An Energy Efficient Flat Routing Protocol for Wireless Ad hoc Networks

Vahid Nazari Talooki, Hugo Marques, Jonathan Rodriguez
Instituto de Telecomunicações
Aveiro, Portugal
vahid@av.it.pt, hugo.marques@av.it.pt, jonathan@av.it.pt

Hugo Água, Nelson Blanco, Luís Campos
PDM&FC
Lisboa, Portugal
hugo.agua@pdmfc.com, nelson.blanco@pdmfc.com, luis.campos@pdmfc.com

***Abstract*—In ad hoc networks some nodes can became a critical spot in the network because they support packet forwarding for most of their neighbours. Critical nodes would consume more energy due to the extra load and deplete battery sooner. These unfairly burdened nodes will lead to node failure, network partitioning, decrease in route lifetime and route reliability. To avoid this problem, this paper proposes a new routing protocol, called Energy Efficient DSR (E2DSR) for balancing the energy consumption amongst the nodes in the network. E2DSR uses some mechanisms of Dynamic Source Routing (DSR) but defines a new structure for control packets, changes the routing behavior in nodes, implements a new "Energy Table" and creates a whole new algorithm for route cache and selection.**

Keywords- E2DSR; Energy Efficiency; Flat Routing; Load Balancing; Mobile Ad hoc Networks; Routing Protocol

I. INTRODUCTION

Ad hoc networks are characterized by multi-hop wireless connectivity, a frequently changing network topology and an absence of an infrastructure. Each mobile node operates not only as a host but also as a router, forwarding packets for other mobile nodes in the network that may, or not, be within direct transmission range of each other [1]. Ad hoc networks don't have a central point of failure, which is good for redundancy, but may unfairly overload some particular nodes in the topology. These nodes are the ones that, due to their location, relay an increased number of packets in the network and consequently, are viewed as critical spots in the network. Besides creating a bottleneck effect, the overload on these nodes also depletes their battery quicker, causing them to fail. Since the majority of traffic in the network crosses these critical spot nodes, they can become an important target for attackers.

To avoid these issues, we propose to implement a new routing protocol, called Energy Efficient Dynamic Source Routing (E2DSR) that is energy efficient [2]. Energy efficiency is a critical issue for battery-operated mobile devices in ad hoc networks. As mentioned, unbalanced power consumption may not only result in earlier node failure in overloaded nodes, but also lead to network partitioning and a decrease in route reliability [3]. So, there is a need to both improve energy efficiency and balance battery consumption among nodes in eMANETs to reduce the number of critical nodes in the network.

The work presented in this paper was undertaken in the context of the project INFSO-ICT- 225654 PEACE (IP-Based Emergency Applications and ServiCes for Next Generation Networks), which has received research funding from the European seventh Framework Programme. The authors would like to acknowledge the contributions of their colleagues from the PEACE consortium.

There are some protocols that use different approaches to achieve energy efficiency or battery consumption balancing. However most of these protocols try to find some network layer mechanism to avoid flooding and unnecessary packet forwarding but one major drawback of most of these ad hoc routing protocols is that they do not have provisions for conveying the load and/or quality of a path during the route setup. Nevertheless there are a considerable number of studies that try to improve these issues, but they usually cause some other drawbacks like requiring global topology information, increase delay or even create a blocking issue [4,5,6,7]. Also these energy efficient algorithms do not contain considerations for routing protocol scalability and QoS.

The blocking issue happens for example, when a source node is impeded by a timeout timer to start a data transmission before receiving all replies for a route request message. Determining this time is very challengeable and finally will turn to more delay.

Another issue for which most of current ad hoc routing protocols don't have a proper mechanism to handle is saving critical nodes. As aforementioned, there is a high probability that some nodes turn into critical nodes and because they randomly have a good position in network topology several routes use them and consequently they consume more battery power which leads to node failure. Our protocol proposes a new mechanism to detect these nodes, balance their load and avoid them to be shut off from the network. Most of other studied protocols try to find the optimized routing mechanism in terms of energy efficiency but they fail to solve problems like end-to-end delay or routing load. E2DSR not only tries to find the best energy efficient route for each transmission but also considers other issues like delay and length of route.

