Routing Protocols for Wireless Sensor Networks

Basil Etefia
Electrical Engineering
Loyola Marymount University
buetefia@lion.lmu.edu

Faculty Mentor: **S. Shankar Sastry**
Research Supervisor: **Jonathan Sprinkle**
Graduate Mentor:**Tanya Roosta**

Summer Undergraduate Program in Engineering at Berkeley (SUPERB) 2004

Department of Electical Engineering & Computer Sciences
College of Engineering
University of California, Berkeley

Routing Protocols for Wireless Sensor Networks

Basil Etefia

Abstract

This paper presents an improvement on the implementation of information routing capabilities in ad hoc wireless sensor networks. Improving the protocols used by each sensor node can increase the network's localization and power conservation abilities. Furthermore, using a more efficient algorithm will improve the overall effectiveness of the entire network in terms of power efficiency. Using novel and creative schemes to generate shortest paths for information routing from source to destination nodes, we have implemented an approach to limit the inefficiencies of routing protocols used by sensor networks for information transfer.

I. INTRODUCTION

Ad Hoc Wireless Sensor Networks have the capacity to revolutionize the contemporary technical arena. Offering a more convenient means of communication, this idea of infrastructure less networks can transform many applications, including military strategy, home security, information transfer, environment monitoring, and surveillance. This concept can initiate wave of wireless interaction that the world has not yet seen.

A. Ad Hoc Wireless Sensor Networks

An *Ad Hoc* Wireless Sensor Network is a system of wireless nodes that dynamically self-organize in random network topologies. This allows internetworking in areas where communication infrastructures were once nonexistent [1]. The nodes of these networks can collaborate with their neighbors to perform functions such as routing, localization, time-synchronization, and data aggregation. However, these networks have many complex problems that need to be solved before this innovative idea comes into being, such as power conservation and localization.

B. Multi-hop Wireless Networks

Multi-hop wireless networks are a type of *ad hoc* wireless networks. This term refers to the idea of the information being transferred having to travel through more that one location to reach its final destination. Based on the scheme of its network's algorithm, a multi-hop wireless network can be costly in terms of power. Also, leach individual node in a wireless network usually requires some complex scheme to manage its localization. Localization is the idea that each node can approximate its location by using algorithms to find the distance of itself from its neighbors. These issues of power conservation and localization complicate the coding schemes of many algorithms designed for the Berkeley motes.

C. Berkeley Motes

In recent years, University of California - Berkeley has emerged as a leader in the technological advancements of *Ad Hoc* Wireless Sensor Networks. Researchers at UC Berkeley have developed these motes; they are a physical interpretation of the wireless sensor network. An example of a mote is shown in Fig. 1.

In fact, Berkeley has developed a test bed in which many of these motes rest on a nylon net sensing its surrounding environment. Furthermore, the motes have seen much advancement over the past years.

Fig. 2 shows the motes as they have evolved from mica1's to mica2dot's. The size of the mica2dot's has decreased to nearly the size of a quarter. The sensor boards, shown in the Appendix, give the mote its sensing capabilities. These motes also use a program board, which is also shown with a mote in Fig. 3.

The board is connected to the computer using the serial and parallel ports. They not only program the motes, but also give the user the ability to listen to an entire network. The system used by these motes is entirely written in TinyOS, an open-source operating system designed for wireless embedded sensor networks.

This work was supported by the National Science Foundation under the Summer Undergraduate Program for Engineering Research at Berkeley–Information Technology (SUPERB–IT) 2004.

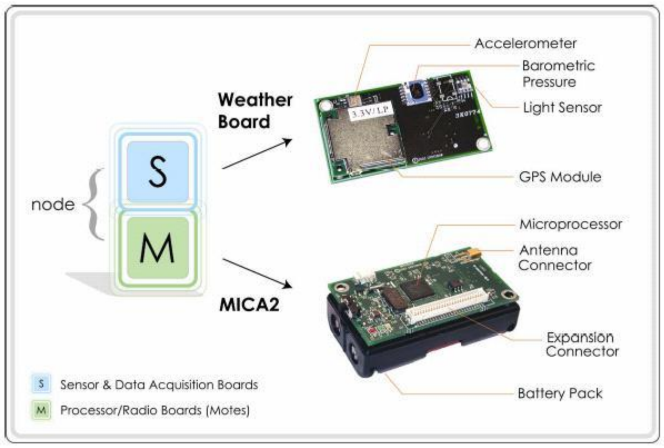


Fig. 1. A sample mote

Fig. 2. This figure shows the evolution of the Berkeley motes.

