PERA: Ad-Hoc Routing Protocol for Mobile Robots

Carlos Agüero Vicente Matellán Pedro de-las-Heras-Quirós José M. Cañas
DIET (Departamento de Informática, Estadística y Telemática)
Universidad Rey Juan Carlos
C/ Tulipán s/n, Móstoles (Madrid, España)
{caguero,vmo,pheras,jmplaza}@gsyc.escet.urjc.es

Abstract

Mobile robots need to be able to communicate among them, and with other hosts participating in a given task. Traditional wired networks are obviously not suitable for mobile robots. Current wireless networks are usually based on a fixed network infrastructure (base stations) to route packets. The best alternative for mobile robots are Ad-Hoc networks, which are wireless networks that do not need a fixed infrastructure. This paper describes PERA, an adaptation of an ad-hoc routing protocol that runs on Eyebot *mobile robots. By using PERA, a robot can send messages to other robots or hosts that are not directly reachable through its radio antenna, by routing messages through intermediate mobile robots. The design, implementation and lessons learned in the initial tests of PERA are presented in this paper.*

1 Introduction

Let's imagine a group of mobile robots working in a rescue situation [14]. In this kind of environment the communication infrastructure may be severely damaged, so robots can only trust on their own capabilities to communicate among them. In an scenario where a robot finds a victim, it should be able to send images to the mobile host where a human operator is supervising the rescue mission, even if this host is out of the range of the robot's radio. This can be done by using other robots as intermediate hops in the route towards the human operator.

Wireless networks can be classified into two groups attending to their dependence of some kind of infrastructure: Infrastructure networks and *Ah-Hoc* networks. This paper explores the use of *Ad-Hoc* networking to satisfy communication needs among mobile robots.

Traditional routing protocols used in fixed networks such as the Internet are not well suited for Ad-Hoc networks, because they do not stabilize under frequent changes in connectivity among routing nodes. In a network of mobile robots the change of connectivity is the norm.

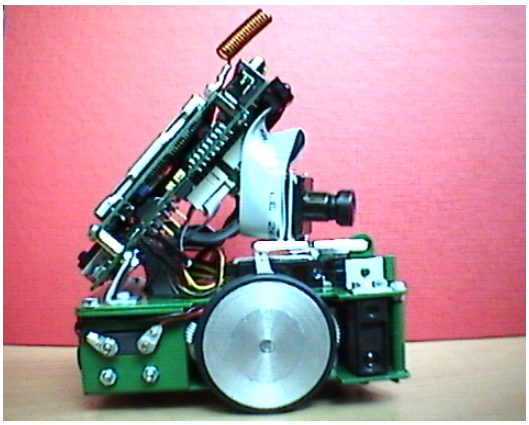

Figure 1: *Eyebot* robot

Many researchers have created routing protocols for Ad-Hoc networks. These protocols incorporate mechanisms to adapt routing tables to the frequent mobility of nodes, while trying to maintain a low power consumption. This is not an easy task, as the frequent change of connectivity induces the need of a frequent exchange of messages that helps to update routing tables.

In this paper we describe PERA, which is an adaptation of one of these protocols to the world of mobile robots. PERA is a protocol inspired in the AODV (*Ad-Hoc On-Demand Distance Vector*) [11] routing protocol. An implementation of PERA has been programmed as a library to communicate robots with independence of their radio scope. An implementation of PERA that runs on the *Eyebot* mobile robot is de-

scribed in this paper.

Eyebot robots are equipped with three infrared sensors, two encoders and a camera. In addition, robots are supplied with a 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: section 2 provides an introduction to *Ad-Hoc* networking in mobile robots. The PERA protocol specification is presented in section 3. Section 4 describes PERA's design library. Finally, the implementation and tests done on the Eyebot mobile robots are described on section 5.

2 *Ad-Hoc* Routing protocols for mobile robots

Each mobile robot must be configured as a router and should collaborate by routing packets it receives from neighbour robots. Before robots can route packets towards its destinations, they need to discover good routes by running routing protocols.

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. All those assumptions do not hold in mobile robots. Other protocols have been proposed that have better efficiency because they accommodate to the special features of robots. We can separate them into two different groups: based in routing tables and based in *on demand* routing.

