Reliable Splitted Multipath Routing for Wireless Sensor Networks

Jian Wu, Stefan Dulman, and Paul Havinga

EEMCS Faculty, University of Twente, the Netherlands
{jian,dulman,havinga}@cs.utwente.nl

Abstract. In wireless sensor networks (WSN) the reliability of the system can be increased by providing several paths from the source node to the destination node and by sending the same packet through each of them (the algorithm is known as *multipath routing*). Using this technique, the traffic increases significantly. In this paper, we analyze the combination between a new multipath routing mechanism and a data-splitting scheme that results in an efficient solution for achieving high delivery ratios while keeping the traffic at a low value. Simulation results are presented in order to characterize the performances of the algorithm.

1 Introduction

Sensor nodes have many failure modes [4], each one of them decreasing the performance of the network. The algorithms presented in this paper will assure 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 [5].

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 *Turbo 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.

H. Jin et al. (Eds.): NPC 2004, LNCS 3222, pp. 592–600, 2004.

2 Multipath On-Demand Routing

Multipath Routing allows the establishment of multiple disjoint paths 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. Based on Dynamic Source Routing (DSR) [3], we designed a new multipath routing algorithm Multipath On-Demand Algorithm (further referred to as MDR). The algorithm provides several paths from sources to destinations. A data splitting algorithm as presented in Section 3 will be used to safely route data while keeping the amount of traffic low. The two phases of the algorithm are described below.

2.1 Route Request Phase

The source initiates the Route Request phase by sending a request message to notify the destination that it has a packet for it. 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

Each node in the network has an unique ID and 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. In 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.

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.

2.2 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). Because each intermediate node keeps information about the neighbors, the complete path between the source and the destination has not to be stored inside the route reply message, which 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 *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 current path length. The *detours* field specifies how many times the reply message is allowed to travel in an opposite direction (from source to destination).

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. 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 re-sending 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.

3 Data Splitting Across Multiple Paths

The route discovery process of MDR provides multiple disjoint paths between source and destination. In this section we try to predict the number of paths that will succeed in delivery subpackets. Furthermore, we will give an approximation that allows for a term that can be used to increase the chance of successful delivery of the entire message at the trade-off of added redundancy.

Of course, one can send the whole message along each of the available paths, but the overhead induced by this will be too high. The entire data package to be sent from the source to the destination over the available n disjoint paths will be split up into smaller subpackets of equal size with added redundancy. The number of created subpackets corresponds to the number of available paths. Only a smaller number of these subpackets will then be needed at the destination to reconstruct the original message. In the following, we will focus on approximating a value k that gives, with high probability, the number of successful paths. This value will then be used to determine the amount of redundancy to be added for the split message transmission in Section 4. The total number of subpackets as well as the added redundancy is a function dependent on the multipath degree and on the failing probabilities of the available paths. As these values change according to the positions of the source and the destination in the network, each source must be able to decide on the parameters for the error correcting codes before the transmission of the actual data subpackets.

Suppose we want to send a data package from a source to a destination and the process of MDR is finished with k different paths whose reputation coefficient is sufficient. Each path has some rate $p_i (i = 1, \ldots, n)$ that corresponds to the probability of successfully delivering a message to the destination. This setting corresponds to a repeated Bernoulli experiment. So the estimated number k of successful path is given by

$$k = \sum_{i=1}^{n} p_i$$

When to estimate the number of successful path k, we consider the possibility of a packet failure independent of the packet size. The effect of packet size on the failure possibility depends on many different factors in the system. One large packet is better for MAC layer and OS scheduling, however has higher chance of failure during physical transmission. Many small packets put more burdens on MAC layer and OS scheduling, while each of them has better chance of success in physical layer. So in this paper, we do not consider the influence of packet size on the failing possibility. According to the previous work in [1], we gave an estimate

for the number of successful paths with a given value α that serves as a desired bound on the delivery. Given the probabilities of failure for n disjoint paths in the multipath routing, the (maximum) number k of paths that are successful in 1-α of all deliveries can be expressed as

$$k = x_\alpha \cdot \sqrt{\sum_{i=1}^{n} p_i(1-p_i)} + \sum_{i=1}^{n} p_i$$

x_α is the corresponding bound from the standard normal distribution for different level of α (see Table 1 for some values). Obviously, the estimated value for k corresponds to a bound of α=50%. In this case, the coefficient x_α is 0, and we obtain the same estimation as in the previous expression.

α	5%	10%	15%	20%	50%
x_α	-1.65	-1.28	-1.03	-0.85	0

Table 1. Some values for the bound α

4 Turbo Erasure Correction

The design of error correction meets the requirements of our split-multipath scheme. The Turbo Erasure correction code (TEC) 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 TEC, 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 1. 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 TEC 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 TEC is able to reconstruct the original packet.