Fig. 3. This figure shows the a Berkeley mote with a programming board.

D. TinyOS

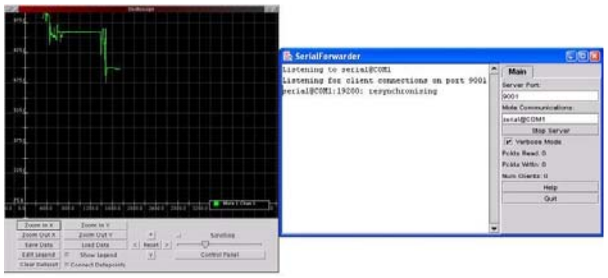
TinyOS is an event-driven operating system intended for sensor networks with limited resources. The TinyOS system, libraries, and applications are written in nesC, a new language for programming embedded systems. nesC is a version of C that was designed to handle the structuring and execution of TinyOS. In nesC, programs are built out of components which are ”wired” together to form whole programs. The behaviors of these components are specified in terms of interfaces. These interfaces are bidirectional; they specify a set of functions to be implemented by their providers and users. The components are linked together by their interfaces. nesC expects that code will be generated by whole-program compilers and is based on run-to-completion tasks and interrupt handlers which may interrupt tasks and each other. To help with the struggles of nesC, TinyOS has a simulator that provides abstractions of real world phenomena, such as bit error [2].

E. TOSSIM

TOSSIM is a discrete event simulator for TinyOS sensor networks. Instead of compiling a TinyOS application for a mote, users can compile it into TOSSIM, which runs on a PC. TOSSIM builds directly from TinyOS code. This allows users to debug, test, and analyze algorithms in a controlled evnvironment. TOSSIM captures the behavior of TinyOS at a very low level. It simulates that network at the bit level, simulates each individual ADC capture, and every interrupt in the system. However, though TOSSIM captures TinyOS behavior at a very low level, code which runs in a simulation might not run a on real mote. Although TOSSIM times interrupts accurately, it does not model execution time. TOSSIM also does not model radio propagation nor does it model power draw or energy consumption. Instead, it provides radio abstraction of independent bit errors between nodes and can be adjusted to add annotations to components that consume power to provide information on when their power state changes, e.g. turned on or off [3]. TOSSIM also has many applications that can be used to accurately simulate these motes.

F. TinyViz

TinyViz allows TOSSIM simulations to have an extensible graphical user interface for debugging, visualizing, and interaction. TinyViz can easily trace the execution of TinyOS apps, set breakpoints, visualize radio messages, and manipulate the virtual position and radio connectivity of motes [4]. Also, TinyViz allows you to write your own modules to visualize data or interact with the running simulation. Among many other features, TOSSIM also uses an Oscilloscope, a Sensor Network Topology window, and also a Serial Forwarder, some of which are shown in Fig. 4.



```
Listening to serial@COM1
Listening for client connections on port 9001
serial@COM1:19200: resynchronizing
```

Server Port: 9001
Mote Communications: serial@COM1
[✓] Verbose Mode
Pckts Read: 0
Pckts Wrttn: 0
Num Clients: 0

Fig. 4. Oscilloscope and Serial Forwarder Tools used in TOSSIM

II. Methodology

A. Problem Statement

The purpose of this research is to design a shortest path routing algorithm for use with the Berkeley motes. The objective of the shortest path routing algortihm is find the shortest path from one node to the next based on the information packet's beginning and final locations. The nodes of this network will have a single path that they will direct information down based on the destination address. This single path is designed to be the packet's shortest path, which is why it would be most beneficial to travel on that route. One problem that must be avoided is that of ”‘free-looping.”' This is when a node in the alogorithm that is on the shortest path malfunctions and the information packet is sent in a continual circle, never reaching its final destination. Also shortest path routing algorithms are usually simple, therefore, using it will most likely reduce the complexity of the overhead.

