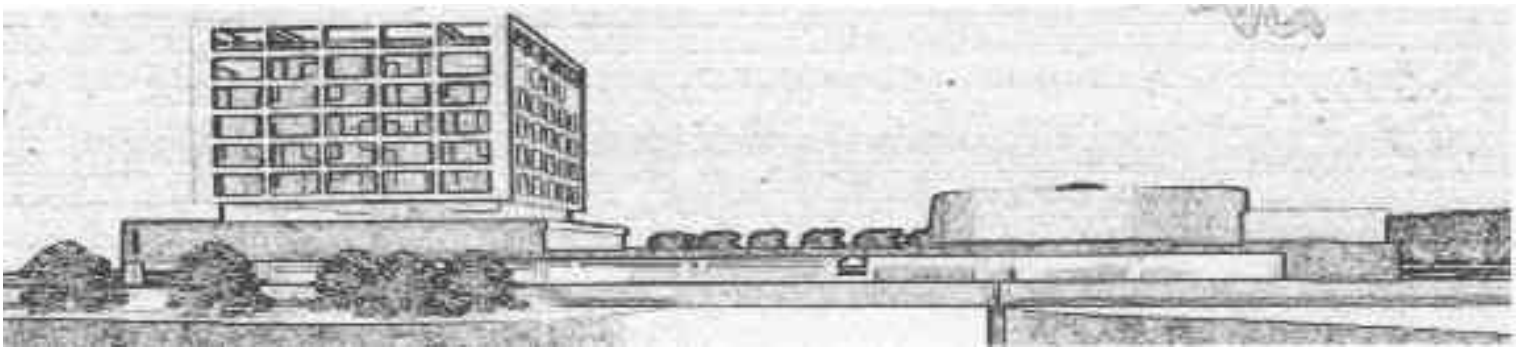
VOLUME V, NUMBER 1



REPORTS ON SYSTEMS AND COMMUNICATIONS

Design and Implementation of an Ad-Hoc Routing Protocol for Mobile Robots

Carlos Agüero, Vicente Matellán, José María Cañas and Pedro de-las-Heras-Quirós

Móstoles (Madrid), December 2005
Depósito Legal: 12345
ISSN: 67890

Table of Contents

Communications

Design and Implementation of an Ad-Hoc Routing Protocol for Mobile Robots *

Carlos Agüero, Vicente Matellán, José María Cañas, and Pedro de-las-Heras-Quirós

Universidad Rey Juan Carlos
C/ Tulipán s/n, Móstoles (Madrid, España)
{caguero,vmo,jmplaza,pheras}@gsyc.escet.urjc.es
Robotics Group

Abstract. Mobile robots need to be able to communicate among them, as well as with hosts participating in the task all of them are involved in. Wired networks are obviously not suitable for mobile robots. Current wireless networks based on fixed infrastructure (GSM, WiFi, etc.) to route packets are neither suitable because this infraestructure does not cover every place. The best alternative for mobile robots are Ad-Hoc networks, which are wireless networks that do not need a fixed infrastructure. This article describes PERA, an adaptation of a well-known Ad-Hoc routing protocol for mobile robots with reduced communications capabilities. This protocol has been iplemented and tested on *Eyebot* mobile robots. Robots using PERA can send messages to other robots or hosts that are not directly reachable through its radio antenna covarage, by routing messages through intermediate mobile robots also running PERA. The design, implementation, testing and lessons learned in development of PERA are presented in this article.

keywords: mobile robots, communications, ad-hoc networks, protocol

1 Introduction

Communication capabilities are nowadays an indispensable component of any robot, both to let human users interact with them, or to let groups of robots to communicate among them. Obviously, mobile robots require wireless technologies.

Wireless networks can be classified into two groups attending to their dependence of fixed infrastructure: Infrastructure based networks and *Ad-Hoc* networks. In the first ones, mobile nodes communicates through different types of infraestructure, large antennas in cellular telephony are a classical example, or Access Points (AP) in the case of WiFi. That is, the mobile node sends information to a fixed network, where the location (anntena or AP) where the other mobile node can be located has been kept, and the information is sent there.

