A multihoming architecture for OneLab

Antonio de la Oliva
Universidad Carlos III de Madrid
Madrid
Spain
aoliva@it.uc3m.es

Benoit Donnet
Université catholique de Louvain
Belgium
donnet@info.ucl.ac.be

Thierry Parmentelat
INRIA
Sophia-Antipolis
France
thierry.parmentelat@inria.fr

Ignacio Soto
Universidad Carlos III de Madrid
Madrid
Spain
isoto@it.uc3m.es

Abstract—This paper describes the work in progress in the European Commission funded OneLab project to extend PlanetLab nodes with multihoming functionalities. These multihoming functionalities aim at enabling application developers, for example, to test the impact of multihoming solutions on their applications, or to find out appropriate parameters for multihoming solutions according to the requirements of their applications.

***Key Wordss*— Multihoming, OneLab, PlanetLab, REAP.**

1. INTRODUCTION

Multihoming is the ability of having different connections to Internet, potentially through different providers. As Internet evolves, there are at least two different trends pushing the importance of multihoming:

- Internet communications are nowadays seen as strategic by companies and institutions. Reliability and degree of independence from particular service providers are an increasingly common requirement for these companies and institutions, and multihoming provided at the site level is the key to achieve the mentioned objectives. Furthermore, nowadays, at least 60% of stub domains are multihomed to two or more providers [8].
- There is a proliferation of access technologies everywhere. Therefore, it is increasingly common to have devices with several network interfaces (3G, WLAN, Ethernet, Bluetooth…). To gain full advantage of robustness in communications and ubiquitous access by the ability of using the different accesses where they are available, a multihoming solution at the host level is required.

With these scenarios, it is expected that more and more multihoming situations will become common in the near future. As a consequence, a lot of work is being done in different Working Groups of the IETF [1] to develop multihoming solutions that can provide the appropriate functionality while fulfilling requirements that guarantee that the solutions are deployable.

In IPv4, multihoming is mainly provided by means of BGP, i.e., multihoming is provided by routing. Such a solution, however, suffers from limitations, in particular scalability, that impede its full deployment up to small sites level, and much less to the host level. Indeed, if a site announces different address blocks with different priorities to obtain some load balancing capabilities, different routes will be needed in the core network for the address space of the site (on per address block with different paths to follow). As more sites are getting interest in obtaining capabilities, this can become a serious concern. Multihoming research in IPv6 aims at providing the full expected functionalities of multihoming. The multihoming architecture solutions for IPv6 are host based and allow not only sites, but also individual hosts, to take full benefit of multihoming.

The European Commission funded OneLab project [2] is currently working on extending the functionalities present in the PlanetLab overlay network environment. OneLab, using PlanetLab software as a starting point, will create an overlay network environment federated with PlanetLab. Eventually, the added functionality could be included in PlanetLab. Among the extensions, OneLab aims at providing multihoming functionalities by means of introducing multihoming components in PlanetLab standard nodes. In this paper, we describe these new components that should allow researchers to experiment with multihoming in a realistic distributed environment over the Internet. Examples of interesting issues that can be studied are effects of multihoming mechanisms in applications, improved performance of multihoming mechanisms by interactions with applications, or even study different multihoming methods.

The paper is structured as follows; section 2 describes the components of the OneLab multihoming solution; section 3 explains the requirements of the multihoming solution for OneLab nodes; section 4 describes how we introduce those components in OneLab nodes; Sec. 5 details how researchers can benefit from these functionalities; and finally, Sec. 6 concludes the paper.

2. Multihoming Components in OneLab

2.1 Objective

The objective of the multihoming solution in OneLab is to enable OneLab nodes with the following host based multihoming related functionalities:

- A failure detection and path exploration method.
- A transparent mechanism able to change the path used by a flow.
- A mechanism to simulate link failures.

2.2 Architecture

The basis of the architecture is a OneLab node with more than one network interface. The multihoming solution consists of three modules, each one providing one of the three functionalities described in Sec. 2.1. The functionality of these modules is described in the following.

2.2.1 Failure Detection and Path Exploration Method:

In a multihoming solution, we need a way to assess the current status of the path between two IP addresses, and detect failures when they happen. If the path is disrupted, a path exploration method is needed to find a new path between the nodes (represented by a new pair of addresses) through which the communication is possible, if it exists.

We have chosen to provide this functionality using a protocol based on the REAchability Protocol (REAP) [3]. REAP is responsible for failure detection and recovery. REAP periodically sends Keep Alive messages when no data is transferred by upper layers. As a host is supposed to receive Keep Alive messages or data, a failure can be detected due to a timer expiry. Failure detection

