NEW PROTOCOL FOR GROUPING DATA USING ACTIVE NETWORK

A. Moreno[1], B. Curto[2] and V. Moreno[2]
Computer Science
University of Salamanca (Spain)
[1]*amoreno@usal.es*
[2]*control@abedul.usal.es*

Abstract: Active networks provide an ideal support for the incorporation of intelligent behaviors into networks. This ones make possible to introduce a more general functionality, that supports the dynamic modification of the behavior of switching networks. By means of this approach, users can dynamically insert code into nodes, thus adding new capabilities.

In this paper we propose a protocol to improve network services, implemented from the point of view of active networks. The specific purpose of the protocol is to reduce the network traffic, thus increasing the ratio of the volume of transmitted data to the number of frames that pass through the network. More precisely, we intend to merge data coming from the same node or from different nodes in intermediate points of the path toward the receiver, so that they arrive as closely grouped as possible. This would reduce the congestion in networks that support a great volume of traffic, like those that are used in industrial environments, which can also suffer from a low bandwidth.

For the implementation of this protocol we use ANTS (*Active Node Transport System*), a freeware tools for the construction of active networks developed by MIT(*Massachusetts Institute of Technology*). ANTS makes use of one of the most innovative and daring approaches for injecting programs into active nodes, by means of a mobile code called *capsules*.

Keywords: Active networks, network protocols, mobile code, data grouping.

INTRODUCTION

A current tendency in distributed control systems is the use of general purpose networks, like *Ethernet*, to communicate different processing elements.

This can produce poor network performance in segments that support high flows of data coming from multiple devices that are linked to it. This situation appears in industrial environments, particularly in data acquisition distributed systems.

A possible solution is the development of new protocols that allow us to improve network services by looking for alternatives that adapt to the specific necessities of each moment. However, the introduction of a new protocol is a slow and difficult task. This is due, first of all to the fact that it is necessary to carry out a standardization process in order to guarantee the system interoperability. Then, once a protocol has been accepted, its development presents many difficulties since there is no automatic form to include the new protocol with those already in existence. And one must add to this, the problem of compatibility with existing versions. This means one must take into account the times needed to solve the previous problems, thus making viable solutions to appear very slow in the evolution of network services. In this sense, it is foreseen that the deployment of IPv6[1] will take no less than approximately fifteen years to be completed.

Thus, *active networks* seem to be a perfect solution to tackle the aforementioned problems[2; 3]. The Active Network Program is a DARPA-sponsored research program born during the years 1994 and 1995. Users can program the network by loading their own programs for the realization of specific tasks. The main idea in active networks is to standardize a communication model, instead of individual communication protocols.

There exist two approaches for the construction of active networks: one of them is *discrete* and the other is *integrated.* In the first one, we separate the procedure to inject programs into active nodes from the processing of the packets that cross the node. Therefore, there will be a mechanism for the users to include their programs into active nodes, and there will be a mark on the packets to indicate which is the program that will process them. A different approach is to include the code with which packets will be processed inside the very same packets. This way, packets (called *capsules*) will contain both data and programs[3].

At the moment, one important research field in active networks[2] is centered in the creation of tools that facilitate their construction to a greater extent. One of them, ANTS (*Active Node Transport System*), has been developed by MIT and is freely distributable[4]. It was the first tool to use the capsules pattern. More recently, MIT has designed PAN(*Practical Active Networks*)[5], which is based on the ANTS architecture, but whose objective is to be more efficient than its predecessor, both in execution and in the code mobility system. Utah University is also working on a system based on ANTS, known as Janos[6], that improves node resource administration and tries to make a clearer and surer separation among the different user programs that will be executed in one

node. The University of Pennsylvania in collaboration with Bell Switchware [7], which has taken the discrete approach as its focus, makes special emphasis on node security. Georgia Tech is creating an operating system for nodes, called Bowman[8], and an execution environment called CANE (*Composable Active Network Elements*)[9], also with a discrete approach.

