Modeling Dynamic Source Routing with Energy Aware Path Selection Using a Rule Based Expert System

Steven Baylor, Jeffrey Corcoran & Alexander Maskovyak
Introduction to Artificial Intelligence – R.I.T.
Graduate Class Research Project
August 11, 2008

Abstract.

Most networks are composed of stationary devices utilizing fixed physical connections, which operate without regard to energy consumption. More recently, there has been an explosion in the growth of ad-hoc networks due to increases in the availability and decreases in the cost of wireless devices like cell-phones, PDAs, and other portable computers. Contemporary protocols such as TCP/IP have dominated fixed networks for years but require the use of routing protocols like (e.g. RIP, OSPF) which send control messages frequently to keep routing tables up-to-date. This is a costly strategy in mobile devices where the cost of transmission is high and power-reserves are limited. This is an energy expensive method of networking that is non-ideal for ad-hoc situations where battery life is critical for maintaining the integrity of both the devices themselves (to perform its own functions) and the network as a whole (to ensure connectivity). This paper explores the authors' design of Dynamic Source Routing, common protocol used in ad-hoc networks, because of its energy saving features. This protocol was extended with several author-created features to enhance its energy savings. Jess, a rules engine, is used to determine the feasibility of implementing a routing protocol using an expert system. The assumptions, implementations, and design choices that went into this model's construction are discussed at length.

1 Introduction

This paper describes two areas of research. The first involves researching the operation of Dynamic Source Routing and prototyping it using an expert system. The prototype uses a rule-based intelligent agent to perform routing and datagram manipulation. The research and companion project specializes in the extension of DSR with energy aware functionality.

1.1 Research Objectives

The Java Expert System Shell (Jess) is used to implement the intelligent agent at the heart of the protocol. The intelligent agent processes rules needed by the DSR protocol to support route discovery and route selection. The rule based agent allows for extensibility when designing a route selection algorithm. The prototype was developed in Java, which provides models for common aspects of a network that the protocol may need to interact with. The protocol is a virtual network layer when considering the seven-layer approach to networking. Data encapsulation is modeled after the 5 layer approach used in modern networking. Several modification of the network layer datagram allow the protocol to collect and process power metrics from other devices in the network, which allow it to make routing decisions designed to distribute energy load.

1.2 Background

Dynamic Source Routing (DSR) is a network layer protocol used to support routing functionality in Mobile Ad-hoc NETworks (MANETs), mesh networks, and sensor networks. Network layer protocols are responsible for end-to-end delivery of prepackaged units of information known as datagrams. This service is provided through a combination of device addressing and route selection. Route selection involves some form of least-cost path-finding algorithm that operates upon a device's internal representation of the network's topology.

This internal representation must be updated periodically to maintain consistency with the topology of the network. The topology of a network is dynamic. Devices connect and disconnect regularly. The cost of using a link increases and decreases dependent upon congestion and environmental effects. Routing devices send control messages to one another with information regarding the state of their links. Traditional, wire-bound routing protocols send this control information at regular intervals. These proactive protocols ensure that their network representation is as up-to-date as possible to ensure reliable communication along best cost paths before the need to send data arises.

Mobile and sensor based devices have a finite reserve of energy which powers communication equipment whose cost for use is energy-expensive. These devices do not have the luxury of sending regular control information since this would quickly exhaust their power supplies. Instead, these devices tend to use reactive protocols which send control information only when a route (of unknown viability) to a destination is not already known. These protocols trade-off the accuracy of their internal network representation in favor of energy savings—allowing them to potentially increase the ratio of data to control messages that can be sent over their lifetime. There are costs to using stale information. The best-cost path of the internal representation may not be identical to the best-cost path in the actual topology, meaning that this device may expend more energy than necessary by using a more costly path. In cases where the environment is highly dynamic, the path selected may no longer be viable. This will force the device to send costly control messages to update its network information which increases the latency of actual data messages. In the worst case, this delay may cause a time-out on the datagram and prompt data resends—further decreasing a device's power supply and network throughput. The assumption is that the energy cost of using a non-ideal path and the need to use control messages when non-viable paths are discovered is less than the energy cost of using frequent control messages.

1.2.1 DSR

DSR is a reactive protocol that uses source routing. Source routing requires that the source of a datagram fully qualify the route that the datagram should follow through every intermediate device along the route from source to destination. This is in contrast to the next-hop routing used in other protocols (typical of proactive protocols) where only the next-hop is specified and routing decisions are made along every hop until datagram reception at the destination. Source routing allows intermediate devices to be ignorant of viable paths in the network since the routing decision has been made in advance for them. The disadvantage is that this fully qualified path must be transmitted with every datagram which costs additional transmission energy. This also places more of the routing burden on end-devices in the network where this might otherwise be the responsibility of a specially-purposed device designed explicitly to efficiently perform routing chores (i.e. an OSI layer 3 router).

1.2.2 Energy Aware Routing

DSR is an energy agnostic protocol. DSR's energy conservation is primarily a product of being a reactive protocol. The energy aspects of a route are not considered during path selection, instead, paths with lower-latency are given preference. Much research has been undertaken to create energy-aware routing protocols. DSR is often employed as a base for creating derivative protocols which make routing decisions based-upon device and network energy states. Theoretically, the trade-off of these approaches is between datagram latency (device performance) and energy savings.

Energy savings can be measured in various ways. The total energy/transmission remaining in the network after some number of datagrams has been sent is one common approach. The standard deviation of energy is another that attempts to measure how adequately a protocol distributes energy load network-wide. Network-health is often an important measurement. The performance of a network depends upon the network's connectivity. Device disconnection can force data to take longer paths in the network or even completely sever communication between ends of the network. To this end, network health can be measured in terms of the time until some number *n* devices completely exhaust their battery (known as single point failure).

An author has previously identified several strategies employed in power aware routing protocols [1]. General strategy categories that were considered for this implementation include those which modify DSR's path selection process and its path usage process. A brief re-examination of these strategies and the methods they depend upon is warranted here. A more thorough discussion on the design implemented by this group, including its trade-offs and the alternatives considered, appears later.

1.2.2.1 Tools for Energy Aware Routing

Energy aware protocols typically require additional information about the state of the network in order to make decisions that are based around the energy health of the said network. Many protocols use a power-metric as an abstract way to represent the health of a device. These power-metrics can represent a number of values including: the number of transmissions a device has made, the number of transmissions a device has left, the current energy level of a device's battery, etcetera. In many cases**,** this value cannot be calculated remotely for a given device. The solution is to have every device calculate their power-metric, and for them to include this value in the control messages of the route discovery process. Thus when a source device requests a path, it not only obtains a path but also the power-metrics of every device along that path. The term for this is "piggy-backing".

Piggy-backing allows a device to have a greater understanding of the energy states of devices in the network. However, this also increases the energy needed for transmitting control messages. It also lengthens the control message thus increasing the chance that some information could be lost during transmission due to environmental effects necessitating further energy expenditure for data retransmission. This also increases the latency of messages and reduces network throughput as it becomes more dominated by control information than by actual data.