B. Shortest Path Routing

Designing an algorithm to implement shortest path routing between nodes for information transfer is a very complex matter. The goal of this algorithm is to efficiently find a path from one node to another using the least amount of ”hops” possible in a Multi-hop Wireless Network. There are many things to take into consideration before a shortest path routing algorithm can function. Power conservation and memory management create complexities that make devising an efficient route selection algorithm rather difficult. However, the research presented implements a possible solution.

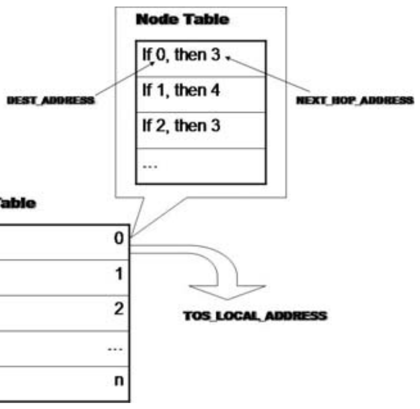


Fig. 5. Node Table and Structures

C. Shortest Path Algorithm

The essence of this shortest path algorithm rests in its ability to create a table and tree like structure. Imagine each node being organized in a tables by their addresses and final destinations as shown in Fig. 5.

Based on the destination address of the information packet, each node in this table also has a tree like structure that describes a shortest path from which they can determine which node to send the packet to next, as shown in Fig. 6.

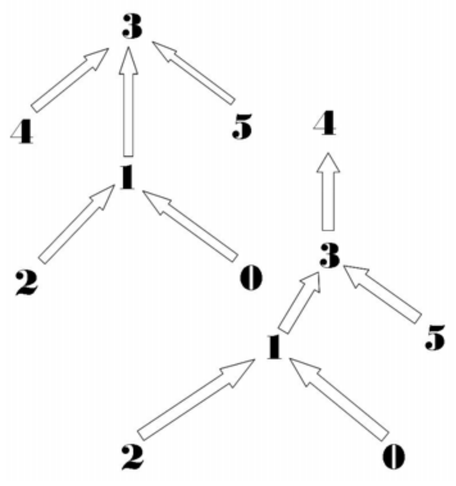


Fig. 6. Tree-like Structures

In theory, each node maintains a table sorted by the destination address of the packet that lists the ”next hop” address. There is a neighbor table that lists all the nodes, as well as a node table that lists the ”next hop” and destination information.

D. Advantages

While this may seem costly in terms of memory management, it provides a simplistic method for choosing which node needs to be on standby at all times, thereby improving the power conservation of the network. This method also gives the system a localization feature. Each node will know its location in proximity to its neighbors.

In most sensor networks, it is common practice to use a base station node to distribute and relay information to its surrounding peer nodes. With a base station system, in order to get the information packet to its destination, a node must first send the packet to the base station node, as shown in “Fig. 7”. The base station is the only node that can send to a final destination address.

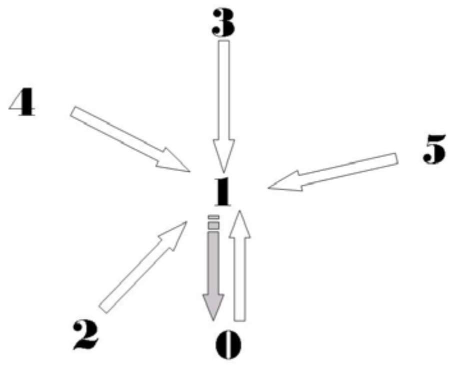


Fig. 7. Sample Base-Station Structure

The beauty of the design in this research is that any node can immediately direct the information to any other node at any time. Each node has equal bearing and status in this network structure. Imagine a node structure similar to that in Fig. 8.

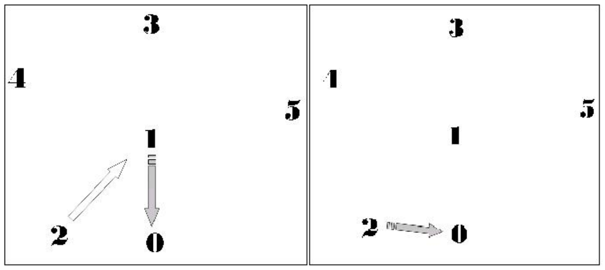


Fig. 8. Base-Station Disadvantage

In a base station system, if the node at address 2 wanted to send information to the node at address 0, it would have to send the packet to the base station at node address 1 first. In the shortest path system, the node at address 2 can directly send the information to the node at address 0, showing an improvement of efficiency between these two models. This also creates a more safe and secure network. For example, in a base station system, if the base station node malfunctions, that immensely affects the productivity of the other nodes. However, in the node structure described in this research, the malfunction of one node mildly affects the efficiency of its counterparts.