* Work suported in part by CYCIT grant No. TIC2001-0447 and by Comunidad Autnoma de Madrid grant No. 07T/0004/2001

The second ones does not use any fixed infrastructure, the mobile nodes are themselves routers, and the whole group of robots will be incharged of delivering data to destination. This kind of technologies are very useful in many situations: Let's imagine a group of mobile robots working in a rescue situation [11]. In this kind of environment the communication infrastructure may had be severely damaged, so robots could only trust on their own capabilities to communicate among them. If a robot would find a victim, it should be desirable that it would be able to send for instance, images to the mobile host where a human operator is supervising the rescue mission. Tipically, this operator will be out of the robot's radio range, so it will need other robots to relay its data.

Classical routing protocols as the ones used in fixed networks (i.e. IP in Internet) are not well suited for Ad-Hoc networks, because their routing tables do not stabilize under frequent changes in connectivity by the mobile nodes.

Besides the routing protocols problems, most extended wireless technologies, i.e. WiFi (IEEE 802.11 standard), cannot be used in many robots. For instance, very small robots hardly can carry the hardware, and more important, they cannot afford the battery cost of Wi-Fi technology, neither the computational power required, nor the communications required (adresses space for instance). In those robots propietary radio communications are usual.

In summary, routing protocols used in wired networks are not a good choice to communicate mobile robots. These protocols assume that the network is fixed, that the batteries are always full and they also assume that the bandwidth is constant. None of these assumptions do hold in mobile robots. Alternative protocols have been proposed that have better efficiency because they take into account the special features of robots. We can divide them into two different groups: based on *routing tables* and based on *on demand* routing.

Robots using protocols based on routing tables have to maintain a table that allows them to route to any destination. This kind of protocols send lots of routing information for updating changes in the network connectivity. These are examples of this category of protocols: DSDV (*The Destination-Sequenced Distance-Vector Routing Protocol*) [10], CGSR (*Clusterhead Gateway Switch Routing*) [12] and WRP (*The Wireless Routing Protocol*) [7].

On the other hand, protocols based on *on demand* routing only store in their tables the routes that are really needed. When a packet addressed to an unknown destination will be received by a robot, a route discovery process will be initiated on demand in order to learn a new route. A route maintenance process is also needed to update the routes learned and to delete unused routes. Some examples of this category are: AODV (*Ad Hoc On-Demand Distance Vector Routing*) [8], DSR (*Dynamic Source Routing*) [6], LMR (*Lightweight Mobile Routing*) [3], TORA (*Temporary Ordered Routing Algorithm*) [8], ABR (*Associative-Based Routing*) [14] and SSR (*Signal Stability Routing*) [5].

In this paper we describe PERA [1], which is an ad-hoc protocol designed taking into account the special requirements of small mobile robots. PERA is a protocol inspired in the AODV (*Ad-Hoc On-Demand Distance Vector*) [8] routing protocol. PERA implementation has been programmed as a library that

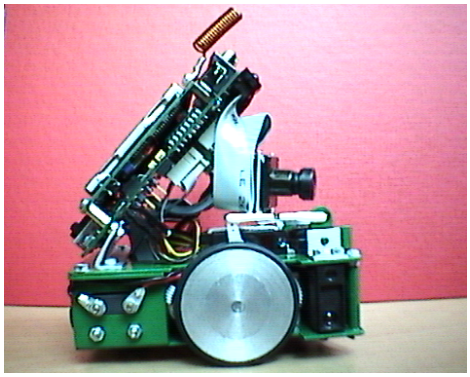


Fig. 1. *Eyebot* robot

can be used by robot applications that need to exchange messages among robots with independence of the radio scope of the robot.

The implementation of PERA described in that paper has been tested on the *Eyebot* mobile robot (figure 1). *Eyebot* robots are equipped with three infrared sensors, two encoders and a camera. In addition, robots are supplied with a radio communication module that is used by PERA. All those devices are managed by the *RoBios* operating system. The PERA library is built using the *RoBios* API.

