Toilers Research Group
Studying Quality of Service in Wireless Sensor Networks

Client: Dr. Qi Han
Team: Teresa Davies and Michael Feineman

Abstract

Wireless sensor networks often measure some condition in an area. In order for the data they collect to be useable, it must be transmitted to a base station. Getting information to the base station effectively can be difficult, so research is being done to develop better methods of collecting data from the network. In order to evaluate these methods, it is necessary to compare their performance to already existing methods. The goal of this project is to measure latency, packet delivery ratio, and energy use for networks of varying size and density in order to create a baseline for comparison with new collection methods.

Requirements Specification

Introduction

Wireless Sensor Networks are becoming more and more important. From remotely recording bird calls to measuring the wear of a factory's machinery, wireless sensors are in demand. This increase in demand has warranted research into the development of better ways to manage these sensors and improve their efficiency. One aspect of wireless sensor networks is how effectively data can be transmitted from the nodes in a network to a base station. Since nodes do not have a very long range, they cannot all transmit directly to the base station. Instead, information is forwarded from one node to the next until it reaches the base station (multihop). Since there is no predetermined path for the data to follow, it does not necessarily get to the base station by the shortest path. A current area of research is trying to improve the way in which data is sent in order to improve the performance of networks. Once algorithms are developed, they need to be tested. This project will provide a way of evaluating the performance of a particular algorithm.

Project Description

The goal of the project is to evaluate the timeliness and reliability of wireless sensor networks with varying parameters. The networks we will be working with will have no intelligent way of transmitting data, so our data will serve as a baseline for future tests of more intelligent protocols. The independent variables are network size and network density. Network size is the number of nodes. Network density is related to how many other nodes each node can communicate with. We intend to measure how latency, packet delivery ratio, and energy consumption vary with each of the variables. Latency is how long it takes for data to get from a node to the base station, and is a measure of the timeliness of the network. Packet delivery ratio is a measure of what percentage of the packets sent are received by the base station, and indicates the network's reliability. Since transmitting takes more energy than computation, energy consumption can be estimated by how many messages are sent.

Requirements

Functional Requirements

Our main tasks are dictated by the following functional requirements.

1. Have a collection simulation in which to gather data

2. Prepare graphs of network size, network density, and link quality vs. latency, packet delivery ratio, and energy consumption.

Our first requirement is the most important. Before we can have any data to work with, we first must have a working simulation. Our client has specifically requested that our simulation use the collection protocol where nodes attempt to pass their information to a base station. Our simulation may be used by researchers to test optimizations of the collection algorithm. However, we are assuming that any other users of our simulation will be familiar with TinyOS and TOSSIM, so that there is no need for an elaborate user interface. In order to make the simulation usable for other people, we include a description of the process of running a simulation and the resulting output.

The second requirement is what the client expects us to deliver at the end of the 6th week. Based on the data we collect from our simulation, we must create graphs that display the results. These graphs will be used for comparison with other algorithms. In order to generate the graphs, we have to run the simulation many times with different parameters.

Non-Functional Requirements

1. Use TinyOS (version 2.x) and its simulator TOSSIM
2. Learn how to use NesC and Python, which are the languages TinyOS and TOSSIM use
3. Use Java and C++ to generate input with pre-existing programs
4. Create documentation describing how to run a simulation
5. Organize directory structure to facilitate running simulation on other computers

Our client specifically requested that we use TinyOS in order to complete this project. TinyOS and its simulator, TOSSIM, are open-source software used for programming and simulating wireless nodes. Our client asked us to use the most up-to-date version of TinyOS and TOSSIM, currently 2.x. Before we could do any simulation, we had to learn how to use TinyOS and TOSSIM.

Scope

By the end of the sixth week we delivered the graphs that our client requested. We tried to make the simulation accessible to other users. The goal of making the simulation available for other users is for them to test their own collection protocols.

System Design

Design Goal

Our goal is to create a collection simulation, and use it to measure how latency and packet delivery ratio are affected by network density and size. Collection is when all the nodes in a network send their data to a single base station (called the root). A necessary characteristic of collection is a routing protocol. Since nodes have a small transmission range, data can not go directly from a node to the base station, but instead must be forwarded through other nodes. The routing protocol defines how data travels from a node to the base station when there is no direct link between the two.

Tasks to be completed

1. Find a computer that will allow us to do our work on TinyOS
2. Configure TinyOS to allow us to compile applications
3. Find or create a way to generate networks of different density and size
4. Create a TinyOS program to simulate collection
5. Add a way to measure latency and packet delivery ratio
6. Create a method for easily generating data for different densities and sizes
7. Create a method to format raw data so it can be easily graphed
8. Plot formatted data on a graph
9. Make simulation user friendly (non-essential)
10. Package application so that it is easily installed on another computer (non-essential)

Risk Items

We identified the following risk items:

- Creating a program that simulates multi-hop collection
- Ensuring that packets are dropped correctly (or at all)
- Ensuring that we measure latency and packet delivery ratio (PDR) correctly

The biggest piece of our project was creating a simulation that implements a collection protocol. If collection is improperly implemented, then our simulation is not accurate or useful. As part of correctly simulating collection, we have to ensure that the simulation does, in fact, drop packets. Since packets are lost in any real network, it is important that our simulation correctly handles packet loss. If no packets were lost in the simulation, it would not generate meaningful data.

