Deev Patel
CPE 400 (Fall 2020)
12/07/20
Final Project Report

Energy Conservation Routing Protocol

Table of Contents

Abstract

This paper proposes a routing algorithm for dynamic networks where nodes are energy constrained. The proposed Energy Conservation Routing (ECR) protocol takes concepts from the Dynamic Source Routing (DSR) protocol and adds novel attributes to create a comprehensive specification that is able to achieve node longevity with minimal latency impacts. The paper starts by discussing the ECR protocol's problem domain and goals. This is followed by a detailed description of protocol's functionality and novelty. Notably, the ECR protocol is a fully complete specification that includes error handling. Finally, the ECR protocol is implemented and its performance is analyzed on four networks. The analysis is done with the help of a simulation program written in Python. The simulation code and its associated documentation are provided alongside this paper.

Problem Definition

Overview

With the growing availability of compact battery-powered electronic devices, it is not hard to imagine a network of energy constrained devices. For example, suppose we wanted to deploy a bunch of battery-powered sensors across a large area and have them wirelessly communicate with each other. We could model the sensors as nodes or routers in a computer network and use a dynamic routing algorithm like DSR to route packets between them. However, for a relatively densely connected network with multiple paths between a given source and destination node pair, traditional routing algorithms may not be the most efficient way to ensure that nodes stay online for as long as possible. This is because an algorithm like DSR cannot make use of the battery level of nodes to ensure that packets are routed in a way that avoids draining nodes. So, we create a new ECR protocol that makes uses of a node's knowledge of its battery level to increase network longevity. The proposed algorithm seeks to route packets in a way that ensures nodes stay online while minimizing the impact to end-to-end network latency.

Modeling Details

We can model the dynamic sensor network as a tradition computer network with some added constraints. Each sensor is represented as a node. Each node has a communication link to every sensor it can communicate with. Additionally, each node has a battery level. The battery level is negatively affected by the node's day-to-day activities (such as collecting data) and the number of packets the node is required to forward by the network. Eventually, when the battery level reaches zero, the node will shut down and no longer function. The battery level can also increase up to a certain limit through charging.

Modeling Details: Simulation Environment

In the simulation environment, we assume the battery level, b, to continuously decrease at a constant rate, Δb_c, and a variable rate, Δb_v. Δb_c is negative and represents the energy requirements for a node to stay online. This includes all the continuous tasks the node must perform, like maintaining communication links to its neighbors by sending periodic updates as defined by ECR protocol. While there might be some variability in this value since different nodes have a different number of links to maintain, the model assumes a constant value for simplicity. In all likelihood, this value would only change slightly between different nodes. The value for Δb_v is assumed to be a negative value proportional to the number of packets the node forwards over a given time. So, the node's battery goes down for every packet the network requires the node to forward. The simulation model does not account for battery level changes due to charging or day-to-day actives outside of the network, like collecting data. However, the ECR protocol should be able handle this as it allows each node to estimate its time-to-live based on a dynamic set of criteria. Figure 1 shows an example simulation network.

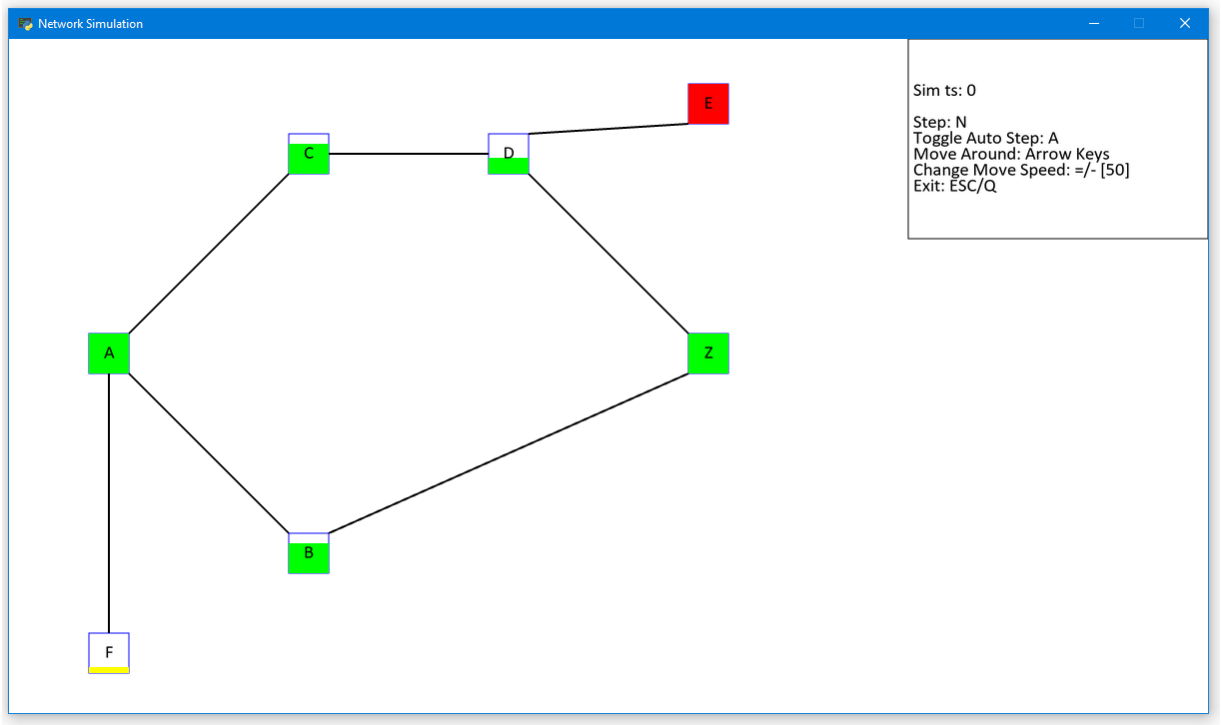


Figure 1: Example simulation network. Each node has a battery level. Yellow nodes are low on battery. Red nodes have no battery and are unresponsive. So, the link *D-E* cannot be used for sending packets However, the link *A-F* can still be used but would likely by avoided by ECR protocol if possible. In the top right, we can see simulation quantities like the elapsed simulation time.

The simulation network assumes that it takes one simulation timestep (which can be thought of as one millisecond) for a packet to travel across each link, regardless of link distance. So, if a node sends a packet at $t_{sim} = 0$, it will arrive at the node on the opposite side of the link at $t_{sim} = 1$. The simulation network assumes no other variable delays in the network like queuing delay at the nodes. This assumption is reasonable because the network is not expected to have enough bandwidth where other limiting factors become relevant. Additionally, the ECR protocol should be able to handle variable amounts of delay and there is no simple way to model realistic variable delays without construction and deployment of a real network.

Protocol Functionality

Overview

At a high level, the ECR protocol requires each node the compute an estimate of its nodal last-alive-time, lat_n, based on an exponential moving average of recent battery levels and number of packets processed. The last-alive-time represents that last time, in the future, that a node thinks it will be alive. The estimate is given to immediate neighbors during link maintenance. For routing, each node maintains a Routing Multi-Table (RMT) containing entries with four columns: (1) destination node, (2) next-hop node, (3) the estimated route last-alive-time, lat_r, and (4) the discount factor, d_f, associated with the lat_r. The d_f is essentially the number of hops away the lat_r originates from. This can be thought of as the number of hops to the energy limited node. Table 1 shows an example RMT for node *A* from Figure 1.

Destination	Next-Hop	lat_r	d_f
B	B	7	1
C	C	7	1
D	C	4	2
E	-	-	-
F	F	2	1
Z	B	7	1
Z	C	4	2

Table 1: Example RMT for node *A* for the network shown in Figure 1. The lat_r values are for reference only and would depend on specific parameters of the ECR protocol. Note that there are two entries for destination *Z.* The route through *C* has an earlier lat_r because node *D* has relatively low battery. Additionally, the d_f for the route through C is 2 because the energy bottleneck happens at *D*, two hops away. So, if *A* wants to send packets to *Z,* the ECR protocol will initially route packets through *B,* but then switch to routing through *C* once *B* loses enough energy from the added packet flow.

The value of lat_r is basically the earliest distance discounted lat_n along the route. When a node is required to send a message to a destination that is in the RMT, the node simply picks the entry in the RMT with the matching destination and latest lat_r. The RMT is built through a serious of route construction messages and maintained through a series of route update messages. The ECR protocol also has error messages that allow for the removal of RMT entries to dead links and well-defined behavior for unreachable nodes.

Message Types

The ECR protocol defines five message types: (1) route discovery message, (2) route response message, (3) route packet message, (4) route update message, and (5) route error message. The discover and response messages extend and modify functionality implemented in various DSR message types. The update message is an entirely novel aspect of the ECR protocol that allows iterative dynamic updates to the RMT based on network changes.

Message Types: Route Discovery (RD) Message

The RD message is used by the ECR protocol when a node does not have a needed entry in its RMT. The message is sent to all nearby links and subsequently forwarded until the *dst desired* node is reached. Each intermediate node that forwards the RD message appends itself to the variable *route* field. The RD message is very similar to the RREQ message in DSR. Figure 2 shows the format of the RD message.

Route Discovery (RD) Message Format

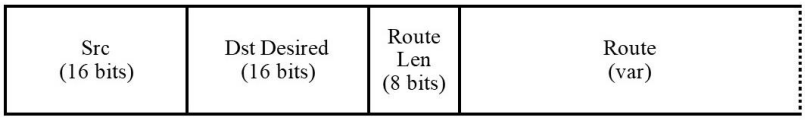


Figure 2: The format of the Route Discover Message.

Message Types: Route Response (RR) Message

The RR messaged is used by the ECR protocol to return a response to the RD message. The message is similar to DSR's RREP message but adds a few novel concepts. The *route src* field is the *src* of the original RD message and *route dest* field is the *dst desired* of the original RD message. Unlike the RD message, the RR message is not sent to all nearby links. Instead, it is selectively sent in the reverse order of the *Route* variable field. This selective forwarding process uses an *ACK* system to ensure that messages are not lost. Each node forwards the RR message to the next node in the *Route*. Before forwarding, each node adds the *route dest* node to its RMT and updates the lat_r and discount factor if necessary. In summary, the RR message is used by nodes to add or update entries in the RMT for future routing. Figure 3 shows the format of the RR message.

Route Response (RR) Message Format

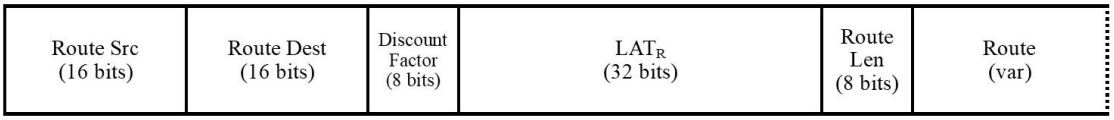


Figure 3: The format of the Route Response Message.