The rest of the paper is organized as follows: the PERA protocol specification is presented in section 2. Section 3 describes the design of the PERA library. Finally, the implementation, and the tests made on the Eyebot mobile robots are described on section 4.

2 The PERA Ad-Hoc routing protocol

PERA has been designed to fulfill the following requirements:

- Every robot should be able of sending data to any other robot.
- Every robot should be ablo of receiving data from any other robot.
- Movement of robots has to be allowed without disturbing ongoing communications.
- Multiple applications running concurrently on the same robot can use PERA in order to send / receive data independently of each other.
- Every robot should be able of sending data to a particular application running on a given robot (end-to-end communication).
- The library providing the PERA protocol should allow to choose between unicast and multicast transmission.

These requirements are not always possible to fulfill, in particular, when a path of intermediate robots is not available between the source robot and the destination robot of a message, obviously PERA can not help in this point.

Besides the previous requirements, that refers to the protocol itself, the implementation is going to be tested on *Eyebot* robots. The major limitations that *Eyebot* robots pose to PERA lie on the *RoBios* OS. For instance, the maximum size of a data packet is limited to 35 bytes when sending between adjacent robots.

Some protocols based on *on demand* routing include the complete path in each data packet. This path is included in the packet when it is first sent, so that intermediate nodes can route the packet by consulting the path included on it. Due to the small size of data packets, this option was discarded.

PERA uses a protocol based on *on demand* routing that is table driven: each mobile robot maintains a table with routes. Packets only contain the address of the destination robot. Any intermediate robot which receives a packet which is not addressed to it, will consult its table in order to choose the next hop. Contrary to what happens in protocols based on routing tables, as the ones used on the Internet, the protocol used in PERA only updates these tables on demand. Tables are also created on demand.

So, there are two main tasks that the routing protocol must fulfill: route discovery and route maintenance, which are used, respectively, to create an entry in the table when a packet must be routed to an unknown destination, and later, for keeping updated a given route in case it is still needed. For this reason, each entry in the routing table can be erased or updated.

The lifetime field, combined with a sequence number field ensure that a robot does not use old routes, and that routing cycles do not exist.

2.1 Route discovery

This process will be triggered by a robot when it would like to send a packet to another robot and there is not an active route for the desired destination on its routing table.

The process is started by the sender that composes an RREQ (*Route Request*) packet, which includes the identifier of the sending robot and a locally generated RREQ ID, togehter these will identify the request uniquely in the net of robots. Then, the robot broadcasts this message. Closeby robots that receive this RREQ must rebroadcast the packet. By flooding this initial message, the route discovery process ensures that the destination, in case it is reachable through any existing route, will be reached by this RREQ message.

All robots must check the ID and the origin of the RREQ in order to avoid unnecessary flooding. This way an RREQ that has been previously received and resent by a given robot will not be sent again. The RREQ ID is incremented each time a new route discovery is initiated, so that when the conditions of connectivity change a new route discovery will not be discarded by intermediate robots.

Each time an intermediate robot receives an RREQ, it learns the reverse route to the source of the RREQ: the next hop is the neighbour robot that has sent this RREQ to us (we assume symmetrical links).

If an RREQ is received by the final destination, that robot will send back an RREP (*Route Reply*) packet addressed towards the source of the RREQ received.

This RREP packet is sent following the reverse route that followed the RREQ. This reverse route has been already learned and stored by the intermediate routers when the RREQ was flooded. When the RREP reaches an intermediate robot, it learns the reverse route towards the origin of the RREP, and stores it on its routing table. Note that this is exactly the routing information that was originally seeked for by the robot that initiated the route discovery.

An optimization that accelerates the pace of route discoveries is used in PERA. When a robot receives an RREQ, even if it is not addressed to him, it can reply with an RREP in case it already knows a route to that destination. The advantage of this hack is that RREQ's don't need to be flooded everywhere in the net of robots, in case someone already knows a route that is being discovered.

