General Direction Routing Protocol

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Sean Michael Lydon

June 2009

COMMITTEE MEMBERSHIP

TITLE:	General Direction Routing Protocol
AUTHOR:	Sean Michael Lydon
DATE SUBMITTED:	June 2009
COMMITTEE CHAIR:	Hugh M. Smith, Ph.D.
COMMITTEE MEMBER:	John Bellardo, Ph.D.
COMMITTEE MEMBER:	Michael L. Haungs, Ph.D.

Abstract

General Direction Routing Protocol

Sean Michael Lydon

The General Direction Routing Protocol (GDRP) is a Wireless Sensor Network (WSN) multi-path routing protocol which abstracts localization information (commonly GPS coordinates) into relative direction information in order to perform routing decisions. By generating relative direction information GDRP is able to operate with fewer precision requirements than other protocols. This abstraction also allows the integration of other emerging hardware-based localization techniques, for example, Beamforming Sensor Arrays.

GDRP does not specifically address the next hop a packet should take, but instead specifies a direction it should travel. This direction abstraction allows for multiple paths to be taken through the network thus enhancing network robustness to node mobility and failures. This indirect addressing scheme also provides a solution to sensor node unique identification.

GDRP is simulated in a custom simulator written in Java. This simulator supports interfaces for multiple protocols for layers 1, 2, 3, and 7 of the OSI model. For performance comparisons, GDRP is compared against multiple WSN routing protocols. GDRP operates with a significantly lower setup cost in terms of bytes transmitted and a lower setup latency for networks of varying sizes. It also demonstrates an exponentially lower routing cost when compared to another multi-path routing protocol due to a more efficient packet propagation in the network.

Keywords: Networking, Wireless Sensor Networks, GPS

ACKNOWLEDGEMENTS

Thank you Dr. Hugh Smith for your long hours in helping me perfect this work and encouraging me to produce a paper for publication. Thank you Cal Poly Computer Science department for providing the majority of the funding needed to travel to Crete, Greece to present GDRP at the IASTED Sensor Network conference in September 2008. Thank you Dr. John Bellardo for your willingness to have many illuminating discussions about wireless protocols and simulations. Thank you Dan Nelson for keeping me company in the networks lab for many days leading up to our thesis defenses. Thank you my peers and my professors for providing a positive educational atmosphere and many interesting and exciting projects and discussions.

To my family, thank you for supporting me as I spent extra years at school pursuing a Master's degree and for the uncountable hours helping me to revise and perfect my wittings on GDRP. Thank you Rachael, my beautiful and understanding fiancée, for always being there to provide me with encouragement and copious amounts of caffeine.

Contents

List of Tables....viii

List of Figures....ix

Chapter 1....1

1.1 Introduction....1

Chapter 2....3

2.1 Background....3

2.2 Challenges in WSNs....4

Chapter 3....6

3.1 Related Work....6

3.2 Routing Protocol Implementations....6

3.3 Unique Identification....16

3.4 Antenna Types....17

3.5 Beamforming Sensor Arrays....17

3.6 Simulator Types....18

Chapter 4....20

4.1 Simulator Implementation....20

4.2 World Design....22

4.3 Node Design....27

4.4 Default Physical Layer....32

4.5 CSMA/CA MAC Layer....33

4.6 Bordercast 35

4.7 GPSR....38

4.8 Simple Application Layer....40

4.9 Logger....40

Chapter 5....42

5.1 GDRP....42

5.2 Direction Abstraction....43

5.3 Setup Phase....44

5.4 Operation....45

Chapter 6....48

6.1 Testbed....48

6.2 Number of Nodes....49

6.3 Node Degree....54

6.5 Number of Slices in GDRP....56

6.4 Accuracy of GPS....59

6.6 MAC properties....61

Chapter 7....62

7.1 Conclusions....62

7.2 Future Work....63

Bibliography....65

List of Tables

Table 1: Default simulation parameters..25

List of Figures

Figure 1: A Wireless Sensor Network (WSN). 3

Figure 2: Exposed Terminal Problem. 5

Figure 3: Hidden Terminal Problem. 5

Figure 4: Simple flooding logic. 7

Figure 5: Flooding with HTL. 8

Figure 6: Flooding with caching and HTL. 8

Figure 7: Two Bordercast zones with radius two. 9

Figure 8: Pseudo-code of Bordercast operation. 10

Figure 9: Minimum Cost Forwarding Algorithm (MFCA) operation. 12

Figure 10: Dynamic Address RouTing (DART) setup. 16

Figure 11: Simulator Object interaction. 21

Figure 12: Example using Reflection to create an Object. 28

Figure 13: Network layer transition diagram. 31

Figure 14: CSMA/CA packet header. 34

Figure 15: Bordercast (BC) packet header. 35

Figure 16: Bordercast example setup. 36

Figure 17: GPSR routing header. 38

Figure 18: Pseudo-code of GPSR routing. 39

Figure 19: Perimeter routing left-hand rule. 39

Figure 20: Direction abstraction (8 slices). 43

Figure 21: Pseudo-code of GDRP routing operation. 45

Figure 22: GDRP routing header fields. 46

Figure 23: GDRP routing procedure. 47

Figure 24: Transmitted Bytes for Routing vs. Number of Nodes for Bordercast and GDRP. 49

Figure 25: Transmitted Bytes for Routing vs. Number of Nodes for GDRP and GPSR. 50

Figure 26: Transmitted Bytes for Setup vs. Number of Nodes. 51

Figure 27: Setup Latency vs. Number of Nodes. 51

Figure 28: Collisions vs. Number of Nodes. 52

Figure 29: Duplicity vs. Number of Nodes. 53

Figure 30: Reliability vs. Number of Nodes. 53

Figure 31: Transmitted Bytes for Routing vs. Node Degree. 55

Figure 32: Duplicity vs. Node Degree. 55

Figure 33: Reliability vs. Node Degree. 56

Figure 34: Reliability vs. Number of Slices. 57

Figure 35: Transmitted Bytes for Routing vs. Number of Slices. 58

Figure 36: Duplicity vs. Number of Slices. 58

Figure 37: Reliability vs. GPS Accuracy. 59

Figure 38: Transmitted Bytes for Routing vs. GPS Accuracy. 60

Figure 39: Duplicity vs. GPS Accuracy. 61

Chapter 1

1.1 Introduction

Wireless Sensor Networks (WSNs) are a collection of devices referred to as nodes which sense the environment around them and transmit this data via wireless communication to a sink [1]. Typical applications require real time data from small, inexpensive, reliable, and disposable equipment [2]. Therefore, sensor nodes must be inexpensively manufactured and are required to operate at a minimal power cost. These applications require that the nodes self organize into an ad-hoc network for routing information.

Battery lifetime is one of the major limiting variables in WSNs. Each time a sensor node transmits information it utilizes significantly more power than the power required for typical processing [2]. Once a node falls below a minimal energy threshold, it no longer functions. Newer WSN network protocols attempt to conserve power by limiting the number of node transmissions.

Applications also require that data arrives reliably. The sensor nodes are responsible for routing the data to the sink. Obstacles to reliable data delivery include network congestion, random world noise, and inaccuracy in any node assumptions about the network. Network congestion causes packets in sensor nodes to be dropped by queues overflowing with packets waiting for transmission. Some routing protocols make assumptions about the network and the location of the nodes in the network. For example, GPSR assumes that a sensor node is at a specific location as determined by its GPS hardware. Inaccuracies in these assumptions can cause packets to become lost in the network and eventually dropped.

General Direction Routing Protocol (GDRP) [3] is the multi-path routing protocol presented in this thesis. This protocol abstracts localization information into relative direction information, which allows for inaccuracies in the localization information. GDRP routes data towards a destination called the sink. Any nodes in the direction of the sink will forward the data. This routing behavior includes the additional benefits of multi-path routing and the dropped requirement of unique identification for sensor nodes.

GDRP is implemented in a custom WSN simulator written in Java. The simulator allows for multiple network protocols to be dynamically loaded. The simulator enforces an abstraction between network layers and requires that all sensor node communication be done by wireless transmission.

GDRP is compared to Bordercast [4], a multi-path routing protocol, and GPSR [5], a localization-based routing protocol which uses GPS hardware. Network properties such as setup latency, setup cost, routing cost, network congestion, reliability, and duplicity are measured for networks of varying number of nodes, node degree, MAC properties, and localization information correctness.

GDRP operates significantly better than Bordercast in terms of setup cost, setup latency, network congestion, and routing cost for networks consisting of a varying number of nodes and node density. GDRP operates more reliably than GPSR when faced with localization inaccuracy problems. It also operates with a reduced setup cost and setup latency. GDRP does have a greater congestion level for networks with a very large node degree when compared to GPSR. Other than the results mentioned above, GDRP operates similarly in all other categories to GPSR.

Chapter 2 is devoted to background information related to WSNs, and Chapter 3 is an analysis of related work. The design and implementation of the custom Java simulator is addressed in Chapter 4. Chapter 5 introduces the implementation of GDRP. Chapter 6 presents the simulation results and provides a comparison between the GDRP, GPSR, and Bordercast protocols. Finally, Chapter 7, serves as a conclusion and discussion of future work.

Chapter 2

2.1 Background

Wireless Sensor Networks (WSNs) are a collection of battery powered, wireless, sensor nodes which self organize into a multi-hop wireless network, often called a peer-to-peer or Mesh Network [6]. An edge device in the network which performs storage of sensed data or translation to a different medium is called a sink. The figure below illustrates a simple WSN.

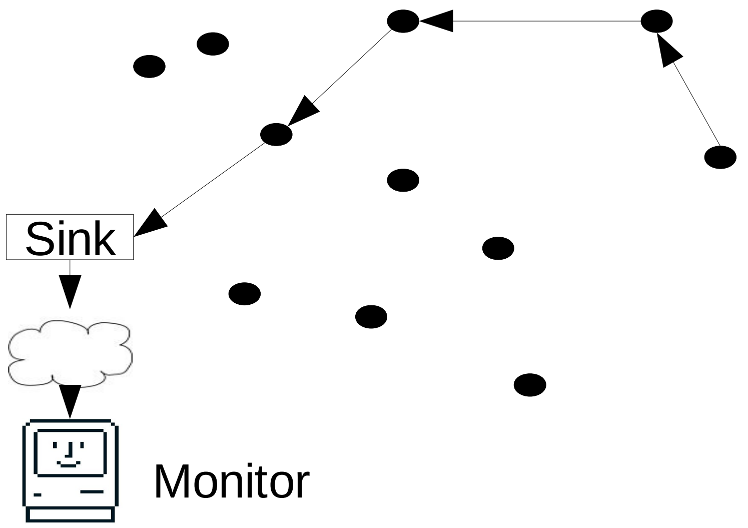


Figure 1: A Wireless Sensor Network (WSN).

There are many network parameters which contribute to the power efficiency of a routing protocol in WSNs. These include mobility, network node density, average node degree, the number of nodes in a network, and the traffic pattern.

Mobility continually changes the network topology requiring a routing protocol to relearn network state information. Based on how fast the network topology changes and the latency of a routing protocol's setup phase, sensor nodes can usually rerun a setup phase to relearn this information.

The average node degree is the average number of neighbors any node in the network. Node degree is controlled by altering the distances a node can transmit and the network node density. Network congestion increases as node degree increases. Network congestion usually leads to an increase in energy cost for network operation due to retransmissions required by collisions happening in the network. Network congestion also leads to a decrease in reliability, because a sensor node will start dropping packets if its routing queue is full.

2.2 Challenges in WSNs

WSNs provide some unique challenges that make historically accurate routing and data-link protocols less efficient and reliable. Most of these problems are directly attributable to WSNs being multi-hop wireless networks. As such, each sensor node is in the same collision domain as all of its neighboring nodes. This makes broadcast storms, routing loops, and packet collisions much more likely than in wired networks. There are also some challenges introduced by routing protocol implementations for WSNs that use GPS coordinates to perform greedy routing.

Most MAC protocols attempt to avoid packet collisions by checking the medium before attempting to transmit a packet [7]. This can result in a false positive error known as the exposed terminal problem. It is depicted in the figure below. Node A is transmitting to node B, and node C wishes to transmit to node D. Node C will determine that another node is currently transmitting, so it will hold off transmitting until the medium is free. This is a false positive, because node C could successfully transmit to node D without interrupting B's ability to receive the message from node A.

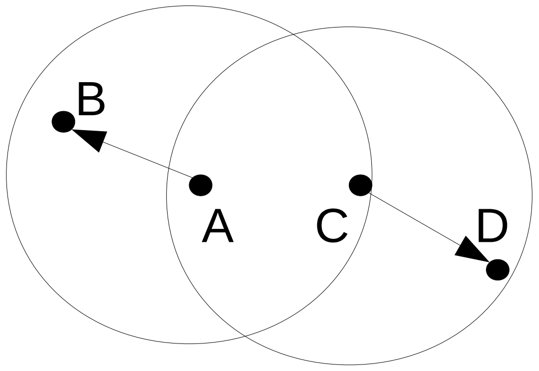


Figure 2: Exposed Terminal Problem.

There is also the possibility of receiving a false negative when sensing the medium before transmitting a packet. This problem is called the hidden terminal problem, and it is illustrated in the figure below. Node A is sending to node B, and node C wants to send to node B. After node C finds the medium free of transmissions, it starts to send to node B causing a collision in the network.

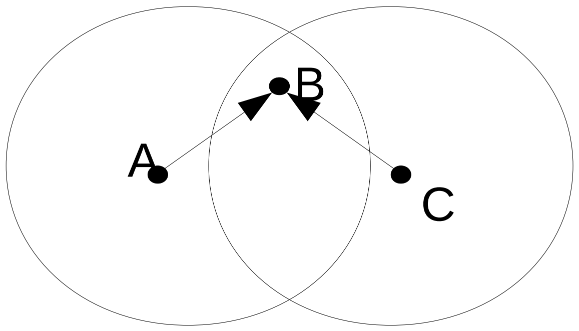


Figure 3: Hidden Terminal Problem.

Both of these inaccuracies can be improved by using a Request-To-Send(RTS)/Clear-To-Send(CTS) scheme [8]. The RTS/CTS model can only improve congestion at the transmitting node, and it is often the receiving node that needs to know about congestion. There have been other attempts to share congestion information between sensor nodes and many are presented in reference material [9].

Chapter 3

3.1 Related Work

There are several types of technologies that help to improve the capabilities and performance of Wireless Sensor Networks, and distributed algorithms and network protocols is one of these technologies [1]. This category of research defines more efficient operation in terms of fewer wireless transmissions to communicate data, resulting in overall power conservation. The rest of this Chapter is devoted to the routing technologies related to GDRP, covering an analysis of these protocols, and a discussion of solutions to other less known networking issues.

3.2 Routing Protocol Implementations

There have been many routing protocols proposed for wireless sensor networks. These routing protocols take differing approaches in order to exploit some property of the network to optimize routing for a given application. As described in Chapter 2, there are numerous types of categorizations of WSN routing protocols. This section will analyze three specific categories: multi-path routing, single-path routing, and localization-based routing. Particular routing protocols were selected to represent each of these categories. Flooding and Bordercast (ZRP) represent multi-path routing protocols. MCFA represents a single-path non-localization routing protocol, and GPSR, PAGER, and GeRaF represent localization-based routing protocols. The rest of this section explores implementations of these types of routing protocols.

Flooding

Flooding is an elementary routing protocol. The crucial part of its operation is demonstrated in pseudo-code in the figure below.

Forward (transmit):

```
if ( pkt.dst != me )
        broadcast ( pkt );
else
        sendToApplication ( pkt );
```

Figure 4: Simple flooding logic.