E2DSR aims to be usable not only for Mobile Ad hoc NETworks (MANET) but also for Ad hoc sensor networks. Energy efficiency is of great importance in sensor ad hoc networks due to limitations in terms of battery, memory and processing power. E2DSR is based in the Dynamic Source Routing (DSR) protocol [8] and improves it to be energy efficient. E2DSR uses a new structure for control packets, changes the routing behavior in the nodes, implements a new "Energy Table" and creates a whole new algorithm for route cache and selection. Although we opt to use DSR, the same principle is applicable for other current flat ad hoc routing protocols such as Destination Sequenced Distance Vector (DSDV) or Ad -Hoc On-demand Distance Vector (AODV).

The paper is structured as follows: Section II briefly reviews DSR protocol, Section III describes the proposed E2DSR protocol, Section IV provides performance metrics to evaluate the performance of E2DSR that and finally, section V presents the conclusions and future work regarding the implementation of E2DSR in simulation environment and in real sensors.

II. Dynamic Source Routing (DSR) Protocol

Routing protocols in ad hoc networks can be divided in flat, hierarchical, geographical, or a hybrid version of any of the previous, routing protocols [1]. DSR is considered to be a flat routing protocol, where all nodes participating in routing play an equal role. Flat routing protocols may generally be classified into three categories: (i) flooding protocols; (ii) proactive protocols and; (iii) reactive protocols. In flooding protocols every incoming packet is sent out on every other link by every node. These protocols are very simple to implement but generate significant traffic overhead during the flooding procedure.

Proactive protocols attempt to find and maintain consistent, up-to-date routes between all source-destination pairs regardless of the use or need of such routes. Each node maintains one or more tables to store routing information. In proactive protocols, routing techniques are either link-state or distance vector and require each node to use periodic control messages to maintain routes up to date. These protocols are based on three algorithms: (i) Distance Vector (DV), like DSDV; (ii) Link State (LS), like Optimized Link State Routing Protocol (OLSR) and; (iii) Hybrid DV and LS, like Wireless Routing Protocol (WRP). Proactive protocols need more memory and also use periodic routing messages, which create more traffic overhead. The main advantage of these protocols is that they react quickly to events in the network such as the discovery of new available routes to a destination.

In reactive protocols routes are created only when a source node requests them. Data forwarding is accomplished according to either source routing like DSR or hop-by-hop like AODV. Reactive protocols normally produce less control packets and have smaller memory requirements in comparison of proactive protocols but sometimes choose non-optimal routes.

DSR which appeared in [9] is a flat routing protocol that uses an on demand flooding mechanism for route discovery, uses source routing to discover routes and construct routing tables, doesn't exchange periodic control messages (like Hello or Beacon messages) and can optionally use a "flow state" identification to avoid the big header problem, typical in source routing protocols [8]. This makes DSR a lightweight routing protocol for MANETs. The behavior of DSR can be resumed in the two following phases: route discovery and route maintenance.

The route discovery phase is based on flooding and is used to dynamically discover new routes and also maintain them in the node's routing cache. In this phase, a source node *S* that wants to discover a route to a destination node *D*, creates a Route Request (RREQ) packet and sends it to all of its neighbours. This RREQ includes fields such *S's* address, *D's* address, hop count, and an array of intermediate node addresses (at beginning only includes the address of source node). Each node that receives a RREQ packet increases the hop count by one and checks to see if it is the destination node. If it is not (it is considered to be an intermediate note), it checks to see if it has any known route to *D*, if it has, then the node creates a Route Reply (RREP) message and puts its own address, as well as the addresses of the other nodes this node uses to reach *D*, in the address field array of the RREP message and send it back to *S*; if this node does not have any route to *D*, it will change the received RREQ packet by putting its own address in the addresses field of the RREQ message and then forwards it to all of its neighbours (except the one for which it has received the RREQ packet), with the objective of reaching *D* (flooding process). When the node receiving the RREQ is *D*, it replies to the RREQ with a RREP packet that is routed back to *S*. This RREP contains *D's* address as well as the addresses of any other node that has forwarded the RREQ message to *D* (source routing). In DSR intermediate and destination nodes only process the first received RREQ message and ignore subsequent RREQs.