Message Types: Route Packet (RP) Message

The RP message represents a standard packet. When a node wants to send an IP datagram and the destination is in the RMT, the IP datagram is wrapped in a RM messaged and forwarded to the next-hop node. Each node along the route uses the *discount factor* and the *expected* lat_r to check if the route matches its own RMT. If the lat_r or discount factor need to be updated, then an update (RU) message is sent back. If the destination is not found in an intermediate node's RMT, then the node will send out a series of RD messages to look for a route. If a route is still not found, then an error (RE) message will be sent back. Figure 4 shows the format of the RP message.

Route Packet (RM) Message Format

Figure 4: The format of the Route Packet Message.

Message Types: Route Update (RU) Message

The RU message is a completely novel aspect of the ECR protocol that is used to update RMT entries dynamically. If a RP message is being sent and an intermediate node finds that it has updated information on the route, then it will send back an RU message so previous nodes can update their RMT. This allows the ECR protocol to dynamically update to changing conditions and route packets in a way that maximize node uptime. Figure 5 shows the format of the RU message.

Route Update (RU) Message Format

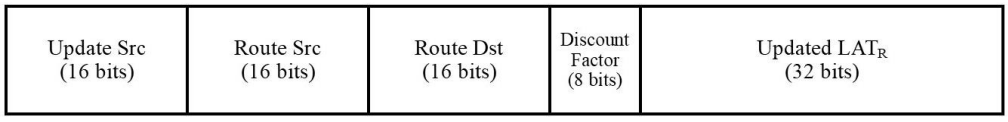


Figure 5: The format of the Route Update Message.

Message Types: Route Error (RE) Message

The RE message is used by the ECR protocol to indicate errors. The *error code* is an integer that can be used to look up the exact error in an error table that all nodes have a copy of. The RE message has an *error route* field that can be used for certain error codes to keep track of the route the RE message takes on its way back. For the initial version of the ECR protocol defined in this paper, the RE message is only really used to indicate that a destination node could not be reached. However, the RE message is defined in such a way that it can be expanded to handle more complex error scenarios in the future. Figure 6 shows the format of the RE message.

Route Error (RE) Message Format

Error Src (16 bits)	Route Src (16 bits)	Route Dst (16 bits)	Error Code (8 bits)	Error Route Len (8 bits)	Error Route (var)

Figure 6: The format of the Route Error Message.

Algorithm

Algorithm: Computing Nodal LAT

At the core of the ECR protocol is the concept of the last-alive-time for each node (lat_n). Each node is individually responsible for computing its nodal lat_n. This value is a 32-bit integer representing the number of timesteps past a common reference time at which a node estimates its battery will be depleted. For the simulation environment, we use a reference time of $t_{sim} = 0$. In general, with a timestep of 1 second, 32-bits can represent approximately 68 years. Thus, the reference time of a network would rarely need to be changed once set.

In terms of implementation, each node can compute its lat_n however it wants. The ECR protocol will work as long as the computed value is relative stable between time iterations and reasonably predictive of when the node will go down. This flexibility allows each node to consider its local conditions in making a good estimate. So, a node can consider the specifics of its battery chemistry and hardware as well battery drainage due to computations independent of the ECR protocol. That being said, a simple uniform computation methodology is provided in this paper as an example. This computation is used for the simulation environment.

From the simulation modeling assumptions discussed earlier, we can define the following equation to compute the nodal last-alive-time.

$$lat_n = t_n + \frac{b}{|\Delta b_c| + |\Delta b_v|} = t_n + \frac{b}{d_c + \hat{p} d_p} \quad \text{(Eq. 1)}$$

Here, t_n is the current time, $0 \leq b \leq 1$ is the current battery level, d_c is the constant drainage factor that represents the battery drainage over a fixed time interval, and d_p is the constant drainage factor that represents the battery drainage associated with forwarding a single packet of data. $\hat{p}$ is the number of packets that is estimated to be sent over the next time interval. For consistency in results, $\hat{p}$ is computed using an exponential moving average over the historical number of packets sent per time interval.

$$\hat{p} = \alpha \hat{p} + (1 - \alpha) p_s \quad \text{(Eq. 2)}$$

Here, $0 \leq \alpha \leq 1$ is a factor representing the weight of historical values and p_s is the number of packets sent in the last time interval. Normally, (Eq. 2) is computed every half a second and (Eq. 1) is computed every second. For the exact values of constants used in the simulation environment, see the associated code.

Algorithm: Maintaining Links

The ECR protocol requires each node to maintain links to its immediate neighbors. This is done by having each node send the lat_n computed by (Eq. 1) to its neighbors at a fixed interval. For the simulation environment, this interval is two simulation time-steps. Note that the energy requirement for computing the lat_n and sending it periodically to neighbors is modeled by the d_c factor in (Eq. 1).

Algorithm: RMT Structure & Discount Factor

As mentioned in the *Algorithm Overview*, each node, *A,* has an RMT that keeps track of cached paths to a given destination node. Recall the RMT has four columns: (1) destination node, (2) next-hop node, (3) route lat_r, and (4) discount factor. Note that *A* does not contain itself in its RMT. Since the table is a multi-table, there can be multiple rows for the same destination node. However, the destination and next-hop together uniquely define a table row. So, each destination and next-hop pair can only have one associated (lat_r, d_f) pair.

As alluded to by the message formats and RMT structure, there is a discount factor, d_f, associated with the last-alive-time of farther away nodes. This discount factor prevents the battery levels of faraway nodes from adversely effecting routing. If at time t_n, *A* has a self-computed nodal lat_n and gets a RR message with an integer discount factor d_f and route last-alive-time lat_r, then (Eq. 3) can be used to compute the (lat_r, d_f) pair for the associated RMT entry.

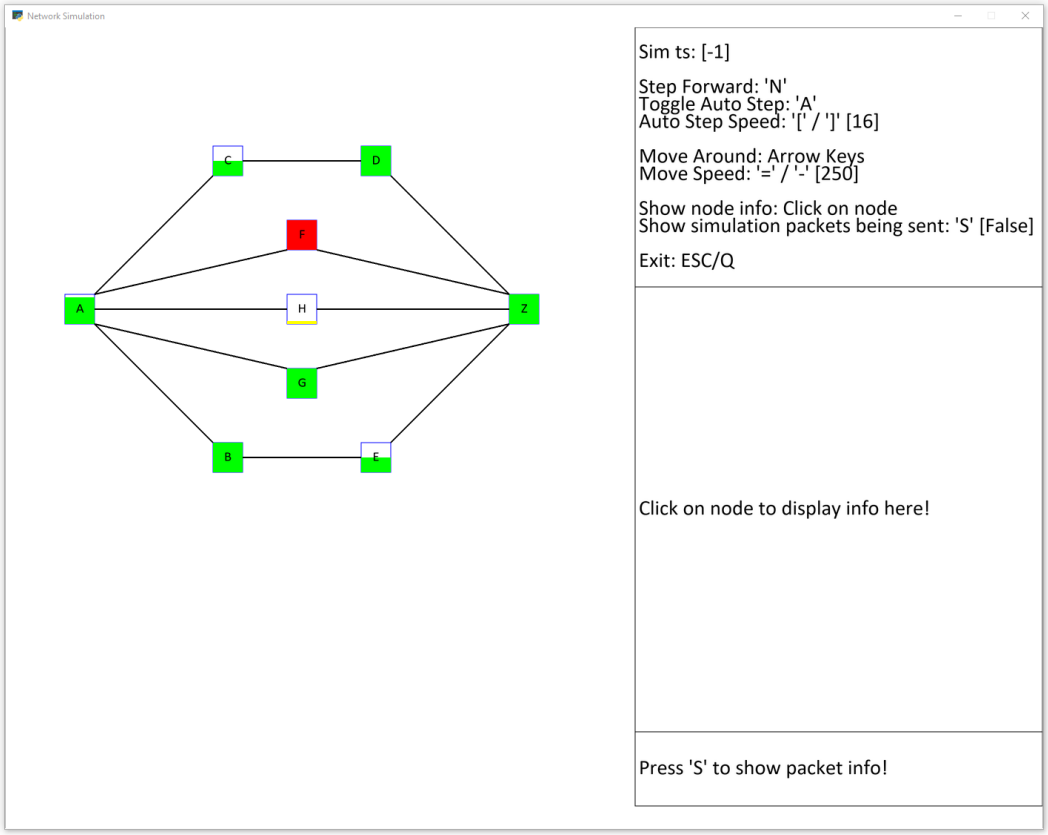
$$lat_r = \min\{lat_n,\ t_n + \max\{0,\ \gamma * (lat_r - t_n)\}\}$$
$$d_f = \begin{cases} 0 & lat_r = lat_n \\ d_f + 1 & \text{otherwise} \end{cases} \qquad \text{(Eq. 3)}$$

For (Eq. 3) we update the lat_r first and then increment the d_f by one if we choose to keep the lat of the message. In essence, d_f measures the number hops away the energy limiting node is. Notice that we apply a discount rate $0 < \gamma \leq 1$ to the time left for the value that comes from the further source. So d_f can be thought of as the number of times a discount rate, γ, has been applied to the lat.

Note that (Eq. 3) assumes that the node has a lat_n estimate. For the simulation environment, each node creates this lat_n estimate according to (Eq. 1). Moreover, each time the lat_n is updated, the node must also update each RMT entry through (Eq. 4) given below. In practice, for efficiency, a node only applies (Eq. 4) to all matching entries when searching for a route to a specific destination.

$$lat_r = \min\{lat_n, lat_r\}$$
$$d_f = \begin{cases} 0 & lat_r = lat_n \\ d_f & \text{otherwise} \end{cases} \qquad \text{(Eq. 4)}$$

Removal of RMT entries for dead links is only done only through error messages for unreachable routes. Note we do not remove routes even if the $lat_r \leq t_n$ after application of (Eq. 4) and/or (Eq. 3). However, we do remove routes for immediate neighbors that fail to send link maintenance messages for a prolonged period. Figure 7 shows an example network and some RMT entries that clarify how the discount factor and (Eq. 3) work.



Nodal Values

Node	Battery Level	lat_n Computed by Node (Eq. 1)
A	0.9	27+90 = 117
B	1.0	27+100 = 127
C	0.5	27+50 = 77
D	1.0	27+100 = 127
E	0.5	27+50 = 77
F	0.0	N/A (dead)
G	1.0	27+100=127
H	0.1	27+10=37
Z	1.0	27+100=127

Select RMT Entries of *A*

Destination	Next-Hop	lat_r	d_f
Z	G	117	0
Z	C	74.5	1
Z	B	72.125	2
Z	H	36.5	1
Z*	F*	20*	1*

*The RMT entry for the route through F is an example of a stale entry that would be removed via error messages if *A* tries routing through it.

Figure 7: A network and example RMT entries for node *A* and destination *Z.* Note that $t_n = 27$ from the top right. For this example, we assume $\hat{p} = 0$ for all nodes and $d_c = 0.01$ for simplicity. The battery levels of each node are also given in the example. We use $\gamma = 0.95$. The ECR protocol would choose to route through G.

Assuming a $\gamma = 0.95$, let us go through and verify each RMT entry in Figure 7 to clarify things.

Node *G* returns a $(lat_r, d_f) = (127, 0)$. So, (Eq. 3) will yield $(lat_r, d_f) = (117, 0)$. This means that A is itself the limiting node for the route through G.