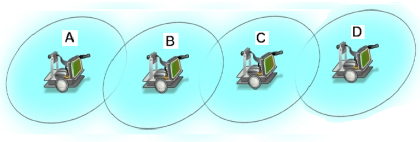


Fig. 2. Route discovery

Let's us consider a simple example to illustrate the process. For instance, consider the scenario of figure 2, where circles represent the radio range, and were all routing tables are empty. In this situtaion, robot A wants to send some information to robot D. First, A needs to discover a route towards D. It is thus necessary to initiate a route discovery process. Robot A creates a RREQ packet. This packet contains the source node address (A) and the current sequence number at node A, as well as the destination address (D). The RREQ also contains a broadcast ID (1), that is incremented each time the source robot initiates a new RREQ.

After creating the RREQ, robot A broadcasts the packet. When neighbour robot B receives it, it first checks whether it has seen this RREQ before, by checking the source address and broadcast ID pair. Each robot maintains a record of the source address / broadcast ID for each RREQ it receives.

In this example, robot B processes the packet. Robot B learns how to route packets to A and stores this information on its routing table. Then, robot B broadcasts the RREQ to its neighbors. This second RREQ is received by robot A, that silently discards the packet because it recognizes it as a packet already broadcasted by him (in fact A was the originator of this RREQ). However, robot C, that also receives the RREQ sent by B, rebroadcasts it to its neighbors, after storing on its routing table a new route towards A that passes through neighbour B. When robot B receives the RREQ broadcasted by C, it discards the packet.

When node D receives the packet broadcasted by C, it learns a route towards A, that passes through C. Then, it unicasts an RREP packet back to the source A. Now, node D already knows to which neighbour it must send the RREP

addressed to A, that is C. The RREP then reaches C, that in this way learns a route towards node D, and stores it on its routing table. Then, C checks its routing table and there it sees that the RREP addressed to A must be sent to B. When B receives the RREP, it learns a route towards D (through C) and stores it on its routing table. Finally, after checking its routing table, B sends the RREP directly to A. When A receives the RREP, it finally stores on its routing table a route towards D (through neighbour B).

This concludes the discovery process in our example, once A has finally learned a route towards D. Now A can send the data to D, using a DATA package type (see section 3 to know the format of this package). Each time a robot sends a packet to a neighbour, it expects to receive an ACK from it. In this way, source robot knows its packets are being routed.

2.2 Route maintenance

If a route is affected by the movement of an intermediate robot, an RERR (*Route Error package*) packet will be sent towards the source of data in order to inform him that the route is no longer available. This RERR is sent by the robot that is one hop before the culprit, when it sends packets to the culprit and does not receive an ACK for them (because the culprit is too far away after moving).

When a neighbour receives a RERR, it deletes its routes towards the unreachable robot, and then propagates the RERR. When an RERR is received by its destination robot, it initiates a new route discovery.

Neighborhood information is learned through broadcasts sent by neighboring robots. Each time a robot receives a broadcast from a given neighbour, it updates a lifetime field associated with that neighbour in its routing table. If at that time there is no entry for that robot in the table, the robot creates one. In addition, all robots broadcast a *Hello* packet to inform its neighbors periodically that it is still in the vicinity. Lifetime field is decremented as time passes.

3 Design of the PERA library

We have built a communications library that can be used to send messages between any pair of robots in a herd, even when they are not directly reachable. This functionality drastically increases the possibilities of communication between Eyebots provided by the *RoBios* library.

The PERA library is structured in hierarchical modules following a traditional communications stack architecture. Each level in the hierarchy provides services to the level above, and uses services of the level below through well defined interfaces. In this way it can be easily ported among different types of robots.

In PERA we have consdered four levels, ordered from lowest to highest in the hierarchy: link, net, transport and application, following the TCP/IP architecture. Each layer has an independent goal explained in the next subsections.

3.1 Link layer

The service this level provides to the net layer is a transmission channel between neighbour robots. This layer is the only one that depends on the type of robot. In case we want to use the PERA library with other robots (*Lego, Pioneer,...*), other links layer must be implemented, adapted to the physical communication channel.

The functions of this level are to send and receive data to / from robots that are directly reachable through the Eyebot radio (in *Eyebot* the range is about 1.5-2 m.).

As the *Eyebot* robots allow more than one application to run simultaneously in one robot (see section 3.3). This feature forced us to develop a new non-blocking *receive* function in this layer.