Measuring latency presents some difficulties. Each node has its own internal timer which is not synchronized with the other nodes. Nodes report when they send or receive data using their timer. In order to calculate how long a packet took to reach the root, it is necessary to compensate for the differences in the timers. A possible solution is to implement a clock synchronization protocol. When the clocks are synchronized, calculating the latency becomes easy. In the end, we did not use a clock synchronization protocol and took another approach to solve the clock synchronization problem.

High-Level Design

There are steps that must be taken in order to run the simulation. Our collection application must be compiled with TOSSIM which is written in Python. The compiled application is run with a Python script which uses TOSSIM's methods for creating a network and stepping through the simulation. TOSSIM needs to know the gain between each pair of nodes and the noise at each node. From this information, it creates a radio model for the network. A radio model defines how well nodes can communicate by determining the link quality between each node. To simplify this process, there is a pre-existing Java application that will create a gain and noise model given certain parameters. Once TOSSIM has generated the radio model, the simulation can begin. Each simulated mote has been programmed with the collection behavior. Just like in a real network,

each node follows its individual programming which includes ways of interacting with other nodes.

Figure 1 shows the steps involved in running a simulation. If someone wished to test a different collection method, the only piece they would have to change would be the TinyOS collection application. The TinyOS application is compiled into TOSSIM. The Python script loads TOSSIM and runs the simulation. Our shell script automates the creation of topologies and running of the simulation. Once the simulations have finished running, our C++ program converts the raw output into useable data. We then plot the data on a graph using a program like Excel. Using this process with varying parameters, and running each simulation multiple times, we will create a set of data to present to our client.

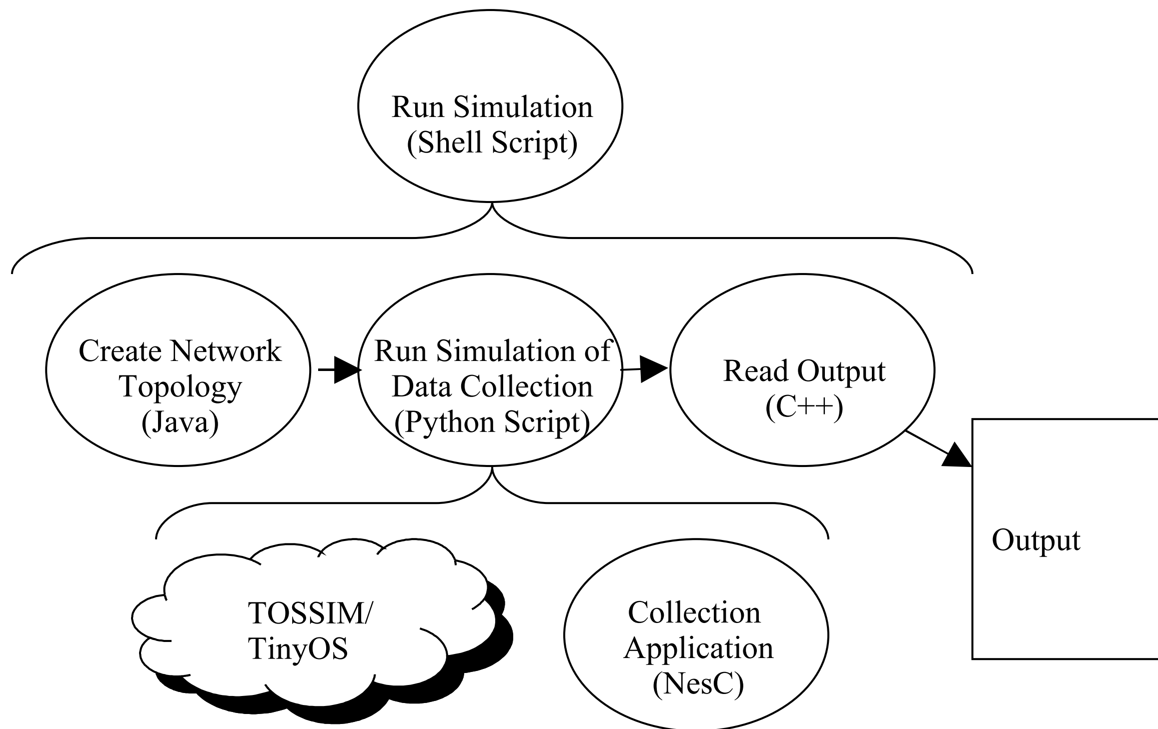

Figure 1 - Organization of application

Design Details

Collection and Forwarding

A collection protocol is not sufficient without giving the motes something to collect. However, we cannot make simulated motes sense simulated data. To solve this problem, we have them send a predefined message periodically.

In order to get data from the nodes to the base station, the network needs a forwarding protocol. Since this project is intended to provide a baseline for other collection methods, it uses a very simple way of routing. The routing protocol builds a routing tree by having each node arbitrarily choose one of its neighbors to be the node that it sends to (its parent). The tree is not static; a node will send to another of its neighbors if its parent is currently busy.