The route maintenance process detects link breakage and changes in the network topology. These changes are then propagated by the use of a Route Error (RERR) control message.

III. Description of Energy Efficient DSR (E2DSR)

This section describes E2DSR having DSR as a base. E2DSR makes significant changes in control packets, changes the routing behavior in the nodes, creates a new table, called the "Energy Table" and defines a new algorithm for route cache and selection

A. Structure of Control Packets in E2DSR

1) Structure of RREQ

In E2DSR we added an energy field in the form of an array to RREQ message. This energy array field contains the remaining power (battery energy level) of each node that forwarded the RREQ. Hence, when a node forwards a RREQ, it will append a value that corresponds to its current remaining battery power to the end of energy array field. The value for the remaining battery for each node is implemented by a 4 bit field which gives 16 different levels. Level zero means battery critical and level 15 means full battery. The use of 16 levels gives a good granularity level while maintaining header size at a reasonable overhead.

2) Structure of RREP

In E2DSR, the RREP message was also modified to include the energy array field. Destination nodes, upon receiving a RREQ message, create a RREP message, add an energy array field that is a copy of the one received in the RREQ message, and send the RREP to the originator of the RREQ. If a RREQ is received by an intermediate node that already has a route to destination, this node uses the energy values of that route when responding to the RREQ message.

The energy array of the new RREQ and RREP message directly shows the energy level of a route

B. Routing Behavior of Nodes in E2DSR

1) Intermediate Nodes

Contrary to DSR, in E2DSR the intermediate node forwards or replies to a maximum of K RREQs (from *S* to *D*). When the first RREQ from S arrives to an intermediate node, it processes it immediately and stores the RREQ in its request table (as in DSR). The intermediate node then caches all subsequent received RREQs and will choose the *K-1* routes with the highest energy level to reply. To do that, the intermediate node will extract the energy array of every RREQ and then calculates the energy parameter of path (energy parameter is discussed in section III.D.4)). The energy parameter of first RREQ is E_1.Only the RREQs which have a better energy level than E1 (the first arrived RREQ energy level) should be forwarded. To achieve this goal, we save E value (which is E1 plus a threshold) in the header of first RREQ in its request table (*E* is introduced in equation 1).The value of threshold is controlled by a coefficient which is set to a reasonable value Ce=0.2 in equation 1. A bigger coefficient yields a bigger threshold. The time at which the first RREQ is received (T_1) will also be saved on the request table and a related timeout clock ($T_{Inter.Wait}$) will be started.

$$E = E_1 + C_e(1 - E_1) \qquad (1)$$

$$0 \le E \le 1;\ \ 0 \le E_1 \le 1;\ \ \mathrm{C_e} = 0.2$$

When the *ith* route request ($\mathrm{RREQ_i}$) arrives, the intermediate node will again calculate E_i (energy parameter of $\mathrm{RREQ_i}$) and if $E_i > E$ then it forwards $\mathrm{RREQ_i}$ and also updates E by $\mathrm{E_i}$ (i.e. $E{=}E_i$) otherwise it simply drops $\mathrm{RREQ_i}$. By using this RREQ processing mechanism, each intermediate node will forward *K* RREQ at maximum; also the intermediate node will not forward RREQs that have arrived after $T_{Inter.Wait}$ timer expires. The algorithm for forwarding RREQs in intermediate nodes is shown in Figure 1.

```
ForwardRREQs() {
  If this is first RREQ {
    Forward RREQ;
    E=E1+Ce(1-E1)         //Energy Parameter of first RREQ
    T1=Current Time;
    Save RREQ, E and T1 in table;
    Counter=1;
  }
  Else {
    Calculate Ei;          //Energy Parameter of ith RREQ
    If ((counter<K-1) && (Ei>E) &&
    ((Current Time - T1)<TInter.Wait))
    {
    Forward RREQi;
    E=Ei;
    Counter++;
    }
  }}
```