Node *C* returns a $(lat_r, d_f) = (77, 0)$. So, (Eq. 3) will yield $(lat_r, d_f) = (74.5, 1)$. Here we note that the limiting node is *C* and discount its lat_r.

Node *B* returns a $(lat_r, d_f) = (74.5, 1)$. So, (Eq. 3) will yield $(lat_r, d_f) = (72.125, 2)$. Note that *B* returns an already once discounted lat_r because the limiting node is *E.* Thus, the RMT protocol will prefer to route through C instead of B because the limiting node comes earlier. It is better to have the limiting node come earlier because it is easier to respond to dynamic updates, which leads to less overall errors later in the route.

Node *H* returns a $(lat_r, d_f) = (37, 0)$. So, (Eq. 3) will yield $(lat_r, d_f) = (36.5, 1)$.

Node *F* is dead and does not return anything. The value in the RMT table is stale. It is there to show that stale entries remain in the RMT and are only removed by error messages. To save space, we could drop entries that have $lat_r \ll t_n$.

Algorithm: RMT Initial Population

All nodes have an initially empty RMT. However, as neighboring links establish communications and send their lat_r, we can use the rules defined in (Eq. 3) and (Eq. 4) to construct and maintain initial RMT entries. Table 2 shows an example of the initial RMT entries for Node *A* from Figure 7.

Destination	Next-Hop	lat_r	d_f
B	B	117	0
C	C	74.5	1
G	G	117	0
H	H	36.5	1

Table 2: Example of initial RMT table of Node *A* from Figure 7. The values are derived from the constant flow of communication maintenance messages. For initial entries, $d_f = 0$ indicates that the current node, *A,* has a lower expected discounted time-to-live than the neighbor. Conversely, $d_f = 1$ indicates that the current node, *A,* has a higher discounted expected time-to-live than the neighbor.

Algorithm: Populating RMT through RD & RR Messages

If a node *A* needs to route to packets to destination *Z* for which it has no RMT entry, then *A* will try and create the entries. *A* will construct a RD message with itself in the *src* and *route* fields and *Z* in the *dst desired* field. Then, *A* will forward the RD message to all its neighbors. Each neighboring node will append itself to the *route* field and forward the message to its neighbors until the *Z* is reached. To prevent infinite back-and-forth communications between circularly connected nodes, each intermediate node will not forward messages to any nodes already in the *route*. Additionally, each intermediate route will keep a buffer of recently sent RD messages to ensure it does not send extra messages. This process is the same as the DSR protocol's route discovery through RREQ messages.

For each RD message that *Z* receives, it will send a RR message back to *A* through the specific route in the RD message. This is where the ECR protocol differs from the DSR protocol. *Z* will put $(lat_r, d_f) = (lat_n, 0)$ in the RR message. Then, each intermediate node will update its own RMT and the (lat_r, d_f) pair in the

message the according to (Eq. 3). So, by the time *A* receives the RR message, *Z* will be in the RMT of every intermediate node.

Since the ECR protocol may send updates backwards along the route, when *Z* receives a set of RD messages from *A* with itself as the destination, it will broadcast more RD messages looking for a route back to *A.* Sending these messages will allow each intermediate node to store a route back to *A*. This will be needed for sending route updates back to *A* in the future.

Once *A* receives a single RR, it can start routing packets. As more RRs are returned with different routes, a more optimal route may be found. However, if *A* does not get a RR message after a certain timeout period, then that means there is no possible way to get to *Z* and an error will be returned to the application layer. Additionally, if *A* finds the optimal route to have a lat_r that is close to the current time (indicating the route may die), it will send out another batch RD messages to look for routes that could have been added since the last exploration time.

Algorithm: Sending Messages

Now that we have a way to create RMT entries for any destination node, we can start routing packets and defining the framework necessary to perform dynamic updates to stale RMT entries. If node *A* wants to send a packet to destination *Z* for which it has one or more RMT entries, then *A* will wrap the packet in a RP message. *A* will pick the RMT entry that has the largest lat_r as that is the route that is expected to keep the network online for as long as possible. If there are multiple entries with the same lat_r, then *A* will pick the optimal next-hop by looking for the next-hop with the largest lat_r. If this is still not possible, then A will choose randomly between the matches. Based on the (lat_r, d_f) pair of the appropriate RMT entry, each intermediate node, including *A*, will populate the values in the RP message through (Eq. 5).

$$lat_r = \begin{cases} t_n + \dfrac{lat_r - t_n}{\gamma} & d_f > 0 \\ lat_r & d_f = 0 \end{cases}$$

$$d_f = \begin{cases} d_f - 1 & d_f > 0 \\ 0 & d_f = 0 \end{cases} \qquad \text{(Eq. 5)}$$

Note that each intermediate node will apply (Eq. 4) to its own RMT entry before applying (Eq. 5) to update the RP message.

Algorithm: Updating Route LAT

The ECR protocol can dynamically adapt to changing conditions by updating RMT entries that are stale. The update process is triggered whenever a node that is forwarding a RP message finds an inconsistency in the update process described by (Eq. 5). Inconsistencies occur when the original $(\widehat{lat_r}, \widehat{d_f})$ pair of RP message does not match the updated (lat_r, d_f) pair produced by (Eq. 5). More specifically, if $\widehat{d_f} \neq d_f$ (indicating the limiting node has changed) or $lat_r < \widehat{lat_r}$ (indicating the limiting node no longer as much energy has previous estimated), then the forwarding node will return a RU message back to source through all possible known routes. The RU message will contain the (lat_r, d_f) pair from the RMT table. Note that we return the (lat_r, d_f) pair from the RMT table *before* application of (Eq. 5). This is because each intermediate node along the reverse route will treat the RU message like a RR message and apply (Eq. 3) to update the values in the message and associated RMT entries in its table. Being able to send the RU message back to the source is guaranteed to work because of how we populated the RMT tables before sending the first message.

Notice that we do not have to worry about the destination node doing any updates because its latest lat_n value will always be stored in the RMT of its immediate neighbors. So, the destination node does not have to apply (Eq. 5). For efficiency purposes, the ECR protocol specifies that once a node sends out a set of RU message for a specific route, it cannot send another update message for that route until a cooldown period has passed. This cooldown period is needed to prevent too many update messages from being sent back and to allow inflight RP messages to be flushed from the network.

While the process described so far allows for updated information on the chosen route to be propagated back, we have yet to describe any way for the sending node to get updated information on routes that are not chosen. We can accomplish this by either having nodes randomly choose suboptimal routes at a small probability, or selectively sending out RD messages at some fixed interval along each known path. The latter is what is implemented in the simulation environment.

Algorithm: Error Handling & Dead Links

The ECR protocol supports error handling through the RE message. This message allows for the removal RMT entries to dead nodes. Earlier, in Figure 7, we saw that stale RMT entries to dead nodes can persist unless we apply some kind of heuristic (like removing entries where $lat_r \ll t_n$). However, even with a heuristic, we cannot remove all stale entries. So, we must rely on the RE message to remove stale entries. If a node receives a RP message that cannot be forwarded because there are no RMT entries or all entries have a non-existent next-hop node, then the node will send back an RE message to the source. Each node along the route back will append itself to the *error route* field and use the information to selectively remove RMT entries to the *route dst* node.

Protocol Novelty

The ECR builds upon the basics of common dynamic routing protocols like DSR and adds novelty through a unique routing table structure and route update process. In general, the ECR protocol's route discovery process is very similar to DSR. However, unlike DSR, each route has an expected last-alive-time and discount factor that is used to choose optimal routes. To facilitate this process, each node must compute its own estimated last-alive-time and efficiently send it to other nodes. This computation of the last-alive-time and maintenance is the core novelty of the ECR protocol that allows it to route packets in a way that maximizes network uptime.

Protocol Novelty: LAT Estimates & Discount Factor

One of the major novel concepts in the ECR protocol is the idea of the last-alive-time (lat). Each node has its own nodal lat_n that represents the future time when the node estimates it will die from a lack of battery. Note that this number can be infinite in the case of a node actively gaining charge. The ECR protocol allows each node to employ whatever technique it wants to compute this number. However, the ECR protocol is very specific about how the lat_n will be shared and used.

Each node's lat_n is shared with neighbors and used to compute a route lat_r between each possible source and destination node. The route lat between a source node s and a destination node d is defined as the least distance discounted lat_n of all the nodes $n \in N(s,d)$ along the route $R(s,d)$, with $D(n,R(s,d))$ being the distance (measured in number of hops) between n and s along the given route.

$$lat_{R(s,d)} = t_n + \min_{n \in N(s,d)} \gamma^{D(n,R(s,d))}(lat_n - t_n) \qquad \text{(Eq. 6)}$$

In (Eq. 6), t_n is the time now and $0 \leq \gamma \leq 1$ is the discount rate. We have a discount rate to prevent the ECR protocol from unnecessarily choosing to send packets through longer routes. When describing the optimal $lat_{R(s,d)}$, we often have an associated discount factor, $d_f = D(n, R(s,d))$. Notice that this is just the distance (number of hops) to the minimal element in (Eq. 6).

Protocol Novelty: RMT & Route Choice

Now that we have defined the concept of the lat, we can build upon it to create a novel routing table structure that allows nodes to cache routes and choose locally optimal paths. The ECR protocol defines the concept of a routing multi-table (RMT). The RMT holds unique routes and their associated (lat_r, d_f) pair. Since each node will only ever care about routes with itself as the origin, the RMT of a specific node can describe each route through a $(next\_hop, destination)$ pair. Thus, RMT entries will be a four-tuple $(next\_hop, destination, lat_r, d_f)$. When asked to route packets to a specific destination, each node will pick the best RMT entry it can find. This process is described in full detail in the *Algorithm* section.

Protocol Novelty: Information Propagation & Route Update Process

Thus far, we have defined a methodology for nodes to be able to quantify the usefulness of a route through the lat and choose between different routes through the RMT. However, we have not defined how the lat values are maintained and updated. This is where the final novel aspect of the ECR protocol comes in. The ECR protocol has a well-defined methodology for nodes to be able to propagate updated lat information though RP and RU packets. At a high level, when sending data, a node will wrap the data in an RP packet. The RP packet contains the expected lat value based on the local RMT cache of the sender. Then, each intermediate node will verify that the expected lat is conservative with regards to its own estimate. If the estimate is not conservative, then the node will send back an RU packet with the more conservative estimate. Since (Eq. 6) defines the lat as the minimal (or most conservative) estimate along the route, this update process will guarantee that each node will have the latest information. Note that because we define the lat_r as the minimal *distance* discounted lat, each node must ensure to multiply/divide by γ when comparing lat values.

Simulation Results & Analysis

The ECR protocol was tested by simulating its output in various networks that were specifically designed to highlight how the protocol works and responds to common conditions. The simulation code and config files are given alongside this report. Details on how to reproduce the results can be found in the code documentation paper.