From this pseudo-code one can see that packets are exponentially created by being continuously broadcast in this network. This leads to a network broadcast storm, which exponentially decreases the lifespan of a sensor node and exponentially increases medium contention due to multiple nodes attempting to transmit at the same time. Flooding also induces many routing loops since it is perfectly valid in this setup for two adjacent nodes to just send the same packet back-and-forth.

On the positive side, flooding will route packets to the destination if a path to the destination exists. It does not rely on any forwarding path setup or maintenance, and it does not store any routing information in the sensor nodes.

Flooding does in fact work if you introduce the idea of a “hop’s to live” or (HTL) field. This field is initialized by the first node, and each node that receives the packet decreases the HTL value before retransmitting. When the HTL field equals zero the packet is no longer retransmitted. This approach can be seen in the figure below.

Forward (transmit):

```
if ( pkt.dst != me && --pkt.HTL)
          broadcast ( pkt );
else if ( pkt.dst == me )
          sendToApplication ( pkt );
```

Figure 5: Flooding with HTL.

The use of the HTL field reduces the exponentially increasing number of packets in the network while still allowing the protocol to successfully route packets to the destination. This routing protocol is classified as a multi-path stateless routing protocol. Packets will take several paths through the network to communicate a packet from the source to the destination. Flooding is a stateless protocol since it operates without the need to know any network information. This stateless feature allows the protocol to accommodate all mobility and failure models.

Although the above modification allows the flood routing protocol to operate, it does not prevent the case of two neighboring sensor nodes from transmitting a packet back and forth until the HTL field is zero. The way to combat these short routing loops in the network is to have a timed cache of recently received packets. This way identical packets that are received are ignored. This is demonstrated in the pseudo-code below.

Forward (transmit):

```
if ( pkt.dst != me && --pkt.HTL && notcached ( pkt ) ) {
          cache ( pkt , time );
          broadcast ( pkt );
else if ( pkt.dst == me )
          sendToApplication ( pkt );
```

Figure 6: Flooding with caching and HTL.

Unfortunately, this improvement requires each sensor node to retain information about packets they have received in the recent past. This modification greatly increases the amount of processing and memory used in an active sensor node, and moves the classification of flood routing to a semi-stateful multi-cast routing protocol.

Bordercast (Zone Routing Protocol)

Bordercast is an implementation of a Zone Routing Protocol (ZRP) [4][10]. This protocol involves maintenance of the network topology information at each node. Rather than maintain topology information on the entire network, Bordercast breaks the network into zones. A *zone* is defined with an integer value, which corresponds to the maximum number of hops a sensor node can be from any other sensor node to be considered within the same zone. This network parameter is also called the *zone radius*. Each node knows the topology of all of the other nodes in their *zone*. For example, if the *zone radius* was defined as three, then each node would maintain a list of all nodes that are within three hops. The figure below portrays two zones with a *zone radius* of two. B's zone includes nodes: S, G, F, E, A, D, C, and H. A's zone includes nodes: G, B, F, C, H, D, I, E, J, and K.

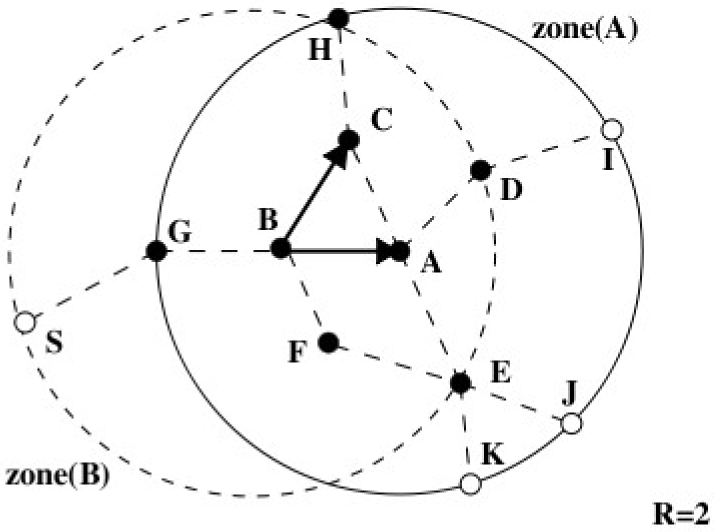


Figure 7: Two Bordercast zones with radius two.

The routing operation of Bordercast is also depicted in the figure above. When node B forwards a packet from G, it set nodes A and C as the next-hops. When node A receives the packet, it will find the next hops that will reach edge nodes that neither nodes C nor B can reach. This means it addresses its next-hops as D and E. This operation is illustrated in the figure below.

Forward (transmit):

```
if ( pkt.dst == me )
    sendToApplication ( pkt );
else if ( contains ( me , pkt.nextHops ) )
    pkt.TTL--;
    pkt.nextHops = getRoutesToUniqueEdgeNodes ( pkt.src , pkt.nextHops );
    pkt.src = me;
    broadcast ( pkt );
```

Figure 8: Pseudo-code of Bordercast operation.

Bordercast has several useful features. It does not allow the possibility of routing loops, the sink is always reached, and only limited network state information is stored by the node.

One of the drawbacks of Bordercast is the overhead generated. Some of the overhead is formed when generating next hop lists and the inclusion of these lists in the packets being transmitted. The next hop list is a variable length field in packet header, which increases packet size and increases the medium access contention in the network.

The Bordercast protocol has a couple of operational inefficiencies. This protocol will send a copy of the packet to every node in the network and not just a subset of all the nodes in the network. Visually, as nodes forward packets through the network the propagation forms an expanding ring. Since no directional information is maintained, packets must propagate out of each ring until the edge of the network is reached.

Other negatives of the Bordercast routing protocol include network topologies where a packet could traverse the network forever. Another drawback is that the operation of this protocol relies heavily on its configured parameters. If it is not parameterized correctly then occasionally fast moving nodes may cause non-optimal routing. In the worst case, there exists the possibility of dropped packets if the zone information is not updated quickly enough.

Minimum Cost Forwarding Algorithm (MCFA)

Another WSN routing protocol is the Minimum Cost Forwarding Algorithm (MCFA). This protocol starts with a network setup algorithm which is initialized by the sink and expands to all the nodes in the network. The setup accomplishes the initialization of a gradient descent tree to the sink from any given node in the network. The creation of the gradient descent tree eliminates the possibility of any routing loops in the network.

The gradient value can be derived from a number of variables. The simplest is the number of hops to the sink, but it could also include available energy, distance, and/or received signal strength (RSS). Each sensor node retains the gradient value for it to get to the sink and the cost of the link to get to its parent sensor node in the gradient descent tree. This parent node is also considered the next hop on the path from the node to the sink.

The gradient descent tree also solves the problem of getting stuck in local minima, which can occur with localization-based routing protocols that use only greedy forwarding. The critical section of the routing protocol is demonstrated in the pseudo-code in the figure below.

Forward (transmit):

```
if ( me != sink && ( pkt.cost – pkt.linkcost == me.cost ) )
        broadcast ( pkt );
else if ( me == sink )
        sendToApplication ( pkt );
```

Figure 9: Minimum Cost Forwarding Algorithm (MFCA) operation.

The MCFA protocol has several advantages and disadvantages. The major advantage is its ability to establish a single path to the sink and this path is guaranteed to be loop free. The disadvantages include the overhead of the entire network setup and its inability to accommodate nodes failing in the network without having to do another entire network setup. Until such a setup takes place all packets are dropped along the branch of the network tree where the failure occurred. Therefore, this routing protocol does not perform well for networks that have node mobility or node failure. As in Bordercast, the work around for this is to do persistent network setup.

MCFA is classified as a stateful single-path routing protocol. It always takes the shortest path, but it also must maintain network state information in each of the sensor nodes. The main criticism of MCFA is the lack of robustness for networks with node failures. A single failure could permanently partition much of the network, even if there were alternate routes that could keep it connected.

Localization (GPS) Protocols

Localization based routing protocols require the addition of new hardware. This hardware, attached to each node, provides each node with its own point of reference. This hardware is often in the form of a GPS locater. In general, the addition of new hardware to sensor nodes decreases their lifespan by increasing their operating cost (power consumption). Also important, when GPS devices are used, each node does not need to be configured to have a unique identifier, because their position can be used as a unique identifier.

Localization routing protocols usually assume that each node knows where the sink is located. This is often not the case in real networks. The position of the sink must be broadcast throughout the network at the start. Also, if the sink moves, then its new position needs to be rebroadcast. Once the location of the base station is established, sensor nodes do greedy forwarding of all packets. This means that each sensor node will only retransmit a packet if it is closer (via Euclidean distance) to the sink than the node from which the packet came.

A problem with localization routing protocols arises when networks have *holes* in them. A *hole* exists if a node has no other nodes closer to the sink within its transmission range, and there still exists a path to the sink from this node. For example, there exists a path in a network, but it requires backtracking away from the sink to reach a path that moves steadily towards the sink. This type of network can have adverse effects on the efficiency of localization based routing protocols [11].

Greedy Perimeter Stateless Routing, Geographic Random Forwarding algorithm, and Partial-partition Avoiding Geographic Routing protocol (GPSR, GeRaF, and PAGER) [5][12][13] all are localization single-path routing protocols which use GPS units as their localization hardware. In solving the problem of *holes* in the network, these protocols define the set of nodes that route a packet to a local minimum as a *concave* or *shadow* region.

GPSR was first proposed by a couple of students from Harvard, and it is extensively accepted as one of the best routing protocols for WSNs (probably because it is very easy to implement in its simplest form). GPSR introduces some very efficient ideas: it does not do any setup or maintenance during the lifetime of the network, and it includes all state information in the packet headers so that all future transmitting nodes have enough context to make a correct routing decision. These two things alone make this protocol very efficient in terms of maintenance overhead and packet delivery.

GPSR solves the problem of *holes* by taking a possibly non-optimal graph traversal of *concave* regions. It accomplishes this by maintaining state information in the packet and following the left-hand rule for following an edge around a *hole*: the node that is the closest to the sink on the left of the transmitting node will retransmit the packet. There are two problems with this. First, in the rare case the *hole* is shaped like a fractal, the left-hand rule could lead the protocol to get stuck and data is then lost. Secondly, since state information for traversal is included in the packet, the packet size grows linearly, and eventually if there are enough sensor nodes in the *concave* region, the packet could exceed the maximum transmission unit (MTU) size. Once this happens, the problem becomes more complex because the packet will need to be fragmented into several packets.

The PAGER protocol solves this problem by having all sensor nodes not in *concave* regions set up a gradient descent tree out of the region (the nodes entering the region have the highest cost). The *concave* regions are identified as containing a node that is closer to the sink than all of its neighbors. This node then initiates the setup of a gradient descent tree out of the *concave* region. The sensor nodes at the border of this region act as gatekeepers and disallow any packets to enter the *concave* region. Any packets that are generated in the *concave* region are routed out of the region via the gradient descent tree. The setup internal to the *concave* region is very similar to a MCFA setup. This state information needs to be retained on each of the sensor nodes in the *concave* region. In this way, PAGER is no longer a stateless routing protocol, which means if there is any node movement or any node failures, then the *concave* region will need to be set up again.

GeRaF is slightly different than the other localization routing protocols, because it introduces the idea of sleep/wake duty cycles for nodes to increase their lifespan. There is a decrease in reliability when sensor nodes randomly sleep/wake, but there has been research that shows this is less of a problem with an increased node degree [14]. GeRaF circumvents this problem by having sensor nodes sleep and wake on an established duty cycle. In this fashion, all sensor nodes know when their neighbors will be awake versus sleeping.

Although GeRaF is slightly different than GPSR and PAGER, it still relies on localization based routing. The way it combats the problem of *holes* is to have the network partitioned into good and bad regions. Sensor nodes located outside *concave* regions are classified as belonging to the *priority zone*. Nodes not in this zone broadcast their packets until they reach the *priority zone* where normal greedy forwarding is used. This technique is less effective than the one used in PAGER, because the nodes in the *concave* region are not able to know when the packet has reached the *priority zone*. Since sensor nodes in the *concave* region do not know when the packet reaches the *priority zone*, they will continuously rebroadcast the packet in the *concave* region. This will cause the sensor nodes in the *concave* region to exponentially lose power. It will also results in multiple instances of the same packet be transmitted into the *priority zone.* This results in the sink receiving the same packet multiple times.

There are other inherent problems which occur with GPS units which can limit their usefulness in WSN routing protocols. This includes the fact that there are regions where a sensor network would be useful but GPS is not available. Such physical locations could be around mountains, inside buildings, or in caves. Another difficulty that arises in networks which use GPS units is a possible lack of precision [15]. This lack of precision could cause the loss of packets or the possible formation of routing loops by nodes with erroneous GPS location data.

3.3 Unique Identification

One of the limitations in previous network routing protocols was how sensor nodes obtain unique identifiers for routing operation. This unique address often takes the form of a Medium Access Control (MAC) address. Programming individual sensor nodes with unique addresses at manufacturing time costs more, and sensor nodes are meant to be manufactured as cheaply as possible. There has been very little research done on how to actually dynamically assign identification numbers to a network where none exist.

Currently, the most sophisticated algorithm for assigning numbers is Dynamic Address RouTing (DART) [16]. Once this complex algorithm is completed, the network is configured into a hierarchical routing tree with sensor nodes knowing who their parents are and how to route traffic to the sink.

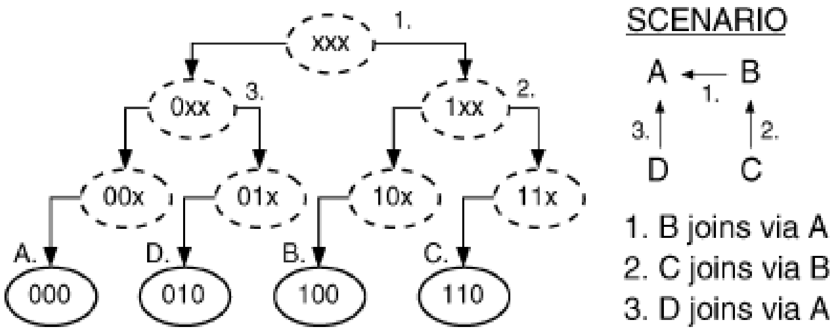


Figure 10: Dynamic Address RouTing (DART) setup.

The figure above contains an example of a hierarchical routing tree setup. The sink (portrayed by node A in this diagram) chooses the lowest possible number, 000. As a single neighbor joins A (node B in this case), it receives the number 100. This is because when a new branch joins at an equal level, it receives half of the remaining address pool. Once a node joins with B, it also receives half of the remaining pool and selects the lowest number possible. This is the same operation performed by node D when it joins with node A. If the number of nodes in a single branch exceeds the number of available numbers the tree needs to be re-balanced. This re-balancing is expensive in terms of network packets.

3.4 Antenna Types

The type of antennas used in a network make a significant difference in the way sensor nodes can communicate. This is due to different antennas having different radiation patterns. The radiation pattern, in addition to other important parameters, determines where a receiver must be located to accurately receive and decode the signal.

Antenna models are typically broken into three different types. The dipole antenna is typically considered an omni-directional antenna, at least in terms of the axis perpendicular to the direction the antenna is pointing. There are also directional antennas which transmit signals with a higher gain in a certain direction. In comparison to the omni-directional antennas, nodes in the direction of the higher gain can be further away from the transmitter, while the nodes not in that direction would need to be located closer to the transmitter.

Radiation patterns, gains, and interference can be very difficult to model in a simulation environment, so quite often in research the selected antenna model is an *isotropic radiator*. This antenna is a fictional type of hardware, because it occupies no space, has no mass, and has a perfectly spherical radiation pattern.

3.5 Beamforming Sensor Arrays

GPS is not the only type of localization hardware which can be used in Wireless Sensor Networks. Beamforming Sensor Arrays (BSA) [17] can determine the direction of an incoming signal by looking at its strength and phase shift at several different points in the sensing array.