Our work, based on the integrated approach, tries to find a definition for protocols that intend to solve the overload problems or network congestion that appears in industrial environments, typically in data acquisition distributed systems. The proposed solution contemplates the grouping of data generated by different sources that head toward a common point. We try to increase the ratio of useful information to the overhead of these protocols. This leads to a reduction of the traffic that goes through the network, thus producing a better response time. One must also take into account the possibility that the time necessary to carry out a grouping could be detrimental when trying to have data arrive without undue delay. As we shall see, we have thought about the introduction of a timeout procedure that guarantees as much as possible that packets are sent in a reasonable interval of time.

The remainder of this paper is organized as follows: in second and third sections we describe the motivation and the goals that have guided the realization of this work. In the next section we detail the specifications of the grouping protocol. In the fifth section we describe the implementation details, and afterward we show a study case. The seventh and last section closes the work and we present our main conclusions.

1. MOTIVATION

Let us consider a communication scheme in which a group of sender nodes distributed in the network must transmit a large amount of data that are being produced, and a receiving entity must operate taking those data as input. This situation can be found in a distributed data acquisition system (Figure 1), where several nodes capture data coming from different sensors. These nodes send the data they have picked up to a receiver, all within some established time constraint.

If each node sends its data one by one and data arrive separately, coming from different nodes, the network will experience an unnecessary overload. This problem is specially severe in the segment connected to the receiver, since it collects all incoming traffic.

If we use this approach, each data is encapsulated inside a frame when crossing the network, and we can have the paradoxical situation that the information of the protocol can actually take up more space than data being sent.

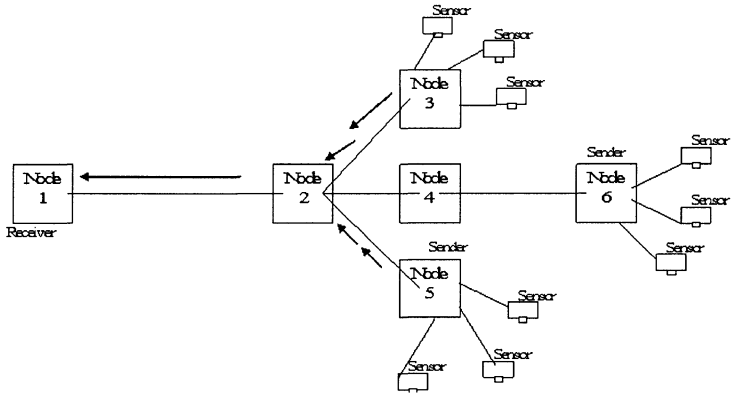

Figure 1. Data transmission without group

A possible solution to the problem would be that senders themselves grouped their data before emitting them. However, it can be interesting not to overload to the originators with grouping tasks. Then this solution would not be feasible. In any case, this scheme does not solve one of the problems, since data coming from different nodes will still reach the receiver separately.

2. GOALS

The main goal of this work is to propose a solution to improve the ratio between the amount of transmitted information and the protocol-related overhead in environments with heavy traffic. In this way we try to enhance the performance of low-bandwidth links, thus enhancing their response time.

The solution is based on an active network system that makes it possible to generate new network protocols that adapt to the specifications of different services. The solution of the problem we have described will be a protocol in which data are grouped (Figure 2) as they traverse the network toward their target. The points where data can be grouped would be the active nodes, in such a way that data can reach the receivers in as close a group as possible.

The new data frame maintains the same size, with a bigger amount of useful information. The grouping of data sent from the same node would be no longer be the responsibility of sender, but rather it could be delegated to the closest node in the network. The protocol could be also built in such a way that the sender could configure the emission, thus indicating the node that must do the grouping, its size, the maximum wait time for data grouping, etc, all this in an attempt to enhance the time response of the network.

