ABSTRACT

The proliferation of ad hoc networks with a limited power supply presents a new challenge to ad hoc wireless routing. To support routing in an energy-constrained ad hoc network, energy-aware routing policies can be combined with existing protocols to reduce node failure. In this paper, we provide a detailed description of the Dynamic Source Routing (DSR) and Energy Saving Dynamic Source Routing (ESDSR) protocols, present an application-level energy-aware Dynamic Source Routing (DSR) implementation that can be modified to conserve energy, and identify ideas and variations of DSR that can be explored in future research.

General Terms
Experimentation, Measurement

Keywords
Wireless ad hoc networks, sensor networks, routing, energy saving, network lifetime

1. INTRODUCTION

Wireless ad hoc sensor networks have gained importance and popularity in both research and industry. In many situations, the nodes in the network are designed to operate for long periods of time without human intervention. Due to the absence of a constant power supply and the limited lifetime of batteries, energy-harvesting techniques extract energy from the environment to provide power to nodes. Each node in the network will likely differ in the amount of energy it can harvest at any point in time. For example, [6] describes a possible network of solar-powered nodes where some nodes receive a high amount of solar energy in the morning, while other nodes receive more energy in the afternoon. Each node in this network should consider the energy amounts of other nodes and thus route packets in such a way as to maximize node lifetime.

An important consideration in these types of networks is conserving energy when routing. In the past, wireless routing protocols have been developed that enable nodes in an ad hoc network to efficiently discover routing paths to other nodes. The proliferation of ad hoc networks with a limited power life-span presents a new challenge to ad hoc wireless routing protocols. In addition to considering end point reachability, path edge count, available bandwidth, and latency in network edges, wireless routing protocols will also need to factor in available power when constructing routing paths. To support routing in energy constrained ad hoc networks, power-aware routing policies can be integrated and evaluated with existing features in the Dynamic Source Routing (DSR) routing protocol.

The purpose of this paper is to present the results found from implementing an energy-based routing policy derived from the DSR protocol that attempts to maximize the longevity of the nodes in an ad hoc wireless network. Routing updates utilize energy data to determine which nodes in the
network to use, enabling both reliable packet transmission and power conservation. This project leveraged existing work on energy based routing by exploring and implementing the DSR protocol as well as identifying the benefits and deficiencies of an energy-based routing policy.

The rest of this paper is organized as follows. First, we provide a detailed description of both the DSR and ESDSR routing protocols (section 2). Second, we describe and show results of our own DSR implementation that allows for ESDSR modifications (section 3, 4, and 5). We then identify ideas and variations of ESDSR that can be explored in future research (section 6) and conclude (section 7).

2. BACKGROUND

Unlike fixed networks, routing in an ad hoc wireless environment must account for nodes that arrive and leave the network. In order to be scalable, ad hoc network routing protocols should avoid active probing to maintain awareness of the network topology.

A common approach is to utilize a source-path routing approach where the entire forwarding path is included in the network packet. This is in contrast to normal IP routing, where packets are forwarded hop-by-hop.

2.1 Dynamic Source Routing

[1] describes the Dynamic Source Routing Protocol (DSR), a routing protocol designed for multi-hop wireless ad hoc networks. Because DSR can support mobile nodes, the protocol enables the network to be self-organizing and self-configuring. Packets sent from one node to another in an ad hoc network using DSR include the entire route path within the packet itself. Route discovery is performed when a node does not know the full path to a destination. During route discovery, a node will periodically broadcast a route discovery packet. Each node in range of the original sender that receives the broadcasted route discovery packet will forward the route discovery packet until the final destination is discovered. Before forwarding, the intermediate node will append its address to the route path of the route discovery packet. Once a destination has been resolved, the entire path is returned to the sender using a Route Reply. Route Maintenance is the mechanism by which DSR detects a change such as a lost link in the network topology. Once a link has been detected as lost, a sender can either attempt an alternative path or invoke route discovery.