Some work has been done to greatly reduce the computations necessary to determine the direction of an incoming signal. This reduction in computation reduces the power required to perform the operation, and therefore makes this type of solution viable in WSNs.

By determining a signal's direction, sensor nodes are able to know relatively where they are located with respect to their neighboring nodes. This is assuming that all nodes have a common point of reference, which can be easily provided by a digital compass. A BSA also can share the precision problems of GPS caused by environmental randomness and/or interference, requiring fixing or adapting to these errors in the routing protocol.

3.6 Simulator Types

An ad-hoc wireless network simulator is the piece of software which allows researchers to gather statistical information generated by their routing protocol as compared to other routing protocols. It brings together various models, hardware interactions, parameters, and environmental variables to test a WSN under various modes of operation. Typical measurements from simulators include throughput, latency, bandwidth, packet count, and energy cost. Quite frequently packet count can be directly mapped to an energy cost by some clever heuristics.

The biggest difference in types of simulators is how they simulate the passing of time for many parallel operations on a single (or multi-core) processor. It is very inefficient to require a full overlay network with a single processor for each sensor node in a network to simulate network operation. In a simulator, time can be simulated as either a discrete time value or in a vector approximation.

The *discrete time* model operates by having simulators allocate the same amount of *time* for each sensor node. After all sensor nodes have finished a particular operation in this block of time, the simulator increases the *time* value. A tunable parameter would be how large to make each time slice. If too small, the simulator may take an unreasonable amount of time to run because lots of sensor nodes might be idle while others run a single instruction. If the time value is too large, the accuracy of the simulation decreases, because sensor nodes can perform multiple operations seemingly instantly.

The *vector time* model operates on an event driven basis. The simulator knows how much time each operation takes, and sensor nodes register the start time of each operation. The simulator schedules when sensor nodes can perform their registered operations, and in this manner dead times can be eliminated by skipping simulated time to the next scheduled event. In this model, time is stretched and compressed and maintained across all nodes. This model can significantly speed up the simulator and increase its accuracy.

Chapter 4

4.1 Simulator Implementation

General Direction Routing Protocol (GDRP) is implemented, tested, and compared to Bordercast and GPSR using a custom, ad-hoc wireless network simulator written in Java. The simulator is a discrete time simulator. It provides generic physical, data-link, network, and application interfaces for each of the sensor nodes allowing researchers to quickly implement new protocols and dynamically test different combinations of protocols. The simulator provides typical research models for dispersion, communication, and failures. Also important, the simulator includes a very helpful logging mechanism which allows researchers to easily log information and then gather statistics at the end of the simulation.

The simulator is organized into several parts. The main simulator Object is named the World Object and is responsible for setting up the simulation parameters, setting up the sensor nodes in the network, directing sensor node operation, and controlling the physical medium. The World Object has many Node Objects, each representing a sensor node in the network (or World). Each Node Object has several interfaces. These interfaces line up with layers 1, 2, 3, and 7 of the OSI Communications model. These are, respectively, the physical layer, medium access control layer, routing layer, and application layer. Each interface allows for future protocol development by abstracting the general operation of a particular layer. This allows researchers to easily integrate and test new protocols while leveraging previous work. A diagram of the Object interaction in the simulator is visualized in the figure below.

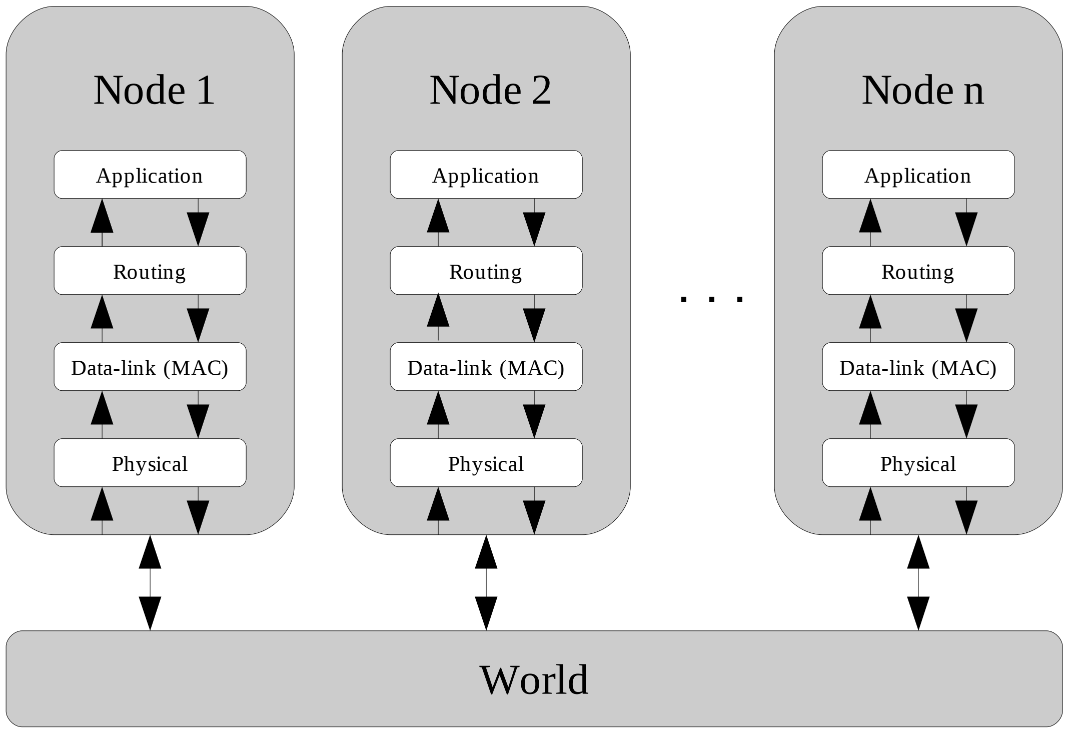


Figure 11: Simulator Object interaction.

The simulator is a discrete time simulator. This means that each sensor node will run for exactly one time-slice, then the time will increase to the next time-slice and all the sensor nodes will run again. In this fashion, all of the sensor nodes are kept synchronized with “real” time. This model is used because the design of a vector time simulator (discussed in section 3.6) is very complex. A vector time simulator would also have the additional problem of parameterization: knowing which actions to group into events that can be scheduled, and the run times for each type of event. Data for these parameters would have been very difficult to collect accurately and are not the focus of this research.

In this simulator, the time-slice represents the amount of time to process and transmit a single byte. Not all sensor nodes fully utilize their time-slices, but this puts an upper limit on the amount of time available to a sensor node to run, which keeps all sensor nodes synchronized in a congruent time-line of actions. A different length of time could have been used, but some of the common ones come with some disadvantages. For example, if the time-slice is the time to transmit a single packet, then the probability of encountering hidden and exposed terminal problems is very unlikely. If the time-slice was set to be the time to transmit x% of a packet, then any measures of throughput will be identical for all protocols: a packet's header size would be irrelevant. The amount of time to transmit a single byte seemed to be the longest chunk of time per run cycle that allowed sufficiently good modeling of real world operation and was selected for use.

Java provides the programmer with the capability to easily enforce the abstractions between World, Nodes, and the Objects implementing the network layers. A strict hierarchy is maintained, which disallows any information to be communicated between sensor nodes except by wireless transmissions. If this enforcement was not present nodes could have capabilities not possible in a real-world environment. This all is achieved through protected, public, and private Object declarations and Object ownership.

Writing a simulation in the simulator is relatively easy. By design, the World Object contains all simulation variables initialized for a default simulation. All a simulation driver needs to do is: instantiate a new World Object, change any of the publicly available variables in the World Object, call the member function *createWorld* with a String array containing the Class names for each of the implementations of the network interfaces, and then call World's member function *run*. When the simulation is finished the World Object combines the logs for each of the nodes in the network into one big log and returns from the *run* function. The World's log Object is public and available for statistics gathering by the driver.

4.2 World Design

The World Object is responsible for maintaining all simulation parameters, all simulation variables, sensor node instantiation, sensor node dispersion, the mobility model, the failure model, and control of the physical medium (the transmission model). As mentioned in the previous section, World itself is instantiated, created, and then run. When it is instantiated, it initializes all simulation parameters to default values.

When the World is created, it takes the simulation parameters and creates a network. This is done by instantiating a number of Nodes and positioning them according to the dispersion model. Before this function returns, the World checks to see if network is fully connected via the transmission model. If it is not, then it tries the dispersion model again (if it is possible to obtain a different result) or just returns a boolean false to the invoking Object. This means that if World is not able to create a network that is fully connected, then the simulation fails to run.

Having a fully connected network is important for simulation. If a network is not fully connected the parameters that define the simulation are effectively no longer correct. For example, having 2 disjoint nodes in a network results in a simulation of (n-2) nodes. Another example would be events witnessed by only disjoint nodes resulting in a reduced measurement of protocol reliability.

There is a significant amount of randomness in the models implemented in the World Object. This is why, when the World Object is instantiated a pseudo-random number generator (RNG) seed is supplied. In this fashion, the operation of World is deterministic because the same path is always taken through the code and calls to the RNG occur in the same sequence. A different seed provided to the World effectively results in a different simulation bound by the same simulation parameters. For example, a different seed would result in a different physical network topology.

When the World Object is *run* it calls the *run* function for each Node Object in the network, and each Node can register a single byte of a packet to be broadcast if they are transmitting. After each of the Nodes run, the bytes are broadcast to each Node via the transmission model. Once each Node has run, an event generator is run. When this is finished, the time-slice is increased and the whole process starts over. The World Object continues to perform these actions in sequence until all events have been generated and all of the Nodes are idle (not transmitting/receiving or processing).

Sensor data at each of the sensor nodes is collected by sensing the World Object. The World Object contains an event generator which will randomly create events in the World environment. An event is guaranteed to be able to be witnessed by at least one Node: it will always occur within a Node's sensing distance. The reasoning supporting this design decision is analogous to the argument of a tree falling in the woods with nobody around to witness it. These events only last one time-slice, so the Application layer of a Node needs to sense the World every time the Node runs. If it fails to do this, then it is possible it misses an event.

Events can only occur after all of the Nodes in the simulation have attempted a setup procedure and the network becomes idle. The parameters determining the occurrence of events limit how frequently events can happen and how many events will occur in a single simulation. This shapes the traffic model of the network.

The World Object contains the master log. At the end of a simulation, it combines the logs from all of the Node Objects for total simulation statistics. This log is very general and typically contains counts of packets, collisions, number of witnesses for a particular event, and how many events the Sink received. This information is then combined with simulation parameters, such as the number of sensor nodes in the network, to measure protocol performance.

Parameters

There are many simulation parameters contained in the World Object. The large number of parameters was the main motivation for allowing the parameters to be publicly mutable. The parameters being publicly accessible also allows individuality in the creation of Simulation drivers. These simulation parameters and their default values are the subject of the rest of this section.

name	value	description
sink_on_outside	TRUE	This parameter sets a Node on the edge of the network as the Sink.
num_nodes	80	The number of Nodes in the network.
node_density	0.030	The minimum Node density of the network.
node_failure_rate	0.0	The rate at which nodes might cease to run.
trans_failure_rate	0.0	The rate at which network transmissions will be randomly corrupted.
num_events	50	The number of random events generated in the World.
event_chance	0.001	The random chance of an event being generated in the World.
debug	FALSE	A debug flag for printing out simulation status at runtime.
node_trans_dist	8	The length a broadcast transmission from a Node will travel.
node_view_dist	5	The distance a Node can view the World for an event.
mac_buffer_size	500	The maximum buffer size at the MAC layer.
mac_backoff_step	30	The initial back-off step taking in the back-off period of CSMA/CA.
cache_timeout	4000	The duration entries are valid in the routing layer cache of recently seen packets.
node_zone_radius	2	Bordercast's zone radius parameter.
num_slices	26	The number of slices of the 360° space in GDRP.
node_loc_accuracy	90%	The accuracy of GPS location information for GPSR.

Table 1: Default simulation parameters.

The number of nodes in the network directly effects the number of routing decisions that need to be made. In multi-path routing protocols, an increase in the number of nodes leads to a large increase in the number of collisions in the network. The rate at which events are generated and the average node degree also effect network congestion. Too high of network congestion leads to congestion collapse and packets being lost due to a finite buffer size in the nodes.

The maximum buffer size serves as a hard limit on the congestion that can happen in a network before congestion collapse occurs. If the buffer size were infinite it would be possible for simulations to take a millennium to finish. Having the buffer size hard capped allows for simulation time to be reasonable and to serve as a typical hardware limitation. This is also the case of the routing level cache timeout value. It serves as a hardware limit on memory size and an equalizer for performance and reliability when a network is forced to deal with congestion.

The location accuracy of GPS coordinates effects the reliability of GPSR's network transmissions. This is similar to the number of slices used for GDRP. The greater the number of slices for GDRP, the higher the likelihood is of accurate information.

Node degree is a very important network parameter, but creating a network from a known node degree is a difficult problem. This results in measuring the node indirectly by changing the node transmission distance and the network density.

Node Dispersion Model

The node dispersion model in the network is a random dispersion limited by the node density and the transmission range of the nodes in the network. Since the network is on a two dimensional plane and guaranteed to be connected, the network density only serves as a lower bound for the actually network density.   There is quite a bit of randomness in the dispersion model, so it is a required that the World is seeded with a known value to ensure simulation repeatability.

The model is an iterative algorithm. First an area is bounded by the node density and the number of nodes in the network. This defines the area of a network in which nodes can reside. Then all nodes are randomly generated inside this area. Once this is done, the iterative part of the algorithm starts. A connected matrix is created to keep track of which nodes are within the bounds presented by the transmission model. All nodes not in this connected matrix (all disjoint nodes) are randomly placed in a new position in the network. This continues until all nodes are in the connected matrix. At this point the fully connected network is created with a minimum density of the value specified.

Transmission Model

The transmission model used in all simulations is one using isotropic radiating antennas. These are antennas that: have no mass, occupy no space, and transmit with the same gain in all directions. This radiation model is illustrated by a sphere with the antenna at the center and a transmission radius limiting the distance of decipherable transmissions.

Since the network is on a single plane, all transmissions are modeled by circles. All communication can be modeled by bidirectional links, because each node has an equal transmission radius.

Each packet transmitted in the network is transmitted at a rate of one byte per time slice. There allows plenty of opportunity to measure how well packet collisions and congestion are handled by MAC and routing implementations, because transmissions are streams of bytes over time and not discrete events.

Mobility/Failure Model

The mobility model is no mobility, and the failure model is no failures. There is no model to mimic transmission errors resulting from random world noise. The implementation of a failure model is a subset of a mobility model. Failure can be modeled by the removal of a node from the network whereas a mobility model requires that a node be effectively removed from the network and inserted at a new location. A mobility model basically requires a routing protocol to setup state information again. Therefore, a routing protocol's ability to accommodate mobility can be achieved by performing well on setup latency and setup cost. Both of these are able to be measured without directly implementing mobility or failure models.

4.3 Node Design

Each Node Object models an entire sensor node. It contains Objects for each layer of the network stack that is implemented for a Node. These Objects are dynamically loaded which allows for a researcher to implement a particular layer's interface without changing any other code. The Node Object itself represents the backplane of the sensor node, and provides interconnection methods between all of the networking components. The Node Object forces separation between all of its member Objects, and requires all communication to pass through the Node Object.

The Node itself is run via its *run* method from the World Object during its turn in a time-slice. At this point it handles the operation between each of the network Objects in the Node. It only has one major constraint in that it can only transmit a single byte during its turn. It can do as many calculations it needs, but if it tries to transmit more than once, only the final transmission will actually be sent. This is because a Node is allowed to only transmit a single byte during a single time-slice.