Topology
An important aspect of our simulation is correctly varying the parameters. To get meaningful data, we must only vary one parameter at a time. The density must stay the same when we change the number of nodes, and we must use the same number of nodes when we vary the density. The topology is generated by a Java program that takes parameters that determine the network size and density. Once the topology is generated, it can be input to TOSSIM as a network. The topology file has gain between nodes. A separate file has a noise trace taken from Meyer Library at Stanford University. From this information, TOSSIM generates a radio model for the network, which has noise that varies with time and determines whether packets can be sent or not.

TinyOS Application
Each mote needs to be programmed to know what to do. This is achieved by installing a TinyOS application on the motes. The application is written in NesC and dictates how the node behaves. In a real network, each node would have the same application installed on it. TOSSIM simulates this by having virtual nodes which behave like their real world counterparts.

Timing
We are trying to measure the time it takes for a message to reach its destination, so the time events occur is important. Measuring latency is not conceptually a difficult problem; simply mark the time from when a packet was sent to when it was received. However, each node has its own timer, and the timers are not synchronized. When a node boots, its timer is started from zero. However, each node boots at a different time which introduces the synchronization problem. Our solution is for the simulation to keep track of when each node starts, and use that as an offset to each node's timer. This approach would not work with a real network because we would not know when each node booted. Thus, if it turns out to be necessary to run our application on a real network, we will have to add a clock synchronization protocol.

Python
When a TinyOS application is compiled with TOSSIM, it generates a Python file. This file encodes the simulator behavior which includes the virtual nodes with the TinyOS application on them. In order to set up and run TOSSIM, we use a Python script. The script will load the compiled TOSSIM file, read in the topology (noise and gain), and then run the simulator. Also, the script defines where debug messages go, such as a file or standard output. We use the debug messages to collect data about what is happening in the simulation.

Output Interpretation
In order to get the correct information from the simulation, we have to process its raw output. The simulation prints a message when a message is sent or received. It tells the time the event occurred according to the timer of the node that sent or received the message, as well as a message identification number. With the identification number, we can track the messages, and determine how long each one took to send. In order to correctly find the latency, it is necessary to correct the timers as described above. By counting how many messages are sent and received, we can also determine the packet delivery ratio.

We convert the raw data into a format that can be read by a program such as Excel.

Summary

We wish to measure the effect of changing network density and size on PDR, latency, and energy consumption. After carefully analyzing the problem, we broke it down into a list of tasks to be completed. From this list, we identified potential risk items. Taking these risks into consideration, we created our design.

Future Work

Our application would be a much more useful tool for comparing the performance of collection protocols if other users could easily run their own simulations with it. Therefore, a possible next step is to make it possible for other users to reuse parts of our application with their own collection protocol.

Results

Our main goal was to produce graphs for our client. The following graphs display some of the data we collected.

References

TinyOS 2.01. Documentation at http://www.tinyos.net/tinyos-2.x/doc/

Figure 2 shows how the PDR varies with the size of the network. The PDR decreases as the network gets bigger because the nodes interfere with each other when they are transmitting.

