Multipath Routing with Erasure Coding for Wireless Sensor Networks

Jian Wu,Stefan Dulman,Paul Havinga,Tim Nieberg
Department of Electrical Engineering, Mathematics and Computer Science, University of Twente, The Netherlands
{j.wu, s.dulman,p.j.m.havinga, t.nieberg}@utwente.nl

Abstract—**Multipath routing algorithm in wireless sensor networks (WSN) increase the reliability of the system at the cost of significantly increased traffic. This paper introduces a splitted multipath routing scheme to improve the reliability of data routing in WSN by keeping the traffic at a low level. Our proposed on-demand multipah routing algorithm offers the data source several paths to any destination. It is used in combination with a data splitting method based on Erasure Coding. The algorithms presented in this paper assures that the gathered data will reach its destination in the network by assuming as a regular fact that nodes may be not available during the routing procedure. Additional energy will be required only for a small amount of computations; this is almost negligible compared with the energy used for communications. It greatly increases the reliability of packet delivery in wireless sensor network, while keep the total network traffic much lower than the traditional multipath routing. At the same time the latency of splitted mulitpath routing is shorter than any retransmission scheme.**

Keywords— **wireless sensor network, multipath routing, reliability, mobility, erasure code**

I. Introduction

In Wirless Sensor Netowrk (WSN), sensor nodes have many failure modes [6], each one of them decreases the performance of the network. Usually, acknowledgements and retransmissions are implemented to recover the lost data. However, these generate large amount of additional traffic and delays in the network, and it gets worst when the failure rate of the node increases. The reliability of the system can be increased by using multipath routing [1]. Multipath routing allows the establishment of more than one path between source and destination, which provides an easy mechanism to increase the likelihood of reliable data delivery by sending multiple copies of data along different paths. Obviously, its drawback is the overall increase of traffic.

In this paper, we present a new multipath routing algorithm Multipath On-Demand Algorithm (further referred to as MDR). MDR is an on-demand algorithm, meaning that a new path from a source to a destination is created only when a data packet has to travel between them. The algorithm provides several paths from sources to destinations. A data splitting algorithm as presented in [1] will be used to safely route data while keeping the amount of traffic low. The algorithm starts by discovering n multiple paths from the source to the destination. Sending the same data over all discovered paths is a solution in case of node failures but it requires large quantities of network resources (such as bandwidth and energy). Our contribution is to develop a new multipath routing algorithm that will discover several disjoint paths between a source an a destination nodes. Then, we will make use of the *Erasure Correction codes* to split the original data packet into k parts (further referred to as *subpackets*) and then compute n-k redundant packets. Finally send these n subpackets instead of the whole packet, across n multipath. The basic principle is to transmit a sequence of n subpackets, out of which only k subpackets are necessary to reconstruct the original packet. The receiver's robustness to missing packets is increased, which also implies that a return feedback channel is not needed anymore.

This work is performed as a part of the European EYES project (IST-2001-34734) on self-organizing and collaborative energy-efficient sensor networks [2]. It addresses the convergence of distributed information processing, wireless communication and mobile computing.

II. Previous work

A. Multipath Routing in Wireless Sensor Network

Multipath Routing allows the establishment of multiple paths between a source and a destination, which provides an easy mechanism to increase the likelihood of reliable data delivery by sending multiple copies of data along different paths.

Several different multipath routing algorithms have been studied by the prior work. The *Temporally Ordered Routing Algorithm* [8] provides loop free multiple alternate paths for mobile wireless network by maintaining a "destination oriented" directed acyclic graph (DAG) from the source. It rapidly adapts to topological changes, and has the ability to detect network partitions and erase all invalid routes within a finite time.

Dynamic Source Routing (DSR) [5] depends on query

floods to discover routes whenever a new routed is needed. Intelligent multipath extensions by Napsipuri and Das [7] have been added to reduce the frequency of routing discovery flooding, while maintaining several disjoint alternate paths between source and destination.