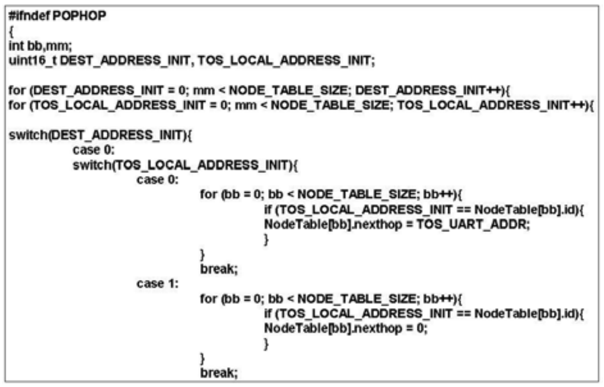
Some networks also use an algorithm that sends the information packet to all of its neighbors, no matter what the destination is. Each of the nodes that just received this information would then send the same information packet out to each of their neighbors. This progression continues until the packet has finally reached its destination. This algorithm is known as ”flooding.” Unlike this ”flooding” approach, the shortest path algorithm of this research eliminates this inefficient network behavior, directing a packet down one single path.

E. Disadvantages

Like most algorithms, this shortest path scheme does have its disadvantages. Being that each node maintains a route structure to each different destination address, this system uses a lot of memory, which hinders the overall efficiency of the network. Also, this design assumes a static network structure. It would not be nearly as effective in mobile or hybrid systems.

F. Coding

The coding of TinyOS was done in nesC, a programming language based on the C programming language. Four tables had to be defined in the header file for the necessary structures of this algorithm to be created; they include a node entry table, a node table, a neighbor entry table, and a neighbor table. The node entry table contained the node address and the address of the ”next hop.” The node table organized each of the node table entries by a final destination address. The neighbor entry table contained a node address as well a node table for each node. Lastly, the neighbor table organized each of the neighbor table entry by node addresses (TOS LOCAL ADDRESS). In the route select module, a static network of nodes is formed. Here, there are given their shortest path structures as well as their table entry location. Sample code of this process is shown in Fig. 9.



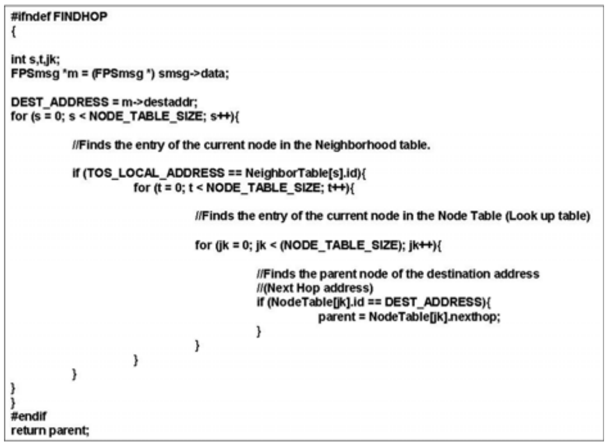
```
#ifndef POPHOP
{
int bb,mm;
uint16_t DEST_ADDRESS_INIT, TOS_LOCAL_ADDRESS_INIT;

for (DEST_ADDRESS_INIT = 0; mm < NODE_TABLE_SIZE; DEST_ADDRESS_INIT++){
for (TOS_LOCAL_ADDRESS_INIT = 0; mm < NODE_TABLE_SIZE; TOS_LOCAL_ADDRESS_INIT++){

switch(DEST_ADDRESS_INIT){
          case 0:
          switch(TOS_LOCAL_ADDRESS_INIT){
                    case 0:
                              for (bb = 0; bb < NODE_TABLE_SIZE; bb++){
                                        if (TOS_LOCAL_ADDRESS_INIT == NodeTable[bb].id){
                                        NodeTable[bb].nexthop = TOS_UART_ADDR;
                                        }
                              }
                              break;
                    case 1:
                              for (bb = 0; bb < NODE_TABLE_SIZE; bb++){
                                        if (TOS_LOCAL_ADDRESS_INIT == NodeTable[bb].id){
                                        NodeTable[bb].nexthop = 0;
                                        }
                              }
                              break;
```