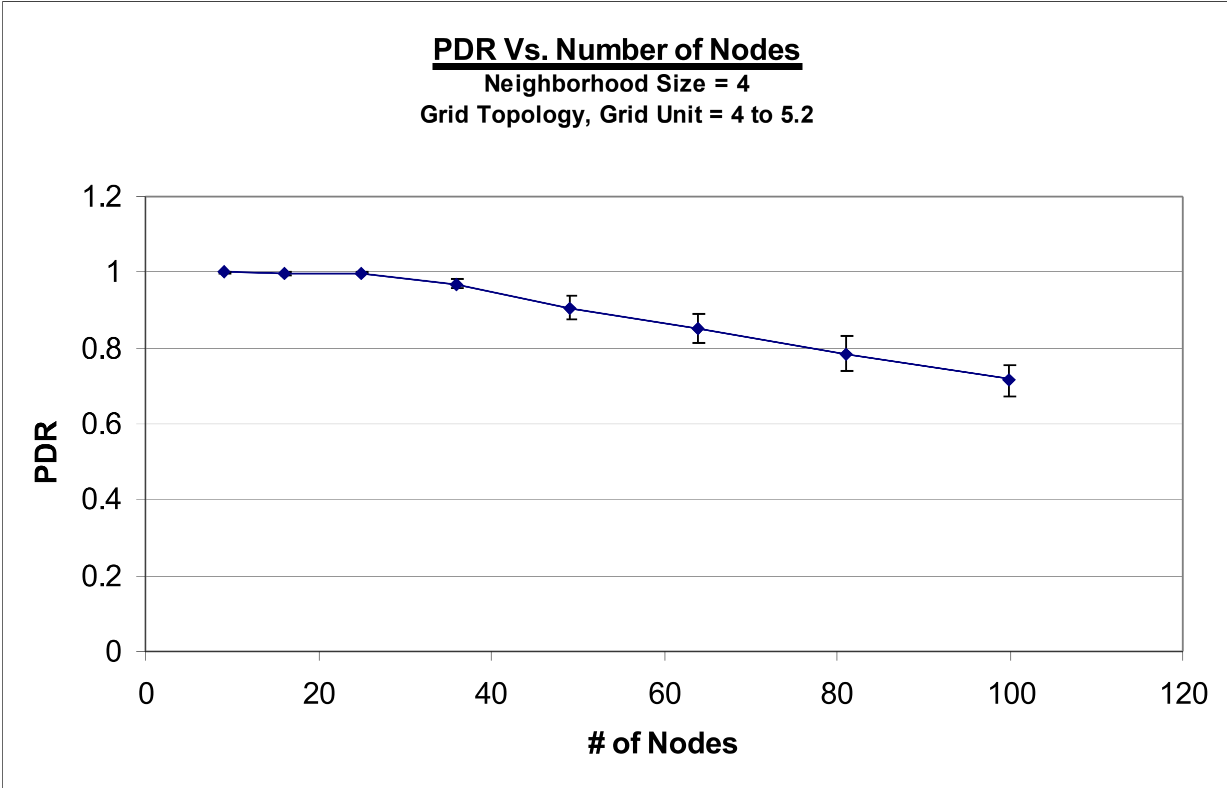

Figure 2

Figure 3 shows how latency varies with network size. Here latency is the maximum amount of time a message took to reach the base station. As the network gets bigger, the time increases because the number of hops to get from one end of the network to the other increases.

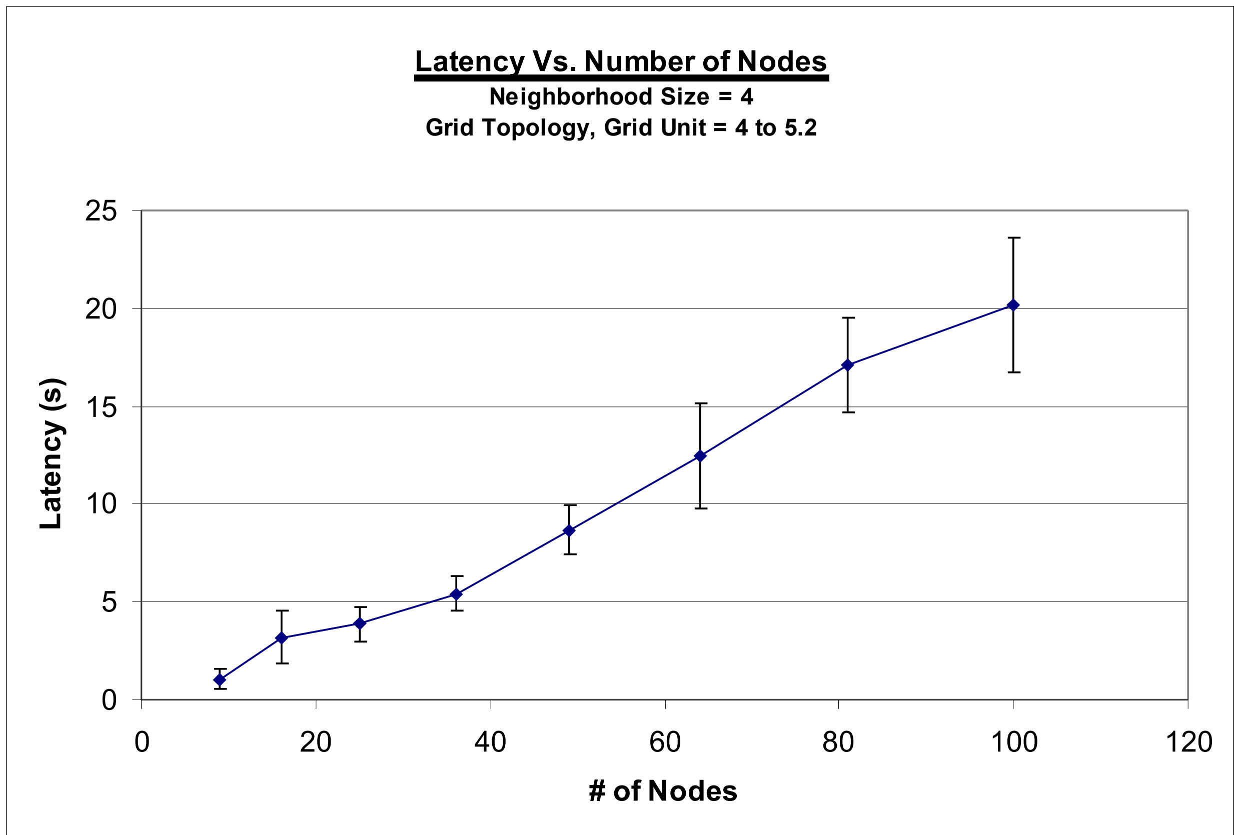

Figure 3

Figure 4 shows transmissions as a function of the network size. The number of messages sent includes forwarded packets, and is an approximation for the energy usage. Since each node sends the same number of messages in each simulation, there are more total packets sent. However, the number of messages sent also increases because more messages need to be forwarded in a larger network.