Simulation Network 01

Simulation network 01 was constructed to show how the ECR protocol dynamically responds to changing conditions and can route packets through longer routes to yield longer network uptime. The simulation can be run with “sim_01_nodes.txt” network file and “sim_01_packets.txt” packets file. Figure 8 shows the initial network layout. In this scenario, *A* is asked to route packets to *Z* starting at time zero and not stop until all possible routes are exhausted. Notably, *B* and *C* start out with the same battery level. However, *D* has an even lower battery level, which initially limits the route *A-C-D-Z*. Figures 9 and 10 show how the ECR protocol alternates between routing packets through the A-*B-Z* and A-*C-D-Z* in response changing conditions. Figure 11 shows an overview of how the packets were routed. Figure 12 shows a timeline of the energy level of both available routes. We can see how the ECR protocol tried to ensure both routes stayed online for as long as possible.

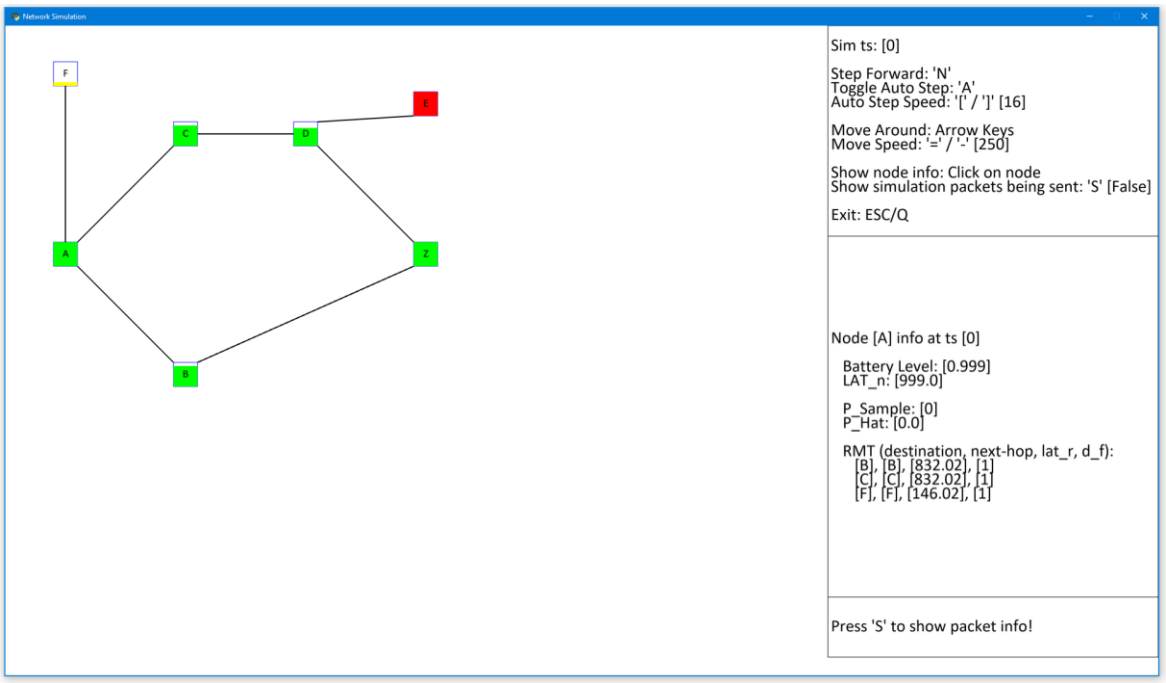


Figure 8: Initial layout of simulation network 01 at time 0. In this simulation, *A* is asked to route packets to *Z* until there is no path left. As seen in *A's* initial RMT table (shown on the right), nodes *B* and *C* have the same initial battery level (85%). Node D has a slightly lower battery level (75%). In this simulation, the ECR protocol will alternate between sending packets through the *A-B-Z* and *A-C-D-Z* routes to maximize the time before any of the relevant nodes die.

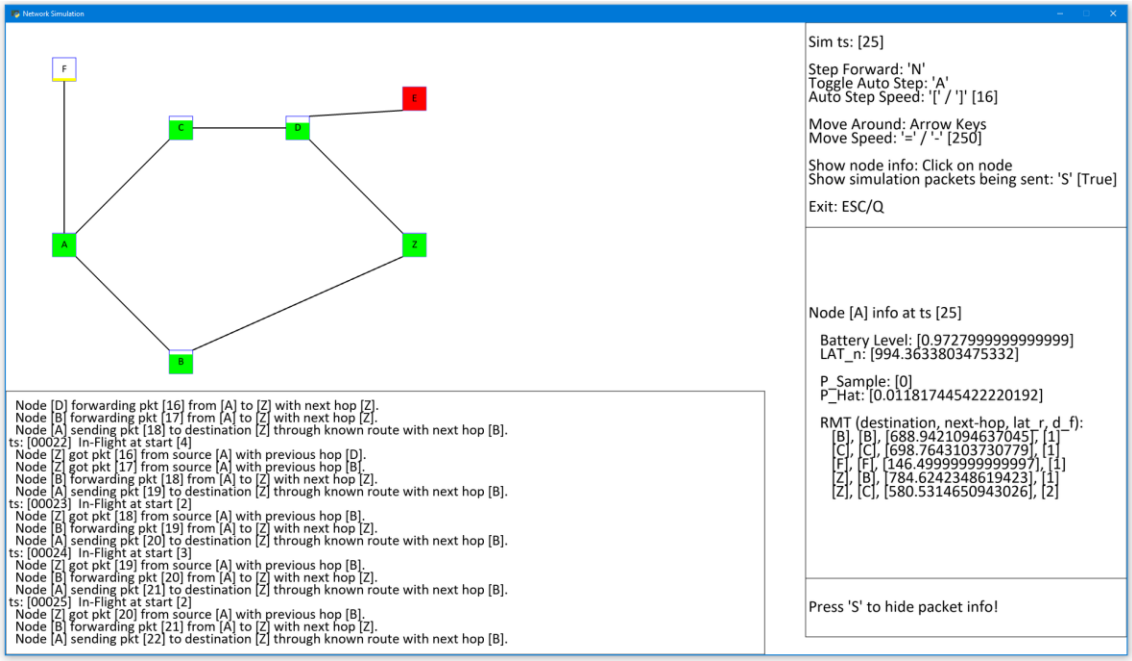


Figure 9: Snapshot of simulation environment for network 01 at time 25. Here, we can see that *A* initially routes packets to *Z* through *B* because the route through *C* is limited by *D.* Note *A's* RMT (shown on the right) indicates that the route through *C* is not chosen because it has a lower lat_r then the route through *B*. Additionally, the RMT shows the route through *C* is limited by a node that is two hops away.