While this process is helpful for the initial discovery and path selection phase, it does nothing to help a device understand the current state of a path it is using. Power-metrics can be piggy-backed on normal data messages as well, allowing a device to get an up-to-date view of a path's integrity without the excessive overhead of a control message. It allows a device to refrain from using a path when it knows of a preferable alternative path. This greatly aids in distributing energy load more evenly across the network to avoid the complete energy exhaustion of a device in a network. The drawbacks are similar to those when piggy-backing power-metrics on control messages.

This power-metric information can also be sent in a similar fashion to the control messages of a proactive protocol. In this case, devices "gossip" their power-metric (and possibly the power-metrics of other devices which they have stored) to their neighbors when they learn of an appreciable change in the network condition. Gossiped messages have a smaller energy footprint than discovery control messages. This means that they can be sent more frequently than normal discovery control messages and still potentially use the same amount of power for transmitting information. These gossiped messages may also be used to supplant the piggy-backing of power-metrics on normal data transmissions possibly saving additional energy. This approach is not without its drawbacks. Increased message sending means increased contention and congestion in the network which lowers throughput and increases the incidents of retransmission events. The energy savings associated with removing piggy-backed information may not be enough to overcome the increased energy cost of retransmissions.

Power-metrics that deal with distance may have additional hardware requirements. Devices must often employ GPS equipment to determine the location of other devices along a path in use and hence the power necessary to devote to transmission to communicate with those devices. This information allows a device to scale back power transmission to the minimum required to reach the next hop along a path. Location information also allows a device to infer additional link connections and paths that may not otherwise be presented to it during the normal path discovery process. This may allow a device delay or even forgo the expense of initiating the path discovery procedure. The use of GPS is not without its own costs. GPS equipment may be financially or spatially infeasible especially in the case of large sensor networks. The use of the equipment itself is also power-expensive.

1.2.2.2 Discussion of Energy Aware Protocols

Path selection focused routing protocols modify DSR's selection of a path when a source device, which previously sent a request control message to determine a path in the network, receives multiple reply control messages each of which contains path information to the requested destination. These protocols include: Min Power Routing (MPR) (versions 1 and 2) as well as Min-Max Battery Cost Routing (MMBCR) [2] [3].

MPR is implemented as two separate protocols. The first protocol selects paths that minimize the total amount of power required to transmit a datagram from the source to the destination device. This is done by assigning a link-cost between devices based upon the amount of energy required to send a datagram. These link-costs are affected by the retransmissions necessary along links with excessive congestion and/or contention. This protocol adequately avoids congestion and retransmission to extend the energy expended by the network as a whole. However, it cannot ensure network health since it does not deal with single point failures [2].

The second version assigns energy costs to devices based upon the number of transmission they have completed during their life on the network. Paths which minimize energy cost are given preference even where they involve a larger number of intermediate devices/hops than alternatives. This has the affect of causing longer paths to be taken in the average case. Notably, this does not measurably increase delay. Like the preceding variant, this protocol deals with path health and not individual device health, meaning that network longevity is not addressed [2].

MMBCR takes a different approach. It does not attempt to minimize the total amount of energy expended on transmission for a path. Instead, it attempts to address the issue of single point failures and network longevity. MMBCR attempts to distribute energy load more evenly across the network. It assigns a path cost based upon the inverse of the weakest device in the path. MMBCR selects those paths with the lowest cost (e.g. those whose weakest device is the strongest amongst the set of all weakest devices along all paths known). This means that MMBCR will often use paths longer paths with more devices which will cost more in terms of total energy expended for the end-to-end transmission. While this effectively reduces the energy level of a greater quantity of devices; it does successfully spare the weakest devices in the network from premature disconnection [3].

1.2.2.2 Path Usage Strategies

Path usage protocols go one step further and attempt to actively monitor the state of the active path as compared to alternative paths. These protocols will switch to an alternative path when it becomes preferable to do so. These protocols include: Consumption Based Load Balancing Routing (ECLB) and Multi-path Power Sensitive Routing Protocol (MPSRP) [4] [5].

ECLB stores multiple paths to every destination device in the network. It selects paths based upon their ability to support the data to be transmitted. Alternative paths are selected when the current path is calculated to be unable to successfully deliver the information during a transmission session. Additionally, devices can choose to “opt-out” of routing a message either during the route discovery or data forwarding stages if the device has determined that it will be unable to transmit due to battery exhaustion [4].

MPSRP similarly stores multiple paths, but uses a weighted round-robin approach to path use to more equitably distribute energy load throughout a network. Weight assignment is used to ensure that energy level variance across devices is maintained below a certain threshold. It also employs path aging to remove stale paths from the cache to avoid failed transmissions. Experimentally, both MPSRP and ECLB have been demonstrated to extend network lifetime appreciably and that this increases network throughput by a significant degree [4] [5].

1.3 Expert Systems

Expert systems are rule-based software systems designed to replicate the decision-making capability of human experts. Expert systems separate information into two categories: a knowledge-base which contains information about the domain of operation and a database which contains information acted upon by rules in the knowledge-base. Rules-engines are used in situations where the computational problem involves decision trees with significant conditional branching. They are ideal for situations where the capabilities of a system may need to change or extended with time due to the ease of inserting rules into the knowledge-base to augment or overshadow existing rules.

2 Feasibility Study

Research of the data exchange process in Dynamic Source Routing and collaboration dealing with energy considerations has allowed for the design of an on-demand route discovery and energy aware network protocol.

2.1 Project Scope and Limitations

The primary goal of this research is to determine whether the rules of a network protocol can be modeled by an Expert System. In addition, the research addresses the capabilities needed for dealing with energy utilization. Although based upon the DSR protocol, this solution is a prototype which ignores some functionality due to time constraints. Thorough testing might demonstrate the need for additional functionality that is unrelated to energy awareness or most scenarios for path selection. This research attempts to model the routing aspect of DSR and energy awareness and address some interaction issues between the transport and data-link layers. Additional abstractions were created to aid this undertaking and are described in Section 2.3 and the Appendices. Due to time limitations, the research emphasizes the set of rules needed to perform learning and intelligent routing.

2.2 Protocol Description

The protocol that was created is based on a reactive routing approach to wireless mesh networks. In this approach, nodes attempt to learn the topology of the network, by extracting and storing information that is embedded in the transmissions of other nodes. The information that is typically communicated amongst nodes, in a wireless mesh network that utilizes a reactive routing strategy, is the availability of the routing paths to other nodes. The authors' protocol collects this information, in addition to energy metric information of other nodes, and the cost associated with transmissions between node pairs. This energy related data is collected for not just adjacent nodes, but for all the nodes whose membership in the network is known. This enables the system to deliver messages in a way that minimizes the total energy required to send messages and also to avoid needlessly expending the energy of nodes with relatively low energy levels.

Using the information that is collected, a node is able to determine the complete route that a message must take to reach the destination. This is typical of reactive protocols that employ source routing but contrary to proactive protocols where each router determines only the next hop that a message should take to reach the destination.

Data that is required to determine paths is collected in three different ways. In the first situation, the node is either the desired recipient of a message, or is in the routing path between source and destination. In this case, the node extracts valuable data relating to the network conditions before delivering the message to the host application or forwarding the message to the next node in the specified path.

The second possible way to receive network related information is by receiving a message, where the node is neither the next intended recipient along a path nor the final destination. This occurs when the node is within transmission range of either the source node or a node along the path of the message. In this case, the node extracts data from the message, but does not forward the datagram along the path.