Another candidate for multipath routing is *Directed Diffusion* [4], which features data centric dissemination and in network data aggregation. It can realize robust multipath delivery, empirically adapt to a small subset of network paths, and achieve significant energy savings when intermediate nodes aggregate responses to queries. Based on directed diffusion, a novel braided multipath routing scheme, which results in several partially disjoint paths, is studied by D.Ganesan [3]. Results show it is viable alternative for energy efficient recovery from failures in WSN.

All these algorithms focus on using multipath routing only to reduce the effects of failures. As it is shown in section V our algorithm also makes data transmission less dependent on the average speed of the nodes.

B. Dynamic Source Routing

DSR involves the following phases:

- *Route Request* - the source floods the network with messages, trying to find in this way the destination. The messages increase in length by each hop they travel. If more than one route request messages reach a node, only the first one is processed and the others are discarded.
- *Route Reply* - if the destination receives a route request message from the source it will reply with a message containing the path used to reach the source. In the case of bi-directional links, this path is simply reversed. The reply messages have constant length between the source and the destination. Still, their initial length depends on the number of hops between the source and the destination.
- *Route Maintenance* - after the source has received a path to the destination, it sends the data packet on it. Each node is responsible for ensuring that the message travels to the next hop (this can be done for example by passive acknowledgement [5]). If a node detects that a link is broken, it sends this information back on the path to the source. A new path has to be constructed or another cached path can be used. The length of the messages involved in this phase is dependent on the number of hops between them.

C. Data Splitting across multiple paths

A way of increasing the reliability of the routing mechanism is to send the data packet across multiple disjoint paths. This way, even if some paths will fail, there is a higher probability that at least one data packet will reach the destination. The obvious drawback of this mechanism is the larger amount of traffic used.

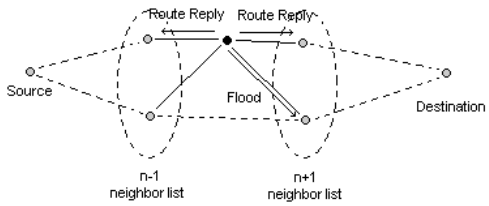

Fig. 1. Algorithm details

We have investigated a way of reducing the amount of traffic [1]. The idea is to split the data packet in subpackets (one subpacket for each disjoint route from the source to destination). By adding redundancy to each subpacket (e.g. by using forward error correction codes) it is possible to reconstruct the original data packet by using a smaller number of subpackets. In this way, even if some paths fail, the original data packet is not lost and the amount of traffic is kept at a minimum.

A requirement of this mechanism is a routing algorithm that will provide several disjoint paths from the source to the destination. This is one of the goals of the MDR algorithm.

III. Multipath On-Demand Routing

The Multipath On-Demand Routing algorithm (MDR) has as a main goal finding on demand multiple paths between the source and the destination while minimizing the amount of traffic in the network. The algorithm follows the basic ideas behind the DSR algorithm. The details of the algorithm are described bellow.

A. Multipath On-Demand Routing

The MDR algorithm has following phases (see Figure 1):

- *Route Request* - when the source wants to find a destination it floods the network with a short message announcing this. The message contains the source ID, the destination ID and the ID of the request. Thus, the length of the message remains constant during the route request.
- *Route Reply* - the destination will eventually receive one of the route request messages. It only knows that there exists a path. It is not interested in what the path is. The destination just returns a route reply to the neighbor from which it received the route request message. The message contains a supplementary field that indicates the number of hops it traveled so far. Each node that receives a route reply, increments the hop count of the message and then forwards the message to the neighbor from which it got the original route request.

This mechanism reduces the size of the messages considerably when compared to the original DSR. In fact we are moving the information stored inside the messages to the sensor nodes themselves. The sensor nodes are responsible to ”remember” where the flooding message came from.

One can notice that there is no route maintenance. This approach will be discussed more in detail after the way the multiple paths are handled and the simulation results are presented.