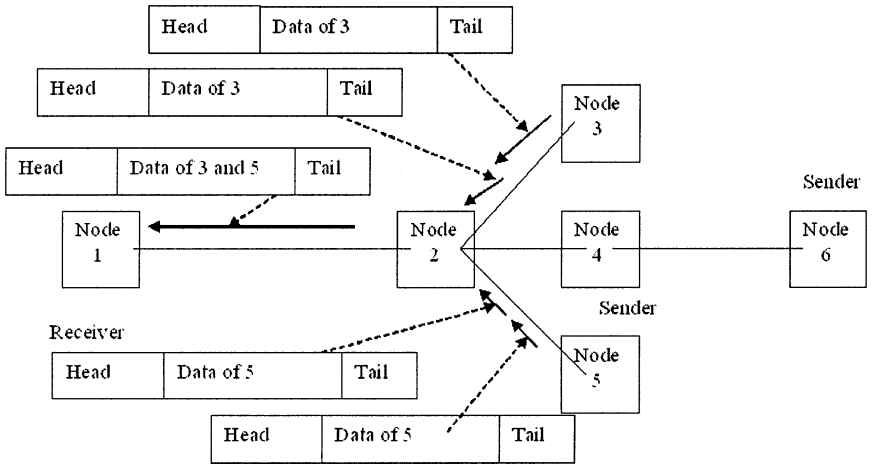

Figure 2. Data transmission with group

3. THE PROPOSED GROUPING PROTOCOL

The data grouping protocol makes it possible to join data coming from one or different sources in best possible way. In order to carry out this task, it performs three important functions:

- Configuration of the grouping system. The receiving node has the capability to configure the system so that an efficient grouping of data is done.
- Grouping of data coming from the same originator. This task is that of building groups with data that come from the same node. It can be the responsibility of the same sending node, or that of any neighbor.
- Grouping of data coming from different senders. The nodes that receive data coming from different sources should encapsulate them all in one frame in an orderly way. This task will be carried out by nodes placed at the intersections.

Configuration of the grouping system

First, the receiver must select the nodes from which it requires data. All these senders will be part of a *merge data group*. The receiver will send a set of parameters (Table 1) to each data source. This process shall be called *registration*.

Table 1. Registration information

sender	Sender node
receiver	Receiver node
group	Group-id to which the sender will belong
units	Highest node number of data that can be grouped in a frame
time	Available maximum time to group the data before sending them
jump	Distance limit among the sender node and the node that clusters their data

The intermediate nodes that will participate in the data grouping system are registered in a simultaneous way with the sending nodes. This way, we generate what we call the *groupings tree*, which lets them know from which path they will receive data. In order to build the tree, it is necessary that each of the registered sender nodes confirms its registration with the receiver, so that, when a node receives confirmation, it can know about the network links that will belong to the tree branch and store them in the *registration array* (RM).

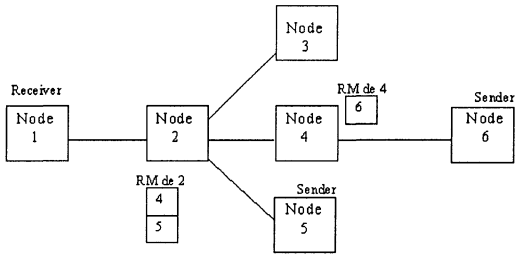

Figure 3. Registration operation

In order to show the registration process we refer to the network topology that we have shown previously. Node 1 sends a registration frame to nodes 5 and 6, and with their answers the tree is built (Figure 3). Nodes 2 and 4 have generated a *registration array* (RM) that serves to locate the tree branches. Node 3 is excluded because it has not been registered.

Grouping data from the same sender node

Nodes that have been registered as data sources send their data in an individual way toward the node where they will group. This will be configured by the *jump* parameter or in an mandatory way a intersection node (a node whose *registration array* has more than one element). Frames with individual data are named *unit capsules* and the place where the units will group shall be called *grouping point*. At this point, the unit capsule itinerary is finished and this will

be the place where data are stored in the cache together with the data from other unit capsule that come from the same source. The first capsule of the grouping should start a timer whose timeout value is specified in the configuration.