Figure 1. Example DSR Topology

Table 1. Example Routing Table

<table>
<thead>
<tr>
<th>Node 1 Routing table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Destination</strong></td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Consider the simple four node network illustrated in Figure 1. Suppose all nodes have empty routing caches and that Node 1 wishes to send a packet to Node 2. Node 1
will broadcast a route discovery packet (destined for 2) to all neighbors in range. In this example, nodes 2 and 3 are assumed to be within broadcast range of node 1 but not within range of each other. In the first branch of the route discovery, Node 3 receives the route discovery broadcast, detects that it is not the intended destination, appends itself to the path, and rebroadcasts the packet. This process is repeated again at node 4 and eventually a path from \{1, 3, 4, 2\} is discovered and returned back to node 1. In the second branch, Node 2 detects the route discovery from Node 1 and returns a routing path of \{1, 2\}. By default, DSR will choose the routing path based on the shortest number of hops. Therefore, packets sent to Node 2 from Node 1 will take the path of \{1, 2\}. Table 1 summarizes the routing tables for all nodes in the example.

Although DSR is intended for use in limited hop (5-10) ad hoc networks, it can be integrated with heterogeneous wireless/radio network technologies using logical interface indexes and gateway nodes. Different logical interface indexes allow a node in an ad hoc network to leverage both long and short-range radios to transmit packets, while gateways enable nodes to transmit and receive packets from the Internet. Routing optimizations can also be incorporated into the protocol by listening on broadcast packets.

The DSR protocol is flexible enough so that custom routing policies can be implemented for selecting network paths to traverse. For our study, we modify DSR to support an energy-aware routing policy.

### 2.2 Energy Saving Dynamic Source Routing

[2] presents the design, implementation, and performance of the Energy Saving Dynamic Source Routing (ESDSR) protocol. ESDSR is a modified version of dynamic source routing (DSR) that integrates the advantages of a transmission power control approach and a load sharing approach to save energy and maximize the lifespan of a mobile ad hoc network (MANET).

ESDSR combines two approaches that try to minimize the amount of energy consumed during routing. The transmit power control approach minimizes total transmission energy. The disadvantage of this approach is that it always chooses the same least-transmission power path, which causes this path to be overused and hence 'die' faster than other paths. The load sharing approach focuses on balancing energy usage among nodes by avoiding over-utilized nodes. The disadvantage of this approach is that it assumes transmission power is the same for all nodes. Energy can actually be saved by transmitting at a lower power for closer nodes. ESDSR chooses a path as follows. First, it gives a score to each path. The ratio of the remaining battery energy and the current transmit power of each node in the path is calculated. The minimum ratio along the path is considered the score. The path with the highest score is then chosen to be the one used for routing. ESDSR still performs route discovery and route maintenance in the same manner as DSR except that it stores the extra energy and power information in its route reply packets. [2] shows results of this protocol from a simulation using ns-2. Compared to DSR, ESDSR can save energy up to 40% per packet and can send 20% more packets with the same battery power.

We used the idea of selecting a path based on the remaining battery energy and implemented it on a real platform. We then compared this with the performance of DSR.

### 2.3 Other Related Work

[3] describes the Destination-Sequenced Distance Vector (DSDV) algorithm. In response to the insufficiency of the well-known link-state and distance vector
methods for ad hoc networks, [3] proposes the DSDV algorithm that maintains the simplicity of distance-vector while avoiding the looping problem of link-state. DSDV uses sequence numbers to distinguish old routes from new ones. Like distance-vector, DSDV uses periodic updates and new-information updates to populate and maintain routing tables at each node. All updates contain the following information for each destination: the destination’s address, the number of hops needed to reach the destination, and the most recently received sequence number of the destination. When choosing an optimal route, the route (or next-hop) is determined first by the highest sequence number and then by the optimal metric. This ensures that the routing decision is based on the most recent information, in case a node fails or moves. Unfortunately, DSDV, like distance vector, periodically shares routing table information with its neighbors whether or not a route is needed. For sensor networks, this extra work may not be useful or necessary if certain routes are not required. [5] presents an overview of design considerations for Energy Harvesting Embedded Systems (EHES) at both the system and software level. Among other discussions, [5] provides several software power management policies to consider in designing EHES systems. Specifically, power management at the network level addresses how spatial variations in harvesting opportunities can be managed across different nodes. In the past, energy aware routing protocols have used battery power as a metric for routing cost as a way to spread out the work; however, with the emergence of energy-harvesting nodes, optimal routing protocols should consider both the battery level as well as the energy harvesting prediction of each node. [5] provided the inspiration for this project with a detailed scenario of a sensor network where nodes receive different amount of solar power. The specific algorithm presented that requires an energy prediction algorithm may be too in depth for our purposes, but is a consideration for future work.