The second group of modifications involve the multiple paths management. In the original DSR, if the same route request message was received several times by a node, only the first one was considered and the rest were discarded. MDR considers all the messages and uses the whole information it can get out of them.

By using these changes we obtained a controlled flooding in the first phase of the algorithm by using small messages with fixed length. The second phase also uses small fixed length messages that involve only a fraction of nodes existent between the source and the destination.

B. Route Request phase

The Route Request phase is the mechanism by which the source of the data packet notifies the destination that it has a packet for it. The route request message involves all the nodes of the network or at least, only the nodes to which the message arrives before it expires (the message contains a field saying how many hops the message is allowed to travel).

Message description

The route request message contains the following fields:

- *snodeID* the source node ID
- *dnodeID* the destination node ID
- *floodID* the route request message ID
- *lasthop* the ID of the node forwarding this message
- *ack* the ID of the last hop

For the algorithm to work, each node in the network has to have an unique ID. Each message source maintains a counter of the requests sent, such that each route request message in the network is uniquely identified by the first three fields. The *ack* field is needed to distinguish between the messages received by a node. This way, a route request message can be immediately classified as being received for the first time, or being just a passive acknowledgement of a previously sent message.

Route Request phase description

When a source node has to transmit a message to a destination, it first checks its cache to see if there are any routes to that destination that did not expire. If the number of routes found is big enough for the maximum given failing probability of the nodes in the network, it uses them. If not, it generates a new route request message filling the ack field with its own ID.

When receiving such a message, a node checks its local data structure to see if it has received another route request message having the same three fields identical. If not, it creates a new entry in the data structure and stores this information plus the ID of the node from which it received it. From additional messages received the node has to store only the name of the neighbor. It can easily check and mark if the source of the message is a first order neighbor by looking at the lasthop or ack fields.

The node will forward only the first route request message it gets. It has to change only the ack field with the lasthop value and the lasthop with its own ID. After receiving several such messages each node knows who are its neighbors and more than that which ones are closer to the source (further referred as the *n-1* neighbor list) and which one closer to the destination (further referred as the *n+1* neighbor list). In fact, each data structure stores them in two separate lists according to the previous rule. If the node identifies itself as being the destination of the message it initiates the second phase of the algorithm.

C. Route Reply phase

The Route Reply phase is the part of the algorithm in which several paths between the destination and the source are reported to the source (if they exist). The reply messages have fixed length. Because in the previous phase each node stored information about the neighbors that forwarded the route request message, the complete path between the source and the destination has not to be stored inside the message.

Message description

The route reply message contains the following fields:

- *snodeID* the source node ID
- *dnodeID* the destination node ID
- *floodID* the flood message ID
- *lasthop* the ID of the node forwarding this message
- *nexthop* the ID of the node to which the message is forwarded
- *ack* the ID of the last hop
- *hops* the number of the hops the message traveled through
- *detours* the number of detours a message can take

The meaning of the field names is the same as in the previous phase. There are two new fields: the *nexthop* field contains the ID of the node that has to receive this message. This information is provided by each node from their local data structure. The *hops* field is incremented with each hop the message travel and represents the cur-

rent path length. The *detours* field specifies how many times the reply message is allowed to travel in an opposite direction (from source to destination).

Route Reply phase description

When the first route reply message arrives at the source, this node stores the ID of the node that forwarded the message and the path length. It also sets up a timer to measure the interval that it will wait for other reply messages to come. When this timer expires it splits the original data message according to the number of paths, the maximum probability of failure and the length of the paths and forwards it. The paths can also be stored in a local cache (together with time information) for future usage (this feature is not implemented yet).

A node that receives a route reply addressed to it, will modify the last four fields of the message according to the new parameters. Afterwards, it will forward it to the first neighbor in the *n-1* neighbor list. If this list is empty and the detours field is not empty, it chooses the first neighbor in the *n+1* neighbor list and also decreases the *detour* variable by 1. A node that receives a route reply not addressed to it, searches its own data structure to find the entry corresponding to the first three fields. If such an entry is found, it removes the forwarding node from both *n-1* and *n+1* neighbor lists.