Protocols based in routing tables 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*) [13], CGSR (*Clusterhead Gateway Switch Routing*) [15] and WRP (*The Wireless Routing Protocol*) [10].

On the other hand, protocols based in *on demand* routing only store in their tables the routes that are really needed. When a packet addressed to an unknown destination is received by a router, a route discovery process is initiated on demand in order to learn a new route. A route maintenance process is also needed to keep fresh the routes learned and to delete unused routes. Some examples of this category are: AODV (*Ad Hoc On-Demand Distance Vector Routing*) [11], DSR (*Dynamic Source Routing*) [9], LMR (*Lightweight Mobile Routing*) [4], TORA (*Temporary Ordered Routing Algorithm*) [11], ABR (*Associative-Based Routing*) [17] and SSR (*Signal Stability Routing*) [6].

3 The PERA Ad-Hoc routing protocol

PERA tries to fulfill the following requirements:

- Each robot can send data to any other robot.
- Each robot can receive data from any other robot.
- Movement of robots is allowed without disturbing communications.
- Multiple applications running concurrently on the same robot can use PERA in order to send / receive.
- Every robot can send to a particular application running on a given robot.
- The library providing the PERA protocol should allow to choose between unicast and broadcast transmission.

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

The major limitations that *Eyebot* robots pose to PERA lie on the *RoBios* OS. The maximum size of a data packet is limited to 35 bytes.

Some protocols based in *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.

Protocols based on tables were evaluated as a possibility to implement PERA, but they were soon discarded, because they would collapse the network just with control traffic only to maintain a real time vision of connectivity.

Instead, PERA uses a protocol based in *on demand* routing that is table driven: each mobile robot maintains a table with routes. Packets only contain the address of the destination robot. An intermediate robot consults its table in order to choose the next hop. Contrary to what happens in protocols based in routing tables, as the ones used on the Internet, the protocol used in PERA only updates tables on demand.

There are two main tasks that the routing function must fulfill: route discovery and route maintenance, which are used, respectively, to initially fill the table when a packet must be routed to an unknown destination, and later, for keeping updated a given route updated in case it is still needed. For this reason, each entry in the route table that is stored by each robot can be erased or updated. This is why all the routes stored in the routing table include a field named *lifetime*, as shown in figure 2. When route maintenance detects a route is too old, its lifetime field will not be refreshed. The Lifetime field, together with the sequence number field ensure that a robot does not use old routes and that routing cycles not exist.

Destination	Next Hop	Sequence Number	Hop Count	Lifetime	Last Modified
⋮	⋮	⋮	⋮	⋮	⋮

Figure 2: Routing table in PERA

3.1 Route discovery

This process is triggered by a robot when it wants to send a packet to another robot and the first one has not an active route for the desired destination on its routing table. This process ends when a route is discovered (in case it exists), and has a side side effect on intermediate routers.

The process begins with the sender composing a RREQ (*Route Request* packet, which includes the identifier of the robot which wants to later send a message. The identifier of the source robot and a special ID identifies uniquely this RREQ. 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 originator of the RREQ in order to avoid unnecessary flooding. This way a RREQ that has been previously received and resent by a given robot can be discarded. The RREQ ID is incremented each time a new route discovery is initiated, so that when the conditions of connectivity change this new route discovery is not 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 suppose symmetrical links).

In case the RREQ is finally received by the final destination, it will send back a RREP (*Route Reply*) packet addressed towards the source of the RREQ received. This RREP packet is sent towards the sender of the RREP, 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 this protocol. 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. It is always necessary to consider that a RREP packet originated by an intermediate robot should not have an older sequence number than the sequence number included in the RREQ for the final destination that it is responding to.