Dynamic Loading

The Node Object uses the Java Reflection Libraries to create the Objects for each of the network layer interfaces. This allows researchers to implement a new network layer Object and then just specify its name to have it Dynamically loaded at runtime. A sample set of code to perform Object creation for a particular interface is in the figure below.

```
Object[] o = new Object[] { this };
Class[] params = new Class[] { Node.class };
Routing_protocol r;

try {
        Class cls = Class.forName( arg[0] );
        Constructor c = cls.getConstructor( params );
        r = ( Routing_protocol ) c.newInstance( o );
} catch ( Exception e );
```

Figure 12: Example using Reflection to create an Object.

In the above example, Routing_protocol is an Interface. A researcher will create a Class that implements this interface, and then pass in the name of this new Class as *arg[0]*. The constructor for all network interfaces require that the Node Object be passed as the only parameter. This is why the Object[] and the Class[] contain just the invoking Node Class and its Class type respectively. The Reflectance library will get the Class of the network layer implementation (in this case a routing layer) by the name specified in *arg[0]*. From here, the Reflectance library will get the Constructor that matches the parameter set specified by the variable *params* (a single variable of type Node). After this is done, the Reflectance library will then create a new Object using the constructor *c* and the parameters *params*. The returned Object is then cast as the type of interface it represents, a Routing_protocol.

Each packet transmitted in the network contains the headers of each of the network layers encapsulating the application data. Each network header has an interface of basic methods that need to be implemented for basic operation for a new network layer. The networking layers are responsible for knowing their type of header and casting the Interface descriptor accordingly. This serves as an easy and generic way for any layer to implement custom headers.

Dynamic loading is very useful for leveraging previous work and implementing slight changes without needing to change non-relevant code and recompile everything. It allows Nodes to be of a single type and have any number of interior workings: otherwise, a researcher would need to implement a new type of Node for each set of interior protocols.

Layer Interaction

During a time-slice a Node is run exactly once. When it is run, the Node makes sure that the correct OSI layers are run in the correct order. This means the Node needs to model an interrupt driven interaction between all of the OSI layers to handle three different types of data flows which can occur at any time. These flows are: received data from the medium, received data from the application layer, and sent data to the medium. The Node uses a state machine type structure to manage the run order of the each of the layers in the network stack. Transitions between states occur based on the state of each of the layers. There are some general limitations which help to design the run order: the physical layer runs *at most* once (only sending or receiving one byte per time-slice), the application layer runs *at least* once (to sense information from the World), the MAC layer runs *at least* once (it manages the buffers of the Node and handles the back-off sequence due to congestion), and the Routing layer *may not need* to run at all. This is all diagrammed on the next page in a state diagram, including a legend for each type of flow.

As mentioned previously, each of the layers communicates through the Node Object. For example, for the MAC layer to communicate the newly received packet with the routing layer, it needs to call a wrapper method in the Node Object. This prevents out of band communication between the components in a Node, allowing for easy monitoring and logging. This model simulates an asynchronous component interconnect bus controller, managing the flow of data between components, which might not be the optimal model for a communications device where all operations are usually contained in a single micro-controller.

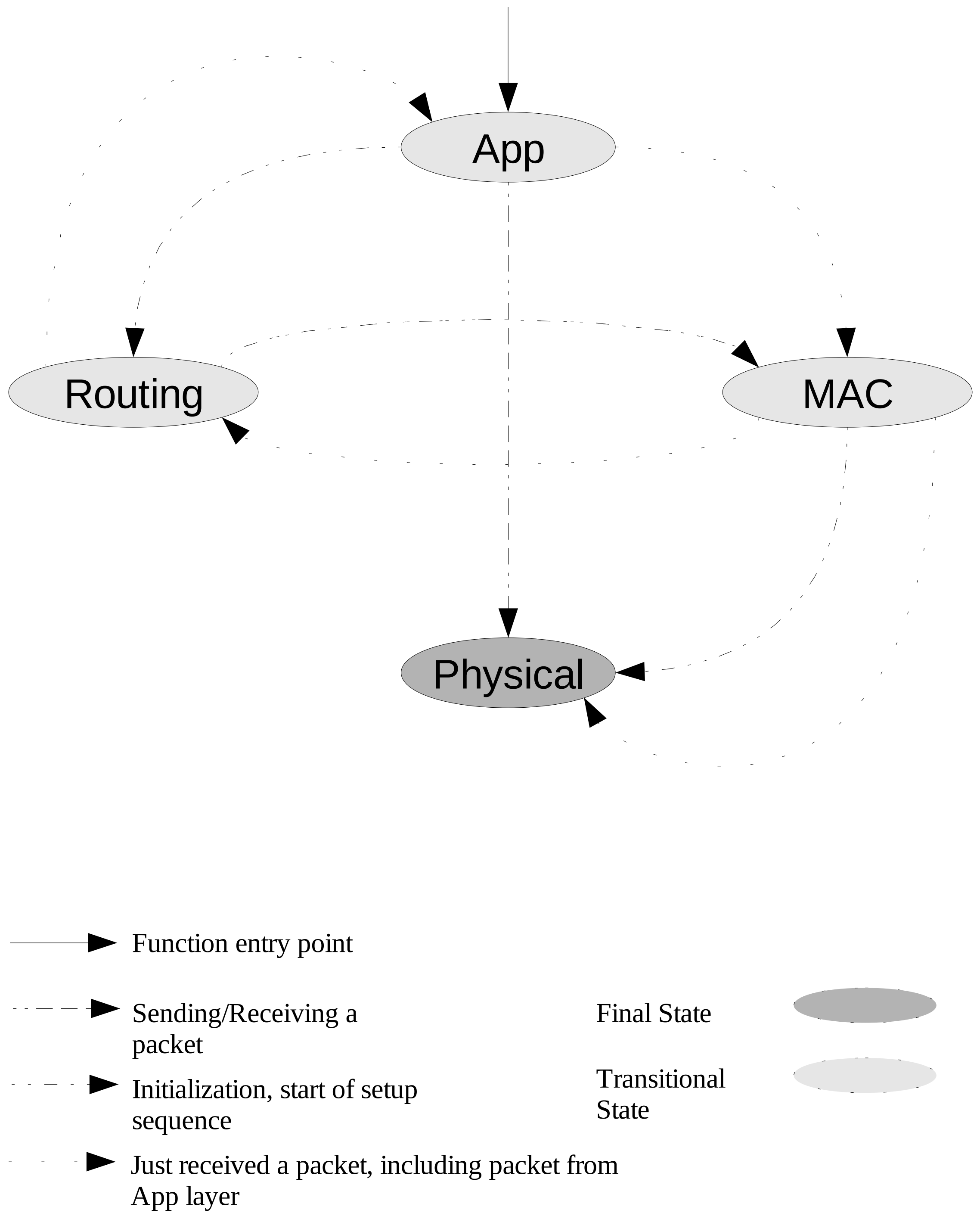

App
Routing
MAC
Physical
Function entry point
Sending/Receiving a packet
Initialization, start of setup sequence
Just received a packet, including packet from App layer
Final State
Transitional State

Figure 13: Network layer transition diagram.

4.4 Default Physical Layer

The Default Physical layer is the implementation of the OSI layer one in each of the sensor nodes. It handles the direct transmitting and receiving of bytes in the network, combining the bytes into packets, and working with the MAC layer. The current methodology requires that the Physical layer sends or receives only a single byte at a time.

The Default Physical layer has only one buffer. It is the size of the maximum packet length in the network, and it will contain a single packet when transmitting or receiving. If the MAC layer does not immediately read the packet from the physical layer after it is done being received, then a later transmission will wipe away the data. Conversely, if the MAC layer does not wait for an entire packet to be sent before sending another, the second packet will overwrite the first, and the first packet is lost.

The Default Physical layer's interaction with the World is via wrapper functions in the Node Object, *recvPacket* and *sendPacket*. The receive function provides the physical layer with the entire packet and the index of the currently transmitted bit. The send function requires the same pieces of information. The entire packet is passed in this transaction for the physical layer to easily determine if a problem occurred. Typically this would be done by running a checksum over a received packet, but in this simulator the packet is considered to be complete when all bytes have been received.

There are several failure recovery mechanisms. If sufficient progress is not made on receiving a packet due to a collision happening at the sending node, the packet is dropped. If a node starts to receive a new packet while receiving another packet, a collision is detected. The World Object will also detect collisions due to the hidden terminal problems in the network and notify the corresponding Node Objects: this is similar to a notification mechanism used in

CSMA/CA when a collision is detected. The physical layer will clear all state information when a collision is detected.

4.5 CSMA/CA MAC Layer

Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) is the MAC layer implemented in the simulator. It attempts to control access to a medium that can have multiple accesses (multiple nodes trying to transmit simultaneously). If it has a packet to transmit, it senses the medium to see if any other transmissions are taking place before forwarding the packet to the physical layer. If it detects that another transmission is taking place, then it will attempt to transmit after a period of time. Typically, this period of time is initially the minimum size of a packet, and then as the number of failed attempts increases the amount of time increases exponentially. This can allow some nodes to monopolize the medium, which in turn prevents other nodes from ever being able to transmit anything. This MAC protocol is able to detect if a collision happens in the network. When a collision is detected, the same back-off mechanism is used as when a packet is unable to be transmitted. In the back-off equation below, the $delay_{step}$ variable is a network parameter, and is usually the time to transmit a single packet. The $failure_{number}$ is the number of failures that have occurred since the last successfully sent or received transmission.

$$delay = delay_{step} * rand\left(2^{\left(failure_{number}\right)} + 1\right) + rand\left(delay_{step}\right)$$

The MAC layer is often responsible for buffering packets when there is congestion in the network, and this simulator's implementation of CSMA/CA performs the same function. The buffer size is limited as a simulation parameter, because it often controls the reliability of a network: for example, if the buffer size was infinite, the network could be completely reliable, but would take a millennium to complete any useful work. When the buffer is full, any new packets that wish to be transmitted are dropped. A node is still able to receive a single new packet at a time because the physical layer maintains a single packet buffer: if this was not the case, then once the CSMA/CA implementation's buffer is filled all network operation would cease.

There are relatively few operations to be handled in the CSMA/CA implementation in the simulator. The CSMA/CA implementation maintains a queue (the buffer), where new packets are added to the end of the queue (from the routing layer), and the front packets in the queue are dequeued for transmission when the medium is idle. The CSMA/CA implementation senses the medium before each transmission to make sure no other transmissions are currently taking place. This can lead to exposed terminal problems, but solving this network issue for CSMA/CA is not in the scope of this work. The CSMA/CA implementation also maintains the back-off delay. When the back-off delay is in effect, the CSMA/CA implementation will perform no actions other than decrementing the count of the remaining delay.

The CSMA/CA implementation keeps track of the current packet being sent by the physical layer and is notified when a collision occurs in the network. When this happens the packet is put back at the front of the queue, and the back-off procedure takes place.

The following figure illustrates the CSMA/CA implementation's header. It only utilizes two fields, a flags field and a packet size field. In normal operation, the header would often contain other useful information like version number and header checksum.

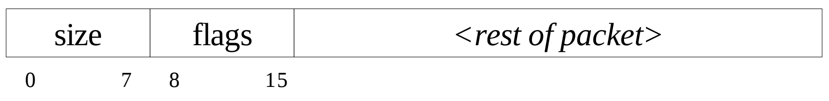


Figure 14: CSMA/CA packet header.

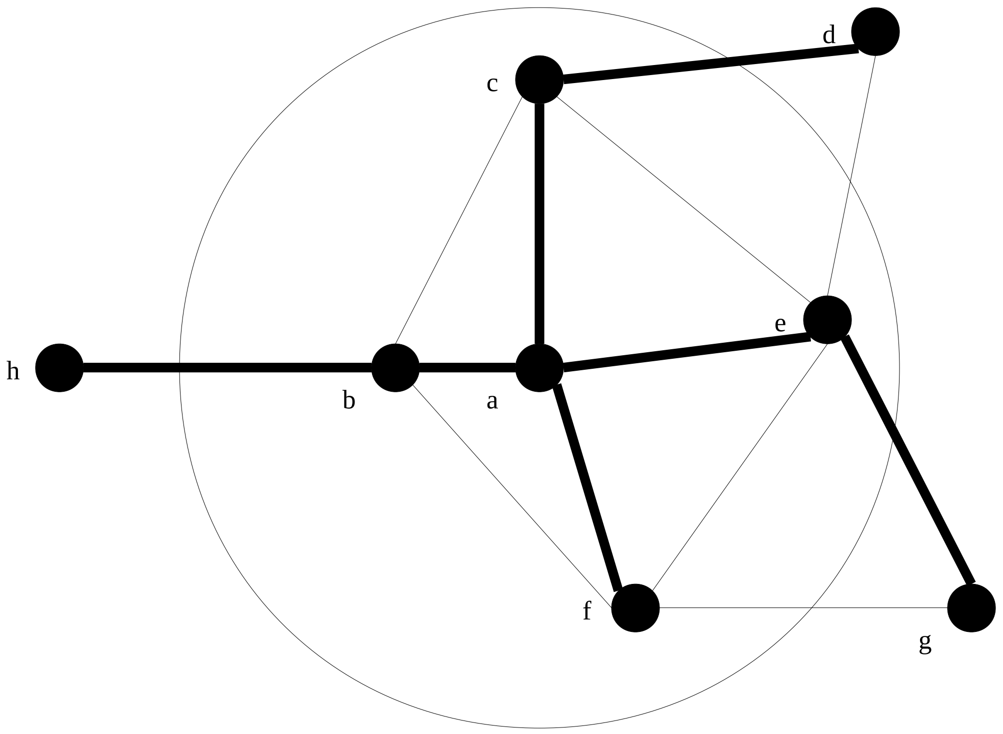
4.6 Bordercast

The main operation of Bordercast was defined in the Related Work section 2.2. It is the first of the two comparison routing algorithms used for GDRP validation and analysis. This section serves as an explanation of how it is implemented in the simulator.

The setup part of the protocol is initiated during the first time-slice. Each node will broadcast out a setup request packet. This involves setting: the source of the packet as the node index, the Time-to-Live (TTL) value to equal the zone radius, and the destination as the broadcast address. Each node that receives a request packet with a valid TTL value (greater than zero) will decrement the TTL and retransmit it in the network. In the case where the TTL is equal to the zone radius, then the node labels the sender as a neighbor. When the TTL of a setup request packet equals one, then the packet should be at the edge of the zone for the setup node. The node at the edge of the zone will generate a setup reply message. This reply message is set to have: the source of the node where the TTL expired, the TTL value equal to the zone radius, the destination as the source of the request packet, and, lastly, the via value as the node index. At each hop along the way back to the source of the initial setup packet, nodes will decrement the TTL and set their own identities as the via value of the packet. When the packet finally reaches the source of the initial setup packet, the setup node has now identified a node that is at the edge of its network. The header information is in the figure below, and the setup process is highlighted on the next page.

dst	src	flags	ttl	*n* nextHops	<*rest of packet*>
0 7	8 15	16 23	24 31	32 31 + *n* * 8	

Figure 15: Bordercast (BC) packet header.



Setup paths:	**Action taken:**	**Reply paths:**	**Action taken:**
a→b→a	drop		
a→b→c	drop		
a→b→f	drop		
a→b→h	generate reply	h→b→a	add h via b
a→c→a	drop		
a→c→b	drop		
a→c→d	generate reply	d→c→a	add d via c
		d→e→a	add d via e
a→c→e	drop		
a→e→a	drop		
a→e→b	drop		
a→e→d	generate reply	d→c→a	ignore
		d→e→a	ignore
a→e→f	drop		
a→e→g	generate reply	g→e→a	add g via e
		g→f→a	add g via f
a→f→a	drop		
a→f→b	drop		
a→f→e	drop		
a→f→g	generate reply	g→e→a	ignore
		g→f→a	ignore

Edge nodes by their via: h ← b | d ← c | d,g ← e | g ← e

Figure 16: Bordercast example setup.