A node that forwarded a message has to take care of two more things: first it sets a flag in his data structure saying that it will not forward any other message and second, it waits for the passive acknowledgement. If this does not arrive it assumes that the node to which it send the message is no longer there, is broken or it forwarded a message previously and it deletes it from his lists. It will try resending the message to the next neighbor in the lists, until the lists become empty or the *detour* field becomes 0.

The previous step of removing nodes from the list is needed to ensure that the source will receive only disjoint paths. If for various reasons, the paths from the destination to the source have to be known, each node that forwards a route reply message can append its ID to it. This way, the messages will grow in length, but this growth is controlled and involves only a subset of the nodes.

IV. Erasure Correction

The design of error correction meets the requirements of our split-multipath scheme. The Erasure correction code (EC) described in this section is based on the well know Reed-Solomon error correction code (RSC).

RSC codes are linear block codes, which are often denoted $RS(n, k)$ with s-bit symbols. The encoder takes k data symbols and adds check symbols to make an n symbol codeword. RSC codes correct up to t errors in a codeword where $2t = n - k$. For a symbol size s, the maximum codeword length (n) is $n = 2s - 1$. Because a RSC codes correct symbol errors, they can potentially correct many bit errors. This makes RSC code very good at correcting large clusters of errors. Moreover if the position of the error is known (error which is called erasure), then the decoding procedures can correct up to $2t$ erasures. It means that RSC could correct the same number of errors as the redundancy added. In EC, when a data packet arrives, it is divided into k subpackets each with L bits. Then these subpackets were put into a two-dimensional array with $k \times L$ bit as shown in Figure 2. Further let $L = L' \times s$ and every s bits form a symbol in finite field GF(2^s). The encoding could be carried out in two stages. The outer-codes are Reed-Solomon codes over GF(2^s) which protect against subpacket loses. Each column of information symbols in GF(2^s) is encoded into a code word of $C_0(n, k)$, where the number of redundant symbols $R = n - k$. In total there are L' outer code words in this array. Then a header h is added to each row, which keep the index of each subpacket and the number of padding added. The inter encoding is optional which gives extra reliability over link errors. A binary BCH code could be used for each row as an inner correction code. After the EC encoding, each row of subpacket is sent on n different path established by the multipath routing algorithm. As along as more than k of them are received in the destination node the EC is able to reconstruct the original packet.

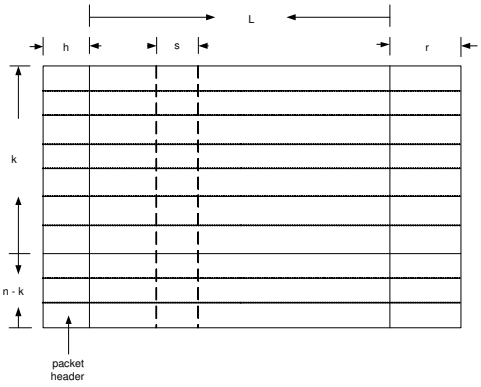

Fig. 2. Turbo Erasure Correction Code

The desired characteristics of the EC are summarized below:

- BURST CORRECTION: Errors occurs because of link failure. Normally the whole subpacket is lost instead of bit errors.
- ERASURE CORRECTION: The index in the subpacket header help to location the error in the decoding, which resulting in erasures,
- ADAPTABILITY: The number of multipath degree and