Figure 1. Algorithm for forwarding RREQs in intermediate nodes

By varying *K* and $T_{Inter.Wait}$, the behavior of the protocol can be adjusted to be more efficient. The chosen value for *K* is 3. Higher values for *K* imply forwarding and processing more RREQs which have a significant impact in traffic overhead and energy consumption. A smaller value for *K* limits the number of redundant routes in a node's routing cache, which can lead to higher delays and also to more traffic in the network, due to the discovery phase flooding mechanism. Also $T_{Inter.Wait}$ is set to 2 seconds because it's assumed that a RREQ which is received after this reasonable long time, has encountered problems inside its path (such as traffic congestion or interference), so it should be ignored.

```
ReplyToRREQs() {
  If this is first RREQ {
    Reply RREQ;
    E=E1+Ce(1-E1)         //Energy Parameter of first RREQ
    T1=Current Time;
    Save RREQ, E and T1 in table;
    Counter=1;
  }
  Else {
    Calculate Ei;          //Energy Parameter of ith RREQ
    If ((counter < K´-1) && (Ei>E) &&
    ((Current Time-T1)<TDest.Wait)) {
       Reply to RREQi;
       E=Ei;
       Counter++;
    }
  }
}
```

Figure 2. Algorithm for replying to incoming RREQs in destination nodes

2) Destination nodes

In E2DSR, destination nodes will immediately reply to the first received RREQ (avoiding the blocking problem described in Section I) and also to the subsequent *K*-1 received RREQs which have the highest energy. The algorithm of replying to an incoming RREQ in destination nodes is shown in Figure 2.

By changing two numbers (*K´*, $T_{Dest.Wait}$), we can customize our protocol for most efficiency; for destination nodes their default values are (3, 4s), respectively.

3) Source Nodes

E2DSR uses a new function, called *Route Priority Function*, to calculate the priority of each discovered route and select the best route to use for a specific destination (see section III.D). After *S* had received one or more RREP to *D* (during discovery phase), *S* runs the *Route Priority Function* on all of the discovered routes to find the best one. The chosen route to use for data communication will be the one that has the highest priority. Taking in consideration the behavior of intermediate and destination nodes described previously, *S* can still receive other RREPs with higher energy level shortly after the first calculation (during $\mathrm{T_{Source.Wait}}$ seconds), so again S should run *Route Priority Function* to find the new best route. This (re)selection process doesn't add additional delay since it can run in the background.

If a data exchange, between *S* and *D,* takes too long, the energy of all the nodes in the path will change and eventually some nodes will first achieve a critical battery level. To avoid this, during the data transmission time window, a source node may change the route to *D* to an alternate route that has better energy. *S* should run *Route Priority Function* in the following cases:

- The first run should happen at the beginning of the communication, $T_{run1} = 0$.
- The second run is at time $T_{run2} = T_{Source.Wait} = 6s$. Note that during this time, *S* has probably received other routes to *D*.
- The next runs will take place each $T_{interval} = 180s$. This number is chosen for test setup and can be changed based on new results.

By using this mechanism, source nodes will balance the energy consumption between the best routes they have to a specific destination.

...	S	11	12	13	14	15	...
...	S	12	11	1	2	10	...
...	S	2	6	5	8	15	...
...	S	6	2	1	11	...	
...	S	2	6	5	8	15	...

Figure 3. Snapshot for routing cache of node *S* (path part)

C. *The Energy Table*

In E2DSR, nodes create a new table called *Energy Table*. This table is used to save the value of the remaining battery power of discovered nodes in network. By extracting data from this table a node can determine the energy level of whole path towards a destination.