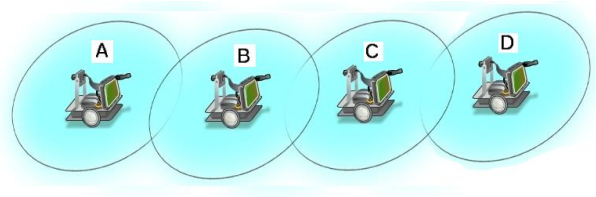

Figure 3: Route discovery

As an example, in figure 3 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's address (A) and the current sequence number at node A, as well as the destination 's 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 our 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 his neighbors. This second RREQ is received by robot A, that silently discards the packet because it recognizes it as packet already broadcast by him (in fact A was the originator of this RREQ). However, robot C, who also receives the RREQ, rebroadcasts it to his neighbors, after storing on its table a 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 which neighbour it must send the RREP addressed to A. By consulting its routing table, it sees it's node C. The RREP then reaches C, who this way learns a way towards node D, and stores it on its routing table. Then, C checks its routing table and there it learns 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. Note that in fact the route is not completely known to A. What A knows is that it must send packets addressed to D to B. But B knows that packets addressed to D must be sent to C. And C knows that packets addressed to D can be send directly to D, as D is directly reachable through C.

As a final step, a counter is added in each RREQ, to know the number of hops that packet has followed. This counter is attached in each entry of a routing table, and when a robot rebroadcasts an RREQ, it must increment it.

3.2 Route maintenance

Once a route has been discovered for a given source / destination pair, it is maintained as long as needed by the source node. Movement of nodes within the *Ad-Hoc* network affect only the routes containing those robots on its paths. Movements of robots that do not affect those routes do not trigger any protocol action. If during an active session the movement of the source robot affects the route being used to route packets, the source robot must reinitiate route discovery in order to discover a new route to the destination. Each time a robot sends a packet to a neighbour, it expects to receive an ACK from it. This is the way the source robot knows its packets are not being routed.

However, if the route is affected by the movement of an intermediate robot (let's call him "culprit"), 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 toward 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 the 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 node creates one. In addition, all robots broadcast a *Hello* packet to inform its neighbors periodically that it is still in the vicinity.

4 Design of the PERA library

We have build 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 EyeOS 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 provides services to the level below through well defined interfaces.

In PERA we use 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 next subsections.

4.1 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 link 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.). *Eyebot* robots allow more than one application to run simultaneously in one robot. This feature forced us to develop a non blocking receive function.

4.2 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

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

We have created an addressing scheme adapted to the peculiarities of the Eyebot communications infrastructure. Each robot must have an unique address (see 1).

PERA uses one byte of each packet for this purpose, subdivided in three fields. 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 (broadcast is a particular address of multicast). 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 4.3).

Data packet

Type	*Hop Source*	*Hop Destin.*	*Destin. Addr.*	*Origin. Addr.*	*Size*	*Data*
0	1	2	3	4	5	...

Table 2: Data message format

The format of the data message is illustrated in table 2. It contains the following fields:
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*)

Type	*Hop Src*	*Hop Dest.*	*Hop Count*	*RREQ ID*	*Dest. Addr.*	*Dest. Seq.Num.*	*Orig. Addr.*	*Orig. Seq.Num.*
0	1	2	3	4	5	6	7	8

Table 3: RREQ (*Route Request*) message format

The format of the RREQ (*Route Request*) message is illustrated in table 3. It contains the following fields:
0-Type–Message identifier (1).
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*)

Type	*Hop Src.*	*Hop Dest.*	*Hop Count*	*Destin. Addr.*	*Destin. Seq.Num.*	*Orig. Addr.*	*Lifetime*
0	1	2	3	4	5	6	7

Table 4: RREP (*Route Reply*) message format

The format of the RREP (*Route Reply*) message is illustrated in table 4. It contains the following fields:
0-Type–Message identifier (2).
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*)

Type	*Hop Src.*	*Hop Destin.*	*Unreachable Destin.Addr.*	*Unreachable Destin.Seq.Num.*	*Destin. Addr.*
0	1	2	3	4	5

Table 5: RERR (*Route Error*) message

The format of the RERR(*Route Error*) message is illustrated in table 5. It contains the following fields:
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.

4.3 Transport layer

The transport layer offers the abstraction of ports. It provides the service of multiplexing/demultiplexing the radio channel among different applications. The advantage of using ports is that now it's possible to establish a communications flow between two applications instead of between two robots.

It is necessary that an application *binds* to a free port when it wants to receive packets addressed to that port on that particular robot. Function *bind* associates one application to a port and does not permit two applications to listen on the same port.

Having ports it is much 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.

4.4 Application layer

Applications that use PERA can create a maximum of 8 threads. Each one can send or receive the *send*, *receive* and *bind* primitives offered by the transport layer. Currently the PERA library does not provide any communications protocol on the application layer. In future releases we intend to provide application layers adapted to the communications needs of the applications we run on our robots.