Fig. 9. Sample Code

Also in the route select module, a function named ”getParent” is called that is passed through the actual FPS message. From this, the final destination address is gathered. Based on the destination address from the FPS message, an algorithm determines the shortest route for the information packet by searching through the tables and structures created. A sample of this code is included in Fig. 10.



```
#ifndef FINDHOP
{

int s,t,jk;
FPSmsg *m = (FPSmsg *) smsg->data;

DEST_ADDRESS = m->destaddr;
for (s = 0; s < NODE_TABLE_SIZE; s++){

          //Finds the entry of the current node in the Neighborhood table.

          if (TOS_LOCAL_ADDRESS == NeighborTable[s].id){
                    for (t = 0; t < NODE_TABLE_SIZE; t++){

                              //Finds the entry of the current node in the Node Table (Look up table)

                              for (jk = 0; jk < (NODE_TABLE_SIZE); jk++){

                                        //Finds the parent node of the destination address
                                        //(Next Hop address)
                                        if (NodeTable[jk].id == DEST_ADDRESS){
                                                  parent = NodeTable[jk].nexthop;
                                        }
                              }
                    }
          }
}
}
#endif
return parent;
```

Fig. 10. Sample Code

After it finds this value, it will return the address of the ”next hop” that is on the shortest path. Then, this address is then placed in the appropriate location in the structure of the message. This message the gets placed on the send queue, dequeued and sent, and then received by the node at the ”next hop” address. This cycle repeats until the information packet has reached its final destination.

III. Results

A. TOSSIM

Using the TOSSIM application of TinyOS, the shortest path algorithm was simulated. TOSSIM was used as a very helpful debugger and also gave insight on the code procedures. Before simulating the algorithm, debugging statements were put in to see the processes of the program. The algorithm was then simulated with six motes and the information packet was given a final destination address of five (nodes had addresses zero through five). The simulation cycled through each node as the source address, modeling every individual circumstance. Using TOSSIM's debugging window, the packet appeared follow the direction of a node neighborhood structure. This structure is shown in Fig. 11.

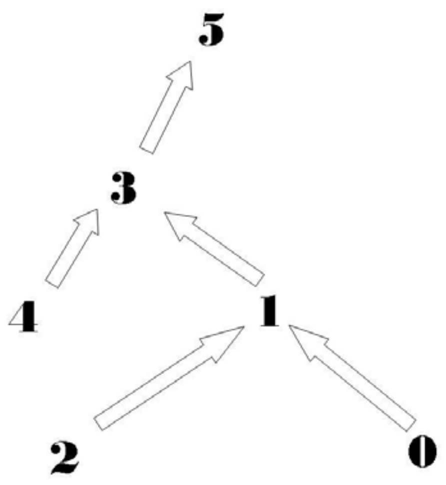


Fig. 11. Node Neighborhood Structure

For example, when the packet originated at node address 0, its ”next hop” address would be node address 1, then node address 3, and then node address 5, its final destination. However, if the node originated at node address 4, then the packet's next hop address would be node address 3, then node address 5, its final destination.

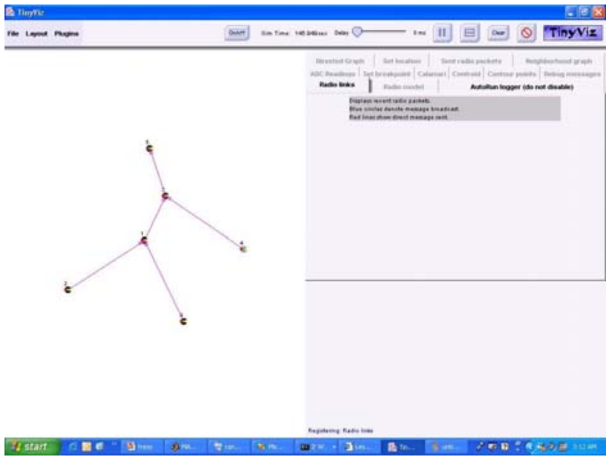
B. TinyViz

In addition to TOSSIM, TinyViz was used to gain insight about the code processes. TinyViz was actually used to visualize the movements of the information packets amongst the nodes in this simulation. A capture of this application in action is shown in Fig. 12.