3.2 Net Layer

The service offered by this layer to the transport layer is the routing of packets between any pair of robots, even if they are not neighbours. This is core layer of PERA. It is here where we find the routing algorithms that PERA uses, and where some data structures are implemented in order to store routing and control information.

Addressing We have created an addressing scheme adapted to the peculiarities of the Eyebot communications infrastructure. Each robot must have an unique address. PERA uses one byte of each packet for this purpose, subdivided in three fields (see 1).

Unic./ Mult.	*Host 3*	*Host 2*	*Host 1*	*Host 0*	*Port 2*	*Port 1*	*Port 0*
0	1	2	3	4	5	6	7

Table 1. Addressing scheme

First field (bit 7), selects between a unicast or multicast address. When bit 7 is set to 0, it specifies an unicast address and when it is set to 1 it specifies a multicast address, that is an address that does not represent a single robot, but a set of robots. The second field (bits 6-3) choose the destination robot (we can address a maximum of 16 robots). Finally, the last field (bits 2-0) selects the port inside the destination robot (see section 3.3).

Data message format The DATA package contains the following fields, where each field correspond to a byte, except by the sixth one, whose size is specified by the fifth:

0-Type–Message identifier (0).
1-Hop Source–Source address of next hop.
2-Hop Destination–Destination address of next hop.
3-Destination Address–Destination address of data packet.
4-Originator Address–Source address of data packet.
5-Size–Data size in Bytes.
6-...-Data–Data.

Route request packet (*RREQ*) The RREQ package (*Route Request*) described in previous section is made up by the following 9 bytes:
0-Type–Message identifier.
1-Hop Source–Source address of next hop.
2-Hop Destination–Destination address of next hop.
3-Hop Count–The number of hops from source address to the robot handling the request.
4-RREQ ID–A number uniquely identifying the particular RREQ when taken in conjunction with the *Originator Address*.
5-Destination Address–The address of the destination robot for which a route is desired.
6-Destination Sequence Number–The last sequence number received by the source robot for any route toward the destination.
7-Originator Address–The address of the robot that originated the route request.
8-Originator Sequence Number–The current sequence number to be used for route entries pointing to (and generated by) the source of the route request.

Route reply package (*RREP*) The RREP package is composed by the following eight bytes:
0-Type–Message identifier.
1-Hop Source–Source address of next hop.
2-Hop Destination–Destination address of next hop.
3-Hop Count–The number of hops from destination address to the originator address.
4-Destination Address–The address of the destination for which a route is supplied.
5-Destination Sequence Number–The destination sequence number associated with the route.
6-Originator Address–The address of the source robot that issued the RREQ for which the route is supplied.
7-Lifetime–The time for which robots receiving the RREP consider the route to be valid. This field will be reduced in each hop, and if its value is 0, the message will be discarded.

Route error package (*RERR*) Finally the RERR package needs the following six bytes:
0-Type–Message identifier (3).
1-Hop Source–Source address of next hop.
2-Hop Destination–Destination address of next hop.
3-Unreachable Destination Address–The address of the robot that has become unreachable because of a link break.
4-Unreachable Destination Sequence Number–The last known sequence number associated to the unreachable robot.
5-Destination Address–The address of the destination robot towards the RERR goes.

3.3 Transport layer

The transport layer provides end-to-end communication by means of the abstraction of ports. It provides the service of multiplexing the radio channel among different applications running in the robot.

Ports makes easier to program applications that are composed by different threads of control. For example, a thread can run a reactive controller which avoids obstacles by using infrared sensors, while another thread is running the code that guides the robot towards a ball using the camera. Imagine that another robot needs to send data to one of those threads on the first robot. Without ports it would be more difficult to do this task because we could not select between applications.

Ports have been incorporated in the addressing scheme as shown in 1. Three bits have been reserved to identify the port, that is, eight different applications can be addressed in a robot.

3.4 Application layer

Currently the PERA library does not provide any communications protocol on the application layer. In future releases we intend to provide application protocols adapted to the communications needs of the applications we run on our robots. In particular, we want to implement a subscrition protocol that let application get periodical information, for example sensor data.