There are a couple of edge cases that can cause problems if allowed to occur. The most important case of neighbor nodes thinking they are edge nodes is solved by setting the source of the packet as a neighbor in each of the recipient's routing tables if the packet has a TTL equal to the zone radius. This will greatly limit the number of setup replies generated in each zone. If the via value is set to the same value as the source address, then the reply packet is ignored, because the zone edge node is really a neighboring node.

All of this information is contained in a large routing table at each node. Each neighboring node, or via, entry in the table contains a list of edge nodes that are accessible through it. When a packet comes in with the current node in the next-hops list, the node will determine the next-hops the packet should traverse. If the packet destination is in the via list, then it is the obvious next-hop. Otherwise, the node finds the set of vias disjoint from the union of the next-hops specified in the current packet and the source of the packet. From this set, the node finds the edge nodes that are not accessible by these neighbors, or vias. If all of the edge nodes are accessible by this set, then the packet is dropped. If there are edge nodes not accessible from the set, then a list of the fewest vias to access these nodes is generated (using a greedy method since the problem is NP complete). This list is included in the packet that will be forwarded as the next-hops field. The packet will also have its TTL and source fields update before transmission.

Between the setup of the network and the routing operation, this is a very bursty routing protocol. As such, it creates a large amount of congestion in the network which causes a decrease in performance and reliability, which in turn leads to a decrease in network performance in general.

To help with some of the inefficiencies, the routing protocol has a timed cache to filter duplicate packets that were recently seen in the network. The routing layer saves a hash of each packet for a certain amount of time, and when an identical packet is received the packet is ignored. This helps to eliminate some of the network congestion, therefore increasing reliability.

This technique is implemented, for consistency, in each of the other routing protocols in the simulator.

4.7 GPSR

Greedy Perimeter Stateless Routing (GPSR) is a routing protocol that utilizes GPS localization information to make routing decisions (section 3.2). It serves as the second of the two comparison protocols to GDRP. It operates on the simple premise that nodes which are closer to the destination should retransmit a packet, and the ones further away should not. This is called greedy routing. An important edge case occurs when the current node responsible for forwarding a packet does not have any neighbors closer to the destination. When this happens, the node is a local minimum in the network. At this point, GPSR falls back to using perimeter routing (using the left hand rule) to move around the network. The location of a node serves as a unique identifier for addressing.

GPSR goes through a small setup phase. This setup phase disseminates the location of the sink node to all of the rest of the nodes. It also serves to populate the neighbor list of each node. Each node maintains a neighbor list, and when a packet needs to be routed, the closest (or the left-most during the the fall-back perimeter routing) neighbor is selected as the next-hop of the packet. The packet structure is defined in the figure below.

nextHop	src	dst	flags	ttl	*<rest of packet>*
0 127	128 255	256 383	384 391	392 399	400

Figure 17: GPSR routing header.

Forward (transmit):

```
if ( me == sink)
        sendToApplication ( pkt );
else if ( pkt.nextHop == me )
        pkt.nextHop = bestNodeToDest ( pkt.dst );
        pkt.TTL--;
        broadcast ( pkt );
```

Figure 18: Pseudo-code of GPSR routing.

The above figure shows pseudo-code of the main portion of routing decisions made in a sensor node using GPSR. When a new packet comes in with the current node set as the next-hop, then the current node will find the closest neighbor via Euclidean distance. In the case where the node is a local minimum, the next-hop is the left-most node from the vector from the source to the destination. This decision is made by selecting the node with the smallest angular value according to the equation below. This is illustrated in the next figure. The entire operation is illustrated in the following figure.

$$\alpha = \pi - \arctan\left(\frac{dest_x - neighbor_x}{dest_y - neighbor_y}\right)$$

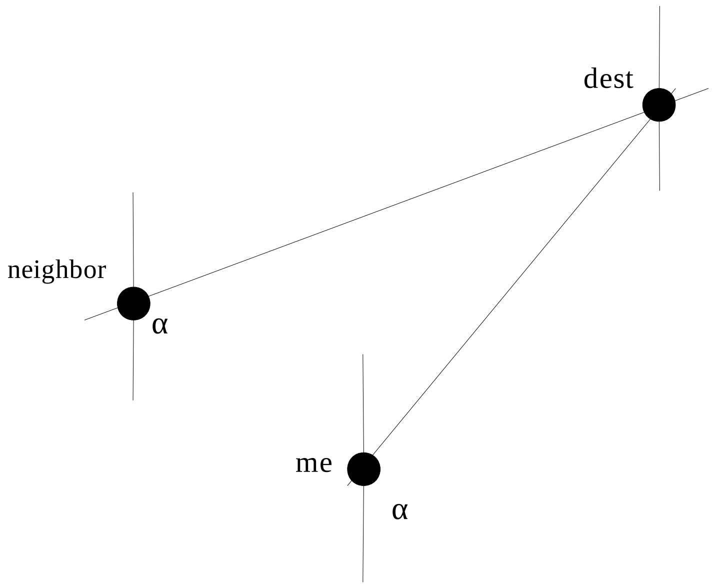


Figure 19: Perimeter routing left-hand rule.

GPSR is implemented with an assumed 90% accuracy value in all cases not explicitly stated. A 95% confidence level is used for localization accuracy, and this is defined to be 10% of a node's transmission radius. Assuming a Gaussian distribution, 100% of sensor nodes are within the measure d as defined in the equation below. This bounding region is then multiplied by the error rate.

$$d=\frac{0.1*trans_{radius}}{0.95}$$

4.8 Simple Application Layer

The application layer implemented in the simulator is very simple. Each time nodes are run they will sense the environment around themselves, via the wrapper methods provided in the Node Object. If an event is detected, then a new packet is created with the event data and sent to the routing layer.

The sink node's application layer has one additional responsibility. When it receives a packet, it will log the data it receives. This is the typical operation of the sink node, because it is the target of all data packets in the network. From here, in a real implementation, the data could then be relayed to a monitor over different channels or saved for retrieval at a later date.

4.9 Logger

The Logger Object provides a naturally intuitive logging mechanism. The Logger Object maintains a list of attributes, and each attribute contains two Strings identifying the attribute being logged and a count variable to give it a value. The first String, referred from here on as the top level String, serves as a categorical identification. The second level String serves as a specific identification. For example, when logging packet counts, the category could be received packets, the specific identification could be setup packets, and the value keeps track of how many have occurred. Even though the count variable is not a floating point number, the Logger Object can save attribute specific information as a String in the second level name. By this method, difficult to log attributes such as node density, network reliability, and duplicated packets seen at the Sink can be easily measured after the simulation is complete.

The Logger Object provides some sophisticated data manipulation methods which will: merge two Logger Objects together, sort the attributes in a Logger Object, find all of the names of specific attributes in a category, and sum the attributes matching a categorical description. These methods allow for easy Logger Object manipulation and information extraction.

Chapter 5

5.1 GDRP

General Direction Routing Protocol (GDRP) is a multi-path routing algorithm which requires the use of some type of localization technology, for example Beamforming Sensor Arrays or GPS. The localization technology provides direction information to the routing layer, and is used to route packets in a certain direction. This direction is typically towards the sink, where towards is defined as in the *general direction* of the target location. This allows for multiple transmission paths to be traversed through the network which enhances robustness of the network. The multiple paths come at a cost and result in a reduction in power efficiency.

The g*eneral direction* transmission technique allows GDRP to compensate for precision errors in the localization hardware. These precisions errors could be caused by nodes not being physically located where they think they are located or the direction of incoming transmissions not being completely accurate. The g*eneral direction* information also serves as a bounding technique for the multi-cast transmissions of nodes using GDRP.

Another advantage of GDRP is that it does not need unique identifiers at any of the sensor nodes in the WSN. This can significantly reduce manufacturing complexity. There only needs to be a distinction between normal nodes and the sink, because the sink is responsible for collecting all data (not routing) and initializing the setup sequence.

5.2 Direction Abstraction

GDRP works by finding the vector (direction) to the best next-hop to route traffic to the sink. This can be done by using the locations of two sensor nodes in the case of GPS. If Beamforming Sensor Arrays are used, then this abstraction is taken directly from the data provided by the hardware. The direction abstraction requires a point of reference from which to start labeling the directions. This is assumed north in GPS solutions, but requires the use of a digital compass when BSAs are used.

This direction abstraction uses a variable number of slices to equally divide up the entire 360º space around the sensor node. The greater the number of slices, the higher the sensor node assumes the accuracy of localization information. The figure below illustrates a 360º space divided up into eight different directions. Each direction has a number associated with it, and the angle between all directions is 45º.

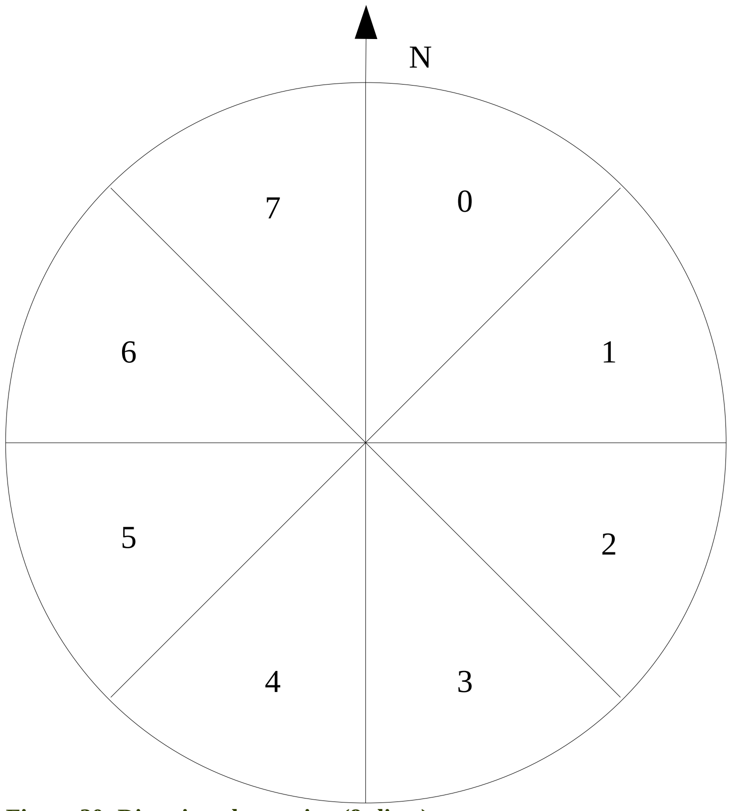


Figure 20: Direction abstraction (8 slices).

The GDRP implementation used for testing assumes the use of Beamforming Sensor Arrays, but GPS could be easily integrated by passing location information around to create direction vectors. The *in general direction* operation is defined as being any direction index adjacent to the direction index provided. In the figure above, directions 2 and 4 would be considered *in general direction* of direction 3.

This abstraction can also be used as a platform for new localization technologies. New technologies would only need to provide some sort of relational information between adjacent nodes to allow for direction information to be extrapolated.

There is no good conversion technique to compare the error percent used by GPS points to the error percent provided by slice abstraction. For the slice abstraction to be 90% accurate, there would need to be 108 slices in the 360º space. GDRP uses 26 slices in regular simulation.

5.3 Setup Phase

GDRP uses a setup phase similar to the one used by MCFA. Initially, all of the nodes besides the sink start unconnected with a path cost of infinity in an unknown direction. The sink starts the setup with a beacon packet that allows all of its neighbors to determine the direction of the sink and correspondingly how many hops away it is (one in this case). Once each node neighboring the sink receives a packet with a better hop count, the node updates it information and sends out it its own setup packet. This cascades until all nodes know the direction and path cost of the sink.

The setup procedure is simple and fairly quick. Similar to a distance vector routing protocol, if a node in the network receives a beacon with a better path hop count back to the sink, then it modifies its state to reflect the new path cost and direction of the neighboring node. If a node never receives a beacon message, then it is disjoint from the network and therefore never tries to transmit. This setup operates with a cost of O(n) in the best case and O(n^2) in the worst case. The worst case requires an exceedingly rare network topology.

This setup procedure has both advantages and disadvantages. As a disadvantage, it needs to be run on a regular basis in mobile networks to accommodate topology changes. As both Bordercast and GPSR also need to do a setup, this is not a major disadvantage.

5.4 Operation

Similar to GDRP's setup procedure, GDRP's routing technique is very simple. Any node which receives a valid packet compares the direction from which the packet was received to the direction it was meant to be transmitted. As shown in the code sample below, if the packet was received in the *general direction* of the intended transmission direction, then it will replace the direction field of the packet with its own known direction to the sink, decrement the time-to-live (TTL), and broadcast the packet.

Forward (transmit):

```
if ( me != sink && --pkt.ttl && inGeneralDir ( pkt.dir , oppDir ( dirOfLastPkt ) )
        pkt.dir = myDirToSink;
        broadcast ( pkt );
else if ( me == sink )
        sendToApplication ( pkt );
```

Figure 21: Pseudo-code of GDRP routing operation.

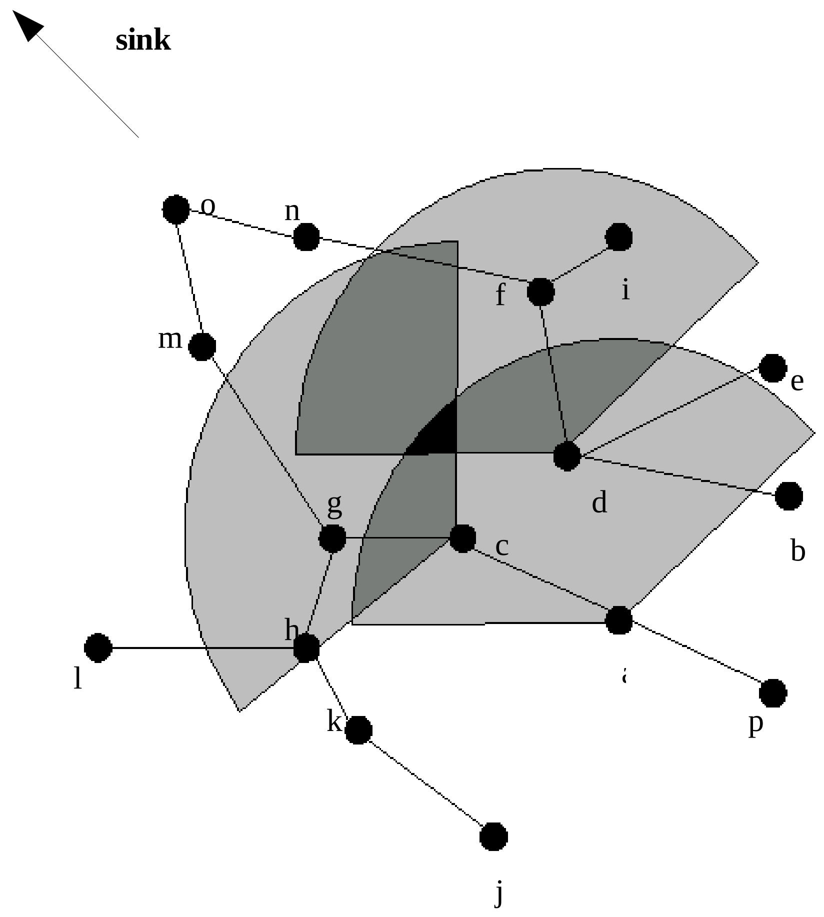
Since all nodes in the *general direction* retransmit the packet, there are in fact multiple paths possible through the network. The typical packet propagation follows the shape of a lens which bounds the number of packets traveling through the network. This is in contrast to Bordercast's packet propagation which is in the form of an expanding ring. This allows for an equally robust network while reducing power consumption. This also combats the main arguments against MCFA's structured networking (the lack of robustness to failures and mobility). The figure below shows the headers for GDRP. It has significantly fewer fields and is therefore smaller than the routing headers for Bordercast and GPSR.

flags	ttl	data(1)	*<rest of packet>*
0 7	8 15	16 23	31

Figure 22: GDRP routing header fields.