When the unit capsule that completes the group arrives, that is to say, when the limit of elements (*units*) fixed in the configuration phase is reached, the whole group is stored in a *grouping capsule* (described in the next section), and this capsule is transmitted to the receiver. The cache is cleared and the timer is stopped. If time runs out before a group is completed, the *grouping capsule* is created with whatever elements are available at the moment; then the frame is marked as urgent and sent, and the cache is cleared.

If we had configured in the previous example the *jump* parameter with a value greater than two, then the unit capsule would cluster in the intersection node 2, since they are not able to group beyond an intersection (Figure 4a). If the value of *jump* is 1, then the node 6 capsules would cluster in 4 and those of the node 5 would cluster in 2. They would only leave the *grouping point* when the time runs out, or when the group is completed (Figure 4b).

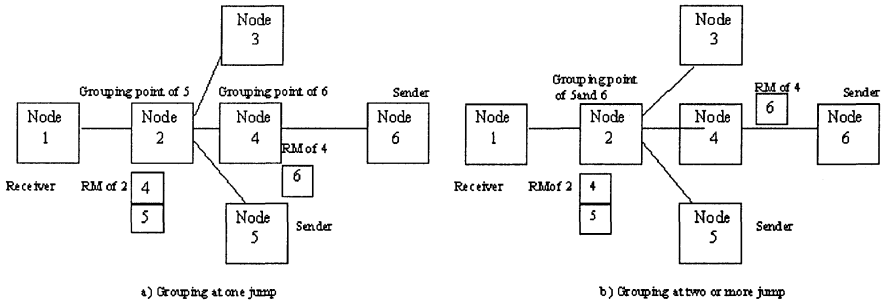

Figure 4. Grouping

Grouping data from different senders

The second phase of the protocol is the grouping of data that come from different senders. In this phase, data are no longer grouped according to source, but depending on their tree branches. It is necessary to ensure that no information about the original data sender is lost.

As in the previous case, when the first datum to be grouped arrives, a timer is started. Data are not sent to the following node unless one of these four cases happens:

- Data of all the tree branches have been stored, that is to say, a complete grouping has been made. In that case a group is formed with all the data

and they are sent to the next node of the tree. Afterward, the timer is stopped.

- Data arrive from some of the branches and there are data remaining in the area corresponding to that branch. A group is formed with the data, and they are sent to the next node. The timer is stopped. The data that have just arrived are stored in the proper place. Since they are the first data of the next *grouping capsule*, the timer is started again.

- Data have arrived marked as urgent. In that case, data are sent as urgent data and the timer is stopped.

- Time has run out. A group is formed and it is sent as urgent.

In the previous network topology example, if we suppose that the unit capsule are grouped at one jump(Figure 5) then the grouping capsule related to the node 6 is generated in node 4. The capsule proceeds directly to node 2, since there is only one source branch (node 4). However, when it reaches node 2, it is stored waiting for data from node 5, and from node 4. The unit capsule of node 5 are grouped in node 2, but when they generate a grouping capsule we take as previous node the same as the previous node of the capsule that created them (node 5).

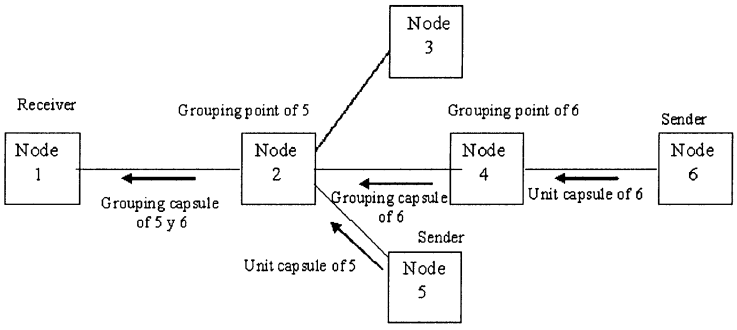

Figure 5. Final scheme of the grouping

In the previous description we have assumed that only one group had been generated. However, the protocol supports the generation of various groups in a direct way.