1) *Structure of Energy Table*

Different nodes have different information in their own routing cache, requests table, flow table and also in energy table. Taking as example the routing cache, we know that for each entry there are associated fields, such as time to live (TTL), hop count, path to destination, and so on. Figure 3 shows partial information (relative to path) retrieved from the routing cache of node *S*. This snapshot shows *S* has currently learned five routes; three routes to node 15 (all with 5 hops), one route to node 10 (5 hops), and another route to node 11 (4 hops). Most probably there are other possible routes or nodes present in network, but *S* doesn't know them for the moment. Taking this as an example, Figure 4 shows partial information of a possible energy table of node *S*. Again, there are probably other nodes, such as 3, 4, etc., for which node S has no information about their current battery power. In E2DSR nodes only keep energy values for nodes which are in the routing cache.

	Node ID	Battery Power Level (between 0 and 15)
...	15	8
...	14	12
...	13	7
...	12	8
...	11	12
...	10	8
...	2	11
...	8	12
...	5	11
...	6	9

Figure 4. Snapshot of Energy table of node *S* (incomplete)

2) *Updating Energy Table*

When S finds a new route either by sending a RREQ or by overhearing or by extracting information from forwarding RREQs, it will add this route to the routing cache and update its energy table. For example, assuming that node S overheard the RREP sent by node 10 towards node 4, partially represented in Figure 5, it should add a new entry to its route table and also update its energy table accordingly.

...	4	3	2	10	8	11	12	7	...

Figure 5. Snapshot of a RREP (incomplete)

The first part is related to path nodes and second part has the level of battery power of those path nodes. After receiving the above mentioned RREP, a new entry regarding destination node 10, is added to the routing cache and also two new entries are added to the energy table. These entries refer to the remaining energy of nodes number 4, 3. Also the energy value of nodes 2 and 10, are updated to new values. The level of battery power for node 10 in Figure 4 was 8 which now is decreased to 7 and for node 2 it is increased from 11 to 12 (it shows that this node was charged during this time).

D. *Route Selection Process*

Taking in consideration the new behavior of intermediate and destination nodes, described in section III.B, S can learn several routes to D. Selecting the best route can be done by taking in consideration well known metrics in wireless ad hoc networks, such as delay, jitter, packet delivery ratio (PDR) amongst others. Original DSR, for example, only takes in consideration the length of the route to reduce delay.

In E2DSR a *Route Priority Function* is defined to determine the priority of each discovered route. For a specific source and destination pair, the route which has the maximum priority will be selected as the best route between candidate routes. The *Route Priority Function* has three input parameters with respect to each route: (i) length; (ii) freshness and; (iii) energy of path. These parameters have values normalized between 0 and 1 and will be explained in detail in the next subsections.

The *Route Priority Function* algorithm is shown in Figure 6

```
RoutePriorityFunction {
//R=Number of candidate paths
//From S to D in routing cache of S
For(r=1 to R)
  {
  //Calculate 3 parameters
  //For rth Route
  F= CalculatePathFreshness;
  L= CalculatePathLength;
  E= CalculatePathEnergy;
  PathPriority(r)=CalculatePathPriority(F,L,E);
  If(Max<PathPriority(r)) {
    BestRoute=rth Route;
    Max=pathPriority(r);
    }
  }
Return(BestRoute);
}
```

Figure 6. Algorithm of *Route Priority Function* (from *S* to *D*)

1) *Priority of a Path*

The *PathPriority* function, introduced in Figure 6, uses (2) to compute the priority of a path.

$$PathPriority(i) = \frac{K_F \cdot F(i) + K_E \cdot E(i)}{K_L \cdot L(i)} \qquad (2)$$

Where K_F, K_E and K_L are the coefficients for *freshness*, *energy* and *length* of route respectively. Desirable values for these coefficients, obtained through simulation, are K_F=1 K_E=3 and K_L=1. It's possible to achieve a higher performance regarding a special metric by giving a higher weight to the related parameter in (2). For example, for end-to-end delay the *PathPriority* function can be customized by a higher coefficient for length parameter (K_L), because delay is typically more dependent on the length of routes.