5 Implementation and Tests

Implementation: An important feature that should have the PERA library is transparency. With this feature in mind, we took some implementation decisions. All applications execute in different threads and use his own calls to send, receive and bind from the transport layer. 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.

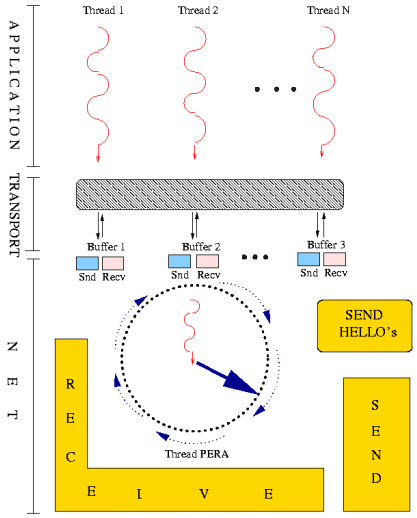

Figure 4: Implementation scheme using various levels

The net layer which is executing PERA is implemented using a different and unique 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 takes it and sends it. Then, the net thread receives and puts data 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.

Tests: We have been testing the new library, simulating various situations in order to check the correct operation of PERA. We have developed an application to configure the topology of the testbed net-

work. This way we can simulate multiple configurations without needing to physically move the robots used in the testbed.

In tests we have verified the proper propagation of RREQs, RREPs and RERRs, the accurate writing, updating and erasing of routes in the routing tables, and the appropriate sending of *HELLO* messages.

Future trials will help us to measure some concrete parameters such as latency, amount of missed packages, etc.

Next, we show an example of one of those tests for checking RREQs and RREPs.

As shown in figure 5 in this trial there are three robots labeled from 1 to 3. The scope of their radios is represented by a circumference around each robot.

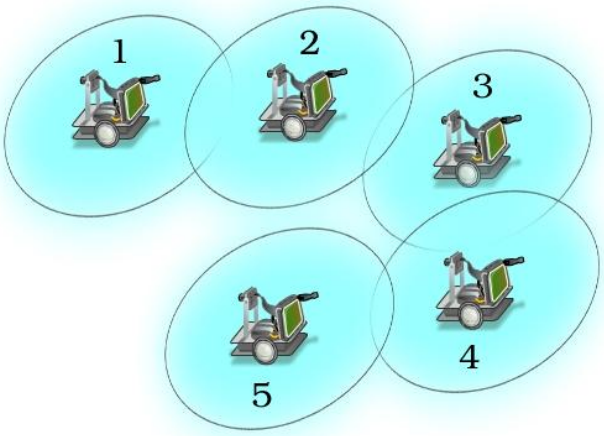

Figure 5: Diagram of robot situation in test

The goal of the test is that robot 1 sends a data packet to port 2 of robot 3, containing the message "Hi!". Once it is received, robot 3 should answer to port 2 of robot 1 with another data packet containing the message "Goodbye". For this test, we have configured the *HELLO_INTERVAL* parameter with a value of 300 (seconds) (we don't wanted to check that feature *HELLO*). The parameter *TTL_START* has been set to 127 (seconds). Adjusting this parameter we make sure that routes do not expire during the test.

Robot 1 begins the trial composing a RREQ because its routing table is empty. Then broadcasts this RREQ which is received only by robot 2. Robot 2 can not reply to robot 1 with an RREP, because its routing table is empty too. Before robot 2 broadcasts the RREQ again, it adds a new entry in his routing table: The destination is robot 1 and the next hop is robot 1 too (this means robot 1 is a neighbour). The RREQ sent by robot 2 is received by robot 1 and robot 3. Robot 1 discards the message because it is in fact who composed the RREQ packet. Robot 3 receives the RREQ message from robot 2 and he notices he is the destination. First, he adds en entry in his route table towards robot 1 using robot 2 as next hop. Then, robot 3 composes and sends a RREP message to robot 2 (consulting the routing table) with final destination robot 1. Robot 2 receives the RREP and looks for an entry in his routing information for send the packet to robot 1. He finds in the routing table that robot 1 is a neighbour. He can send the RREP directly. Robot 2 adds an entry in his route table for robot 3 (neighbour). Robot 1 receives the RREP and adds a new entry towards robot 3, using the neighbour which has received the RREP (robot 2). Figure 6 shows the routing information in all robots at this moment.