The third situation occurs when the node receives a message, or a segment, to send from the host application, but the node does not have enough information to determine a path to the destination. In this case, the node asks nearby nodes for directions to the destination, by sending a Route Request. Each node within receiving distance cascades the Route Request, until it either reaches the destination or it reaches a node which is already aware of a valid path to the destination. At that point, a Route Reply is generated and dispatched backwards along the route that the Route Request took, toward the originating source. The source node, as well as the nodes along the reverse path, can then extract the data from this, and other additional Route Replies, that it receives.

2.2.1 Implementation Assumptions

To model this type of networking protocol, the prototype design must accommodate some assumptions to help focus the scope of the authors' research. All transmissions made are omni-directional. In other words, a transmission at a given power level can be received up to the same distance in all directions. Nodes are configured in a two dimensional plane. There are no obstructions between the nodes that can affect the signal quality. Transmissions decrease the energy level of a node's battery. The amount of energy that is decreased from the battery is proportional to the power at which the transmission is broadcast. The size, in bits, of the transmission does not affect the energy consumption of the node. Receiving or processing messages does not affect energy consumption. The distance at which a transmission can be received by another node has a logarithmic relationship to the power at which the transmission was broadcast. All nodes have equal transmission and reception capability. The maximum transmission power and range is the same for all nodes, and the transmission cost from one node to another, is equal to the transmission cost in the other direction. There is no distinction between network layer and data-link layer addresses.

While the protocol should perform well in mobile scenarios, the prototype deals with a network of stationary nodes. Once a node learns the distance between two nodes, the distance will need no further updates. A node knows the distance, or corresponding transmission power, to the node from which it received a datagram. In other words, the receiving node knows the distance from the source node. This information is given to the receiving node by the java network object similar to how a wireless adapter can detect an incoming signal's strength. For example, a datagram is sent from node A to node B. Once node B receives the transmission, B now knows the distance to A. The sender, A, does not know the distance to B as a direct result of the transmission. The cascading of distance information is explained in the description of the protocol. Restated, learning distances between nodes is possible in application, but the implementation of this detail is not a priority of the research.

2.3 Implementation

The prototype was implemented in a Java environment, using a JESS rule engine and, as such, many objects that interact with the prototype are modeled. A node is synonymous to an entire device and has an agent to handle the protocol. An Agent interacts with its parent node to receive segments and datagrams. A segment originates from an application on the node, and consists of a message and a desired recipient. Appendix A describes each container used in the system such as the message, segment, and datagram.

A datagram is received from the network by the node and describes a datagram type, source address, destination address, segment from the upper layer, path, transmission costs along each hop in the path, and battery metrics for each node in the path. The agent stores data related to the known paths, battery metric information corresponding to discovered nodes in the network, and the cost of transmissions between nodes that are adjacent to each other.

The available outgoing paths are stored and sorted according to destination. A Hash Map relates a destination to a collection of available paths. The collection of paths is implemented with a Hash Set. Each path is stored as a member of the set in an Array List. Appendix B illustrates the details of each data structure used for storing and processing path information.

The battery metrics and transmission costs are also stored in the agent utilizing a Hash Map data structure. The battery metrics Hash Map relates each node's identification value to a corresponding metric value. The transmission cost Hash Map relates a pair of Nodes, which is represented by a Node Pair object, to a transmission cost value. The transmission cost and battery metric data structures are illustrated in Appendix B.

The agent uses a Rete rule engine, provided by JESS, to perform rule-based decision making and provide support methods that are utilized by the engine. The default rules for the protocol are written in the JESS language and stored in a rules file. The rules file is loaded into Rete by the agent during initialization. Dynamic modification of the rules could be performed by the agent by simple insertion in the Rete engine.

The following sections will describe the logic of each rule performed by the intelligent agent.

2.3.1 Segment Handling

If a node receives a command from an application to send data to another node in the network, the node will pass a message to the agent. The agent will decide if a path exists to the desired destination. If a path does not exist, the node will encapsulate the message in a datagram and mark it for follow up. The agent will initialize and transmit a Route Request datagram, RREQ. The datagram is of type RREQ and contains the sending node's ID and battery metric as well as a destination node ID. The RREQ datagram does not contain any segment or path specification. The intention of sending a RREQ is to eventually receive a Route Reply datagram, RREP, which will provide a valid path to the destination. When at least one RREP is received, the agent is aware of the datagram that was marked for later follow up and the learned path information is inserted into the datagram. The datagram is then broadcast by the node with the desired path. A timer could be set after receiving the first RREP to allow additional RREPs to provide alternative paths to the node but in order to minimize initial latency, the first path received is used. It is possible the node is already aware of, at least, one path to the destination. If more than one path has been discovered, the node must decide which path is optimal. The selection process is performed by calculating a cost for each path stored and choosing the path with the lowest cost. The cost is calculated through a mathematical formula, which takes into consideration the most recent battery metric information of each node in the path and the transmission cost between each adjacent set of nodes in the path. Battery metrics are determined by a function of the node's battery capacity and current level of charge as specified in section 4.2. With the previously stated assumptions in mind, the cost of the transmission between two nodes is proportional to the power at which a signal from one node must be transmitted to be received by the other, and is related, by a logarithmic function, to the distance between the nodes. Specifically, the cost of sending a datagram from node n_x to node n_j, using the path n_x, n_{x+1}, n_{x+2}… n_j is equal to the function below where B_k is the battery metric of n_k, and $T_{k,m}$ is the transmission cost between n_k and n_m.

$$\sum_{i=1}^{y-1} (B_i)(T_{i,i+1})$$

The protocol can be calibrated, to adjust the balance between the minimization of the total energy expenditure per transmission and the avoidance of nodes with low energy levels, by modifying the magnitudes of the functions B and T.

Once the node has determined the path for the datagram to reach the destination, the datagram is transmitted. The datagram contains the segment, the node's own identification as the source, the destination, the list of node identifications (the path) and the estimated battery metrics for each the node in the path.

The battery metric values in the datagram are the most recent information the sender has about other nodes along the path. Although these battery metrics may not always be up-to-date, and therefore estimates, the information is provided for nodes along the path, which may have even less recent data, to be updated. Likewise, if a node is in the path and reviews the list of battery metrics, it will modify its metric with current information to benefit additional nodes that will receive the datagram. This is covered in more depth in the next section. Appendix C illustrates how an Expert System decides to handle a segment.

2.3.2 Datagram Handling

When a node receives a datagram, regardless of the type (i.e. RREQ, RREP or Data) the node needs to extract and possibly update the battery metrics and transmission costs that may be provided in the datagram. The datagram may also contain new information relating to the network topology that can be observed by inspecting the datagram's path.

There are three possible cases regarding the node's position in the path information provided by a Datagram. If a node is the datagram's destination, it will be the last node listed in the path. In this case, the reverse of the datagram's path is a valid path back to the source of the datagram, and the node updates its list of paths to the source with it. Likewise, the node also stores paths to each intermediate node on the datagram's reverse path. For example, if source node A sends a datagram to a destination node D with the path A-B-C-D, when D receives the datagram it may learn and store paths D-C-B-A, D-C-B and D-C.