2) *Freshness parameter*

Measuring the freshness of a route is especially important in wireless ad hoc networks due to its dynamic nature. Node's movement constantly changes the validity of a route. Therefore every entry in the node's routing cache should have information regarding how fresh that route is. A route can be old and fresh at the same time, if it was learned a long time ago but used recently. The *F(i)* parameter in (2), indicates the freshness of route *i* and can be measured according to (3):

$$F(i) = \frac{n - i + 1}{n} \tag{3}$$

The freshest route (the one learned or used more recently) has a freshness value of 1 and, for n routes, the oldest route as a freshness value of 1/n. All other routes have freshness values between 1 and 1/n.

3) *Length parameter*

Longer routes can increase the delay in an ad hoc network. Also longer routes have more links, which leads to a higher probability of link breakage. The *L(i)* parameter in (2) indicates the length of route *i* and can be measured according to (4):

$$L(i) = \frac{Length\_of\_Route(i)}{Max\_Length} \tag{4}$$

Length_of_Route(i) is the length, in number of hops, for route *i*. *Max_Length* is the maximum length that a route can have in DSR routing protocol, default value is set to 16.

4) *Energy parameter*

The energy level of a specific route is an important characteristic of that route, since it can assure that a link will not go down due to power issues. However, a route may have nodes that in average have good battery levels and, at the same time, have some nodes with low battery levels. Since there is a high probability of failure in the low battery nodes, this route is undesirable. The *E(i)* parameter in (2) indicates the energy of route *i* and can be measured according to (5).

$$E(i) = \frac{RE(i) \cdot MRE(i)}{M(i) \cdot InitialEnergy^2} \tag{5}$$

M(i) represents the number of nodes in route *i*. *InitialEnergy* is a constant that defines the maximum energy that a node can have. *RE(i)* represents the total of the remaining energy in route *i*. *MRE(i)* is the minimum of the remaining energy between all nodes of route *i*. MRE(i) is important because it will allow the detection of the low battery nodes problem described previously. Knowing that *MRE(i)* is a part of *RE(i)* and is taken into account twice, the E*(i)* calculation presented in (5) will be modified to the one presented in (6).

$$E(i) = \frac{(RE(i) - MRE(i)) \cdot MRE(i)}{M(i) \cdot InitialEnergy^2} \tag{6}$$

IV. PERFORMANCE METRICS

Evaluating a wireless routing protocol involves many varied types of metrics. In this section, we present some common metrics, as well as new ones, that will allow us to evaluate E2DSR.

A. *Average End-to-End Delay (E2E Delay)*

The objective is to determine if the routes chosen by E2DSR are adequate when compared to other protocols, such as DSR or AODV. Longer routes will typically increase the end-to-end delay. In a stream of *n* packets sent by a source node and successfully received by the destination node, the average E2E delay is given by (7).

$$E2E\ Delay = \frac{\sum_{j=1}^{n} End\_Time_j - Start\_Time_j}{n} \tag{7}$$

$Start\_Time_j$ - is the time at which packet j was sent. End_Time_j - is the time at which packet j was received at the destination node

B. *Interarrival Jitter*

Interarrival Jitter determines how stable are the routes in Energy Efficient DSR when compared to other protocols, such as DSR or AODV. This metric also evaluates if E2DSR is suitable for multimedia applications, such as VoIP or Videoconference. The interarrival jitter (J) is defined to be the mean deviation (smoothed absolute value) of the difference (D) in packet spacing at the receiver compared to the sender for a pair of packets. If S_i is the timestamp from packet *i*, and R_i is the time of arrival for packet *i*, then for two packets i and j, D may be expressed according to (8).

$$D(i,j) = \left|(R_j - S_j) - (R_i - S_i)\right| \tag{8}$$

As per RFC 3550 [10], the interarrival jitter should be calculated continuously as each data packet *i* is received from source, using this difference D for that packet and the previous packet *i-1* in order of arrival (not necessarily in sequence), according to (9).

$$Jitter = \mathrm{J(i-1)} + \frac{|\mathrm{D(i-1,i)}| - \mathrm{J(i-1)}}{16} \tag{9}$$

The jitter calculation must conform to (9) in order to allow independent monitors to make valid interpretations of reports coming from different applications. The gain parameter 1/16 gives a good noise reduction ratio while maintaining a reasonable rate of convergence [10].