Similar to TOSSIM, this illustrates a node neighborhood structure of the simulation. Each line represents the path that a packet would take if it has already been sent or if it was about to be sent. This visualization can be described by the same example that was explained earlier in this section.

IV. Discussion

Based on the static node neighbor structures given to the system beforehand, the results received from simulation were very much expected. The route of the information packet from each node was identical to the node structures and trees defined at an earlier time in the program. In fact, the figure used to display the node neighborhood structure in the TOSSIM example of the results section was the same node neighborhood structure illustration used to originally create the node tree of a final destination address equal to five. Therefore, the output of the simulation was exactly what was needed for the algorithm to be successful.



Displays recent radio packets.
Blue circles denote message broadcast.
Red lines show direct message sent.

Fig. 12. Simulation in TinyViz

V. Conclusions and Future Works

A. Conclusions

TinyOS is a very extensive and complex system. It has many applications and tools that need to be studied before one can fully understand the entire system. Doing this research, I learned how to use most of the TOSSIM applications and how to program in TinyOS using nesC. I also learned about mote networks and how they will soon change the world. This project requires a lot of time and effort. I went through a lot of pain trying to get some of the TOSSIM applications running and programming in TinyOS. However, because I went through this pain, I learned a lot more than I probably would have been had someone else just told me been. From this project, I helped contribute meaningful code that might one day serve a purpose in the Ad Hoc Wireless Network Technical Arena. I hope my code can help others who are seeking ideas for a shortest path algorithm.

As of today, the average person would probably look at a mote or node of a wireless network, and consider it harmless. However, these motes have the capacity to develop into many things, especially when they are being used developed by our military. These motes can turn into a rather large ethical problem. One day they will probably be small enough to live inside of a human body. Having the motes sense the typical bodily functions and movements of a human infringes on privacy and human rights laws. Let's hope that the dangers of this new technology will be averted before it gets to this level.

B. Future Works

If I had more time, I would have wanted to put in a link estimation scheme into my shortest path algorithm. This would have allowed the node system to choose its path based on the strength of a signal it recieved from its neighbor. In this scheme, the nodes estimates how far away their neighbors are based on the strength of their signals. The nodes of this system also have a developed node structure as they know how far away their neighbors are. Based on this node structure, a serparate algorithm could have been used to calculate the shortest path from a source node to its final destination, giving me a more efficient shortest path algorithm.

Ad Hoc Wireless Sensor Networks is a field that is advancing very rapidly. The evolution of the Berkeley motes shown in Figure 1 is just one example. There are many companies and institutions around the world working in this field and it is only a matter of time until issues with power conservation, localization, and memory management are diminished. As for now, different algorithms are being developed that incorporate more power efficient routing schemes for use with these sensor networks. Being that most sensor networks operate using some limited power supply, it is important to minimize the power usage of a network as much as possible. Researchers have been looking at this problem from both the software and hardware levels. Also, because of the limited amount of memory that a network has, it is important to cut back the memory usage of a system as much as possible. It is important to note that making a network with cheaper memory usage might also help conserve power. Sometime in the near future, these matters will be worked out which will produce a more efficient wireless sensor network.

VI. ACKNOWLEDGMENTS

I would like to thank the SUPERB-IT program, CHESS program, Jonathan Sprinkle, Barbara Hohlt, Susan Gardner, and Tanya Roosta. They each contributed to my development in Ad Hoc Wireless Sensor Networks.

REFERENCES

[1] M. Ilyas, Ed., *The Handbook of Ad Hoc Wireless Networks*, 1st ed., ser. The Electrical Engineering Handbook Series. CRC Press, 2003.
[2] E. Brewer, P. Levis, and D. Gay, “nesc 1.1 language reference manual, Tech. Rep. 1.1, May 2003.
[3] N. Lee and P. Levis, “Tossim: A simulator for tinyos networks,” UC Berkeley, Tech. Rep. 1.0, Sept. 2003.
[4] “Tinyos tutorial,” Tech. Rep., Sept. 2003.