The second case is that the node is in the path, but is not the last node in the path. In this instance, the node can extract paths to each node preceding itself in the path as well as each node following it in the path. For example, if a datagram contains paths A-B-C-D, then node C may learn and store paths C-B-A, C-B and C-D.

The third scenario occurs when the node is not in the path at all. This occurs if the node is within transmission distance of another node broadcasting a datagram. Since devices operating with this protocol operate in promiscuous mode they intercept all messages sent through the network even when not addressed to them. This action, allows devices to "eavesdrop" on each other—allowing an unintended recipient to learn information. In this case, the node can extract paths to each node in the path, through the node from which the datagram was received. If node E is within transmission distance of C and eavesdrops on a datagram containing the path A-B-C-D, E may learn and store paths E-C-B-A, E-C-B, E-C and E-C-D.

After the transmission cost, battery metrics, and new paths are extracted from a datagram, the agent must decide what to do with the new datagram. The agent inspects the datagram's type which will either be RREQ, RREP or DATA.

If the type is a route request, RREQ, the agent checks if its node is the destination of the RREQ. If it is not, then the agent checks if it contains knowledge of a valid path to the destination. If it has a valid path, it creates a route reply, RREP, with this information and propagates it back to source. If the agent does not have a path to the destination, it appends its node's ID, current battery metric and transmission cost to the datagram before re-broadcasting it. The appended transmission cost is from the last node on the RREQ's path and the current node.

It is possible, and likely, that nodes will receive the same RREQ from more than one other node. The same RREQ should be forwarded only once by any single node, to avoid redundancy and to curtail the creation of paths with loops. Therefore, each RREQ datagram is assigned an identifier by the source node for receiving nodes to store for comparison.

If the node is the actual destination of the RREQ, the corresponding agent changes the type of datagram from a route request to a route reply and reverses the order of the accompanying path, battery metrics and transmission costs established while the datagram cascaded through the network. The node then assigns itself as the source, the destination, and broadcasts the datagram which is now of the type RREP.

If a RREP datagram is received, the agent checks if its node is the destination. If it is the destination, then the agent now has a path to a node which can be used to send a data type datagram. If the node is not the destination, it checks to see if it is in the path. If the node is not in the path, then the agent discards the datagram. If the node is in the path, then it updates its associated battery metric in the datagram and forwards the datagram by broadcast.

When a DATA type datagram is received by a node the agent checks if the node is the destination. If it is, then the segment is extracted and passed to the node which pulls the message and delivers it to the application. If the node is not the destination but precedes the destination node in the datagram's path, the datagram is re-broadcast. If the node is not in the datagram's path, the datagram is discarded. Appendix D illustrates how an Expert System decides to handle a datagram.

3 Testing and Simulation

The Expert System is tested for compliance of each rule. The prototype cannot function in a true protocol stack in a real system so a test suite was designed to verify the correctness of system operation. The agent can take predefined datagrams or segments and conclude appropriate responses or learn specified information. Each scenario illustrated by the decision trees in appendices C and D is tested. For rules testing, the testing functions operate independently of each other. That way, a failure in any one rule will not affect the operation of a function intended to test another rule. Each test starts with a clean Expert System/agent and information is learned from simulated transmissions. The response datagram or segment is inspected to confirm an appropriate response was made. In some cases, the agent is fed simulation information to populate internal data structures. In these cases, the test cases inspect the agent's data structures to verify that the correct information is learned.

A complex test system simulates the protocol in a network environment with multiple nodes. Many components of a real network are modeled in Java to allow different agents to interact. There is a network object which hides certain responsibilities that are otherwise not simulated for the purposes of this research. The network is also charged with the creation of nodes that are assigned a coordinate and a battery. Each coordinate is assumed to exist in a two dimensional environment. Each battery contains information about its maximum capacity and its current level of charge, which is the same for all nodes in the simulation. Nodes are capable of receiving a segment from a user, and transmitting and receiving datagrams with the network. The network determines which nodes will receive a datagram from a transmitting node given the specific distances between devices and the transmission power level. Datagrams that are communicated between nodes are encapsulated, by the Node, in a Frame object. Appendix E illustrates all custom aspects of the Java framework necessary for the simulation.

Due to time constraints, the implementation of the system could only be tested for correctness of operation. Studies involving the performance of the protocol compared to normal DSR were not undertaken.

4 Design Choices

Any project of sufficient scope and planning requires an evaluation of mutually-exclusive design choices. This section justifies the design choices of this protocol, including its use of an expert system.

4.1 Expert Systems

If the routing process could be modeled as a set of rules, an expert system would be viable for a network protocol. Given a set of rules, the system would sense or receive incoming data, follow the rules and conclude what to do with it. This conclusion process is large, in part, because the system must learn information about other devices before it can make an informed decision about the data. The learning process is orchestrated by the use of route requests, route replies and power-metric piggy-backing. The agent inspects all types of datagrams to conclude the topology of the network and status of nodes in it. Defining rules that would allow an expert system to perform the task of routing also requires the support of the encompassing node.

Networking protocols, especially, can benefit from the ability to evaluate multiple conditions in real-time. Most real-world network protocols have a host of data flags that specify either different or additional processing that must occur. An expert system allows each of these special cases to be codified as their own independent rule and added as needed.

The inference engine itself is a fairly efficient piece of software that allows rules to be quickly evaluated and accessed when condition criteria are met. This allows a developer to worry less about evaluation time and more about the correctness of rule operation. As efficient as they are, expert systems are a generalized system that cannot guarantee performance across all domains. In cases where the performance is not adequate for the production system, they can still be used for prototyping software. This is especially the case when it isn't clear how all operations fit together or when only a subset of operations has been finalized since rules can be added and removed easily.

Expert systems allow for mutually exclusive development and easy extensibility. Rules can be created by multiple individuals and added independently to the inference engine for testing. Traditional software development might require extensive refactoring or coordination to mimic this ability. This ability allows for easy extensibility as well.

Rules can be added to the inference engine as additional functionality is needed or deleted as previous functionality is no longer required. The use of an expert system allows these rules to be easily added to the system without major code alterations. The new rules can be set to operate at a higher "salience" than older rules, meaning that they will operate before rules that were added earlier. This allows new rules to either overshadow old rules entirely or for them to augment the old rules' functionality. Traditional systems might require extensive code modifications to achieve the same effect.

Also of benefit is the use of the expert system shell. These rules can be created and tested immediately when the system is in interactive mode. This allows the rules to be created on-the-fly and inserted during actual system operation without the need to halt the system before-hand.

The use of expert systems for the design of protocols on the network stack is not unprecedented. A Media Access Control protocol with Carrier Sense Multiple Access has previously been designed and described [6]. However, it is not known whether such protocols are in active use outside of the lab.

4.1.1 Benefits of JESS

The development of an expert system in the prototype was streamlined thanks to the use of a JESS. JESS is specialized for rapid development of expert systems. The research is not focused on how to develop an expert system but how to use one in a networking protocol. JESS is has been tested against and compares favorably to CLIPS (another expert systems shell) when scaled over large data sets [7]. Networking in general often encompasses large sets of data, namely tables of node ids and routing information. While the rule set of the prototype is not as large as some of the rule-sets used in industry, the associated data is and JESS may offer performance advantages in this aspect.