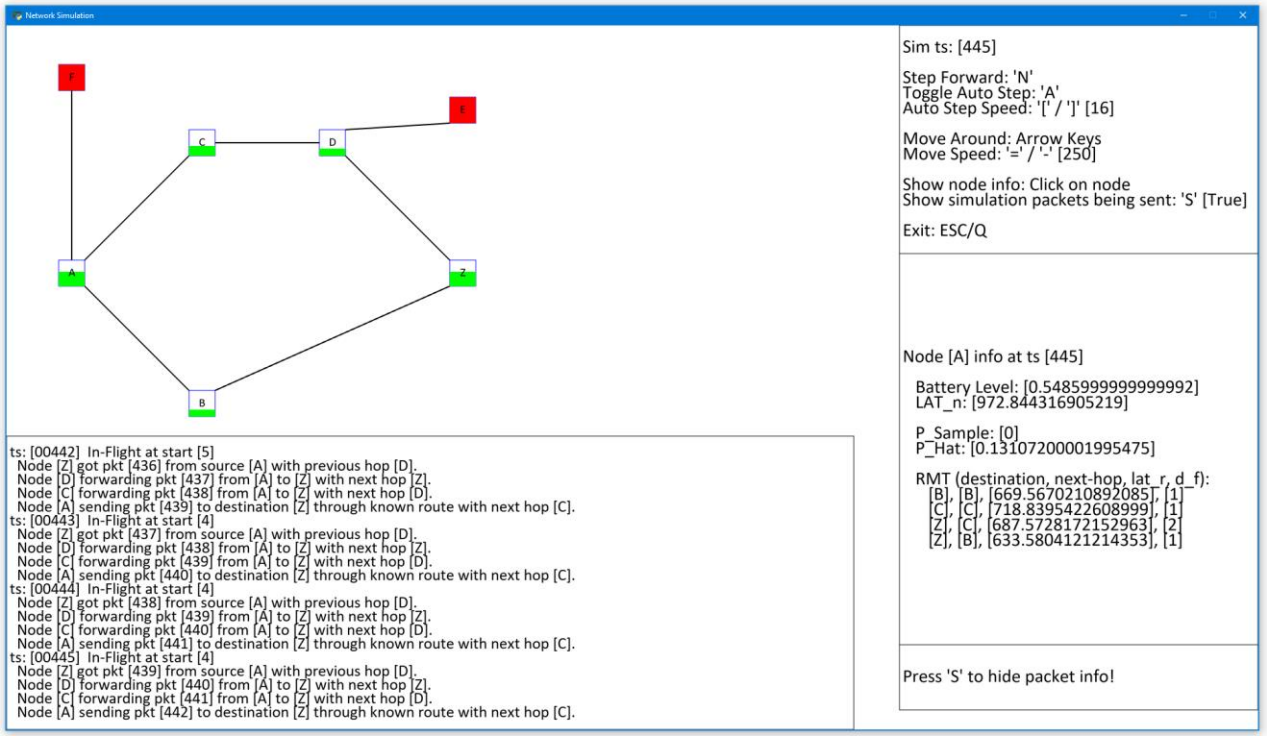


Figure 10: Snapshot of simulation environment for network 01 at time 445. As time progresses, node *B* loses more battery than *C* because *B* has been serving more of the packets to *Z.* So, the ECR protocol dynamically adapts and begins routing packets through *C* even through the route is physically longer. *A's* RMT (on the middle right) shows the lat_r values at time 445 that help drive the transition to the longer route.

```
At [00000] Node [A] was requested to send as many packets as possible to Node [Z]
  Node [A] sent [673] packets (including any necessary retries)
    ts: [00004]-[00010]: [A] sent [6] packets through [B]
    ts: [00011]-[00019]: [A] sent [8] packets through [C]
    ts: [00020]-[00108]: [A] sent [88] packets through [B]
    ts: [00109]-[00116]: [A] sent [7] packets through [C]
    ts: [00117]-[00218]: [A] sent [101] packets through [B]
    ts: [00219]-[00226]: [A] sent [7] packets through [C]
    ts: [00227]-[00328]: [A] sent [101] packets through [B]
    ts: [00329]-[00337]: [A] sent [8] packets through [C]
    ts: [00338]-[00349]: [A] sent [11] packets through [B]
    ts: [00350]-[00368]: [A] sent [18] packets through [C]
    ts: [00369]-[00438]: [A] sent [69] packets through [B]
    ts: [00439]-[00447]: [A] sent [8] packets through [C]
    ts: [00448]-[00454]: [A] sent [6] packets through [B]
    ts: [00455]-[00556]: [A] sent [101] packets through [C]
    ts: [00557]-[00568]: [A] sent [11] packets through [B]
    ts: [00569]-[00577]: [A] sent [8] packets through [C]
    ts: [00578]-[00665]: [A] sent [87] packets through [B]
    ts: [00666]-[00676]: [A] sent [10] packets through [C]
  Node [Z] received [669] packets
    ts: [00006]-[00012]: [Z] received [6] packets via [B]
    ts: [00014]-[00022]: [Z] received [8] packets via [D]
    ts: [00022]-[00110]: [Z] received [88] packets via [B]
    ts: [00112]-[00119]: [Z] received [7] packets via [D]
    ts: [00119]-[00220]: [Z] received [101] packets via [B]
    ts: [00222]-[00229]: [Z] received [7] packets via [D]
    ts: [00229]-[00330]: [Z] received [101] packets via [B]
    ts: [00332]-[00340]: [Z] received [8] packets via [D]
    ts: [00340]-[00351]: [Z] received [11] packets via [B]
    ts: [00353]-[00371]: [Z] received [18] packets via [D]
    ts: [00371]-[00440]: [Z] received [69] packets via [B]
    ts: [00442]-[00450]: [Z] received [8] packets via [D]
    ts: [00450]-[00456]: [Z] received [6] packets via [B]
    ts: [00458]-[00559]: [Z] received [101] packets via [D]
    ts: [00559]-[00570]: [Z] received [11] packets via [B]
    ts: [00572]-[00580]: [Z] received [8] packets via [D]
    ts: [00580]-[00666]: [Z] received [86] packets via [B]
    ts: [00669]-[00676]: [Z] received [7] packets via [D]
```

```
ts: [00677]  In-Flight at start [3]
  Node [A] got route error message for pkt [672] to [Z]. The error originated from [C]. Will reattempt to send the packet through another route
  Node [A] could not find route to [Z]. The sent RD messages have timed out!
  Node [A] could not find route to [Z]. The sent RD messages have timed out!
  ERROR: Node [A] cannot route packets to [Z]! The node may be offline or unreachable! Any future packets to this destination will not be sent!
ts: [00678]  In-Flight at start [1]
  Node [A] got route error message for pkt [673] to [Z]. The error originated from [C]. Will reattempt to send the packet through another route
  Node [A] could not find route to [Z]. The sent RD messages have timed out!
```

Figure 11: On top is the logged performance of how packets were sent in simulation 01. We can see how the ECR protocol alternated between routing packets through *A-B-Z and A-C-D-Z* for efficiently. We can also see that some packets had to be resent (673 were sent by *A,* but only 669 were received by *Z).* An inspection of more detailed logs (displayed on the bottom) shows that resending of packets was automatically done by the ECR protocol's inbuilt error handling mechanism in response to node *D* dying at the end of the simulation.

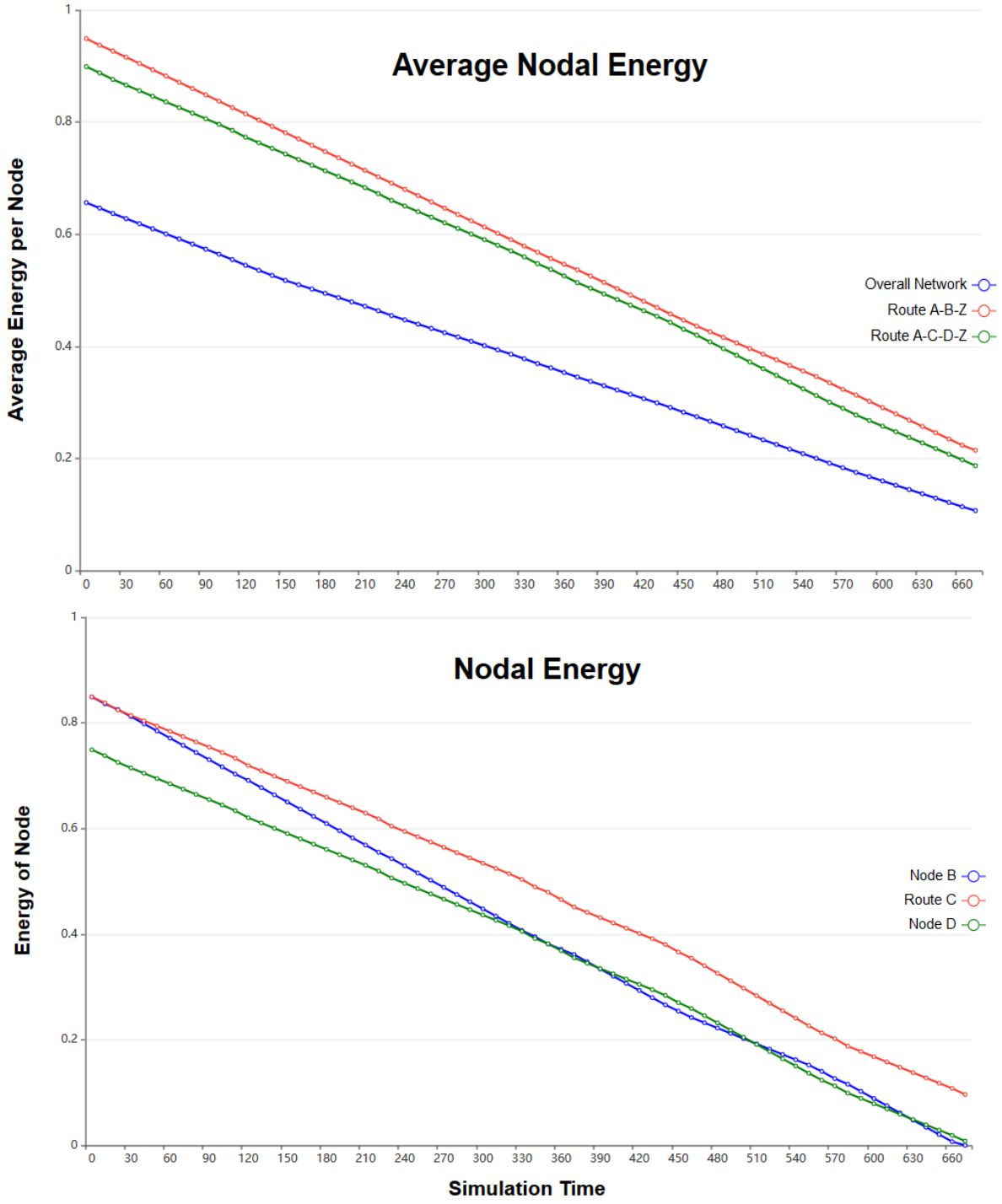


Figure 12: An energy analysis of the ECR protocol on simulation network 01. The top graph shows the average energy of the network and the two available routes. Notable, we can how the protocol managed to keep the decrease in energy across both routes relatively constant. The bottom graph shows the energy of specific nodes. Notable, we can see that the ECR protocol dynamically altered routing to ensure that nodes *B* and *D* died at around the same time. Without the ECR protocol, we would expect Node *D* to die way earlier since it starts with less energy.

Simulation Network 02

Simulation network 02 was constructed to show how the ECR protocol's discount factor works. Figure 13 shows the network structure. The simulation can be run with "sim_02_nodes.txt" network file and "sim_02_packets.txt" packets file. In this scenario, *A* is asked to route packets to *Z* starting at time zero and not stop until all possible routes are exhausted. There are two routes, *A-B-C-Z* and *A-D-E-Z*, that are essentially equivalent in every way. The only difference is that node *C* is the limiting node in the former route while node *B* is the limiting node in the latter route. Figure 14 shows that the ECR protocol will prefer to route packets through *A-D-E-Z* at the start because the limiting node is further away (and its importance is discounted). However, over time the ECR protocol will distribute packets between both routes to maintain throughput and network up time. This is shown in Figure 15 via the final packet route distribution. Finally, in Figure 16, we can see that the ECR protocol is able to balance energy loss to prevent either route from going down prematurely.

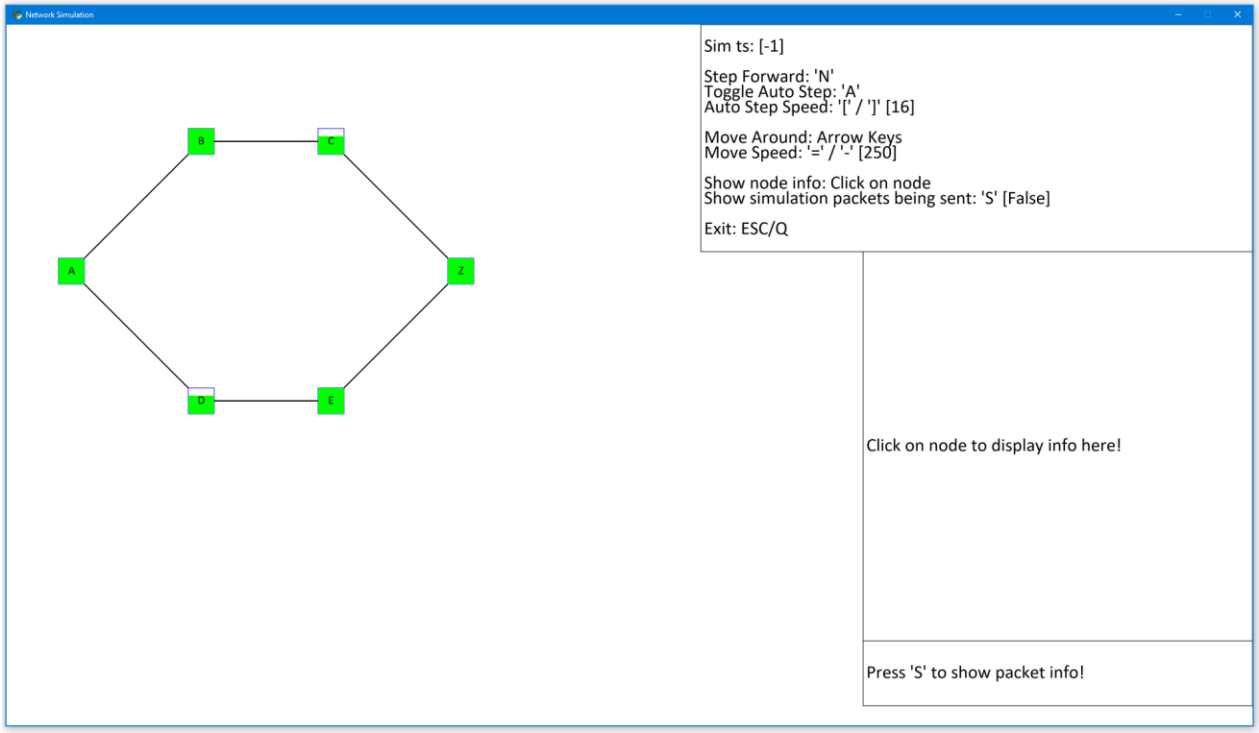


Figure 13: Initial layout of simulation network 02 at time 0. In this simulation, *A* is asked to route packets to *Z* until there is no path left. All nodes have full battery except *C* and *D,* which have 70% battery. This simulation highlights the importance of the discount factor as *A* will discount the importance of *C* and prefer to route through *D* until the energy level drops enough to necessitate routing through *B*.