GDRP also requires less network state information retention for normal operation. This information is just a hop count for path evaluation and a direction towards next-hop on the way to the sink. GPSR maintains a list of all of its neighbors, and Bordercast maintains a table of its zone.

The figure on the next page demonstrates packet propagation over several nodes in a generic section of the network. The lines between the nodes show the known direction to the sink, in this case the general direction of the sink (not shown) is in the upper left corner. The circles represent the broadcast area and the shaded regions represent valid *general direction* regions per the direction set in the packet. Only the transmissions for nodes *a*, *b*, and *c* are shown to simplify the diagram. Extrapolating from these *general direction* regions, one can see how packets would follow a multi-path route to the sink in the shape of a lens (for a uniformly distributed network).



Broadcast 1:	**Broadcast 2:**	**Broadcast 3:**
src = a	src = c	src = d
recipients = b, c, d, p	recipients = a, d, g, h, k	recipients = a, b, c, e, f, i
in-gen-dir = c, d	in-gen-dir = f, I	in-gen-dir = g, h

Figure 23: GDRP routing procedure.

Chapter 6

6.1 Testbed

In this section the performance of GDRP is compared to the performance of GPSR and Bordercast. All three protocols implement timed caching and a TTL value. Each sensor node in the simulated network uses the same exact set of models and parameters for each comparison test. The networks for each test are created with the same parameters and random seed, so they are identical networks. Each test changes exactly one network parameter to test important characteristics of the routing protocols. Unless specified otherwise, the default simulation parameters outlined in Table 1 are used. Each data point is an average of twenty-five simulations on different random networks with the same simulation parameters. This averaging of simulation data, by virtue of the central limit theorem, increases the accuracy of results.

The two standard independent variables used in simulations are the average node degree and the number of nodes in the network. These significantly change a routing protocol's performance in terms of latency, congestion, reliability, duplicity, and energy cost.

Energy cost is measured in the following graphs for both routing operations and setup operations. It is measured in terms of the number of transmitted bytes. This measurement is the combination of both successfully transmitted packets and packets that collided in the network. This measurement also accounts for protocol optimizations to reduce network header size.

Reliability is the percentage of the number of events the sink should have known about compared to the number it actually did know about. Duplicity is the number of extra packets the sink saw per event that it witnessed. This can be above 100% when multiple paths are taken through the network and serves as a method of measuring redundancy in the network.

6.2 Number of Nodes

This section compares the effects of increasing the number of nodes in a network to routing protocol performance metrics. These metrics are: the number of transmitted bytes for routing operations, the number of transmitted bytes for setup operations, the setup latency, collisions, duplicity, and reliability respectively. The number of nodes in the networks tested range from 10 to 300.

The graph below is on the logarithmic scale to show GDRP and Bordercast on the same graph. GPSR is shown separately on the next page, because it would overlap with GDRP. The graph below shows a massive energy savings by GDRP when compared to Bordercast. As the number of nodes increases in the network, Bordercast requires an exponentially increasing amount of energy to perform routing functions.

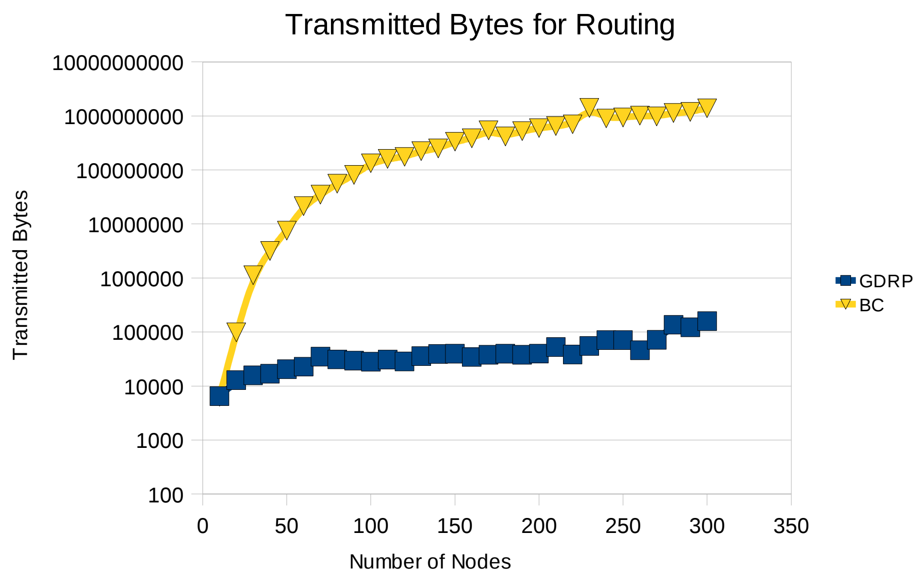


Figure 24: Transmitted Bytes for Routing vs. Number of Nodes for Bordercast and GDRP.

The graph below illustrates the differences in required energy cost for routing operation between GDRP and GPSR on a more reasonable scale. GDRP and GPSR appear to operate with a similar cost, with GDRP operating slightly better. Both curves appear to be exponentially increasing when the number of nodes is above 260.

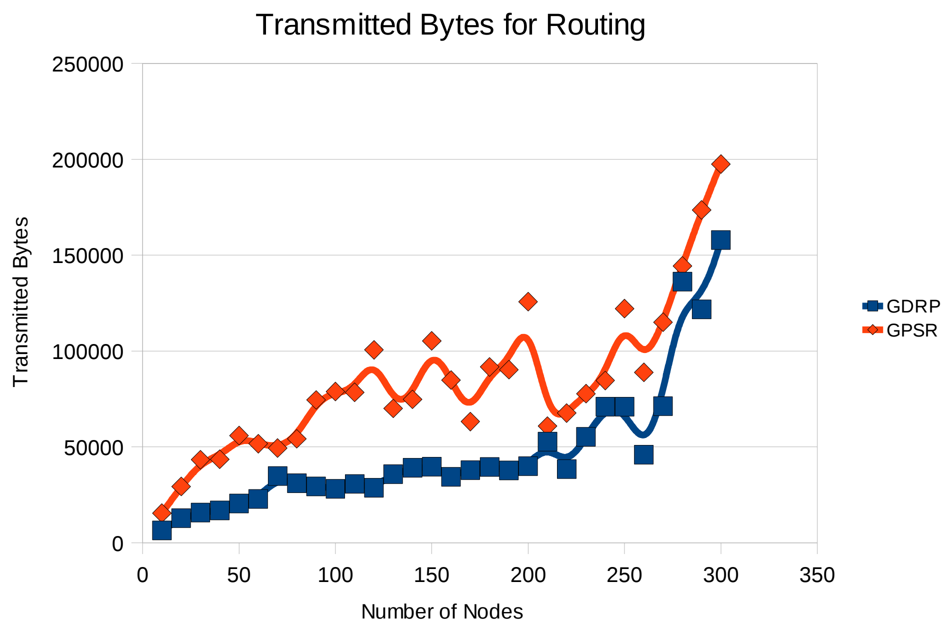


Figure 25: Transmitted Bytes for Routing vs. Number of Nodes for GDRP and GPSR.

The next two graphs show the energy costs to perform the initial setup procedure and the setup latency. These are the important metrics if the routing protocols are implemented in a mobile environment. In the first graph, GDRP operates better than GPSR which operates significantly better than Bordercast. All three trends are linear.

In the second graph, GDRP operates significantly better than GPSR which operates exponentially better than Bordercast. Again, all three trends are linear.

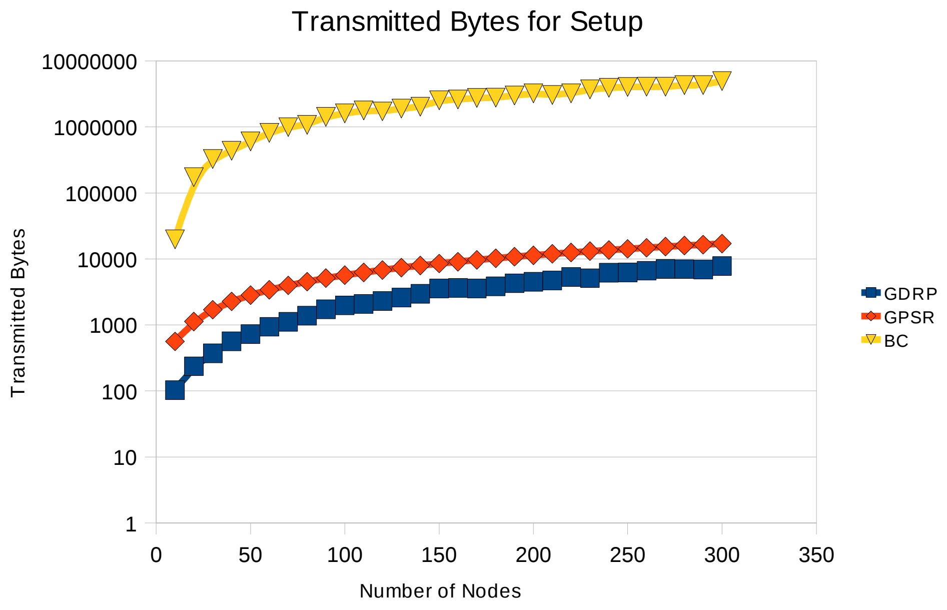


Figure 26: Transmitted Bytes for Setup vs. Number of Nodes.

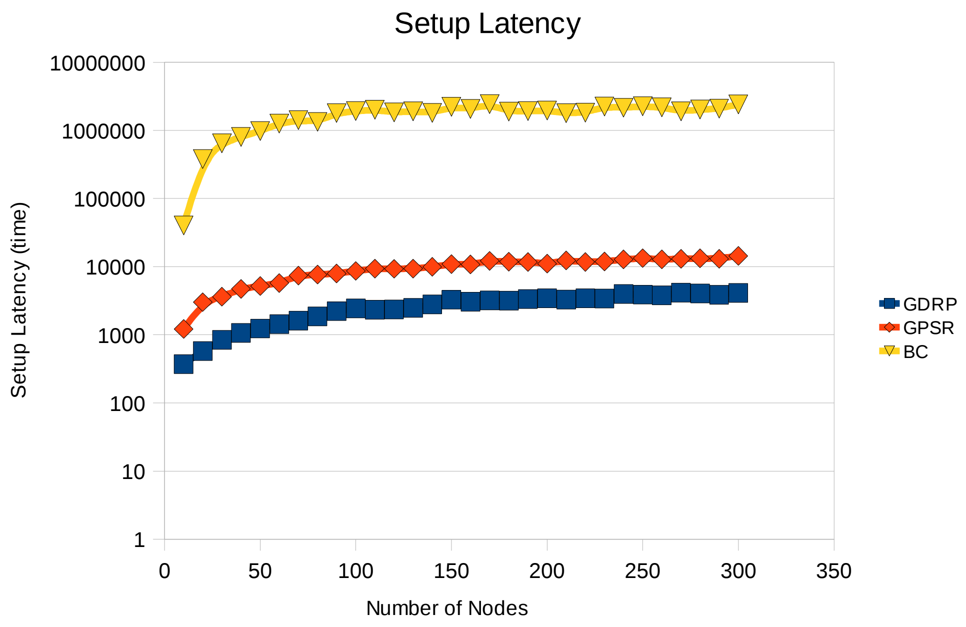


Figure 27: Setup Latency vs. Number of Nodes.

In the previous graphs, Bordercast was shown to be much worse than GDRP and GPSR. The graph below illustrates there are many more packet collisions in Bordercast than in GDRP and GPSR and this metric causes the evaluation of energy cost for routing and setup operations to go up. The network congestion caused by Bordercast could be due to the fact that both its setup and routing operations are very bursty or can be attributed to its routing operation being broadcast in nature, causing every packet to be seen by every node in the network. When more packets see each transmission there is a higher chance of collisions in the network.

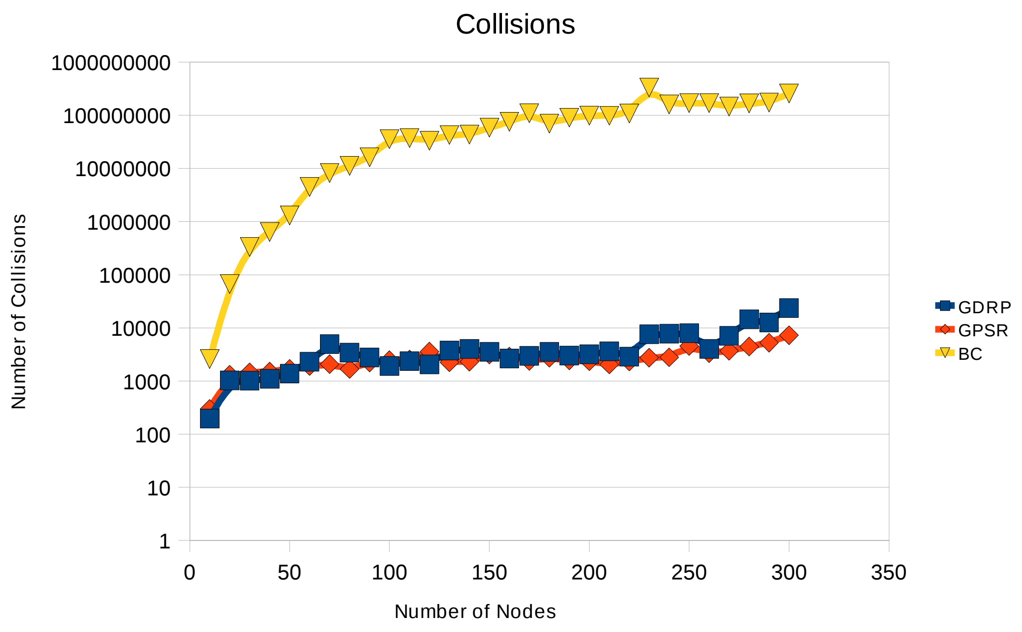


Figure 28: Collisions vs. Number of Nodes.

The last two graphs for this section compare Duplicity and Reliability. In the first graph, Bordercast is shown to operate at an extremely high duplicity. This means that for each event in the network, the sink receives many duplicate packets. GDRP and GPSR are both consistently less than 100%. The second graph shows that while Bordercast has operated at a higher cost in all previous comparisons, it operates at a constant, good reliability for networks containing more than 50 nodes. GDRP operates more reliably than GPSR for any number of nodes.