4 Implementation and Evaluation

One of the more important features required in the PERA library implementation was transparency. This let us take some implementation decisions: every applications execute in a different thread. Send and receive buffers are used in the transport layer to communicate with the net level. One pair of buffers is reserved for each application thread.

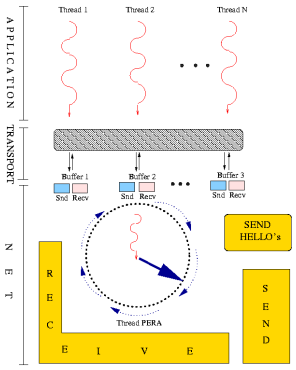


Fig. 3. Implementation scheme using various levels

The net layer which is executing PERA has been implemented using its own thread. This thread checks each send buffer and, if there is some data pending stored on one of them by the transport layer of a sending thread, it sends it. When, the net thread receives data, it stores it in the correct destination buffer according to the destination port. Other functions of the net layer are to route packets to a neighbour and to answer RREP's received. Periodically, it must also check if it must broadcast a *Hello* packet.

Concerning the evaluation, we have tested the library by simulating various scenarios in order to check the performance of PERA. The most significant results are described in this section. Anyway, for further information, the source code and detailed explanation of PERA can be obtained in *http://gsyc.escet.urjc.es/robotica/pfc/pfc-pera.html*.

In the tests, first we wanted to know the overhead introduced in the initialization of the radio modules. Experiments have shown that there is a 15% increase in time, and this increase is lineal to the number of robots. Initialization time depends on how many robots are powered on simultaneously (this is due to the underlying RoBIOS link protocols). This is due to the new threads and structures that have to be created as described in the previous paragraph, and shown in figure3.

Then we want to know the performance in the discovery of new routes. The results once again shown that the time need grows lineally with the number of robots in the route. Note that for any new robot in the route two new messages have to be sent if that robot happens to be in the route, note also that the inherent broadcast nature of robot radio communications, makes it independent from the number of robots that can be "seen" from the new one added.

The performance of the protocol when sending data through a previously discoverd route has also been tested. In figure 4 the results of a simple experiment are shown. First, the time that an increasing number of packages that two robots, that can connect directly, sends to each other takes to be sent is shown (red lower line). Then (green upper line), the same number of packages sent through another

robot, in a three herd scenery, are shown. As it had been previously stated, the time grows lineally.

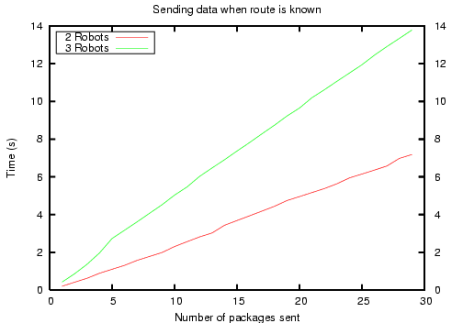


Fig. 4. Time comparison when sending data through a known rotue

Last, the recovery when routes are lost (a relaying robot moves) has also been tested. Figure 5 shows the time that a sending robot needs to discover that the route is not working, till a new route is discovered. The setup of the experiment consisted in simple 3 robots raw scenary, where the middle one was duplicated, that is, in fact there were 2 robots in the middle. The two robots placed in the middle were disconected alternatively. The Time To Life (TTL) of the routes has been set to 5 seconds, and the time or retransmission to 1' sec in order to all the lost packages to generate a new route discovery. Looking at this figure, it can be stated that the reconfiguration time is almost the same as the route discovery time.

5 Conclusions

The aim of this paper was to provide an ad-hoc communications API for robots, that let them communicate in environments where infraestructure is not available, or it does not want to be used. We have proposed a modified communication protocol, an addressing schema, and an implementation for the Eyebot robot. The implementation offers an alternative to the original RoBios API for communications.

The major improvements of this new API are the possibility of sending data beyond the radio scope of the robot, and to a particular program in a multiprocessing robot. That is, the routing algorithm guarantees, that there is a route through other Eyebots, data could be sent; and the addressing schema that allows communication among applications, not among robots.