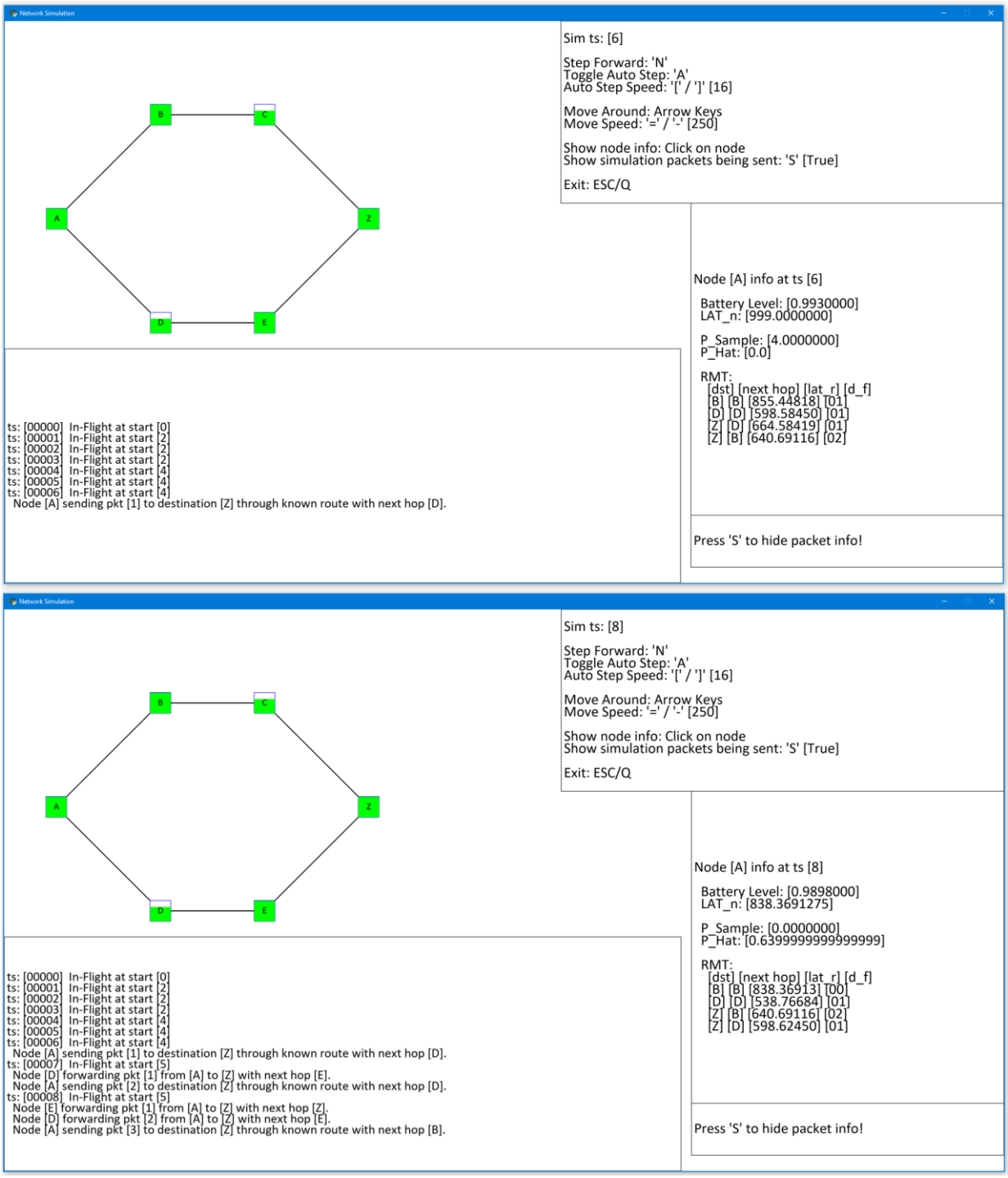


Figure 14: In simulation 02, once the initial route discovery process is done at time 6, *A* will prefer to route through *D* because the lat_n of *C* is discounted. This is shown in the top picture, which shows that *A's* RMT has a lower lat_r and a discount factor of 2 for the route through *C*. However, as soon as *A* starts routing packets through *D*, *D* will update is $lat_n$and send back a RU message that will cause *A to* update its RMT and route through *B* instead. This is shown in the bottom picture taken at time 8. Because the simulation only has packets being sent from *A* to *Z,* it will take a little while for nodes to gain an accurate estimate of their packet throughput. Once this happens, the network routing will stabilize.

```
At [00000] Node [A] was requested to send as many packets as possible to Node [Z]
  Node [A] sent [586] packets (including any necessary retries)
    ts: [00006]-[00007]: [A] sent [1] packets through [D]
    ts: [00008]-[00016]: [A] sent [8] packets through [B]
    ts: [00017]-[00028]: [A] sent [11] packets through [D]
    ts: [00029]-[00042]: [A] sent [13] packets through [B]
    ts: [00043]-[00044]: [A] sent [1] packets through [D]
    ts: [00045]-[00046]: [A] sent [1] packets through [B]
    ts: [00047]-[00059]: [A] sent [12] packets through [D]
    ts: [00060]-[00078]: [A] sent [18] packets through [B]
    ts: [00079]-[00080]: [A] sent [1] packets through [D]
    ts: [00081]-[00082]: [A] sent [1] packets through [B]
    ts: [00083]-[00095]: [A] sent [12] packets through [D]
    ts: [00096]-[00110]: [A] sent [14] packets through [B]
    ts: [00111]-[00120]: [A] sent [9] packets through [D]
    ts: [00121]-[00129]: [A] sent [8] packets through [B]
    ts: [00130]-[00131]: [A] sent [1] packets through [D]
    ts: [00132]-[00150]: [A] sent [18] packets through [B]
    ts: [00151]-[00230]: [A] sent [79] packets through [D]
    ts: [00231]-[00240]: [A] sent [9] packets through [B]
    ts: [00241]-[00247]: [A] sent [6] packets through [D]
    ts: [00248]-[00276]: [A] sent [28] packets through [B]
    ts: [00277]-[00278]: [A] sent [1] packets through [D]
    ts: [00279]-[00350]: [A] sent [71] packets through [B]
    ts: [00351]-[00360]: [A] sent [9] packets through [D]
    ts: [00361]-[00369]: [A] sent [8] packets through [B]
    ts: [00370]-[00470]: [A] sent [100] packets through [D]
    ts: [00471]-[00480]: [A] sent [9] packets through [B]
    ts: [00481]-[00482]: [A] sent [1] packets through [D]
    ts: [00483]-[00506]: [A] sent [23] packets through [B]
    ts: [00507]-[00508]: [A] sent [1] packets through [D]
    ts: [00509]-[00580]: [A] sent [71] packets through [B]
    ts: [00581]-[00593]: [A] sent [12] packets through [D]
  Node [Z] received [582] packets
    ts: [00009]-[00010]: [Z] received [1] packets via [E]
    ts: [00011]-[00019]: [Z] received [8] packets via [C]
    ts: [00020]-[00031]: [Z] received [11] packets via [E]
    ts: [00032]-[00045]: [Z] received [13] packets via [C]
    ts: [00046]-[00047]: [Z] received [1] packets via [E]
    ts: [00048]-[00049]: [Z] received [1] packets via [C]
    ts: [00050]-[00062]: [Z] received [12] packets via [E]
    ts: [00063]-[00081]: [Z] received [18] packets via [C]
    ts: [00082]-[00083]: [Z] received [1] packets via [E]
    ts: [00084]-[00085]: [Z] received [1] packets via [C]
    ts: [00086]-[00098]: [Z] received [12] packets via [E]
    ts: [00099]-[00113]: [Z] received [14] packets via [C]
    ts: [00114]-[00123]: [Z] received [9] packets via [E]
    ts: [00124]-[00132]: [Z] received [8] packets via [C]
    ts: [00133]-[00134]: [Z] received [1] packets via [E]
    ts: [00135]-[00153]: [Z] received [18] packets via [C]
    ts: [00154]-[00233]: [Z] received [79] packets via [E]
    ts: [00234]-[00243]: [Z] received [9] packets via [C]
    ts: [00244]-[00250]: [Z] received [6] packets via [E]
    ts: [00251]-[00279]: [Z] received [28] packets via [C]
    ts: [00280]-[00281]: [Z] received [1] packets via [E]
    ts: [00282]-[00353]: [Z] received [71] packets via [C]
    ts: [00354]-[00363]: [Z] received [9] packets via [E]
    ts: [00364]-[00372]: [Z] received [8] packets via [C]
    ts: [00373]-[00473]: [Z] received [100] packets via [E]
    ts: [00474]-[00483]: [Z] received [9] packets via [C]
    ts: [00484]-[00485]: [Z] received [1] packets via [E]
    ts: [00486]-[00509]: [Z] received [23] packets via [C]
    ts: [00510]-[00511]: [Z] received [1] packets via [E]
    ts: [00512]-[00580]: [Z] received [68] packets via [C]
    ts: [00584]-[00593]: [Z] received [9] packets via [E]
```

Figure 15: Performance log for simulation 02. We can see that packets were alternatingly set via routes *A-B-C-Z* and *A-D-E-Z*. Notably, the number of consecutive packets through each route stabilized as time progressed and intermediate nodes made better estimates of their throughput. The ECR protocol would perform even better if the simulation included more packets from a variety of sources since nodes could better apply (Eq. 2). However, this kind of simulation was avoided to allow for understandable results.