Table 2. Classes and methods of ANTS

Class	Methods
Application	send, receive, node
Capsule	evaluate, length, register, serialize, deserialize
Node	address, get, put, routerfornode, delivertoapp
Protocol	startProtocolDefn, startGroupDefn, addCapsule, endProtocolDefn, endGroupDefn

4. IMPLEMENTATION

In order to implement the protocol[10], we have selected ANTS as a tool for the construction of active networks, because it is written in a general-purpose language (JAVA), as opposed to others that define their own language. Thus we achieve a double goal: the portability of the applications and the possibility of modifying some aspects of the tool, since we can access the source code.

ANTS

ANTS provides a programming model that allows to define protocols, a code distribution system to load new protocols into the network and an environment to execute them. It is based on the capsule-oriented network model, in which the frame can take a reference to some executable code in each node, so that if the node does not have code it can make a request to previous nodes.

In this model, users adapt the network to their necessities by means of the definition of protocols based on ANTS classes (Table 2). Thus, to develop your own protocol, an derivation of the abstract class *Protocol* is created. It is necessary to identify the protocol data units (PDU) that will be inserted in the network and the different processing routines. Each type of PDU and its routines are specified in a derivation of the abstract class *Capsule*. A new application, an entity that makes use of the network to send and receive capsules, must be developed by means of class derived from abstract class *Application*. Active nodes constitute the environment in which a capsule is executed: they receive it, load it and they execute its routines, and they plan its transmission. Thus, an instance of the *Node* class represents the environment of local execution of ANTS.

The protocol

As we have mentioned, in order to use ANTS to codify the protocol it is necessary to generate the classes that will implement the protocol, the capsules and the applications that use them.

Hence, we have defined a class called *FussionProtocol* that registers the different kinds of capsules that compose the protocol: *FussionRegCap*, *FussionUnitCap* y *FussionCapsule*. The first one defines both the registration of sender nodes and the registration acknowledgment. *FussionUnitCap* transports the data toward the *Grouping Points* and it also implements the algorithm that groups individual data. The capsule *FusionCapsule* will be received by the reception node and it will contain grouped data. It will be created by a *FussionUnitCap* and it implements the algorithm to fuse data groups.

Applications constitute the programs that are going to use an active network protocol. They inherit the properties from the *Application* class and in our implementation three classes have been defined: *ReceptionApp*, *EmissionApp* and *MFussionApp*. *ReceptionApp* allows us to register emitter applications, in such a way that the group of fusion data and its parameters can be configured by sending a capsule *FussionRegCap*. It also performs the reception of fused data that arrive from the *FussionCapsule*. The class *EmissionApp* represents the data emitter that makes use of protocol *FussionProtocol*; it receives registrations from a receptor and it sends automatically an acknowledge.

Finally, the *MFusionApp* is executed at every node of the network to monitor their activities. It reads data from the cache when it receives a *FussionUnitCap* capsule or a *FussionRegCap* capsule, and it shows the *GoupingTree* for the node before the last actualization.

The tree branches are filled at the actual moment and the senders nodes that groups their data at this node.

The protocol requires a timing system to control data arrivals. The ANTS nodes do not provide any timing service. Nevertheless, ANTS has a extension system of functionalities that in this work has allowed us to implement a module that offers this timing service. Timing is done by means of a *thread* that will wait for the indicated time period, and then will send a finish signal. In order to perform this task, it has been necessary to create a class named *WaitTime*, that performs a timing task, and a interface, *TimInt*, that must be implemented by every class that need to be timed.

5. CASE OF STUDY

In order to show the effectivity and the performance of the grouping protocol we have built, we shall test with a network topology like the one shown in Figure 6.

