline propagate(src, payload, counter)
{
  atomic
  {
    assert(payload.data[SEQ_NUM] <= 1);

    /* The remote sensors should never propagate. */
    assert(id != NUM_NODES);

    /* Maybe send FLOOD messages. */
    if
    :: true ->
      flood(payload, id, counter);
    :: skip
    fi;

    counter = 0;
    payload.data[FLOOD] = false;
```

```
    /* Send TASK messages. */
    do
    :: (counter < NUM_NEIGHBOURS) ->
      if
      :: (routingTables[id].links[counter].id != src) ->
        send(TASK, routingTables[id].links[counter].id, payload);
      :: else
      fi;
      counter++
    :: else -> break
    od;

    /* Reset local variables to reduce state space. */
    counter = 0
  }
}

/*
 * Function: rtInit
 * Description: Deletes all the upstream links in the routing table.
 */
inline rtInit(counter)
{
  d_step
  {
    counter = 0;

    do
    :: (counter < NUM_NEIGHBOURS) ->
      /* Link is initialized with maximum cost and marked as unusable. */
      routingTables[id].links[counter].cost = MAX_COST;
      counter++
    :: else -> break
    od;

    routingTables[id].seqNum = 0;
    routingTables[id].dNode = false;

    /* Reset local variables to reduce state space. */
    counter = 0
  }
}

/*
 * Function: getMinCost
 * Description: Finds the minimum cost of all upstream links.
 */
inline getMinCost(minCost, counter)
{
  d_step
  {
    counter = 0;
    minCost = MAX_COST;

    do
    :: (counter < NUM_NEIGHBOURS) ->
      assert(routingTables[id].links[counter].cost <= MAX_COST);
      minCost = ((routingTables[id].links[counter].cost < minCost) ->
          routingTables[id].links[counter].cost : minCost);
      counter++
    :: else -> break
    od;
```

```
    /* Reset local variables to reduce state space. */
    counter = 0
  }
}

/*
 * Function: clearPayload
 * Description: Utility function to clear all elements of a payload array.
 */
inline clearPayload(payload, counter)
{
  d_step
  {
    counter = 0;
    do
    :: (counter < MAX_PAYLOAD_SIZE) ->
      payload.data[counter] = 0;
      counter++
    :: else -> break
    od;
    counter = 0;
  }
}

/*
 * Function: handleTaskMsg
 * Description: Used by sensor processes to handle reception of a TASK message.
 */
inline handleTaskMsg(src, payload, oldMinCost, newMinCost, lastSrcCost, counter)
{
  atomic
  {
    assert(payload.data[SEQ_NUM] <= 1);

    /* If this is a new task message then delete all upstream links. */
    if
    :: (payload.data[SEQ_NUM] > routingTables[id].seqNum) ->
      rtInit(counter);
      routingTables[id].seqNum = payload.data[SEQ_NUM]
    :: (payload.data[SEQ_NUM] < routingTables[id].seqNum) ->
      goto handleTaskMsg_return
    :: else
    fi;

    d_step
    {
      /* Find out the current minimum cost link. */
      getMinCost(oldMinCost, counter);

      /* Update the cost metric of the sender in my routing table. */
      assert(payload.data[COST] < MAX_COST);
      counter = 0;
      do
      :: (counter < NUM_NEIGHBOURS) ->
        if
        :: (routingTables[id].links[counter].id == src) ->
          /* Link is also marked as usable at this point. */
          lastSrcCost = routingTables[id].links[counter].cost;
          routingTables[id].links[counter].cost = payload.data[COST];
          assert(payload.data[COST] <= lastSrcCost);
          break
        :: else -> counter++
```

```
    fi
  :: else -> break
  od;

  /* Find out the new minimum cost link. */
  getMinCost(newMinCost, counter);