C. *Normalized Routing Load (NRL)*

NRL is used to determine the overhead caused by E2DSR packets in the network. NRL is the number of routing packets transmitted per data packet successfully delivered at destination. In a stream of *n* packets successfully received by a destination node during a time window *T*, the NRL can be computed according to (10).

$$NRL = \frac{\#E2DSR\_packets\_sent\_during\_T}{n} \tag{10}$$

D. *Balancing of Battery Power Consumption*

This metric will allow measuring the effectiveness of the energy balancing algorithm used by E2DSR. Let's assume that the Energy Load for each node *i*, *EL(i)*, is the relation between the consumed energy in node *i* and the total consumed energy in all the network's nodes. EL(i) can be computed according to (11).

$$EL(i) = \frac{ConsumedEnergy(i)}{TotalConsumedEnergy} \tag{11}$$

The value of the deviation will be the metric for energy consumption balancing of the protocol; the smaller the deviation, the more effective is energy balancing. Figure 7 shows preliminary energy balancing comparison between an early version of E2DSR (called LBDSR [2]) and other protocols. We can see that the early version of E2DSR has a higher performance than DSR. By customizing this early version by giving a higher weight to the remaining energy of routes (LBDSR$_e$), a even higher performance can be achieved.

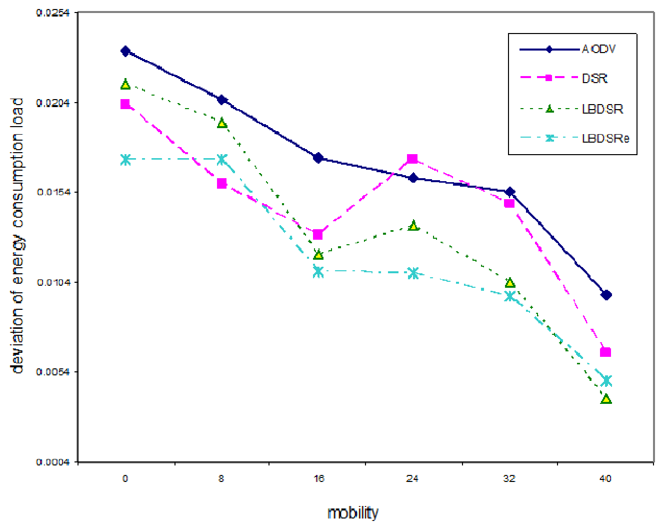

Figure 7. Preliminary results for energy efficiency comparison between an early version of E2DSR and other protocols

E. Node's failure degree

Node's failure degree determines the capability of E2DSR in keeping nodes alive for more time. Node's failure degree is, for a time window *T*, the percentage of nodes in the topology that have failed because of low battery power. This value is calculated according to (12).

$$Node\ Failure\ Degree = \frac{\#failed\_nodes\_in\_T}{\#nodes\_in\_topology} \qquad (12)$$

F. Reaction to Good and Bad News

Reaction to good news measures the capability of E2DSR in detecting new, better routes. If a new better route appears after the discovery phase ends, this metric measures the amount of time needed for this node to discover this route in the worst case scenario. Reaction to bad news measures the capability of E2DSR in detecting that a route is no longer valid. If a route in the routing cache has changed, this metric measures the amount of time needed for this node to reroute traffic, in the worst case scenario.