JESS has additional benefits by virtue of being implemented in Java. JESS can run on any system which can run a Java Virtual Machine (JVM)—which includes many stripped-down Unix/Linux variants. As such, it is theoretically possible for the devices of certain sensor networks to run a version of the JESS rule-set. The extent for which this is the case is unknown.

More pragmatically, JESS was chosen for use because of the authors' familiarity and extensive experience with the Java programming language. JESS allows a developer to easily extend the language's capabilities through the use of Java methods and classes. This proved useful as it allowed for the development of complex algorithms that may otherwise have been more difficult to implement due to the limited range of in-built functions and data-types in JESS proper.

4.1.2 Drawbacks of JESS

JESS is a convenient tool to use when developing expert systems in Java. The system affords all of the functionality needed by the prototype to accomplish the task of networking. One of the drawbacks to using JESS includes its dependency on Java. This alone stands as the largest road-block to producing a field usable prototype. The ability of run a JVM is not a universal property of sensor network devices.

Additionally, there is concern about the performance of Java in real-world environments. Java has historically been a resource-intensive programming language/environment which may not perform well in time-critical situations (i.e. routing decisions). Since a parallel development of the prototype in a comparable level language such as C was not performed, no testing exists to quantitatively defend JESS’s capabilities in terms of performance.

4.2 Battery Representation

In order for the path selection process to choose a path effectively, there must be a method for comparing each node's battery**.** The research assumes that all nodes have the same initial battery capacity, which is clearly untrue in heterogeneous device networks. However, this assumption allows a simple mapping of battery level and capacity to a battery metric value. The battery metric is calculated by dividing the battery level by the battery capacity. For example, a battery with a current level of 4 and a capacity of 10 would result in a 40 percent ratio. Using figure 4.2, a ratio of 40 percent corresponds to a static battery metric of 4. While a linear function could be used to conclude a battery metric, a step function allows for more tune-ability in terms of choosing nodes in a general range with a certain frequency. If nodes with batteries below 25% are assigned higher than linear metric values, the path cost function would be less likely chose a path with a so-called suffering node. In essence, nodes that are almost out of power could be ignored. The current battery metric association function still retains a semi-linear mapping; however, optimization of this algorithm would be an area to further improve the selection process of a best path. The prototype uses the provided table for mapping battery level and capacity to a metric. Additional exploration would be needed when considering nodes that have batteries of different capacities.

Battery Ratio (Level / Capacity)	Battery Metric
100% - 90%	1
89% - 70%	2
69% - 50%	3
49% - 25%	4
24% - 0%	5

Figure 4.2 Evaluating Battery Metrics

4.3 Piggy-Backing Information

All protocols require certain information to be stored in data object to fulfill its function. In the case of network routing protocols, information is stored in datagrams to aid with control evaluation at each node. Fields such as a source and destination address define some of the minimum requirements. In order to support additional capabilities, the prototype system modifies datagrams by appending protocol specific information to the minimum datagram requirements.

This additional information reduces the maximum storage capacity of each datagram, in terms of payload size, but offers the ability to conduct advanced operations. Problematically, this can cause additional datagram fragmentation as the path size grows with each hop. This reduces the throughput of the network as traffic becomes dominated with control information over data content. It also increases the energy cost associated with transmitting this additional control information. The assumption is that most large mesh networks are made up of devices that can communicate over comparatively short paths and that the decrease in throughput for an individual datagram is made up for by the gains from increased quantity of datagrams being transmitted as a result of lengthened network longevity.

4.4 Distance and Transmission Power

After developing a method for representing the current state of a node's battery, path selection appeared to be less complex than initially stated. When provided with multiple paths and the associated metrics, the best path was represented by the path containing a subset of nodes with the lowest cumulative battery metric. In practice, this method tends to favor fewer hops that may use longer transmission distances. In terms of reliability of this approach, the shortest path may not represent the best solution if the battery level at each intermediate node is low. In contrast, the path with the lowest cumulative battery metric may incur unfair congestion. Obviously, some additional knowledge is needed to optimize path selection. In an attempt to trade off the latency advantage of the shortest path with the congestion issues of the lowest cumulative battery metric, transmissions must incur an additional penalty. For the purposes of the research, the transmission cost is defined as the distance between two nodes but is the same as the varying energy requirement needed to send data from a source to a destination node. Most modern wireless transmitters transmit data at a power level correlating to the distance of transmission. For this reason, the transmission cost is represented by a distance between nodes and the network determines if two nodes fall within the communication radius of each other. The mathematical function that relates battery metrics and transmission costs was covered in section 2.3.1.

The current implementation of distance discovery is abstracted out to the “data-link” layer of the simulation environment. The way this is explained in Section 2.2.1 is that devices can detect the signal strength of a transmission and hence can infer the distance. It is possible that this is a non-viable way of determining distance so an alternative method is explored here.

The idea is that the data link layer can determine the distance between two devices through a series of frame broadcasts at varying transmission strengths. The transmission strength is gradually increased until a response is received from the destination device. Thus, the data-link layer can infer that the last transmission strength was the minimal amount required for transmission to that hop. The current version of the simulation environment does not actually model this action in full since it is somewhat outside the scope of the network layer's implementation. Instead, the simulation provides the side-effects of this process: the distance is learned and the power required for this process is deducted from the device’s battery. Ideally, testing would have determined whether the increased cost of discovering these distances (due to having to perform multiple power-intensive transmissions) would have been offset with the savings associated with being able to communicate with minimum power during subsequent communication sessions.

As it stands, there is only speculation that this will result in better battery performance. It is quite possible that this is a needless expense that will not result in power-savings, and may even result in substantial power losses, in the case of networks with highly mobile devices.

5 Conclusion

In conclusion, the prototype has demonstrated that DSR can be modeled with an expert system and energy awareness can be performed by piggy-backing additional information onto datagrams. By the end of the research, many assumptions had to be made to allow for the effective demonstration of the protocol. The research presented many additional opportunities for the use of Artificial Intelligence in network environments. Some advantages of the system in the right environment would be the ability to add filters to the system defining simple rules. Rules could be propagated from a command node on a specialized datagram to all nodes to dynamically modify the protocol. The use of an expert system comes with a resource cost that, in the future, may be justifiable to provide for the dynamic nature of AI learning. Additional research in the field of routing would likely demonstrate other weaknesses of traditional protocols that may be addressable by using an expert system.

Most of the assumptions made for the prototype would have to be addressed if further development would be made. Redevelopment of the system in a historically more resource-frugal language like C may enable the system to run on field devices. In the authors' opinion, Artificial Intelligence is more memory and processing intensive than fixed table based routing protocols but offer many benefits when the primary objective is not speed. There is much evidence to support the conjecture that future network protocols will be developed with a form of artificial intelligence. Network routing occurs in a highly dynamic environment which may require constant adjustment and tuning. An artificial intelligence which can monitor and learn about the network situation over time may be warranted in these situations.

Works Cited

1. Maskovyak, A.G., “Energy Aware Routing Protocols Including EARP”, 4005-740 Data Communications and Networking I, RIT, July 2008.

2. Toh, C.-K., “Maximum Battery Life routing to Support Ubiquitous Mobile Computing in Wireless Ad Hoc Networks”, IEEE Communication Magazine, June 2001.