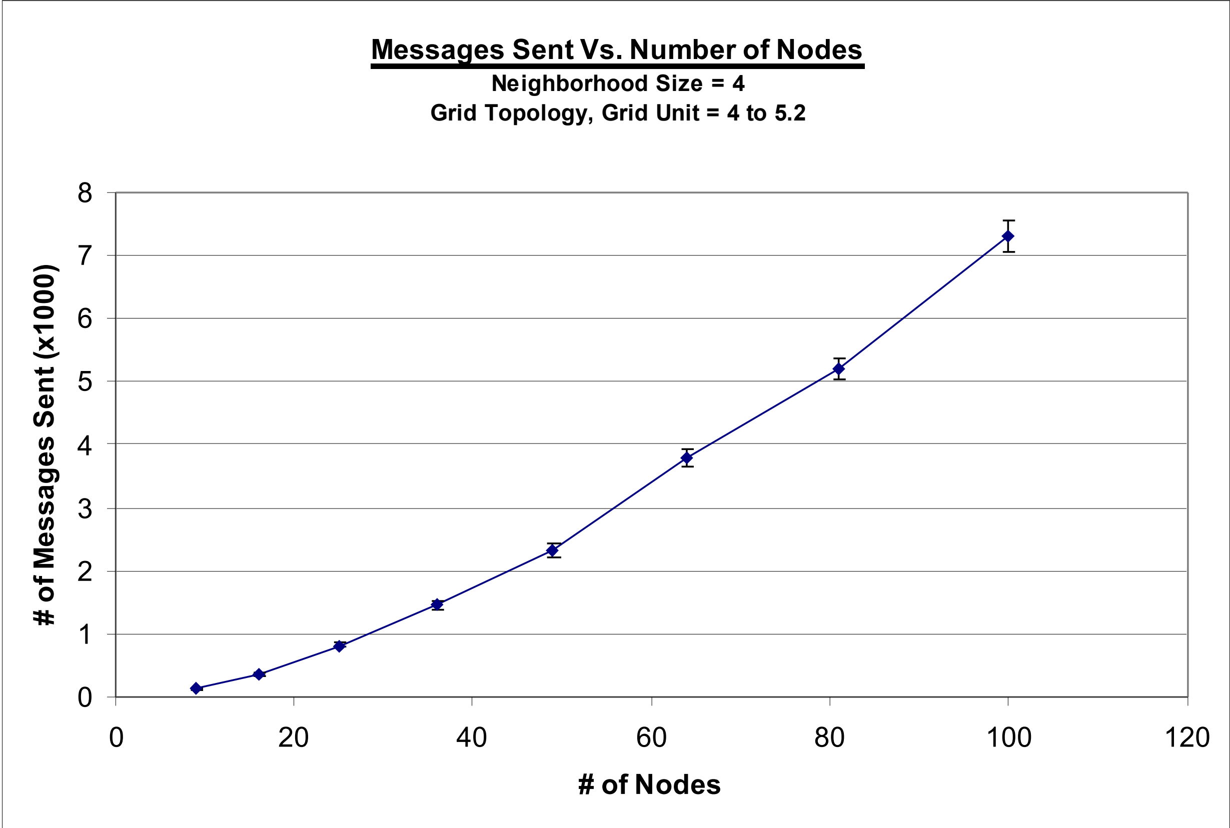

Figure 4

Figure 5 shows how PDR varies with the network density. As the average number of neighbors per node increases, the PDR also increases because a denser network takes fewer hops to get from one end to the other, so there are fewer transmissions required and fewer collisions.