  /* Only remove existing entries if we are already a DN or if this is not a
   * flood message. */
  if
  :: ((routingTables[id].dNode) || (!payload.data[FLOOD])) ->
    /* Remove any links that have a cost greater than or equal to my new
     * offer. */
    counter = 0;
    do
    :: (counter < NUM_NEIGHBOURS) ->
      if
      :: (routingTables[id].links[counter].cost >= (newMinCost+1)) ->
        /* Set link to MAX_COST and mark as unusable. */
        routingTables[id].links[counter].cost = MAX_COST;
      :: else -> skip
      fi;
      counter++
    :: else -> break
    od
  :: else -> skip
  fi;

  /* Reset local variables to reduce state space. */
  counter = 0
}

/* If my new offer is less than my old offer, then update
 * the cost in the TASK message and propagate it on the grid. */
if
:: (payload.data[FLOOD]) ->
  if
  :: (routingTables[id].dNode) ->
    if
    :: (newMinCost < oldMinCost) ->
      flood(payload, src, counter);
      payload.data[COST] = newMinCost + 1;
      propagate(src, payload, counter)
    :: else ->
      if
      :: (lastSrcCost == MAX_COST) -> flood(payload, src, counter)
      :: else
      fi
    fi
  :: else ->
    if
    :: (lastSrcCost == MAX_COST) -> flood(payload, src, counter)
    :: else
    fi
  fi
:: else ->
  if
  :: (routingTables[id].dNode) ->
    if
    :: (newMinCost < oldMinCost) ->
      payload.data[COST] = newMinCost + 1;
      propagate(src, payload, counter)
    :: else
```

```
        fi
      :: else ->
        payload.data[COST] = newMinCost + 1;
        propagate(src, payload, counter)
      fi
    fi;

    /* Final state updates. */
    routingTables[id].dNode = (routingTables[id].dNode) ||
      (!payload.data[FLOOD]);

    /* Reset local variables to reduce state space. */
    clearPayload(payload, counter);
    counter = 0;

    /* Return point for this function. */
handleTaskMsg_return:
    skip
  }
}

/*
 * Function: createTaskMsg
 * Description: Creates a random task message.
 */
inline createTaskMsg(payload, counter)
{
  atomic
  {
    if
    :: payload.data[FLOOD] = true
    :: payload.data[FLOOD] = false
    fi;
    if
    :: payload.data[SEQ_NUM] = 0
    :: payload.data[SEQ_NUM] = 1
    fi;
    /* Cost field. */
    counter = 0;
    do
    :: (counter < (MAX_COST-NUM_NODES-1)) -> counter++
    :: break
    od;
    payload.data[COST] = counter;

    /* Reset local variables to reduce state space. */
    counter = 0
  }
}

/* Process definitions. */
/*
 * Process: injector
 * Description: Simple process that injects packets into the network at random.
 */
proctype injector()
{
  atomic
  {
    Payload payload;
    bit sender;

    /* Choose first sender. */
```

```
    if
    :: sender = 0
    :: sender = 1
    fi;

    /* Possibly send a message to the first receiver. */
    if
    :: (nfull(channels[0])) ->
      /* Send a TIMEOUT_REMOVE message. All payload fields are already clear. */
      sendForged(TIMEOUT_REMOVE, sender, (sender+1), payload)
    :: skip
    fi
  }

  atomic
  {
    /* Possibly send a message to the other receiver. */
    sender++;
    if
    :: (nfull(channels[1])) ->
      /* Send a TIMEOUT_REMOVE message. All payload fields are already clear. */
      sendForged(TIMEOUT_REMOVE, sender, (sender+1), payload)
    :: skip
    fi
  }
}

/*
 * Process: environment
 * Description: This process models all other sensors or sinks in the network
 *  that may send messages to the sensors under study as well as other factors
 *  such as timeouts or delete messages.
 */
proctype environment(byte id)
{
  /* Local variable declarations. */
  bit receiver;
  Payload payload;
  byte scratchByte;
  chan ch = channels[id];

  /* Channel assertions. */
#if (CHAN_BUFFER_SIZE != 0)
  xr ch; /* Only I receive on my channel. */
#endif

  /* Sanity check. */
  assert(MAX_COST > NUM_NODES);