3. Toh, C.-K., Cobb, H., Scott, D.A. “Performance Evaluation of Battery-life-Aware Routing Schemes for Wireless Ad Hoc Networks”, IEEE International Conference on Communications, 2001.

4. Hyun Kyung Cho, Eun Seok Kim, and Dae-Wook Kang, “A Load-Balancing Control Method Considering Energy Consumption Rate in Ad-Hoc Networks”, Networking and Mobile Computing, pp. 324-333, Springer Berlin / Heidelberg, 2005.

5. Anand Prabhu Subramanian, A.J. Anto, Janani Vasudevan, and P. Narayanasamy, “Multipath Power Sensitive Routing Protocol for Mobile Ad Hoc Networks”, Wireless On-Demand Network Systems, pp. 84-89, Springer Berlin / Heidelberg, 2004.

6. Maule, R. A. and Kandel, A. 1985. A model for an expert system for medium access control in a local area network. *Inf. Sci.* 37, 1-3 (Dec. 1985), 39-83.

7. Friedman-Hill, Earnest. "Jess: Implementing a High Performance Symbolic Reasoning Engine In Java." Created March 6, 2008. Retrieved July 13, 2008 from http://aaaprod.gsfc.nasa.gov/teas/Jess/JessUMBC/index.htm

Appendix A Networking data abstractions

Listed below are abstract representations of various data storage terms referenced by this research. The size of each field does not correspond to the field's storage capacity. In practice, each object often contains additional fields not listed in the illustrations and often a single message

Message

Strings, numbers etc.

Above: A message is used to encapsulate and transport a message from one application to the next. A message can contain any combination of values.

Segment

Destination	Message

Above: A segment is used to encapsulate and transport necessary information for moving data from one host to another. A datagram encapsulates a message sent from the application layer.

Datagram

Type	Source	Destination	Segment	Path	Transmission Cost(s)	Battery Metric(s)

Above: A datagram is used to encapsulate and transport necessary information for the network layer to perform routing. A datagram encapsulates a segment sent from the transport layer.

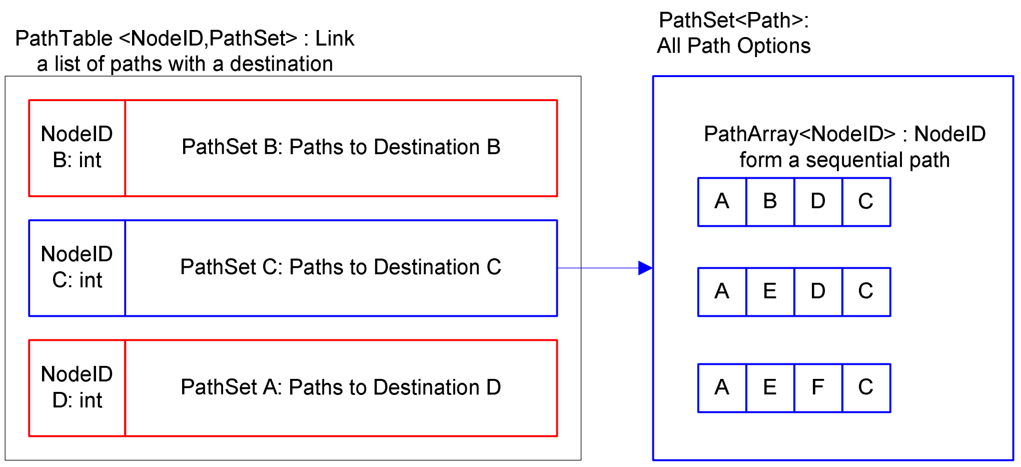
Frame

Source MAC	Destination MAC	Datagram

Above: A frame is used to encapsulate and transport necessary information for the data-link layer to perform device to device transmission. A frame encapsulates a datagram sent from the network layer.

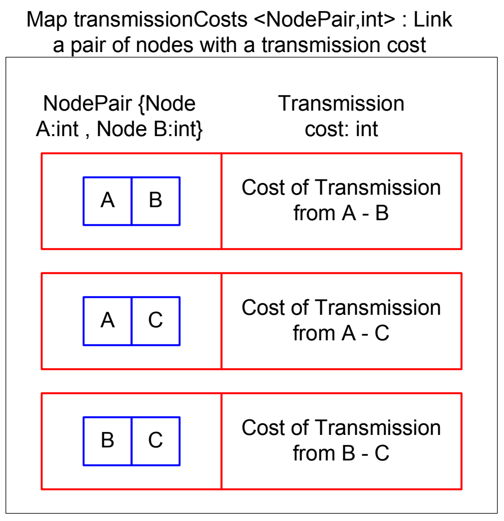
Appendix B   Path and metric data structures

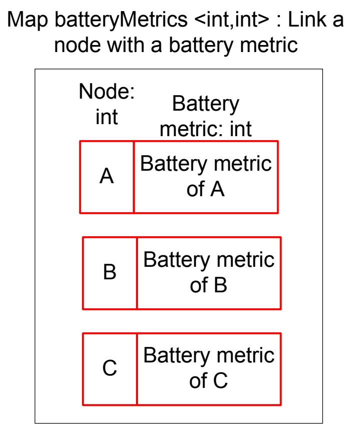
Path evaluation requires various data structures to organize paths for efficient access.


PathTable <NodeID,PathSet> : Link
a list of paths with a destination
NodeID B: int
PathSet B: Paths to Destination B
NodeID C: int
PathSet C: Paths to Destination C
NodeID D: int
PathSet A: Paths to Destination D
PathSet<Path>:
All Path Options
PathArray<NodeID> : NodeID
form a sequential path
A B D C
A E D C
A E F C

Above: The path table is the primary data structure used for storing paths. The path table is essentially a hash map with the key being a destination node and the related object a list of paths to that destination known as a path set. The path set structure organizes paths a node has learned. The set does not allow duplicate paths.

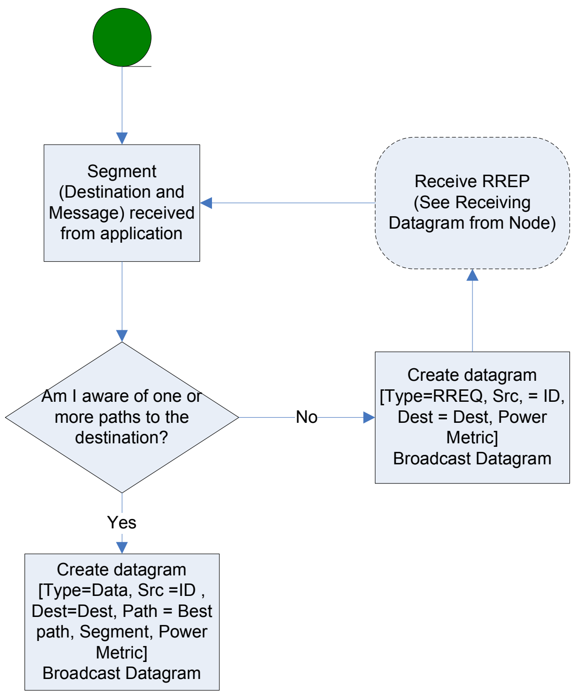
Below Left: A simple structure for storing the related transmission cost from one node to another once received. Aids with path cost calculation.