Destination	Next Hop	Sequence Number	Hop Count	Lifetime	Last Modified
3	2	2	2	127	0:0:50

Routing table in robot 1

Destination	Next Hop	Sequence Number	Hop Count	Lifetime	Last Modified
1	1	2	1	127	0:0:20
3	3	2	1	127	0:0:40

Routing table in robot 2

Destination	Next Hop	Sequence Number	Hop Count	Lifetime	Last Modified
1	2	2	2	127	0:0:30

Routing table in robot 3

Figure 6: Routing information when robot 1 receives RREP

Now, robot 1 sends the data message ("Hi!") to robot 2 and robot 2 sends it to robot 3 (always consulting the routing table). Robot 3 receives the data packet and begins the reply of a data packet containing "Goodbye" to robot 1. He looks for an entry in his routing table and notices it exists. The routing table says he must use robot 2 as next hop towards robot 1. Robot 3 sends the data packet to robot 2 and robot 2 passes on the message to robot 1 using the route table again. Finally robot 1 receives the information and the trial is finished.

6 Conclusions

We have developed an alternative to the original RoBios API for communicate *Eyebot* robots. The major advantages are the independence of the radio scope

and the use of ports. Tests show the correct operation of PERA.

We have noticed that the worst problem of our library is that a packet can be lost with high probability using Eyebots. PERA does not guarantee reliable communication on any of its layers. A place where message recovery could be provided is the transport layer, thus providing recovery end to end. But we think a better place to implement recovery protocols is at the link layer, because it would accelerate the recovery due to the high rate of transmission errors of the Eyebot radio.

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 message in case this is the reason for the absence of ACK reception.

We are also implementing multicast transmission (one sender and a group of receivers). The addressing scheme of PERA already incorporates support for this kind of communication.

References

[1] C. Agüero, V. Matellán, 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] D. Chaparro, R. Rodríguez, J. Pelegrín, "Encaminamiento Ad-Hoc", *http://pantuflo.escet.urjc.es/ir-1-2/crp/*, 2002.

[4] M. S. Corson, A. Ephremides, "A Distributed Routing Algorithm for Mobile Wireless Networks", *ACM/Baltzer Wireless Networks J.*, 1995.

[5] 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.

[6] R. Dube, "Signal Stability based Adaptive Routing (SSA) for Ad-Hoc Mobile Networks", *IEEE Pers. Commun.*, 1997.

[7] J. M. Cañas, E. García, V. Matellán, "Manual del Robot Eyebot", *http://gsyc.escet.urjc.es/robotica/manual-eyebot*, 2002.

[8] Grupo de Sistemas y Comunicaciones Universidad Rey Juan Carlos, "Introducción a las redes de ordenadores", *http://gsyc.escet.urjc.es/docencia/asignaturas/redes*, 2001.

[9] D. B. Johnson, D. A. Maltz, "Dynamic Source Routing in Ad-Hoc Wireless Networks", *Mobile Computing*, 1996.

[10] S. Murthy, J.J. García-Luna-Aceves, "An Efficient Routing Protocol for Wireless Networks", *ACM Mobile Networks and App. J., Special Issue on Routing in Mobile Communication Networks*, 1996.

[11] C. E. Perkins, "Ad Hoc Networking", *Addison-Wesley*, 2001.

[12] 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.

[13] C. E. Perkins, P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routign (DSDV) for Mobile Computers", *Comp. Commun*, 1994.

[14] RoboCup-Rescue Official Web Page, *http://www.r.cs.kobe-u.ac.jp/robocup-rescue/*

[15] E. M. Royer, C. Toh, "A Review of Current Routing Protocols for Ad-Hoc Movile Wireless Networks", *IEEE Personal Communications*, 1999.

[16] A. S. Tanenbaum, "Redes de computadoras", *Prentice Hall*, 1997.

[17] C. Toh, "A Novel Distributed Routing Protocol To Support Ad-Hoc Mobile Computing", *IEEE 15th Annual Int'l. Phoenix Conf. Comp. and Commun.*, 1996.