3. EXPERIMENT

3.1 Experimental Setup

The experimental data reported in this paper was taken on a small 5 node wireless network. Each node is represented by an Intel PXA27x ARM based machine that features functionality commonly found in embedded and sensor devices. Each PXA27x device features a touch-screen console, 16 megabytes of flash storage, 802.11b wireless configured for ad hoc networking, and a port of Linux optimized for the ARM processor.

3.2 Click and Grid

To conduct our experiments, we first attempted to port Click and its associated Grid packages to the PXA27x ARM development environment. Click is a publicly available software-based package developed at MIT/UCLA that enables the construction of network routers. It consists of an object-oriented framework that can be used to compose the functionality typically found within routers. Specifically, power management at the network level addresses how spatial variations in harvesting opportunities can be managed across different nodes. In the past, energy aware routing protocols have used battery power as a metric for routing cost as a way to spread out the work; however, with the emergence of energy-harvesting nodes, optimal routing protocols should consider both the battery level as well as the energy harvesting prediction of each node. DSR is one of the supported routing protocols included in Click/Grid. Our goal with porting Click/Grid was to establish a working baseline routing foundation from which to implement EDSR.

After assessing the options in the Click package, we decided to generate a user-level network router configuration that would utilize the Universal TUN/TAP interface for intercepting packets sent and received by the Linux operating system. The kernel on
several of the PXA27x ARM machines was patched and rebuilt to support the TUN/TAP interface.

After patching the kernels, we successfully compiled Click using the ARM-Linux-gcc cross compiler. However, the configuration files needed to run Click in DSR mode required Perl to be executed on the Click host machine. We attempted to generate a DSR configuration file on the host-compilation machine and port it over to the PXA27x ARM machines. Unfortunately, neither the configuration port nor the attempts to port Perl to the PXA27x ARM machines using our cross-compiler were successful.

3.3 Alternate Method

Due to the limited time frame and the difficulty of porting Click to the platforms, we resorted to writing our own application-level DSR and ESDSR protocols in C/C++.

4. DESIGN

Figure 2. High-Level Design

Figure 2 shows the high-level design of the DSR implementation. The design consists of the following primary components: four threads, three queues, an energy table, and a routing table.

4.1 Threads

The routing protocol is achieved using four threads: send, receive, process, and client. Each thread performs a specific function and communicates with other threads.

The receive and send threads are responsible for packet delivery and both use a UDP datagram socket to communicate with the other nodes in range. When the receive thread receives data (using recvfrom()), it converts the data to a message and places it in the received queue.

The process thread removes messages from the received queue and processes the message according to the content type field. Depending on the message contents, the process thread will do one of four things: drop the message, place the message in the send queue, output the contents to the client, or issue a route discovery and place the message in the pending route discovery queue. The process thread is responsible for updating both the routing table and the energy table, as well as selecting the path and next hop. The send thread removes messages from the send queue and sends them (using sendto()).

The client thread interacts with the user, responding to commands such as sending a message, printing the routing and energy tables, and printing the number of successful messages sent.

4.2 Queues

Inter-thread communication and synchronization are achieved using three queues: send, receive, and pending route discovery. Mutexes are used to protect queues from race conditions, and condition variables are used to signal threads when the contents of a queue has been changed.

4.3 Tables

Each node has both a routing table and an energy table. The routing table stores paths from the source to destinations. Paths are added to this table after route discoveries
and removed when route errors are received. The energy table stores the most updated energy values of each node. This table is used in ESDSR to determine the best path.

4.4 Message Types

The routing protocol is based on a message structure, shown in Figure 3. Every message stores the path from the source to destination to be used when forwarding between nodes. There are five types of messages that can be exchanged among nodes:
- route discovery
- route reply
- message
- ACK
- route error

The route reply message will travel back to the source node through the path specified in the message. Intermediary nodes will simply forward the route reply message to the next hop in the path. When the route reply is received by the source node, it will add the path to its routing table and send the pending messages.

4.4.3 Message

A message is generated by the client thread. If a path to the destination exists, the message will be sent; otherwise, a route discovery will be issued as described in section 4.4.1. All intermediary nodes will forward the message to the next hop in the path. When the destination receives the message, it will output the contents to the client and send an ACK back to the source address.