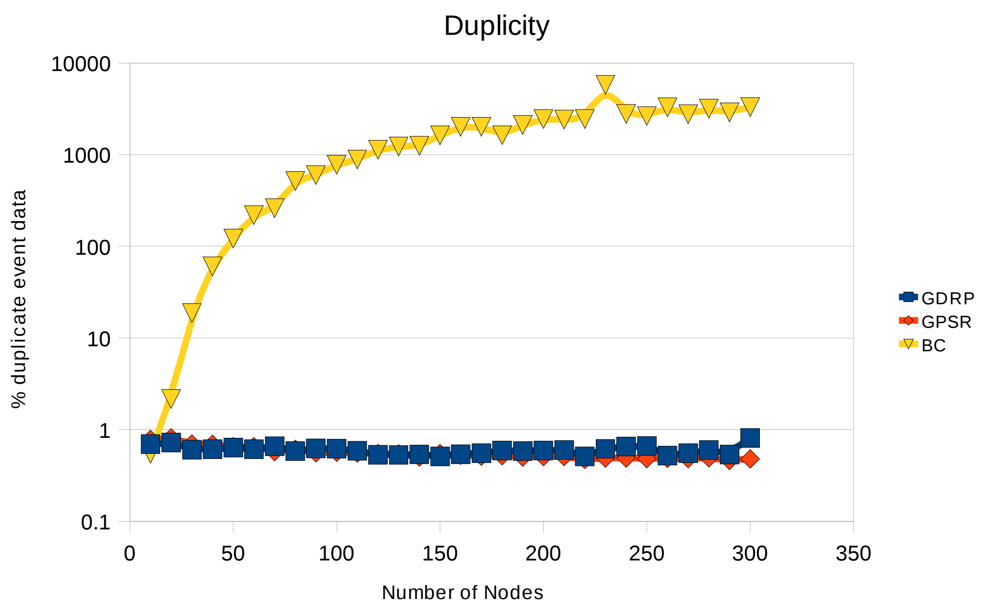


Figure 29: Duplicity vs. Number of Nodes.

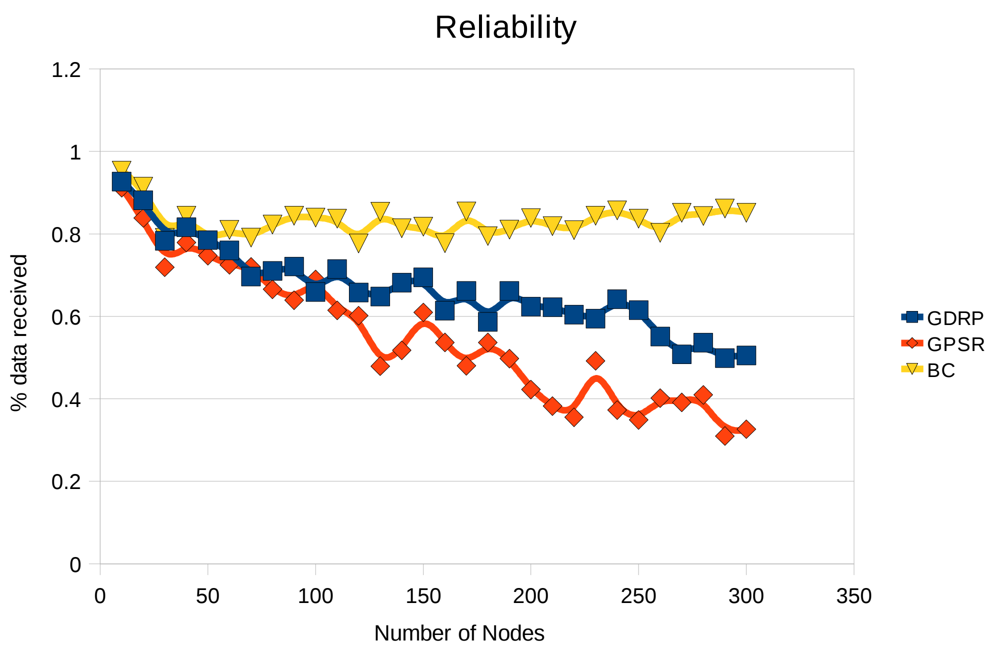


Figure 30: Reliability vs. Number of Nodes.

6.3 Node Degree

The node degree is the number of neighbors of the average sensor node. Increasing this value usually increases the congestion in the network, and congestion increases routing cost, increases setup time, and decreases reliability. The node degree is typically controlled by modifying the network density and the transmission radius of sensor nodes. In this section the transmission radius of nodes is increased to change node degree. In research, it is typical to see the node degree increase by a very large amount, but in practice it does not make much sense for nodes to have an average node degree above 10. The node degrees tested in this section range from 5 to about 60.

The figure below shows a comparison of the three routing protocols for the energy cost to perform routing operations for an increasing node degree. This graph has an abnormal trend for Bordercast with a large decrease in routing energy cost for a node degree above 35. Most nodes are only one or two hops from the sink when the network has a node degree of about 35, and packets are not fully dispersed through the network when this is the case. GDRP operates at an increased operating cost as the node degree increases until it operates with the highest routing cost for networks with a very large node degree. GPSR operates with a decreasing routing cost as the node degree increases.

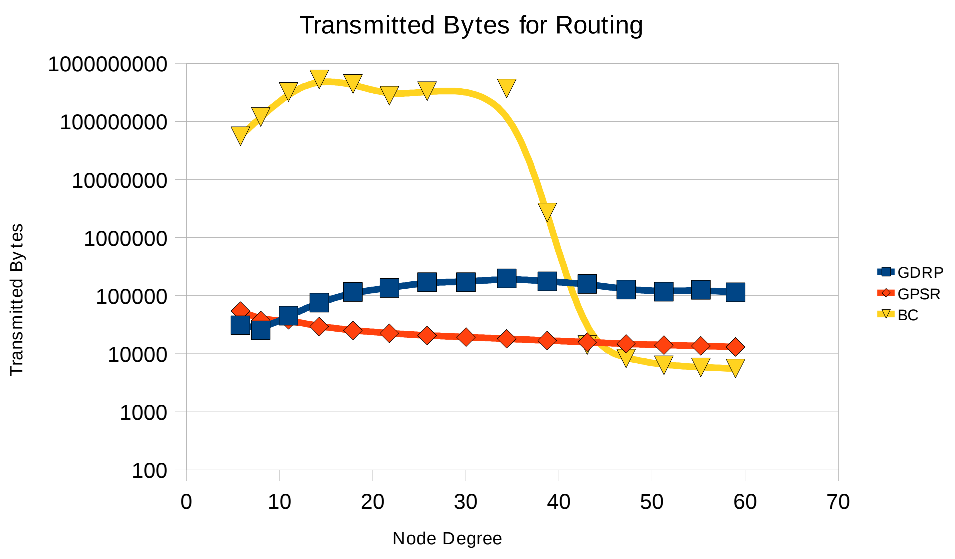


Figure 31: Transmitted Bytes for Routing vs. Node Degree.

The anomaly seen in the graph above can also be explained by looking at the duplicity graph below. When the node degree is above 35, the duplicity of Bordercast drops below 100%. Also seen in this graph, GDRP has an increased duplicity for networks of large node degree. GPSR never has duplicity, measured by its consistent 100% duplicity.

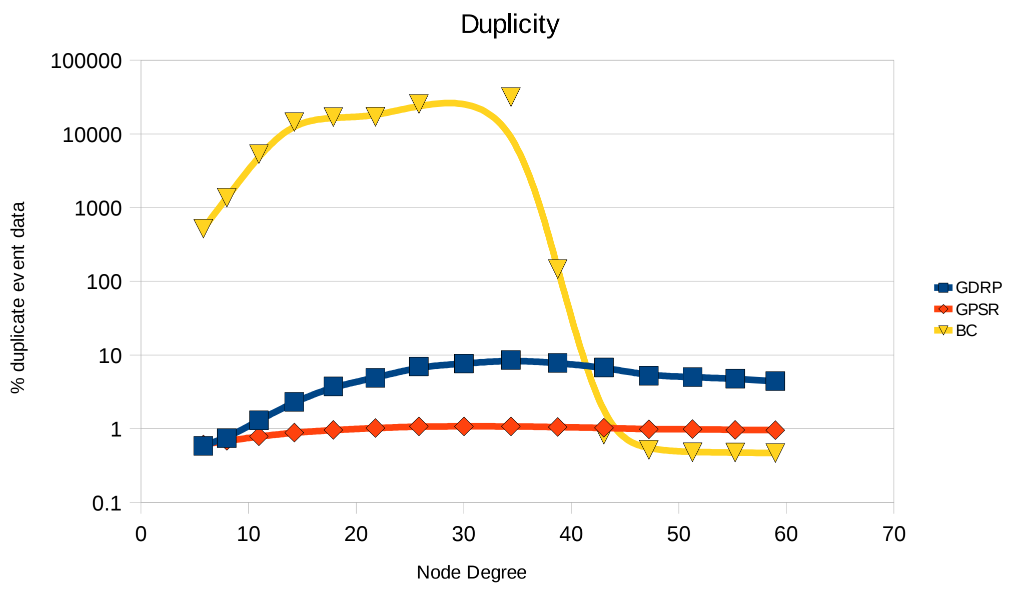


Figure 32: Duplicity vs. Node Degree.

The last figure in this section, below, is another very interesting graph. For a node degree below 10, the standard reliability from the previous section is exhibited. As the node degree increases all of the routing protocols have a decreased reliability. This is attributed to an increase in network congestion. Once the node degree reaches 35, the reliability of each of the routing protocols increases, particularly with Bordercast. GPSR operates at an almost consistent reliability for a changing node degree, which is greatly desired. In general, for networks with a very high node degree, GDRP does not perform as well as the other two routing protocols.

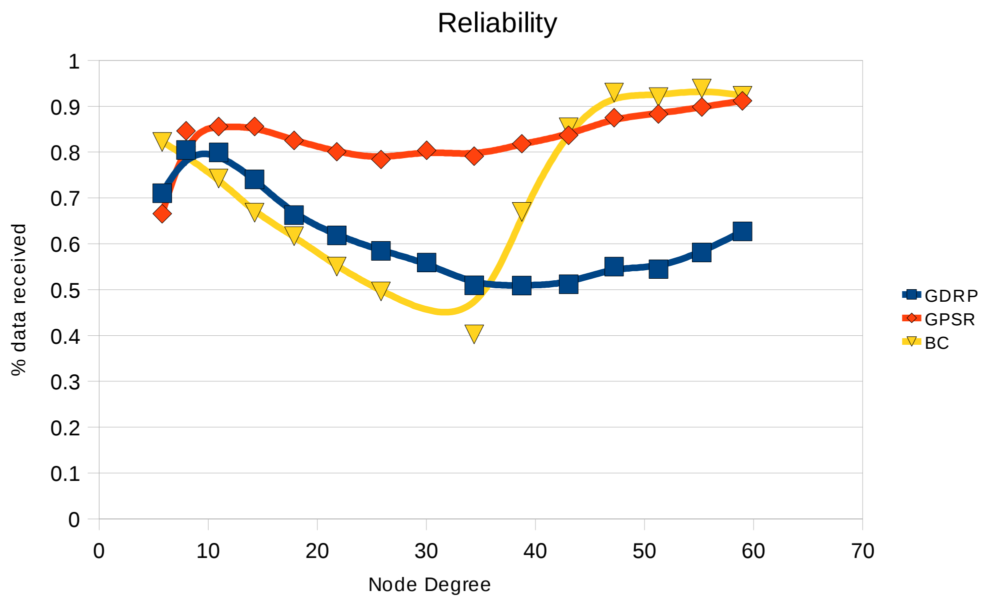


Figure 33: Reliability vs. Node Degree.

6.5 Number of Slices in GDRP

The number of slices in GDRP is how many divisions the 360° space is split into. It is also the way to measure the precision of Beamforming Sensor Array data. As the number of slices increases, GDRP operates with more precise localization data, and can make better routing decisions. When this happens, packets tend to take fewer paths through the network (reducing robustness), but the protocol starts to operate with higher efficiency.

The figure below shows how the reliability of GDRP continually improves for an increasing number of slices. Eventually the amount of increased reliability gained by increasing the number of slices is very small. The reliability increase due to an increase in the number of slices is probably due to a decrease in network congestion from less paths being taken through the network. In default simulation the number of slices is 26, because it approximates the minimum number of slices to gain the maximum amount of reliability.

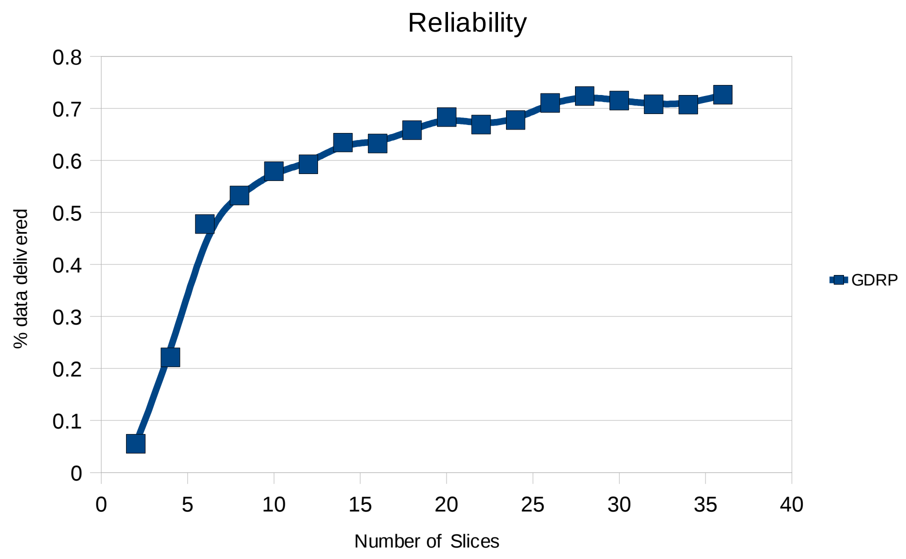


Figure 34: Reliability vs. Number of Slices.

The figure below shows that the energy requirement to perform routing in GDRP is quite asymptotic. When the number of slices is very small, the protocol operates similar to flooding. The value of 26 seems the best number to minimize the accuracy required and minimize the energy cost to perform routing in GDRP.

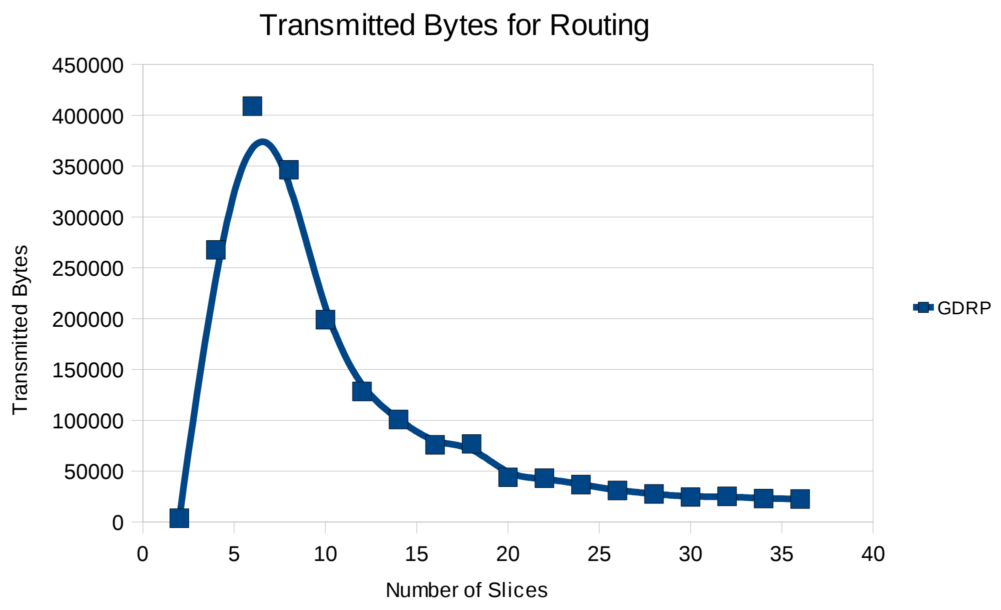


Figure 35: Transmitted Bytes for Routing vs. Number of Slices.

The last figure in this section shows how the duplicity of GDRP decreases with an increase in accuracy, or slice count. This is to be expected, and if an increase in duplicity is desired, then another implementation could increase the number of adjacent slices which are used in the *inGeneralDirection* operation.