Map transmissionCosts <NodePair,int> : Link
a pair of nodes with a transmission cost
NodePair {Node A:int , Node B:int}
Transmission cost: int
A B
Cost of Transmission from A - B
A C
Cost of Transmission from A - C
B C
Cost of Transmission from B - C



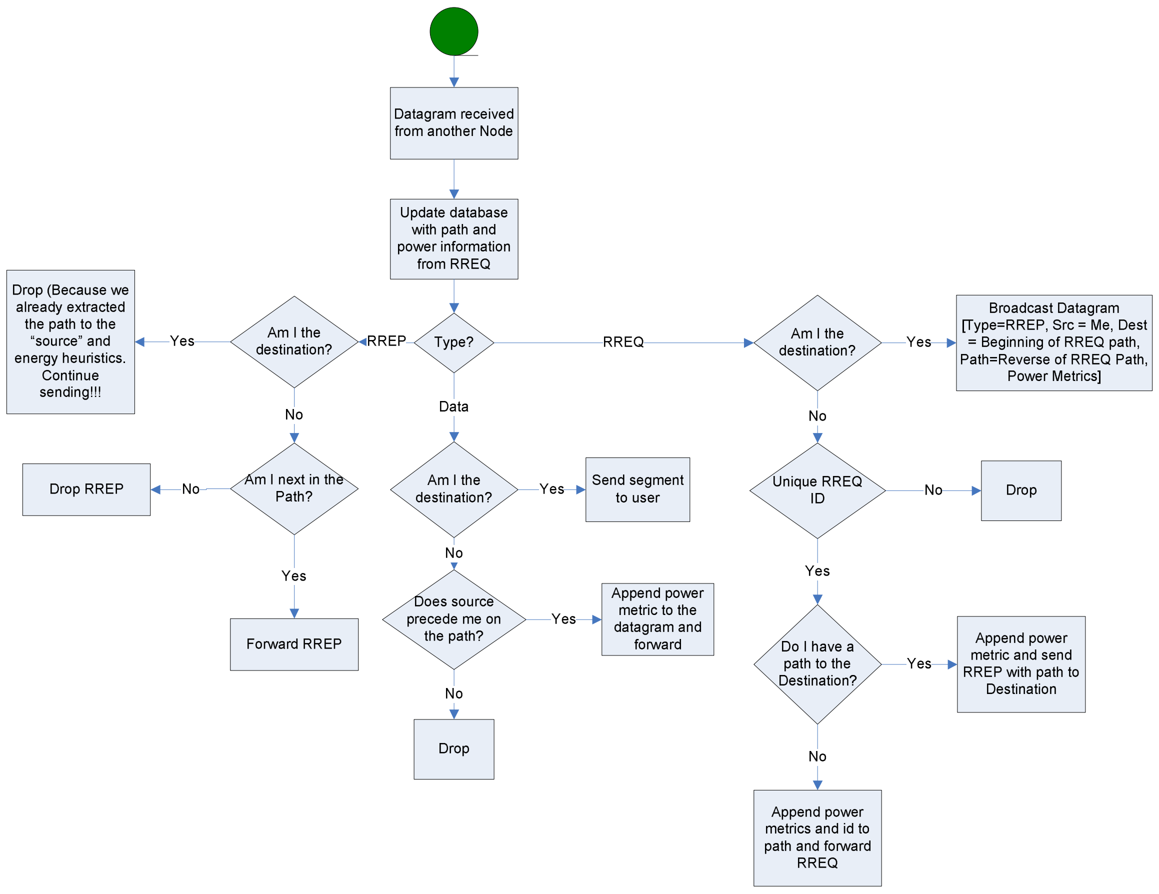
Above Right: A structure for storing the most up-to-date battery metric for encountered nodes. Aids in path cost calculation.

Appendix C Decision tree for segment handling


Segment (Destination and Message) received from application
Receive RREP (See Receiving Datagram from Node)
Am I aware of one or more paths to the destination?
No
Create datagram [Type=RREQ, Src, = ID, Dest = Dest, Power Metric] Broadcast Datagram
Yes
Create datagram [Type=Data, Src =ID , Dest=Dest, Path = Best path, Segment, Power Metric] Broadcast Datagram

Above: This decision tree covers how a segment is handled by the Expert System.

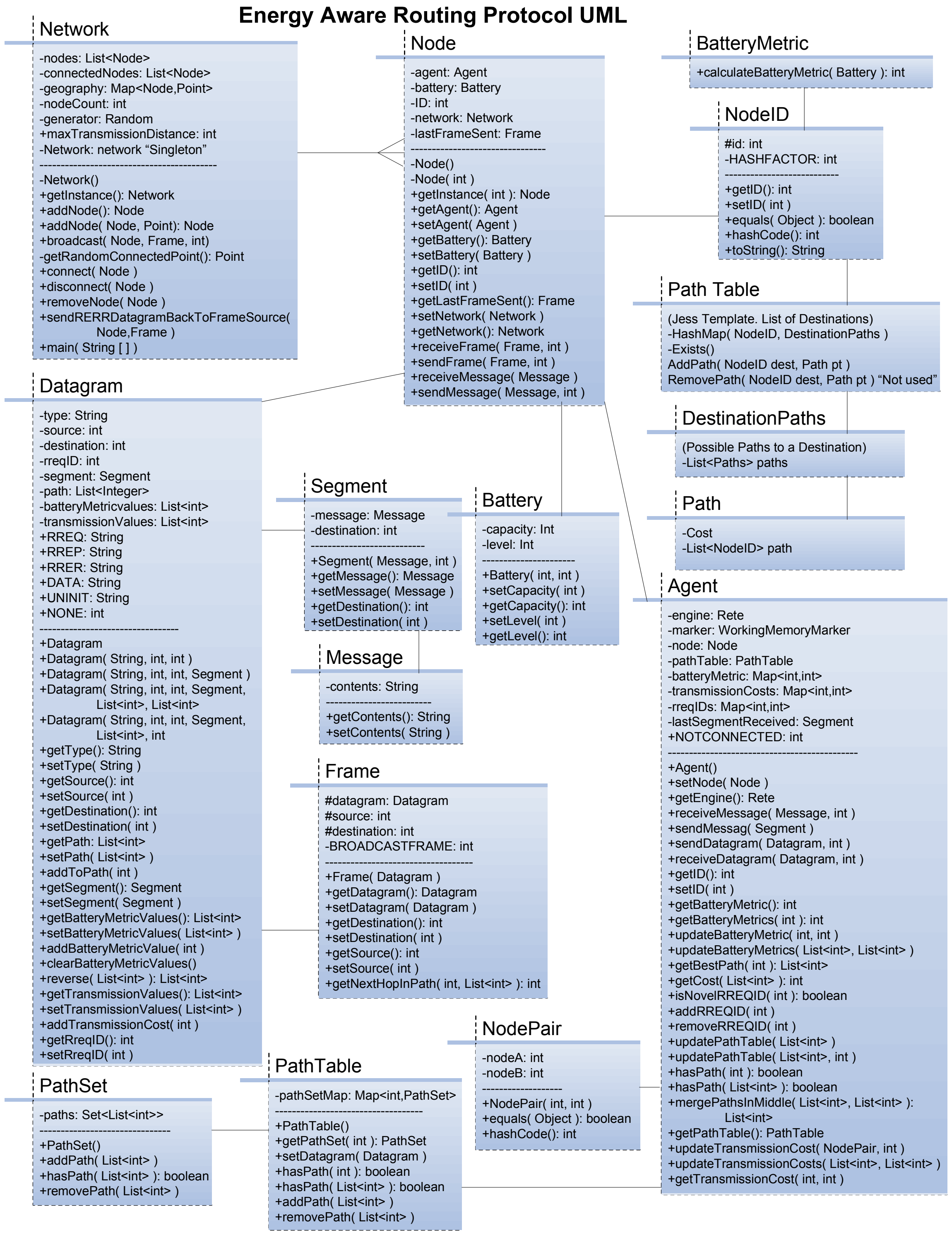
Appendix D Decision tree for datagram handling