4.4.4 ACK

An ACK is generated after the destination receives a message. The ACK is forwarded by each intermediary node to the next hop in the path, which is finally received by the source node. An optimization to our design and area for future work is to add another queue to contain messages pending an ACK. If an ACK is not received in a specified amount of time, the message would be resent.

4.4.5 Route Error

A route error is issued when a node cannot be reached. The route error is returned to the source node, which then removes the path from the routing table.

4.5 ESDSR Modifications

In order to compare DSR to ESDSR, each node tracks its current energy level using “energy units.” When a node receives a route reply or an ACK, it updates its energy field in the packet path, as seen in Figure 3. Once the source node receives either a route reply or an ACK, it updates its
energy table with the energy values in the path.

When choosing a path, the DSR implementation chooses the path with the minimum number of hops. For ESDSR, however, the path is chosen based on energy. First, we calculate the bottleneck energy for each path, that is, the lowest hop energy of the path. The path is then selected by choosing the path with the maximum lowest hop energy. For example, consider the following scenario. There are two paths to choose from. The first path contains three hops with energy values 12, 8, and 100, and the second path contains four hops with energy values 20, 15, 25, and 80. The score for the first path is 8, while the score for the second path is 15. Because 15 is greater than 8, the second path would be chosen.

5. RESULTS

To test the effectiveness of both our DSR and ESDSR implementation, we ran two experiments. The first used a fixed sender and receiver node, while the second exchanged messages randomly between all nodes. In our measurements, we define the network lifetime as the number of messages successfully sent before there is no longer a path to the destination, and node lifetime as the number of messages successfully sent before any node in the network dies.

For both experiments, we tested four topologies, shown in Figure 4. For the first experiment, node 11 is the sending node, and node 16 is the receiving node. In the first topology, each node is in range of exactly two other nodes, forming a circle. In the second topology, there is exactly one path from one node to all other nodes; Because there is only one possible path, all results for this topology should be identical for DSR and ESDSR. Topology three is similar to topology one with an extra communication link between nodes 12 and 14. Finally, topology four connects each of nodes 12, 14, and 15 with both nodes 11 and 16.

5.1 Experiment with Fixed Sender and Receiver

In the first experiment, both the sender (node 11) and the receiver (node 16) are assumed to be connected to a constant power supply and thus, have infinite energy. The other three nodes (12, 14, and 15) each start with a set energy level: 70, 100, and 200 respectively. The setup for topology one is shown in Figure 5.

To measure the node lifetime for both DSR and ESDSR, we sent messages from node 11 to node 16 until the first node died. As seen in Figure 6, ESDSR extended the
node lifetime for all topologies excluding number two. In topologies one and three, ESDSR alternates paths, more than doubling the node lifetime when compared to DSR. Topology three shows a slightly lower number of messages, which results from the additional possible paths that must be traversed during route discovery. Topology four shows the greatest improvement in node lifetime, as ESDSR sent more than five times the number of messages than DSR. Because there are three possible independent paths in this topology, ESDSR fully utilized all paths to extend the node lifetime.

![Figure 6. Node Lifetime](image)

After measuring the node lifetime, we measured the network lifetime. As expected, node 11 was able to send the same number of messages to node 16, regardless of policy. For example, in topology one, DSR will first use the \{11,12,16\} path until node 12 dies, and then use the \{11,14,15,16\} path until it dies. ESDSR will use \{11,14,15\} until the energy values in nodes 12 and 14 are equal, and then will alternate the two paths until nodes 12 and 14 both die. The difference between DSR and ESDSR, in this experiment, is only the order in which paths are taken.

In a network where nodes can recharge their energy values over time, however, ESDSR can extend the network lifetime. This is true for nodes that require human intervention when out of energy (for example, a person is required to physically turn on the node). ESDSR works to extend the node lifetime, so that all nodes have sufficient time and opportunity to recharge. Therefore, the network lifetime will be longer using ESDSR than DSR.