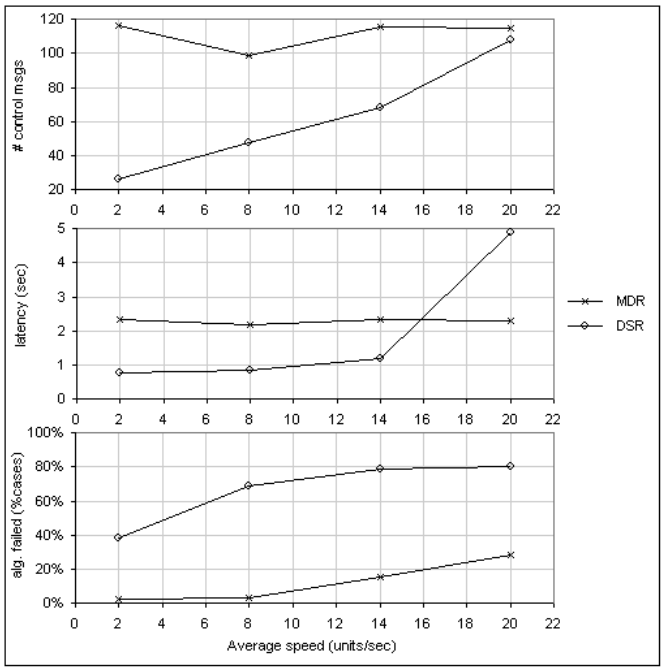

Fig. 3. Comparison MDR/DSR

link quality channel varies over a wide range in a short period of time. TEC can adapt to the changes quickly.

V. Simulation and Results

This section evaluates our MDR algorithm. In order to get a better understanding of its performances, we have compared it with an implementation of DSR. A second group of simulations focused only on our algorithm, trying to find out how to tune the variables it involves to get the better results out of it.

A. Comparison with the DSR algorithm

We have given a brief description of DSR in II-B. We have implemented a round based version of this algorithm. We have run both DSR and MDR for several network configurations. The parameters were identical for both cases and also the generation of destinations. The DSR algorithm had the caching of the paths and the route maintenance enabled. The results are presented in Figure 3.

The figure shows that the number of overhead messages is higher for the MDR algorithm. A closer look at the message sizes shows that the MDR traffic compared to the DSR traffic varies from a 4.04:1 to a 1.02:1 ratio (from the lower average speed to the higher one).

After the paths are created, the source will deliver one data packet that takes one round to travel through one hop. In this case, the latency of DSR is smaller with low mobility. With the increase of speed, the situation changes. In practice we assume data packets far larger than control messages. As future work we are going to investigate the latency from this point of view as well.

The last graph in Figure 3 shows the average number of failed cases for the two algorithms. The MDR algorithm performs way better than DSR. The figure shows clearly the two objectives of our algorithm: it improves the reliability a lot and it makes the network almost immune to higher average speed of the nodes.

A failed case is the situation in which the source had a data message to deliver to the destination but failed reaching it. There are two reasons for it:

- the route discovery mechanism did not return any valid paths between the source and the destination;
- although there were several paths available, the data packets got lost on the way (due to mobility issues).

The MDR algorithm performs way better than DSR, so this is the advantage for which we pay with higher number of control messages and higher latency.

B. Influence of mobility

The first set of experiments took into consideration different degrees of mobility. The metrics taken into consideration were: number of control messages exchanged, number of paths discovered, latency, number of times when the multipath algorithm failed and number of times no data packet was received. For a given network setup we have varied the average speed of the nodes and noticed how these parameters varied. All the experiments were repeated for different transmission ranges of the nodes. It is interesting to notice that when the transmission range was small (100) the network was quite often partitioned. This is why the curves for this value usually have a different shape.

Number of control messages

The curves in Figure 4 represent the total number of control messages divided at the number of cases when at least one data packet was delivered between the source and the destination. As one can notice, the mobility does not introduce major differences in the graphs; the number of control messages usually increases with the increase of mobility. It is interesting to notice that at speed 8 there is a certain equilibrium condition met between the message generation rate, the round time value and the speed of the nodes

Multipath degree

Figure 5 presents the average number of paths discovered between the source and the destination. It is related to the average connectivity of the nodes and to their average speed. As expected, it decreases with the speed and increases with the transmission range.