Datagram received from another Node
Update database with path and power information from RREQ
Type?
RREP
RREQ
Data
Am I the destination?
Yes
Drop (Because we already extracted the path to the "source" and energy heuristics. Continue sending!!!
No
Am I next in the Path?
No
Drop RREP
Yes
Forward RREP
Am I the destination?
Yes
Send segment to user
No
Does source precede me on the path?
Yes
Append power metric to the datagram and forward
No
Drop
Am I the destination?
Yes
Broadcast Datagram [Type=RREP, Src = Me, Dest = Beginning of RREQ path, Path=Reverse of RREQ Path, Power Metrics]
No
Unique RREQ ID
No
Drop
Yes
Do I have a path to the Destination?
Yes
Append power metric and send RREP with path to Destination
No
Append power metrics and id to path and forward RREQ

Above: This decision tree covers how a datagram is handled by the Expert System.

Appendix E UML Diagram of Java project code


Energy Aware Routing Protocol UML
Network
-nodes: List<Node>
-connectedNodes: List<Node>
-geography: Map<Node,Point>
-nodeCount: int
-generator: Random
+maxTransmissionDistance: int
-Network: network "Singleton"
-Network()
+getInstance(): Network
+addNode(): Node
+addNode(Node, Point): Node
+broadcast(Node, Frame, int)
-getRandomConnectedPoint(): Point
+connect(Node)
+disconnect(Node)
+removeNode(Node)
+sendRERRDatagramBackToFrameSource(Node,Frame)
+main(String [ ])
Node
-agent: Agent
-battery: Battery
-ID: int
-network: Network
-lastFrameSent: Frame
-Node()
-Node(int)
+getInstance(int): Node
+getAgent(): Agent
+setAgent(Agent)
+getBattery(): Battery
+setBattery(Battery)
+getID(): int
+setID(int)
+getLastFrameSent(): Frame
+setNetwork(Network)
+getNetwork(): Network
+receiveFrame(Frame, int)
+sendFrame(Frame, int)
+receiveMessage(Message)
+sendMessage(Message, int)
BatteryMetric
+calculateBatteryMetric(Battery): int
NodeID
#id: int
-HASHFACTOR: int
+getID(): int
+setID(int)
+equals(Object): boolean
+hashCode(): int
+toString(): String
Path Table
(Jess Template. List of Destinations)
-HashMap(NodeID, DestinationPaths)
-Exists()
AddPath(NodeID dest, Path pt)
RemovePath(NodeID dest, Path pt) "Not used"
DestinationPaths
(Possible Paths to a Destination)
-List<Paths> paths
Path
-Cost
-List<NodeID> path
Datagram
-type: String
-source: int
-destination: int
-rreqID: int
-segment: Segment
-path: List<Integer>
-batteryMetricvalues: List<int>
-transmissionValues: List<int>
+RREQ: String
+RREP: String
+RRER: String
+DATA: String
+UNINIT: String
+NONE: int
+Datagram
+Datagram(String, int, int)
+Datagram(String, int, int, Segment)
+Datagram(String, int, int, Segment, List<int>, List<int>
+Datagram(String, int, int, Segment, List<int>, int
+getType(): String
+setType(String)
+getSource(): int
+setSource(int)
+getDestination(): int
+setDestination(int)
+getPath: List<int>
+setPath(List<int>)
+addToPath(int)
+getSegment(): Segment
+setSegment(Segment)
+getBatteryMetricValues(): List<int>
+setBatteryMetricValues(List<int>)
+addBatteryMetricValue(int)
+clearBatteryMetricValues()
+reverse(List<int>): List<int>
+getTransmissionValues(): List<int>
+setTransmissionValues(List<int>)
+addTransmissionCost(int)
+getRreqID(): int
+setRreqID(int)
Segment
-message: Message
-destination: int
+Segment(Message, int)
+getMessage(): Message
+setMessage(Message)
+getDestination(): int
+setDestination(int)
Battery
-capacity: Int
-level: Int
+Battery(int, int)
+setCapacity(int)
+getCapacity(): int
+setLevel(int)
+getLevel(): int
Message
-contents: String
+getContents(): String
+setContents(String)
Frame
#datagram: Datagram
#source: int
#destination: int
-BROADCASTFRAME: int
+Frame(Datagram)
+getDatagram(): Datagram
+setDatagram(Datagram)
+getDestination(): int
+setDestination(int)
+getSource(): int
+setSource(int)
+getNextHopInPath(int, List<int>): int
Agent
-engine: Rete
-marker: WorkingMemoryMarker
-node: Node
-pathTable: PathTable
-batteryMetric: Map<int,int>
-transmissionCosts: Map<int,int>
-rreqIDs: Map<int,int>
-lastSegmentReceived: Segment
+NOTCONNECTED: int
+Agent()
+setNode(Node)
+getEngine(): Rete
+receiveMessage(Message, int)
+sendMessag(Segment)
+sendDatagram(Datagram, int)
+receiveDatagram(Datagram, int)
+getID(): int
+setID(int)
+getBatteryMetric(): int
+getBatteryMetrics(int): int
+updateBatteryMetric(int, int)
+updateBatteryMetrics(List<int>, List<int>)
+getBestPath(int): List<int>
+getCost(List<int>): int
+isNovelRREQID(int): boolean
+addRREQID(int)
+removeRREQID(int)
+updatePathTable(List<int>)
+updatePathTable(List<int>, int)
+hasPath(int): boolean
+hasPath(List<int>): boolean
+mergePathsInMiddle(List<int>, List<int>): List<int>
+getPathTable(): PathTable
+updateTransmissionCost(NodePair, int)
+updateTransmissionCosts(List<int>, List<int>)
+getTransmissionCost(int, int)
NodePair
-nodeA: int
-nodeB: int
+NodePair(int, int)
+equals(Object): boolean
+hashCode(): int
PathTable
-pathSetMap: Map<int,PathSet>
+PathTable()
+getPathSet(int): PathSet
+setDatagram(Datagram)
+hasPath(int): boolean
+hasPath(List<int>): boolean
+addPath(List<int>)
+removePath(List<int>)
PathSet
-paths: Set<List<int>>
+PathSet()
+addPath(List<int>)
+hasPath(List<int>): boolean
+removePath(List<int>)