![Figure 7. Network Lifetime](image)

**5.2 Experiment with Randomized Sender and Receiver**

For the second experiment, each node began with an energy value of 100 energy units, and the sender and receiver for each message were selected using a random number generator. The network lifetime was measured by sending messages until a path from the sender to the receiver no longer existed. Figure 8 shows the results of this experiment. In topologies one, three, and four, the network lifetimes were significantly less than the results found in the previous experiment. This is due to the increased number of route discoveries in this experiment. The first time each node sends a message to another node, a route discovery is issued, which consumes a large amount of energy. Route caching, a possible optimization to our implementation, would decrease this overhead. Unfortunately, the results did not show that ESDSR provided a longer network lifetime, which we believe is due to a flaw in our implementation and will
be described in more detail in the following section.

![Network Lifetime Graph](image)

**Figure 8. Network Lifetime**

5.3 Discussion

5.3.1 Design Flaws

During experimentation, we discovered several flaws in our ESDSR implementation. First, the score of a path is determined by the minimum hop energy of the path including the source and destination. For our first experiment, the sender and receiver nodes had infinite energy, but in our second experiment, it is possible that the sender or receiver node had the bottleneck energy of the path. Thus, the score of all the possible paths would be equal, and an arbitrary path would be selected. Instead of choosing the minimum energy of the entire path, ESDSR should exclude the source and destination nodes when determining the minimum energy of the path.

Second, in order to maximize the lifetime of a single node, the selection of the optimal path is based entirely on the bottleneck energy. One can imagine a situation where there are two paths, one path with one hop and one path with 100 hops. Choosing the path with 100 hops may extend the lifetime of the one node in the alternate path, but will certainly hurt the network overall by consuming 100 times the energy. An alternative to our path selection could be to consider both the bottleneck energy as well as the overall path energy, assigning a weight to each score and choosing accordingly.

5.3.2 Experimentation Challenges

Due to our decision to work with actual embedded platforms as well as the limited number of these platforms, testing was extremely time-consuming. Every time changes were made to the application, the program had to be cross-compiled and mounted onto each of the five boards through an Ethernet connection. Testing required manually sending a message from one node to another and checking to see if an ACK was received. Because of these circumstances, we were unable to perform extensive and truly meaningful experiments with our implementation.

One way we overcame this challenge was by using simulated UDP broadcast. Instead of physically deploying the five boards and testing the range of the network, we set the neighbors for each node to the desired topology. During route discovery, each node would send a packet to each of its neighbors. In retrospect, it would have been beneficial to work with a simulator such as ns-2.

6. FUTURE WORK

There are many different routing metrics that can be explored and integrated with DSR and ESDSR. Currently, ESDSR does not consider latency. Designing a metric that factors latency in its calculation could be an idea for future work. In [7], routing decisions are made based on packet loss. This would provide another interesting metric that we could add to ESDSR.

We could also use our experimental setup to test other energy saving methods that we currently did not implement. We could vary transmission power for closer and farther nodes. We could also use power management techniques. If a node does not appear to be needed for a certain period of
time, we could put it to sleep and wake it up once it is needed again. Another idea would be to model energy recharging rates for nodes and investigate the performance of ESDSR with nodes that can recharge over time.

Another area of study in the performance of routing protocols is in comparing static and mobile ad hoc networks. Our current implementation does not support a mobile ad hoc network. It would be interesting to see how our results would differ since [7] claims that DSR performs better for mobile ad hoc networks.

Other work can be done to optimize our current implementation. We could incorporate route caching, investigate a kernel-level implementation, and support timeouts for messages. We could also incorporate a better route maintenance implementation. For route caching, we could store energy values and routes at intermediate nodes. Kernel-level implementations and supporting timeouts for messages would provide an increase in performance.

7. CONCLUSION

We have shown that an energy-aware dynamic source routing protocol can maximize the lifetime of a node in a network. We have shown that it is feasible to build such a protocol to run on a real embedded platform. While there are many improvements that can be made to our design and implementation, we have learned important concepts and lessons in the field of computer networks. We have learned about different routing protocols for mobile ad hoc networks, the difficulties of programming and using sockets with a network where the neighbors are not initially known, the problems associated with porting routing software to an embedded device, the details and design of dynamic source routing, and the challenges of testing on an embedded platform.

Acknowledgements

We would like to thank Edoardo Regini for providing access to the PXA27x platforms used in this project and for his numerous suggestions in design. We would also like to thank Thomas Weng for allowing us to store our equipment in his office. Finally, we would like to thank Dr. Amin Vahdat for his support and paper recommendations.

8. REFERENCES