The receiving application that is executed at node 20.20.20.1 registers the following emitter nodes at group 0: 20.20.20.3, 20.20.20.4 and 20.20.20.5. This means that the grouping task will be performed at the previous node

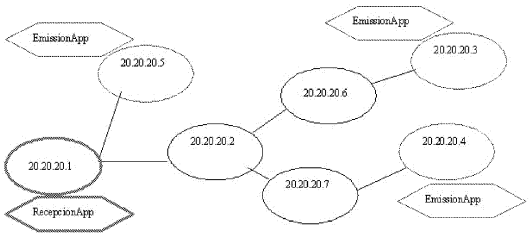

Figure 6. Active Network Topology

(*Jump*=1), where three data will be grouped (*Unit*=3) and that the waiting time will be 10 seconds (*Time*=10).

A register capsule of type *FusionRegCap* is sent to every node and their applications reply with a *CapRegFusion* as an acknowledgment (Figure 7).

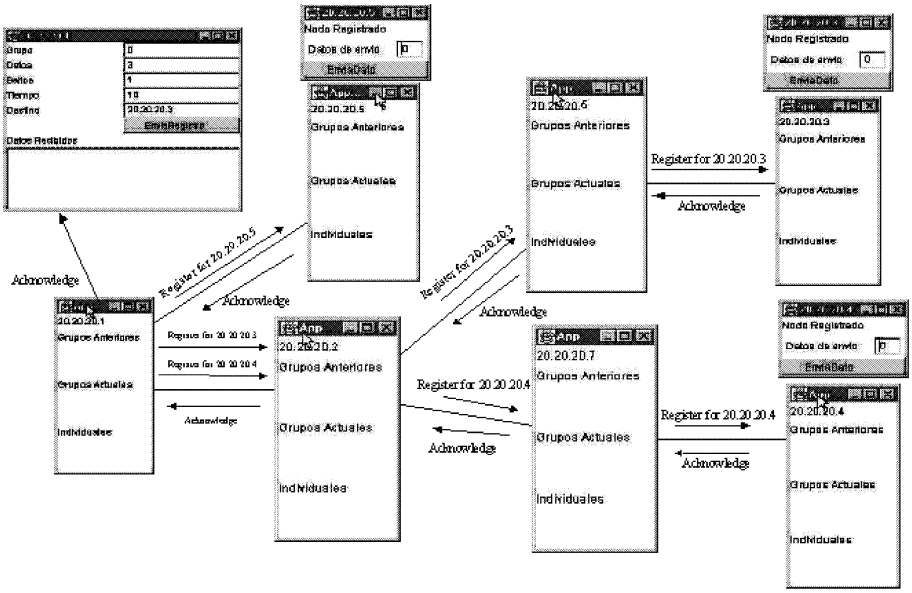

Figure 7. Register Operation

Once all the nodes are registered, data will be grouped by following the schema shown in Figure 8, where one can see that nodes 20.20.20.3 and 20.20.20.4 group their data at 20.20.20.6 and 20.20.20.7 at one jump as it has been indicated at the configuration. The *grouping point* will be the tree intersection (20.20.20.2). Since node 20.20.20.5 is located at distance of one jump from

20.20.20.1, their data will reach the target before grouping, hence data will be received ungrouped (Figure 9).