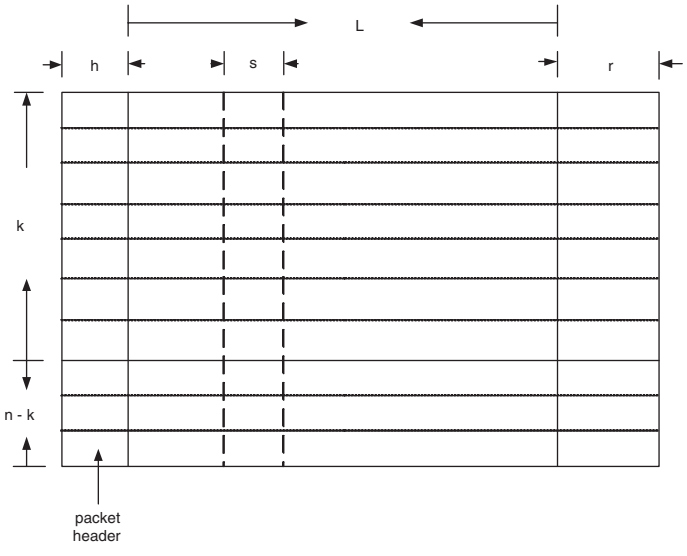

Fig. 1. Turbo Erasure Correction Code

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

5 Simulation and Results

We have implemented the simulation of MDR algorithm and try to quantify the amount of overhead it introduces versus the improvements obtained. The simulations were performed using our own mobility framework designed for the OMNeT++ simulator.

We have considered 50 nodes randomly distributed inside a rectangle surface (500 by 800 units). For the movement of nodes we used the Random Way Point algorithm. The average sleep time of a node was chosen 5.5 seconds. We are assuming that all the links in the network are bidirectional. Sets of up to 10 simulations were performed for different combinations of the average speed of nodes

and transmission ranges. Usually, the speeds were considered in the interval 2 - 20 units/second and the transmission range in the interval 100 - 325 units. One of the nodes defined as being the source node, and randomly chose a destination each 0.5 seconds to forward a message to it. Each simulation had a limit of 200 seconds (in fact after 200 seconds the source stopped generating requests and the simulations ran until all the messages were exhausted in the network).

The main parameters considered were the number of messages, the amount of traffic generated, the latency introduced and the reliability of the algorithm. An implementation of DSR with caching of the paths and the route maintenance enabled was also implemented for comparison. 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 2.

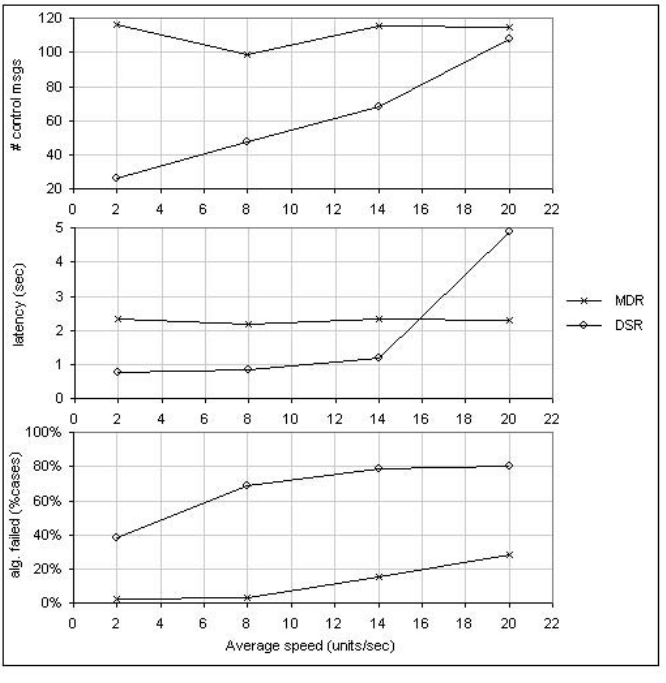

Fig. 2. Comparison MDR/DSR

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 2 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.

6 Conclusions

This paper introduced a splitted multipath scheme to improve the reliability of data routing in wireless sensor networks by keeping the traffic at a low level. An on-demand multipah routing algorithm offers the data source with several paths to any destination (if available). It is used in combination with a data splitting method based on Turbo Erasure Coding.

We have implemented this scheme and estimated the main characteristics. 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

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.

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] 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.
[4] 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.
[5] G.J. Pottie and R. Molva. Embedding the internet: wireless integrated network sensors. In *Communications of ACM, 43(5), 51–58*, 2000.