  /* Run the process that injects packets into the network. */
  run injector();

  atomic
  {
    /* Choose first receiver. */
    if
    :: receiver = 0
    :: receiver = 1
    fi;

    /* Possibly send a message to the receiver. */
    if
    :: true ->
```

```
      /* Create a random, but valid TASK message. */
      createTaskMsg(payload, scratchByte);
      send(TASK, receiver, payload)
    :: skip
    fi;

    /* Clear local vars to decrease state space. */
    clearPayload(payload, scratchByte);
    scratchByte = 0;
  }

  atomic
  {
    /* Receiver is now the other node. */
    receiver++;

    if
    :: true ->
      /* Create a random, but valid TASK message. */
      createTaskMsg(payload, scratchByte);
      send(TASK, receiver, payload)
    :: skip
    fi;

    /* Clear local vars to decrease state space. */
    clearPayload(payload, scratchByte);
    scratchByte = 0;

    /* Receive messages until all have been sent. */
    do
    :: recv(_, _, payload) -> skip
    :: timeout -> break
    od
  }
}

/*
 * Process: sensor
 * Description: This process models a single sensor in the network.
 */
proctype sensor(byte id)
{
  atomic
  {
    /* Local variable declarations. */
    Payload payload;
    chan ch = channels[id];
    byte src, scratchByte1, scratchByte2, scratchByte3, scratchByte4;

    /* Channel assertions. */
#if (CHAN_BUFFER_SIZE != 0)
    xr ch; /* Only I receive on my channel. */
#endif

    /* Forever receive messages and handle them. */
    do
    :: recv(TASK, src, payload) ->
      handleTaskMsg(src, payload, scratchByte1, scratchByte2, scratchByte3,
          scratchByte4)
    :: recv(TIMEOUT_REMOVE, src, payload) ->
      /* In actual implementation we now mark link as unusable. */
      skip
    :: timeout -> break
```

```
    od
  }
}

/*
 * Process: init
 * Description: The first process to start and initialize the simulation.
 *  All other processes are created from this one.
 */
init
{
  atomic
  {
    byte i, j, k, id;

    d_step
    {
      /* Set up the topology. */
      routingTables[0].links[0].id = 1;
      routingTables[0].links[1].id = NUM_NODES;
      routingTables[1].links[0].id = 0;
      routingTables[1].links[1].id = NUM_NODES;

      /* First clear the routing tables. */
      id = 0;
      do
      :: (id < NUM_NODES) ->
        rtInit(i);
        id++
      :: else -> break
      od;

      /* Reset local variables to reduce state space. */
      i = 0;
      id = 0;
    }

    /* Create a random, but valid initial state for the network. */
    /* Setup last sequence number entries. */
    i = 0;
    do
    :: (i < NUM_NODES) ->
      if
      :: routingTables[i].seqNum = 0
      :: routingTables[i].seqNum = 1
      fi;
      if
      :: routingTables[i].dNode = true
      :: routingTables[i].dNode = false
      fi;
      i++
    :: else -> break
    od;

    /* Setup cost entries. */
    i = 0;
    /* Loop through each node in the network. */
    do
    :: (i < NUM_NODES) ->
      j = 0;
      /* Loop through each neighbour of the node. */
      do
      :: (j < NUM_NEIGHBOURS) ->
```

```
        /* Choose the cost for this neighbour. */
        k = 0;
        do
        :: (k < MAX_COST) -> k++
        :: break
        od;
        routingTables[i].links[j].cost = k;
        /* Check if this caused a loop to be formed. */
        if
        :: (!validState()) -> routingTables[i].links[j].cost = MAX_COST;
        :: else -> skip
        fi;
        j++
      :: else -> break
      od;
      i++
    :: else -> break
    od;

    /* Insert breakpoint for simulations. */
    printf("MSC: BREAK\n");

    /* Start sensor processes. */
    run sensor(0);
    run sensor(1);
    run environment(NUM_NODES);

end:
    (!validState()) -> assert(validState())
  }
}
```