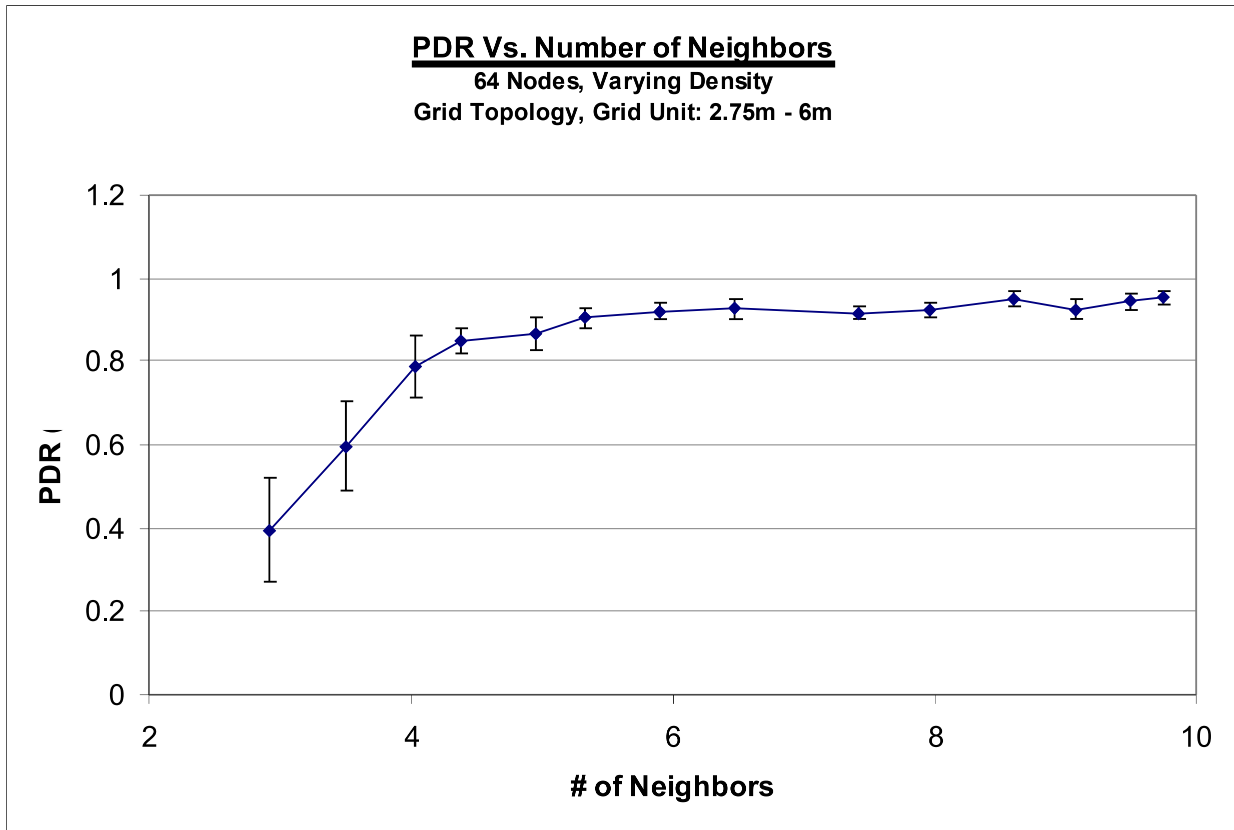

Figure 5

Figure 6 shows how latency varies with density. As the network becomes denser, latency drops. As with PDR, when the graph is denser, there are fewer hops, so it takes less time for messages to be delivered.

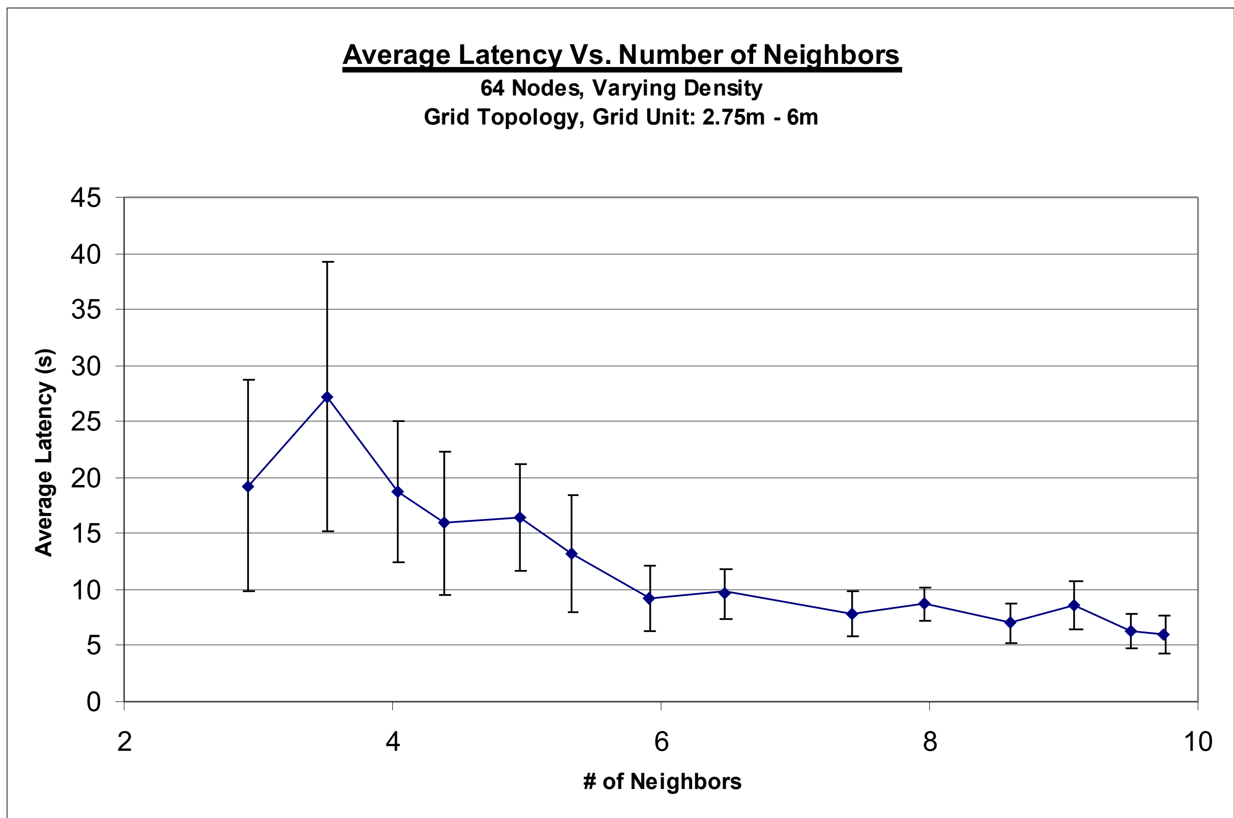

Figure 6

Figure 7 shows how the number of messages sent varies with density. For this graph, the total number of messages sent each time was the same, so any variation comes from a difference in how often messages are forwarded. There is only a small decrease as the density increases.