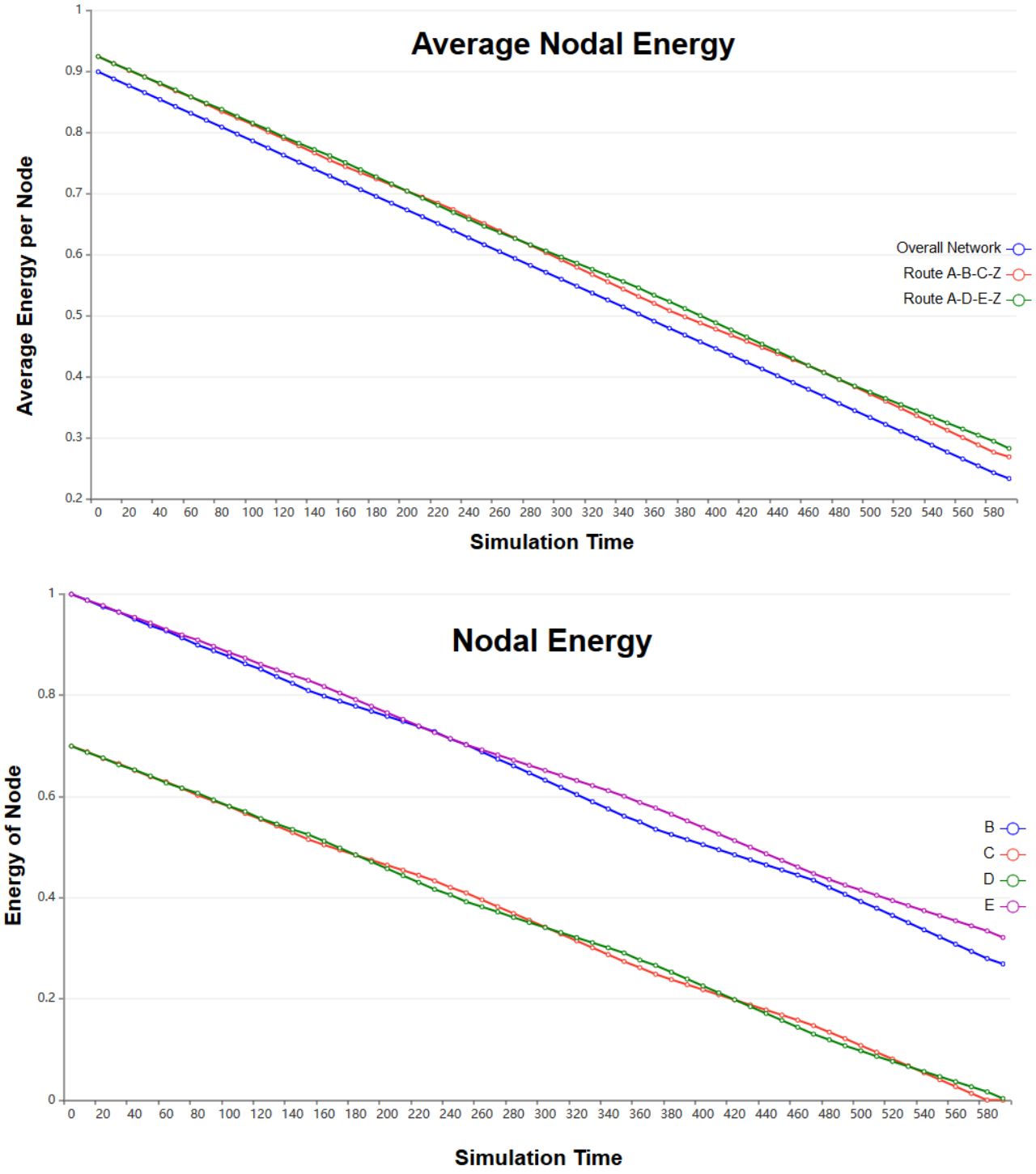


Figure 16: An energy analysis of the ECR protocol on simulation network 02. The top graph shows the average energy of the network and the two available routes. Notable, we can see that the average energy of two routes (in orange and green) crisscrossed depending on how packets were routed. The ECR protocol was able to keep the overall deviation very small. The bottom graph shows the energy of specific nodes. Because of the symmetry of the network, we can see a similar pattern of crisscrossing lines in both graphs.

Simulation Network 03

Simulation network 03 was constructed to see how the ECR protocol handles routing with multiple source and destination routes in a network with a bottleneck. In the network, shown in Figure 17, node *G* is a bottleneck. The simulation can be run with "sim_03_nodes.txt" network file and "sim_03_packets.txt" packets file. In this scenario, *A* is asked to route packets to *Y* and *Z* starting at time zero and not stop until all possible routes are exhausted. Additionally, *G* is asked to route 250 packets to *A* starting at time 50. Figure 18 shows a nodal energy analysis of the simulation. Figure 19 details an example of the ECR protocol's inbuilt error handling abilities.

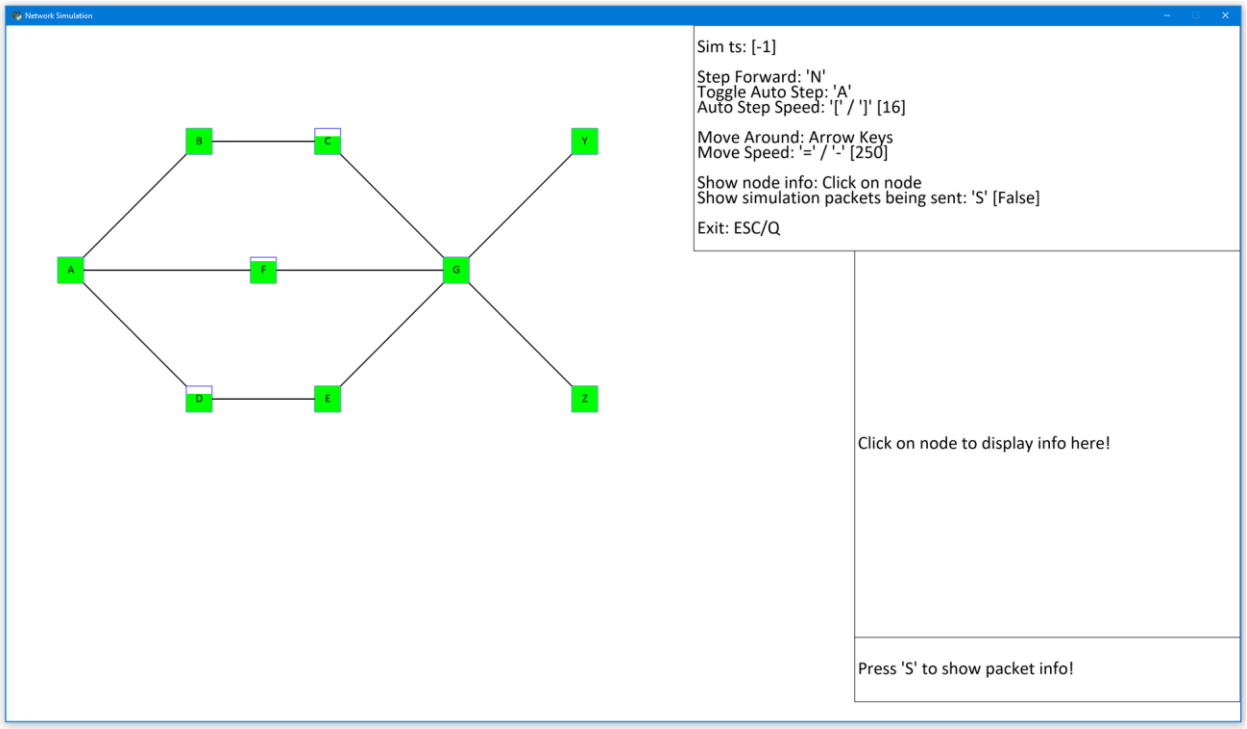


Figure 17: Initial layout of simulation network 03 at time 0. *A* is asked to route packets to *Y* and *Z* until there are no paths left. Additionally, at time 50, *G* is asked to route 250 packets back to *A.* Nodes *C* and *D,* have 70% battery while node *F* has 85% battery. All other nodes have full battery. This simulation shows how to ECR protocol performs when sending multiple packet streams through a bottleneck (node *G*).

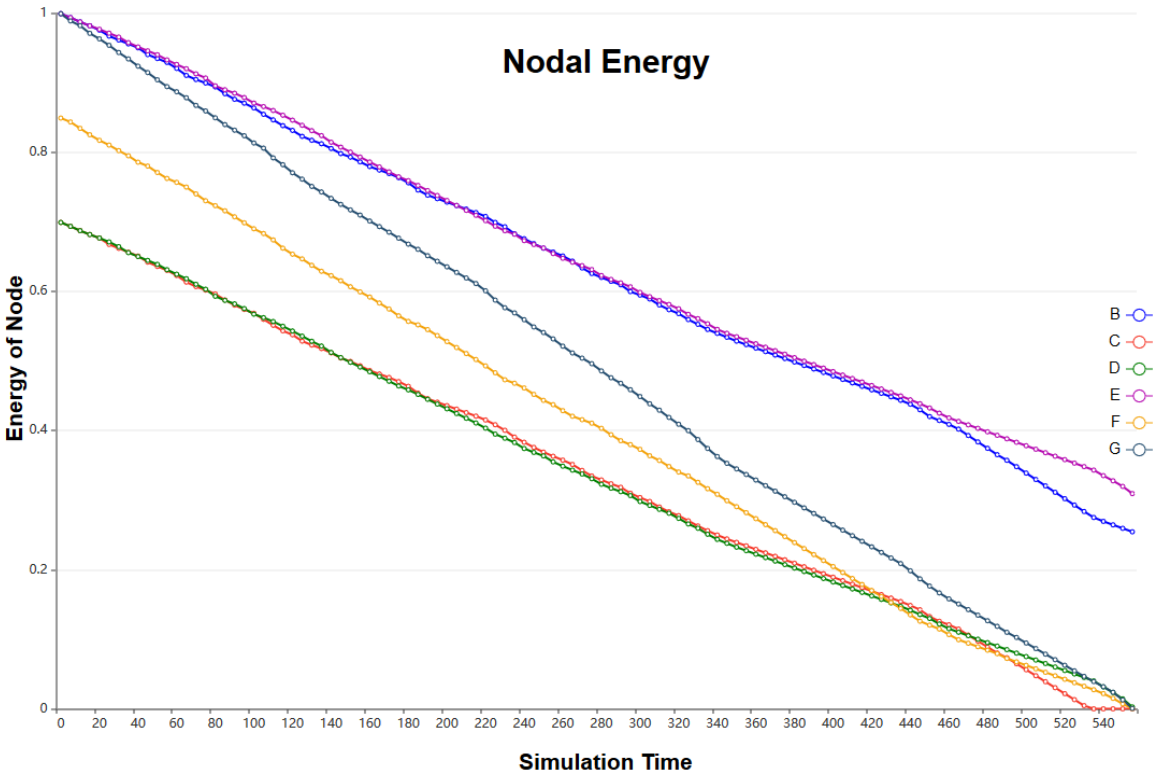
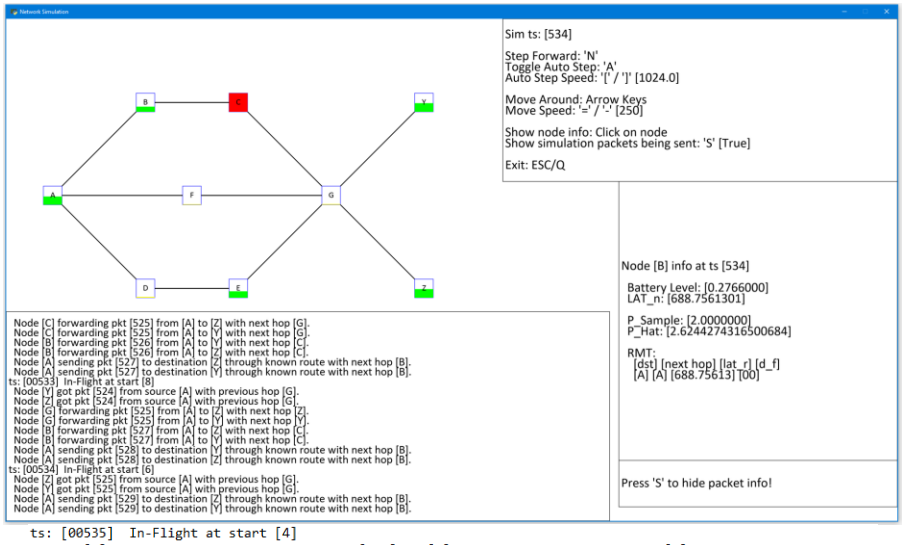


Figure 18: A nodal energy analysis of the ECR protocol on simulation network 03. The graph shows how the ECR protocol was able to route packets in a such a way that no node died too early.




```
ts: [00535]  In-Flight at start [4]
  Node [A] got route error message for pkt [528] to [Y]. The error originated from [B]. Will reattempt to send
the packet through another route
  Node [A] got route error message for pkt [528] to [Z]. The error originated from [B]. Will reattempt to send
the packet through another route
ts: [00536]  In-Flight at start [6]
  Node [A] got route error message for pkt [529] to [Z]. The error originated from [B]. Will reattempt to send
the packet through another route
  Node [A] got route error message for pkt [529] to [Y]. The error originated from [B]. Will reattempt to send
the packet through another route
ts: [00537]  In-Flight at start [6]
```

Figure 19: An example of the ECR protocol's error handling from simulation 03. At time 534, *C* goes offline. As a result, *B,* which is forwarding packets from *A-Y* and *A-Z,* cannot function properly and sends an error message back to *A. A* responds to the error message at 535 and reroutes the packets.