Latency

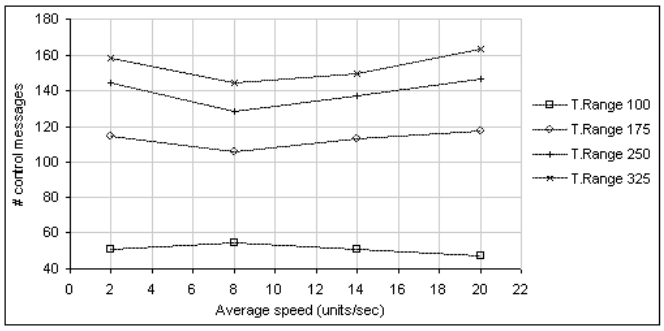

Fig. 4. Number of control messages

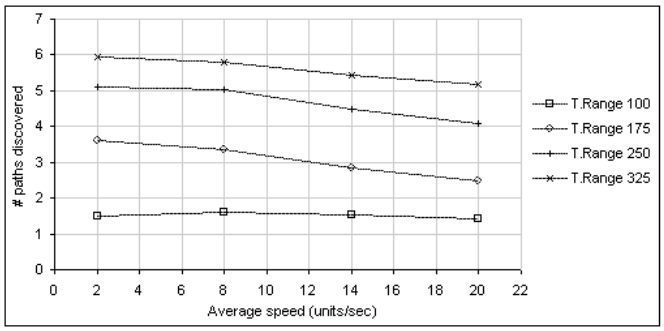

Fig. 5. Number of paths discovered

Figure 6 shows the latency introduced by this algorithm. As in the previous graphs, the speed does not have such a big influence. From the point of view of transmission range, the higher curve corresponds to the smallest transmission range value (the diameter of the network is bigger, implies that the average distance between the source and a possible destination is also bigger).

Algorithm failure cases

We have recorded two different variables. The first one (shown in Figure 7) represents the total cases when the route discovery algorithm failed (there were no replies received at the source from the destination). The second variable (see Figure 8) deals with the number of data deliveries failed. This assumes that there was at least one path found from the source to the destination, and it (or all of them) failed during the data transmission. As one can see, there are very poor results for low transmission range. This is in principle due to the cases when the network was partitioned. More than that, the speed influence can be reduced by increasing the transmission range (so, by increasing the average connectivity). For higher connectivity values, the speed has almost no influence at all. This is a main difference between MDR and DSR. For higher mobility, DSR becomes almost unusable while MDR still performs better.