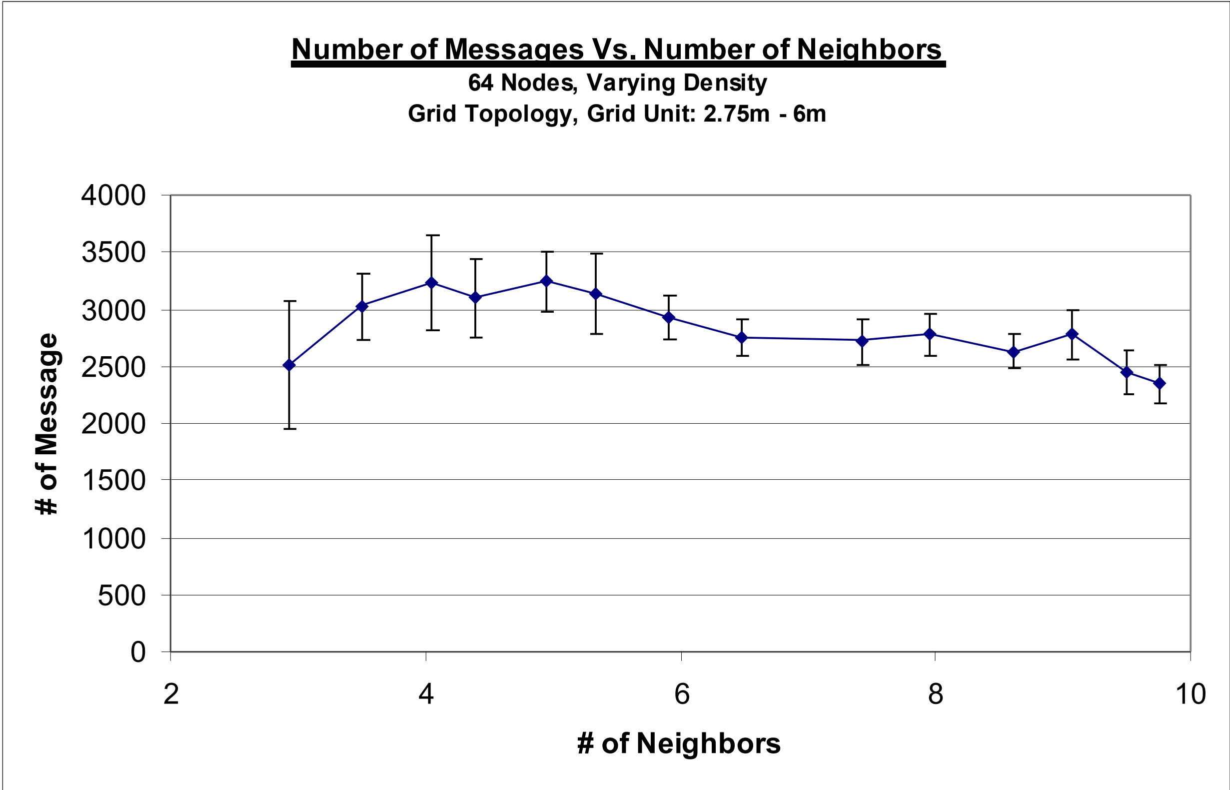

Figure 7

Figure 8 shows how the PDR increases over one period. At the beginning of the period, all of the nodes in the network send messages. Some of the messages reach the base station quickly, and others take longer, usually because they have to be forwarded.