The main problem of this library is that a packet can be lost with high probability because PERA does not guarantee reliable communication on any

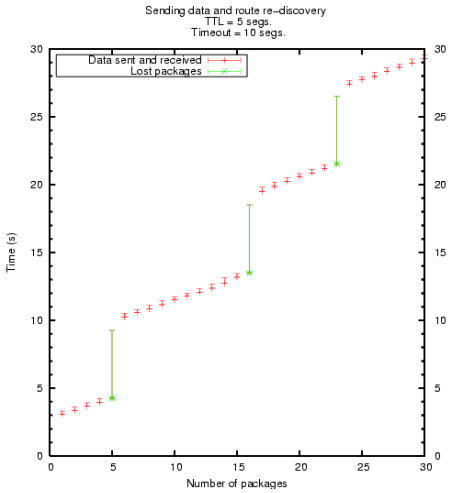


Fig. 5. Route re-discovery times

of its layers. In this way, we could say that we have provided the equivalent to the UDP protocol, not to TCP. A place where message recovery could be provided is the transport layer, thus providing reliable transmission end-to-end by retransmission if it would be required. Another alternative, is to implement recovery protocols at the link layer.

The ACKs that are already used at the network layer in order to detect lost routes could be used also to detect possible transmission errors. This feature would discard false positives in the detection of lost routes, and would accelerate the recovery of lost messages in case this is the reason for the absence of ACK reception.

Finally, we are currently implementing a multicast extension, that is one sender and a group of receivers. The addressing scheme of PERA already incorporates support for this kind of communication, but not the current realese.

References

1. C. Agero, V. Matelln, P. de las Heras, "PERA: Protocolo de Encaminamiento sobre redes Ad-Hoc", *http://gsyc.escet.urjc.es/pera*, 2002.
2. T. Braunl, "Eyebot Documentation", *http://www.ee.uwa.edu.au/braunl/eyebot*, 2002.
3. M. S. Corson, A. Ephremides, "A Distributed Routing Algorithm for Mobile Wireless Networks", *ACM/Baltzer Wireless Networks J.*, 1995.
4. S. R. Das, C. E. Perkins, E. M. Royer, M. K. Marina, "Performance Comparison of Two On-demand Routing Protocols for Ad hoc Networks." em IEEE Personal Communications Magazine special issue on Ad hoc Networking, 2001.

5. R. Dube, “Signal Stability based Adaptive Routing (SSA) for Ad-Hoc Mobile Networks”, *IEEE Pers. Commun.*, 1997.
6. D. B. Johnson, D. A. Maltz, “Dynamic Source Routing in Ad-Hoc Wireless Networks”, *Mobile Computing*, 1996.
7. S. Murthy, J.J. Garca-Luna-Aceves, “An Efficient Routing Protocol for Wireless Networks”, *ACM Mobile Networks and App. J., Special Issue on Routing in Mobile Communication Networks*, 1996.
8. C. E. Perkins, “Ad Hoc Networking”, *Addison-Wesley*, 2001.
9. C. E. Perkins, E. M. Belding-Royer, S. R. Das, “Mobile Ad Hoc Networking Working Group - Internet Draft”, *http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-11.txt*, 2002.
10. C. E. Perkins, P. Bhagwat, “Highly Dynamic Destination-Sequenced Distance-Vector Routign (DSDV) for Mobile Computers”, *Comp. Commun*, 1994.
11. RoboCup-Rescue Official Web Page, *http://www.r.cs.kobe-u.ac.jp/robocup-rescue/*
12. E. M. Royer, C. Toh, “A Review of Current Routing Protocols for Ad-Hoc Movile Wireless Networks”, *IEEE Personal Communications*, 1999.
13. A. S. Tanenbaum, “Redes de computadoras”, *Prentice Hall*, 1997.
14. C. Toh, “A Novel Distributed Routing Protocol To Support Ad-Hoc Mobile Computing”, *IEEE 15th Annual Int’l. Phoenix Conf. Comp. and Commun.*, 1996.