V. Conclusions and Future Work

This paper presents a new energy efficient routing protocol for wireless ad hoc networks called E2DSR. This new protocol is capable of balancing power consumption amongst different nodes in the network effectively delaying earlier node failure due to battery exhaustion and increasing route reliability. E2DSR uses small routing tables, simpler, but still effective, formulas for route priority computation and a simple on demand discovery mechanism. E2DSR was conceived to be a lightweight protocol so it can also be used in sensor networks.

Currently, E2DSR is being implemented in both simulation environment and real sensors. Validation through simulation is being done with ns-2 [11] since it has proven to provide trusted results and it is also widely accepted by the academic research community. The implementation in real sensors is being done in TinyOS 2.1 [12] and being tested on Telos ultra low power IEEE 802.15.4 compliant wireless sensor modules (revision B)[13]. Code regarding E2DSR implementation in sensors is first done in TOSSIM simulator [14], where it is tested, and then exported to the sensors. Currently we have a testbed of ten sensors and have successfully implemented RREQ, RREP and RERR primitives and *Route Priority Function*, as described in the previous sections. The testing in real sensors is allowing us to fine tune some E2DSR coefficients.

Future work in E2DSR includes a full evaluation of protocol performance, using the hereby described metrics, and protocol scalability, by implementing it in a larger scenario. E2DSR is continuously being fine tuned according to the results received by both our simulation platform and implementation testbed; for example, currently the energy field in E2DSR uses a linear quantization, however our latest studies indicate that if we give more granularity to the lower levels of battery energy, by using a non-linear quantization method, we will be able to achieve a more effective energy balance.

References

[1] V. Talooki, J. Rodriguez, "Quality of Service for Flat Routing Protocols in Mobile Ad hoc Networks", ACM MobiMedia, September, London, UK, (2009).

[2] V. Talooki, J. Rodriguez, R. Sadeghi, A Load Balanced Aware Routing Protocol For Wireless Ad Hoc Networks, 16th International Conference on telecommunications, Morocco (2009)

[3] C. Yu, B. Lee, H. Youn, Energy efficient routing protocols for mobile ad hoc networks, Wireless Communications and Mobile Computing, Wireless Com. Mob. Computing (2003)

[4] S.Singh, M. Woo, and C.S. Raghavendra, "Power-Aware Routing in Mobile AD Hoc Networks,´International Conference on Mobile Computing and Networking, MobiCom, 181-190 (1998)

[5] J.-H. Chang and L.Tassiulas, "Energy Conserving Routing in Wireless Ad Hoc Networks," The Conference on Computer Communications, IEEE Infocom, 22-31 (2000)

[6] A. Zhou, H. Hassanein, "Load-Balanced Wireless Ad Hoc Routing", Proceedings of Canadian Conference on Electrical and Computer Engineering, Vol. 2, 1157– 1161 (2001)

[7] G. Chakrabarti, S.Kulkarni, Load Balancing and resource reservation in mobile ad hoc networks, Ad Hoc Networks, Volume 4, Issue 2, 1 March (2006)

[8] IETEF Draft, The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4, available at: http://tools.ietf.org/html/rfc4728 (2010)

[9] D.B. Johnson, D.A. Maltz, Y.-C. Hu, The dynamic source routing protocol for mobile ad hoc networks (DSR).Available from: http://www.ietf.org/internet-drafts/draftietf-manet-dsr-10.txt (2007)

[10] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A Transport Protocolfor Real-Time Applications, IETF RFC 3550, July (2003)

[11] The Network simulator (NS-2), http://www.isi.edu/nsnam/ns (2010)

[12] TinyOS, open-source operating system designed for wireless embedded sensor networks. url: http://www.tinyos.net/

[13] Polastre, J., Szewczyk, R., and Culler, D., Telos: enabling ultra-low power wireless research. In Proceedings of the 4th international Symposium on information Processing in Sensor Networks, Los Angeles, California, April 24 - 27 (2005)

[14] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in: Proc. of the ACM Int. Conf. on Embedded Networked Sensor Systems (SenSys), 126-137(2003)