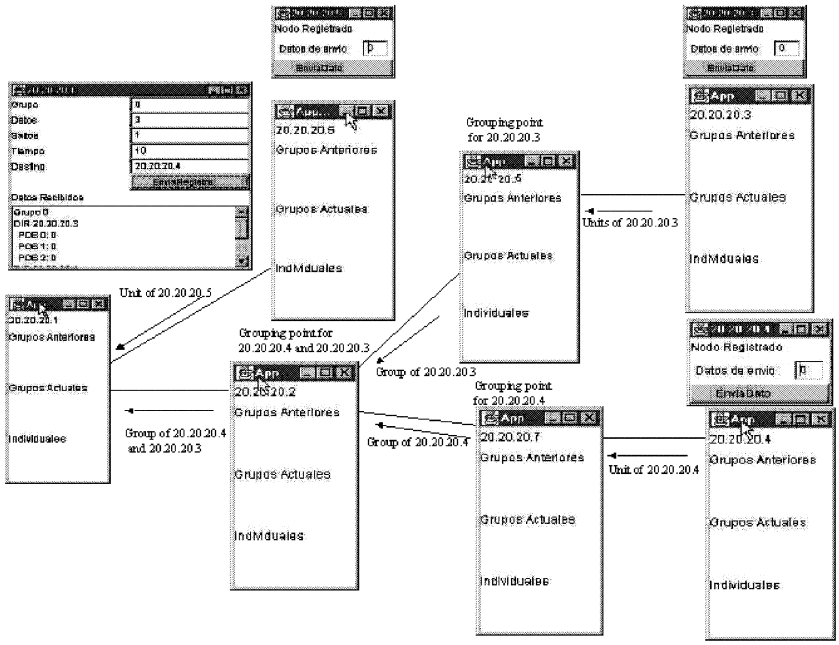

Figure 8. Data sending

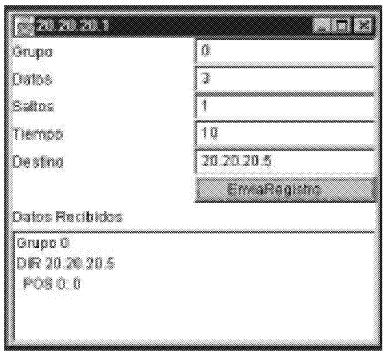

Figure 9. Data from 20.20.20.5

6. CONCLUSIONS

This paper describes an attempt to improve network services by means of active networks. We propose a way to reduce network traffic for possibly low-bandwidth network, thus optimizing their performance and helping to maintain a reasonable response time.

To reach this goal, we have define a protocol that groups data coming from both single and multiple sources. In order to implement the protocol, we have used a tool to construct active networks that is based on the ANTS capsule model. Several applications that use this protocol have been developed, as well as an application used to monitor intermediate nodes. Finally, in order to include timing procedures, it has been necessary to develop an timer extension that can be used in any other research field related to Active Networks.

REFERENCES

R. Gilligan and E. Nordmark. *Transition Mechanisms for IPv6 Hosts and Routers*. Internet Draft, march 1995

D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall and G. J. Minden. *A Survey of Active Network Research.* IEEE Communications Magazine, Vol. 35, No. 1, january 1997, pp 80-86

D. L. Tennenhouse and D. J. Wetherall. *Towards an Active Network Architecture*. Multimedia Computing and Networking (MMCN 96),january 1996, San Jose, CA: SPIE. A revision of this article appeared in Computer Communication Review, Vol. 26, No. 2 (april 1996). http://www.tns.lcs.mit.edu

D. J. Wetherall, J. Guttag and D. L. Tennenhouse. *ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols*. IEEE OPENARCH'98, San Francisco, CA, april 1998

E. L. Nygren, S. J. Graland and M. F. Kaashoek. *PAN: A High-Performance Active Supporting Multiple Mobile Code Systems.* Proceedings of IEEE OPENARCH'99, march 1999, pp 78-89

P. Tullmann, M. Hibler, and J. Lepreau. *Janos: A Java-oriented OS for Active Networks*. Appears in IEEE Journal on Selected Areas of Communication. Volume 19, Number 3, March 2001.

J. M. Smith, D. J. Farber, C.A. Gunter and S. M. Nettles, D.C. Feldmeier, W. D. Sincoskie. *SwitchWare: Accelerating Network Evolution (White Paper).* 1996

S. Merugu, S. Bhattacharjee, E. W. Zegura and K. L. Calvert. *Bowman: A Node OS for Active Networks*. IFOCOMM,2000

E. Zegura, K.Calvert. *Composable Active Network Elements:Lessons Learned.* ANTETS PI Meeting, may 2000

I. Blasco. *Aplicabilidad de las redes activas a la mejora de los servicios de red.* Technical Report, University of Salamanca, june, 2000