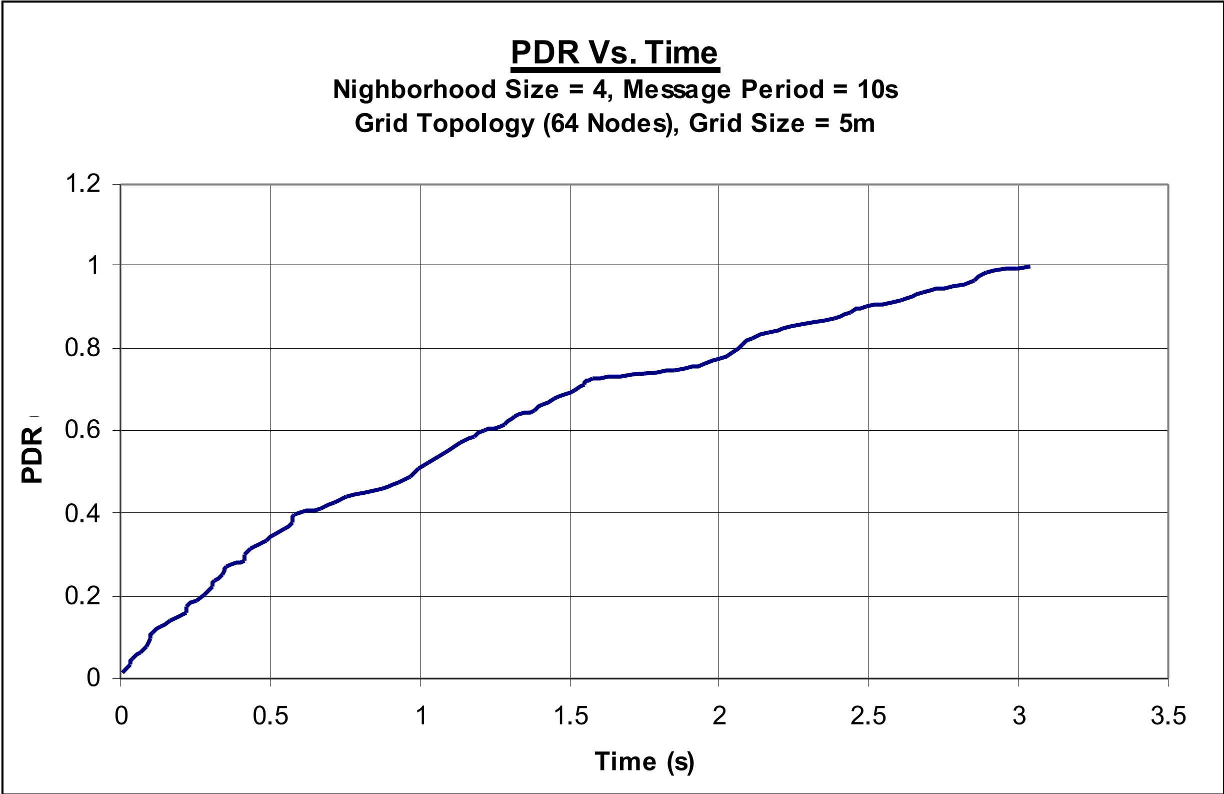

Figure 8

Figure 9 shows how the maximum latency changes as the simulation runs. The general trend is for the latency to decrease slightly after the first few cycles. The latency for the first few rounds of messages sent is probably higher because the forwarding tree is incomplete. However, other factors such as varying noise around the nodes can cause almost as much variation later in the simulation.

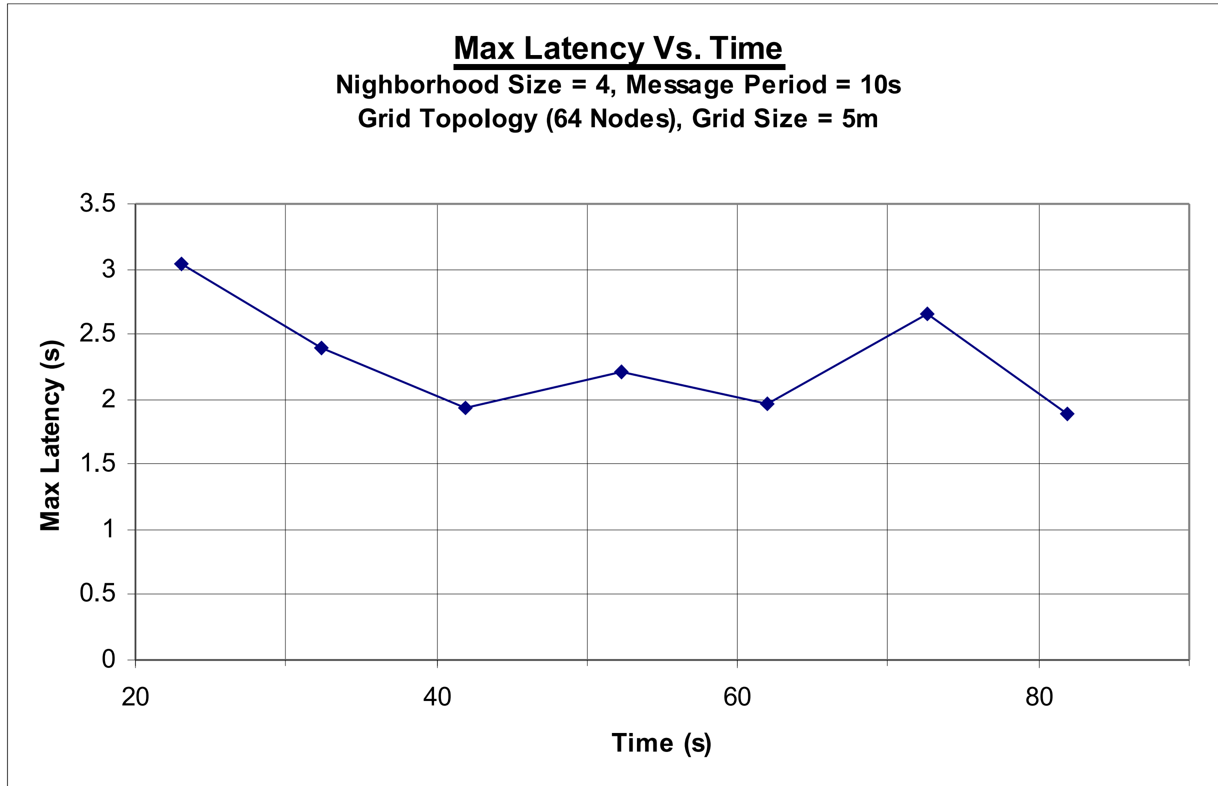

Figure 9

Glossary

Base station – where all of the data from a network is collected

Collection – the nodes in a network send information to a base station

Collision – when a packet is lost because two nodes transmit at the same time and interfere with each other

Dropped packet – a packet that is lost during transmission for some reason

Forwarding – when a node cannot transmit its message directly to the base station, it sends it to a neighbor, which retransmits it

Latency – the time it takes for a message to travel from its source node to the root

Message period – how often the nodes send out messages; all nodes send messages at approximately the same time

Multihop network – a network that is too large for the nodes on one end to transmit to the nodes on the other end, requiring forwarding to get data to a base station

Neighbor – a node that is within transmission range of another node

Network density – a measure of the number of neighbors each node has

Network size – the number of nodes in a network

Packet – a unit of information transmitted by a node; a node sends one packet at a time

Packet delivery ratio (PDR) – the ratio of the number of packets received to the number of packets sent, where the number of packets sent is the number of sending nodes times the number of message periods

Radio model – a model for the noise around the nodes, which affects how well they can connect to each other

Root – the node in a simulated network that acts as the base station

Topology – the relative positions of the nodes and the gain of each link