Simulation Network 04

Simulation network 04 extends simulation network 03 and is designed to highlight how the ECR protocol seeks to dynamically avoid bottlenecks. The simulation can be run with “sim_04_nodes.txt” network file and “sim_04_packets.txt” packets file. In this scenario, *A* is asked to route packets to X, *Y* and *Z* starting at time zero and not stop until all possible routes are exhausted. Additionally, *X, Y,* and *Z* are asked to route 200 packets back to *A* starting at time 100. Figure 20 shows the initial structure of the network. It shows that *G* is a bottleneck across all routes. However, unlike simulation network 03, we now have a way to avoid routing through *G* in many cases. Figure 21 shows a nodal energy analysis of the simulation. Figure 22 details how the various packets were routed to help alleviate the congestion at *G*.

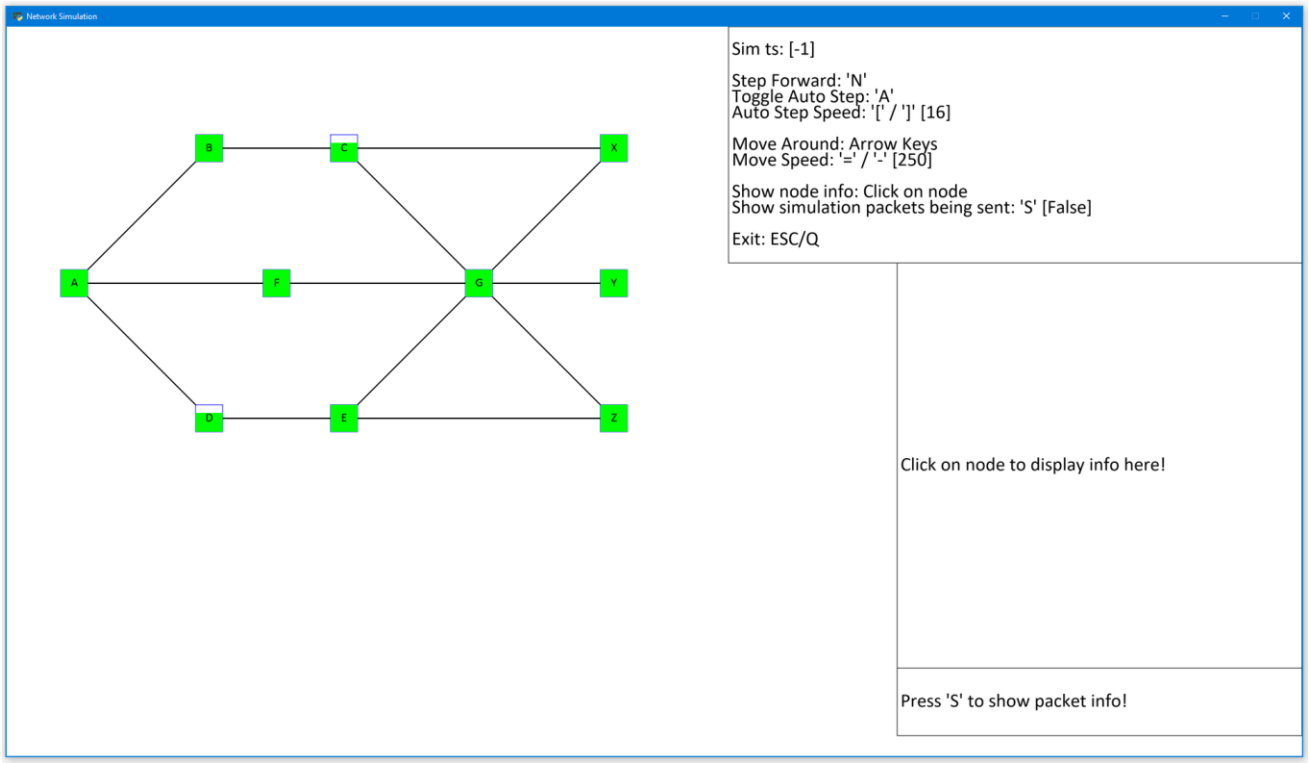


Figure 20: Initial layout of simulation network 04 at time 0. *A* is asked to route packets to X, *Y* and *Z* until there are no paths left. Additionally, at time 100, *X, Y,* and *Z* are asked to route 200 packets back to *A*. In this simulation, the bottleneck *G* can be avoided when routing to and from *X* or *Z*.

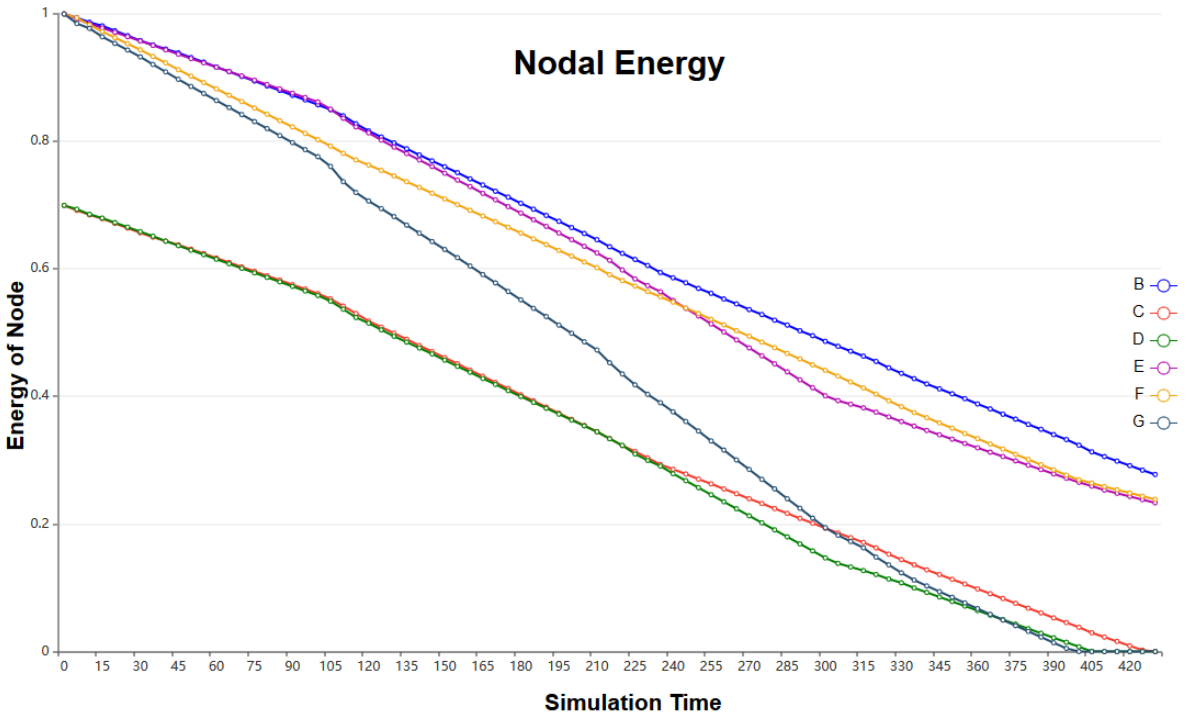


Figure 21: A nodal energy analysis of the ECR protocol on simulation network 04. Notably, we can see the energy drop starting at around time 100 (when *X*, *Y*, and *Z* are asked to route packets back to *A).* We can also see that the ECR protocol was able to extend the life of *G* so that it died at around the same time as *C* and *D.* In a traditional routing protocol (like DSR), *G* would likely have died much earlier.

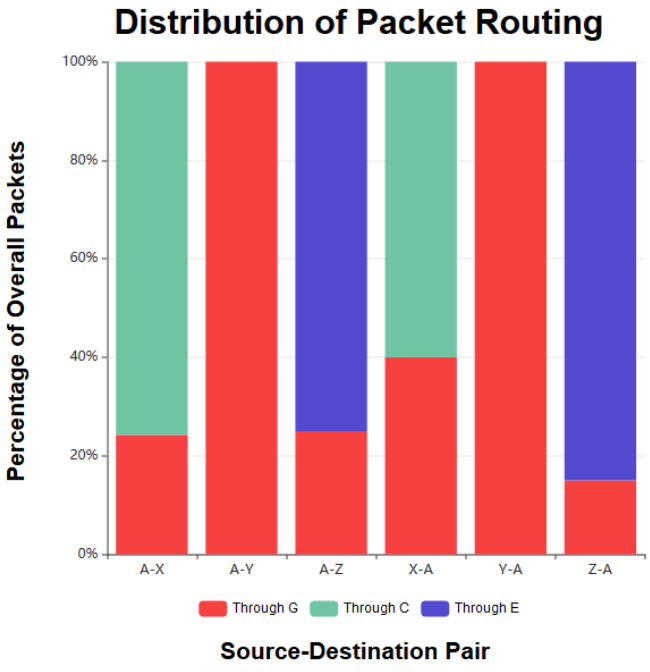


Figure 22: A graph showing the overall distribution of how packets were routed between each source and destination pair in simulation 04. We can see that most packets between *A* and *X* avoided the bottleneck of *G* by going through *C.* Similarly, packets between *A* and *Z* avoided the bottleneck by going through *E.* Packets between *A* and *Y* had to go through *G* as there is no alternative route.

Overall Results

In summary, the simulation results show that the ECR protocol can dynamically adapt to changing conditions. The protocol is able to increase network longevity and deliver packets reliably even when faced with networks that have bottlenecks. Moreover, the protocol can avoid bottlenecks when possible, even if it means routing through longer distances. The tradeoff between efficiency and network longevity is managed by a discount rate applied to longer routes. The protocol has inbuilt error handing scenarios that ensure packets are resent when needed. This allows for the ECR protocol to deal with dynamic networks where nodes may go down at a moment's notice. The update process of the ECR is efficiently designed to prevent the need for sending too many packets while simultaneously ensuring up-to-date information is propagated properly. The novel concept of routing based on the distance discounted predicted last-alive-time of nodes and routes is a very valuable tool in networks that are energy constrained.