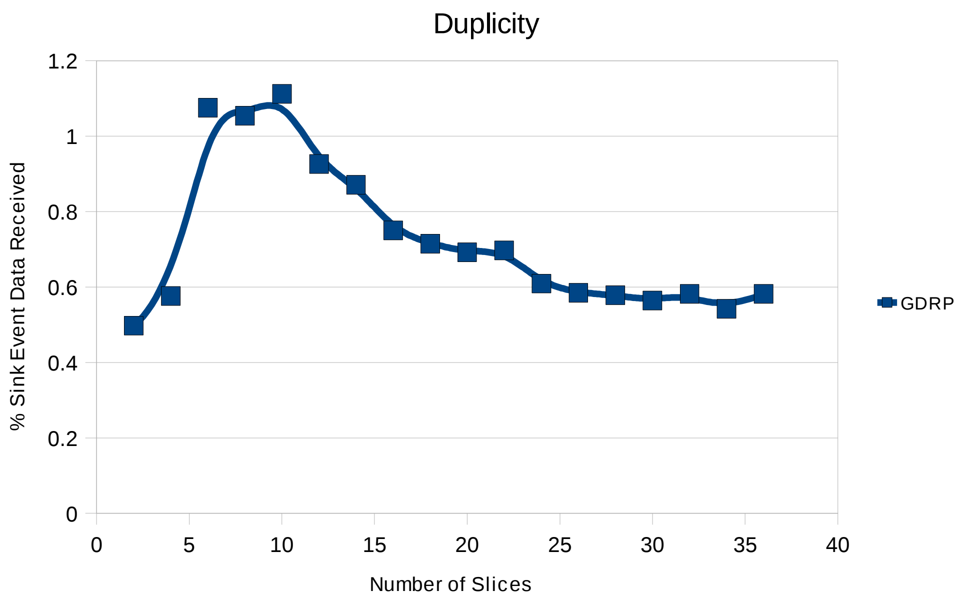


Figure 36: Duplicity vs. Number of Slices.

6.4 Accuracy of GPS

The main contribution provided by GDRP is its ability to provide accurate routing when faced with inaccurate localization information. This section analyzes how GPSR breaks down when its localization information is less accurate. To provide similar comparisons in this section, GDRP was quickly retrofitted to use GPS information to determine direction information. The graphs in this section compare GPSR to a couple of versions of GDRP which uses GPS information. One version of GDRP uses the usual 26 slices and the other uses 52 slices.

The figure below shows how the reliability of GPSR decreases significantly from a decrease in location precision. While both versions of GDRP also decrease, neither decreases as much as GPSR. At the worst accuracy of GPSR, GDRP operates with more than twice its reliability. This graph also shows that increasing the number of slices increases the reliability of GDRP, as seen in the previous section.

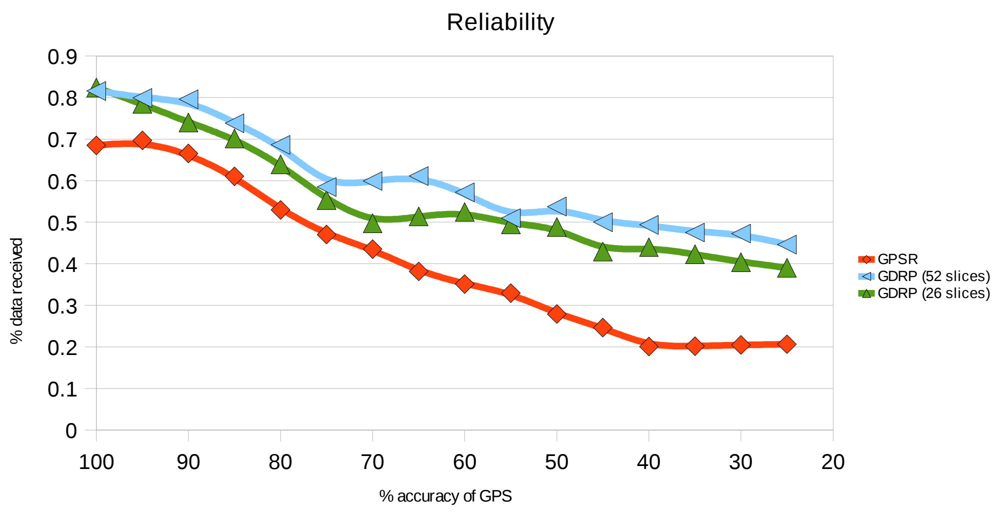


Figure 37: Reliability vs. GPS Accuracy.

The next figure shows that GPSR also starts operating at an increased routing energy cost with a decrease in GPS information accuracy. GDRP with 52 slices is shown to operate at a near constant routing cost. GDRP with 26 slices starts decreasing in operating cost to approach the operating cost of GDRP with 52 slices.

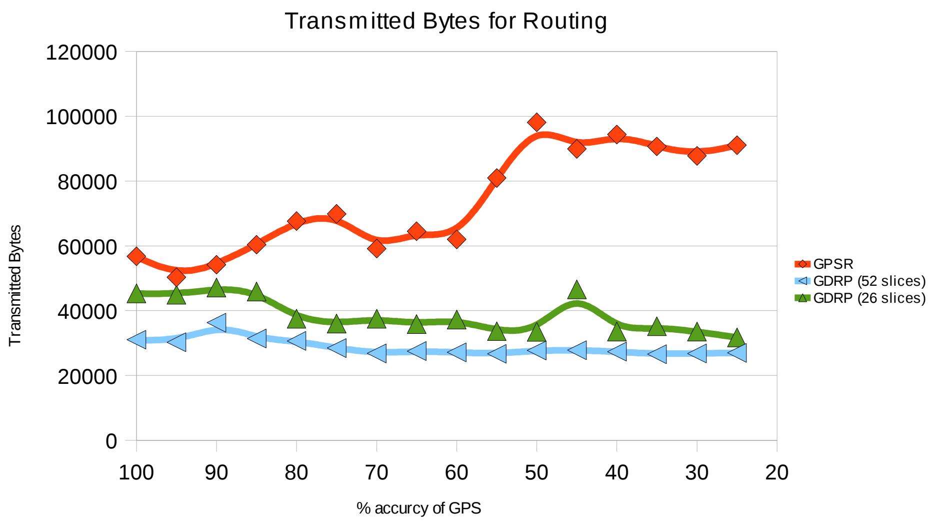


Figure 38: Transmitted Bytes for Routing vs. GPS Accuracy.

The last figure in this section illustrates the duplicity of GDRP and GPSR for a decreasing GPS accuracy. Interestingly, both protocols approach the same amount of duplicity when faced with inaccurate localization information.

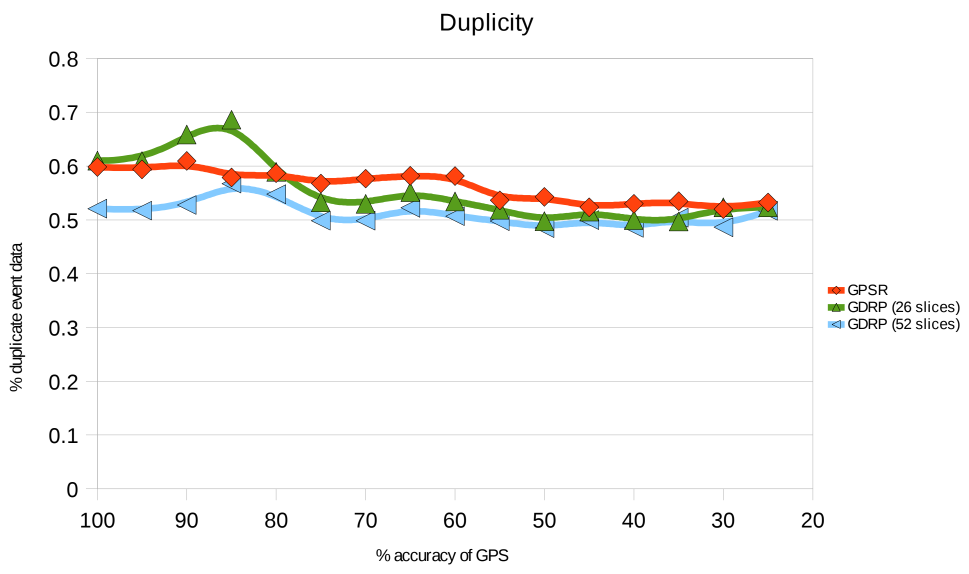


Figure 39: Duplicity vs. GPS Accuracy.

6.6 MAC properties

The MAC Back-off Step size is the time of the initial backoff step plus a random amount of time when a collision happens in the network (section 3.5). As its value increases, it reduces the chances of medium contention while trying to maximize throughput. In simulation, increasing the MAC back-off step size decreased the amount of collisions in the network until it reached the amount of time to transmit an average packet. Although increasing this value increased latency in the network, no other network characteristics were effected.

The MAC buffer size allows the MAC layer to buffer packets during periods of high congestion. During the normal simulation parameter set, GDRP and GPSR did not create enough congestion to show any differences caused by increasing the MAC Buffer size above 100. Bordercast improved slightly, but not enough to even be close in terms of energy cost for routing or setup latency.

Chapter 7

7.1 Conclusions

This thesis introduced some of the important characteristics and types of Wireless Sensor Networks (WSNs). It presented a new routing protocol, the General Direction Routing Protocol (GDRP) which was compared to Greedy Perimeter Stateless Routing (GPSR) and Bordercast. The setup and routing operations of these two other routing protocols were also explained.

The design of a custom, discrete time, WSN simulator written in Java was detailed. This included detailing models for network distribution, communication, failures, and event generation. The complex interaction between various components of the simulator were described and analyzed. The interfaces allowing researchers to quickly test new protocols by dynamically loading their implementations were also explained.

The OSI layers 1, 2, and 4 that were used for testing in the simulator were explained and analyzed. These included a default physical layer implementation which would limit transmissions to a single byte per time-slice, a CSMA/CA MAC layer implementation, and a simple application layer implementation which would transmit location and time information from sensed events.

Through simulation, GDRP performed better than Bordercast and GPSR in terms of energy cost to perform network operations, latency, and reliability for networks of an increasing number of nodes. For networks with an extremely high average node degree, GDRP operated worse in each of the previously mentioned categories, but demonstrated that it takes multiple paths through the network when the node degree is higher improving robustness. It was also shown that an increase in the number of slices used in GDRP increases its reliability while decreasing its energy cost requirements for network operations.

Through a quick retrofit, GDRP was adapted to use GPS localization information to obtain direction information. This allowed for a direct comparison of the effects of a reduced accuracy in GPS localization data between GDRP and GPSR. GDRP operated with a much higher reliability than GPSR for low accuracy localization data. Unlike GPSR, GDRP did not have an increased energy cost to perform routing with increasingly unreliable GPS data.

While having the ability to optimize unreliable localization information to perform routing, GDRP is also able to use different types of localization data obtained form different types of hardware. GDRP operates with a minimum routing header size and performs routing without directly addressing the next hops to be taken through the network. GDRP is well suited to perform routing functions in networks of many nodes with a respectable average node degree, and it is better qualified than GPSR and Bordercast to perform routing in a mobile environment.

7.2 Future Work

Many of the results analyzed in this work were highly dependent on the data-link CSMA/CA implementation, and using a better implementation should improve the performance of the routing protocols and provide a more accurate comparison. There could be important exploration of how a MAC layer change effects the number of slices used in GDRP and resulting reliability. Also, GDRP uses only a simplified setup solution proposed by MCFA (section 3.2). Applying MCFA's more complex setup features to reach a constant setup cost would be desirable.

GDRP could be adapted to use three dimensional directional information for three dimensional network testing against GPSR. GDRP also operates at a higher cost in networks with a large node degree, so providing a scheme to limit the number of nodes in the slice that will route a message would be ideal for limiting the number of duplicate transmissions. Doing research on the effectiveness of GDRP's direction abstraction by using different localization hardware would be interesting.

There are additional testing scenarios that could be executed to test GDRP for a higher level of confidence. Different models could be implemented including multiple mobility approximations, non omni-directional transmissions, different node distributions, and different application traffic approximations. Additional parameters could also be tested including multiple sinks and node power failures.

Adding a visualizer to the simulator for easy protocol validation would greatly reduce development overhead. Also, moving from a discrete time simulator to an event driven time simulator to reduce the simulation time for complex simulations would decrease development overhead.

Bibliography

[1] M. Vieira, C. Coelho, D. da Silva, and J. da Mata, “Survey on wireless sensor network devices,” *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*, 2003, pp. 537-544 vol.1.

[2] T. Arampatzis, J. Lygeros, and S. Manesis, “A Survey of Applications of Wireless Sensors and Wireless Sensor Networks,” *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, 2005, pp. 719-724.

[3] S. Lydon and H. Smith, “General Direction Routing Protocol (GDRP),” *ACTA Press, Sensor Networks '08, from Proceedings*, Sep. 2008, pp. 1-6.

[4] Z. Haas and R. Barr, “Density-independent, scalable search in ad hoc networks,” *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on*, 2005, pp. 1401-1408 Vol. 2.

[5] B. Karp and H.T. Kung, “GPSR: greedy perimeter stateless routing for wireless networks,” *Proceedings of the 6th annual international conference on Mobile computing and networking*, Boston, Massachusetts, United States: ACM, 2000, pp. 243-254.

[6] Bangnan Xu, S. Hischke, and B. Walke, “The role of ad hoc networking in future wireless communications,” *Communication Technology Proceedings, 2003. ICCT 2003. International Conference on*, 2003, pp. 1353-1358 vol.2.

[7] “IEEE-SA GetIEEE 802.11 LAN/MAN Wireless LANS.”

[8] Kaixin Xu, M. Gerla, and Sang Bae, “How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks,” *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, 2002, pp. 72-76 vol.1.

[9] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, “MACAW: a media access protocol for wireless LAN's,” *Proceedings of the conference on Communications architectures, protocols and applications*, London, United Kingdom: ACM, 1994, pp. 212-225.

[10] T. Banka, G. Tandon, and A. Jayasumana, “Zonal rumor routing for wireless sensor networks,” *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, 2005, pp. 562-567 Vol. 2.

[11] G. Xing, C. Lu, R. Pless, and Q. Huang, “Impact of sensing coverage on greedy geographic routing algorithms,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, 2006, pp. 348-360.

[12] M. Zorzi and R. Rao, “Geographic random forwarding (GeRaF) for ad hoc and sensor networks: multihop performance,” *Mobile Computing, IEEE Transactions on*, vol. 2, 2003, pp. 337-348.

[13] L. Zou, M. Lu, and Z. Xiong, “A Distributed Algorithm for the Dead End Problem of Location Based Routing in Sensor Networks,” *Vehicular Technology, IEEE Transactions on*, vol. 54, 2005, pp. 1509-1522.

[14] Fan Ye, A. Chen, Songwu Lu, and Lixia Zhang, “A scalable solution to minimum cost forwarding in large sensor networks,” *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, 2001, pp. 304-309.

[15] R. Bajaj, S. Ranaweera, and D. Agrawal, “GPS: location-tracking technology,” *Computer*, vol. 35, 2002, pp. 92-94.

[16] Jakob Eriksson, Michalis Faloutsos, and Srikanth V. Krishnamurthy, “DART: Dynamic Address RouTing for Scalable Ad Hoc and Mesh Networks,” *Networking, IEEE/ACM Transactions on*, vol. 15, 2007, pp. 119-132.

[17] Yeo-Sun Yoon, L. Kaplan, and J. McClellan, “Pruned multi-angle resolution fast beamforming,” *Sensor Array and Multichannel Signal Processing Workshop Proceedings, 2002*, 2002, pp. 490-494.