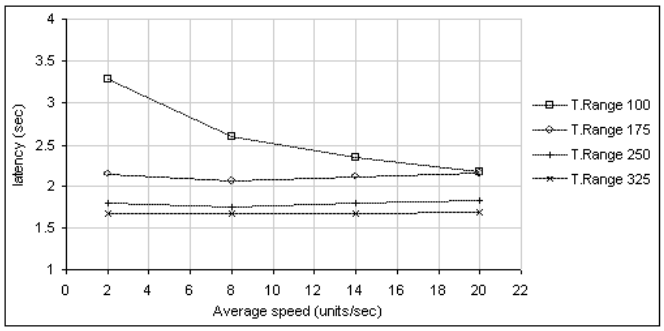

Fig. 6. Latency

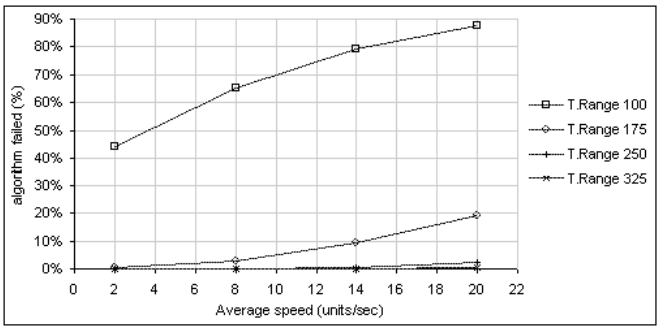

Fig. 7. Algorithm failed cases

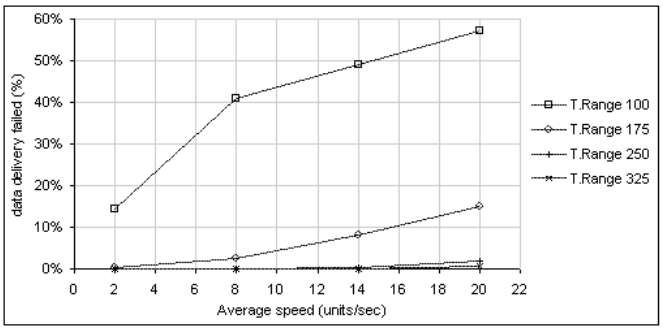

Fig. 8. Data delivery failed cases

C. Waiting time tuning

After receiving multiple route replies messages, the source can decide how to split the data packet and forward it in order to obtain maximum reliability and lower amount of traffic. There is a certain time interval in which the source waits and stores the route replies (further referred to as the waiting interval).

The waiting interval is a key factor for the algorithm. Considering an ideal network in which the nodes do not fail in any way, there is an optimum value for this waiting interval. When using a shorter interval, additional routes discovered are discarded. When using a bigger one, the probability for routes to expire increases. For example, Figure 9 and Figure 10 present the results of varying the waiting interval for different average speed values (the number of paths discovered and the total number of failed

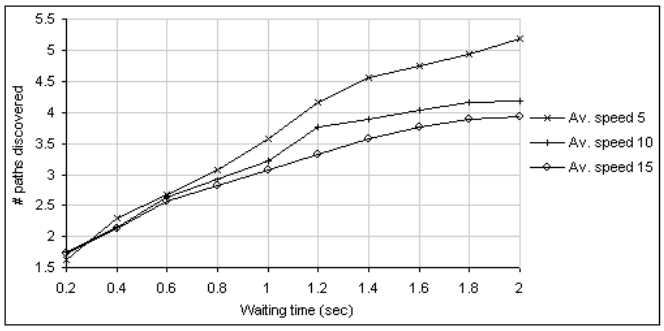

Fig. 9. Number of paths discovered

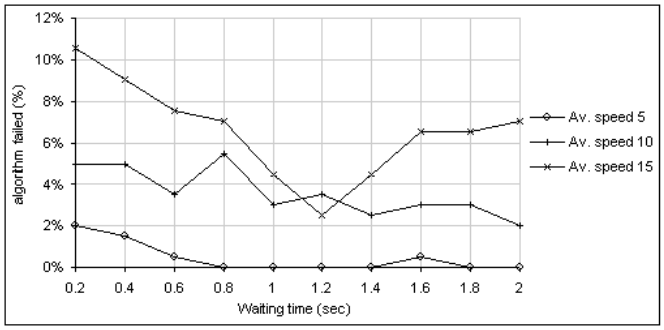

Fig. 10. Algorithm failed cases

algorithm cases). The case of average speed 5 illustrates better the previous idea.

D. Failures

The algorithm we presented was designed having in mind the following reasoning: multipath routing should provide superior reliability to single path routing because it uses several additional paths to deliver the data from the source to the destination. During this process, even if some of the paths fail, the data packet can still be reconstructed at the destination if enough redundancy was added to each subpacket. No acknowledgements are necessary. If a data splitting mechanism the overall latency decreases with the increase of the data packet length (this being paid back by the larger communication and processing overhead).

For our simulations we have considered only the case of temporary communication failures. Analyzing the results of the simulations we have come to a very surprising conclusion (and quite obvious afterwards...): not only the data packets are affected by errors, but also the route discovery messages and also the passive acknowledgements!

When integrating the failures into the simulation, the algorithm is still superior in performance to DSR, but there is quite a difference between this and the ideal situation presented above.

The first method that comes to mind to combat the effects of failures is sending each control message several

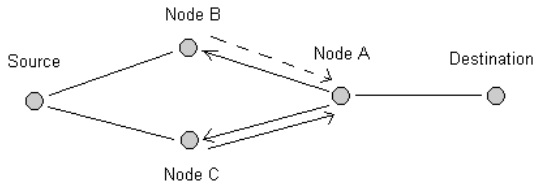

Fig. 11. Node A fails getting a passive acknowledgement

times (2 times in our case). When we applied this to the route request messages, we have obtained certain improvements (up to 8.57% increase in successful data delivery).

When it comes to repeating the route reply messages, the effect is almost negligible. An interesting effect appears. Assume the configuration in Figure 11.

Let us suppose that the route request phase took place and right now we have the route reply phase. Node A selects his first *n-1* neighbor (Node B) and sends him the route reply packet. Node B performs his function but for certain reasons Node A does not get the passive acknowledgement. It assumes that the link is broken and continues with the next neighbor on the list. In the end, the source receives two paths to the destination. If there are other intermediate nodes between the source and Node B or Node C, the source cannot determine from the route reply packet that in fact the two paths are not disjoint.

In this case, retransmitting messages does not solve anything. Even if Node A retransmits the message for Node B, the second one just ignores it (according to the algorithm). We could modify the algorithm to introduce some sort of Acknowledgement mechanism, but if this one fails as well we still end with braided multipaths.

The solution lies in modifying the Route Reply Message. Each node on the return path should add its own ID to the message and give this way the possibility to the destination to remove the braided paths. This is the main objective of our future work.

VI. CONCLUSIONS

This paper introduced the Multipath On Demand Algorithm (MDR). It is a routing algorithm that offers to the data source several paths to any destination (if available). This algorithm is intended to improve the reliability of data routing in sensor networks by keeping the traffic at a low level. It is used in combination with a data splitting method based on Erasure Coding.

We have implemented a round based-version of this algorithm and estimated the main characteristics. It performs better in terms of failures and mobility compared to the DSR algorithm. It greatly increases the reliability of packet delivery in wireless sensor network, while keep the

total network traffic much lower than the traditional multipath routing. At the same time the latency of splitted mulitpath routing is shorter than any retransmission scheme

There were several interesting observations that came up at the end of this study. The most important are:

- the speed of the nodes has a reduced influence on all the parameters of the algorithm. A way of diminishing the effects of mobility is ussually increasing the transmission range of the nodes. This implies a higher energy consumption. By using multipath routing, this is not necessary. This means that the same results can be achieved with a lower amount of energy.
- the failures can affect also the control messages. The solution to this problem is modifying the route reply phase or even change it completely with the route reply phase of the DSR algorithm. Retransmitting the control messages is more a waste of energy than a reliable solution, so other solutions have to be explored.

The future work will focus on integrating path estimation in the MDR, so that the failing probabilities of each node could be obtained in the routing process. Also we have in mind modifying the Route Reply phase to better deal with failures. This will allow also caching of routes also. The effect of caching the routes and maintaining them has still to be determined.Our scheme, although focused on WSNs, can be incorporated into any routing scheme to improve reliable packet delivery in the face of a dynamic (wireless) environment where nodes move and connections break. Also as future work is implementation of our algorithm without using the message rounds. This will allow adding of a MAC layer and quantifying its effects. Still to be investigated are the effects of transmitting large data messages and re-quantifying the control messages overhead. A new more realistic traffic generator has to be employed.

REFERENCES

[1] S. Dulman, T. Nieberg, J. Wu, and P. Havinga. Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks. In *Proceedings of the Wireless Communications and Networking Conference*, 2003.

[2] EYES project. website. `http://eyes.eu.org`.

[3] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(4):11–25, 2001.

[4] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. Sixth Annual International Conference on Mobile Computing and Networks*, 2000.

[5] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[6] S. Marti, T.J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehaviour in mobile ad hoc networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, pages 255–265, 2000.

[7] Nasipuri and S. Das. On-Demand Multipath Routing for Mobile Ad Hoc Networks. In *8th Intl. Conference on Computer Communications and Networks (IC3N 99)*, 1999.

[8] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of IEEE INFOCOM'97 Conf.*, April 1997.