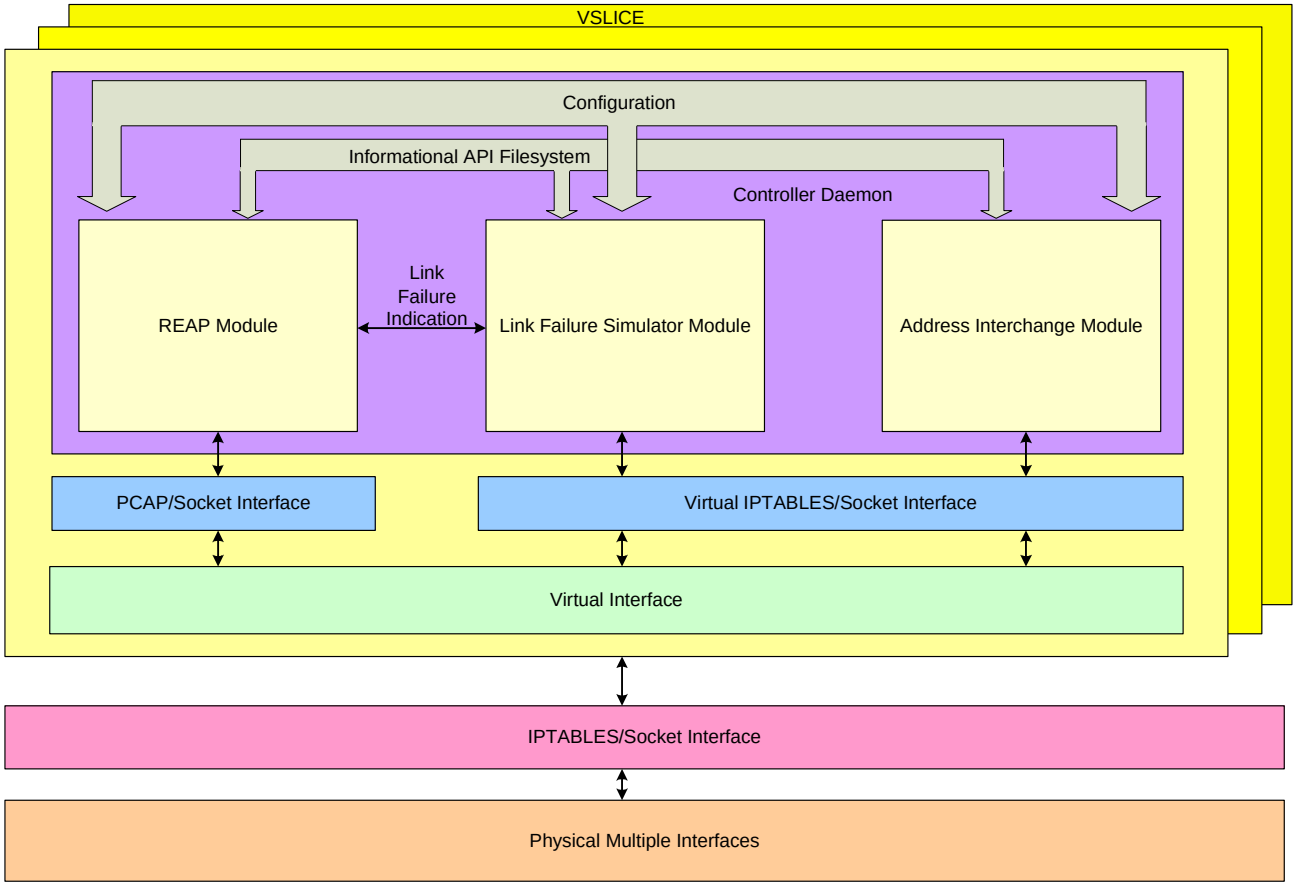

Figure 1: OneLab Multihoming Architecture

triggers a new path exploration procedure. REAP is fully described in section 4.1. The original REAP has been extended in order to support IPv4. Other possible modifications are under consideration. For instance, to provide information not only on path availability but also on some path characteristics such as round trip time (RTT).

2.2.2 Transparent mechanism for path modification:

This module modifies transparently the packets of the applications in order to send those packets through the new path discovered by REAP. The initial version uses a simple static configuration but more complex options for dynamic exchange of addresses (following SHIM6 [6] or MONAMI [7] principles) could be introduced.

2.2.3 Link Failure Simulator:

This module simulates simplex/duplex, random/static failures in a specific link. Notice that these failures do not affect the real network service, only how applications perceive it. It will be implemented using filters.

3. Requirements

Before going into the details of our multihoming implementation in the OneLab testbed, we discuss, in this section, the requirements for the software of OneLab nodes to allow the implementation of the multihoming functionality.

- As we are possibly going to modify packet headers, we require sudo or su capabilities in the host system and the slice system.
- Support for several interfaces. As the multihoming service requires access to the interfaces configuration, a possible scheme to provide this functionality in a secure way is to use one interface as primary (used for managing the OneLab node) and having others with unrestricted access for application use.
- The ifconfig and iproute2 commands are needed in order to configure the non-primary interfaces.
- Raw sockets are needed. The VNET system must support all the raw socket properties, including the possibility of opening sockets at the IP level.
- Ideally, IPv6 should be supported in order to obtain the maximum benefit from the experimental capabilities enabled by the proposed multihoming solution. Unfortunately, OneLab does not support yet IPv6 and its development within the kernel is not on the agenda.

4. Introducing the Multihoming Components in a OneLab Node

The multihoming components defined in Sec. 2.2 must be introduced in the architecture of OneLab nodes. This is done following the PlanetLab architecture principles [4][5]. Having these principles in mind we plan to implement the modules as services in a *sliver* (see Figure 1). A *sliver* is the instantiation in a node of a *slice*, and a *slice* is a set of virtual machines, each one created in physical nodes in the overlay environment. We argue that, implementing the link failure and multihoming *services* inside the sliver and not inside the node itself, will provide a greater deal of flexibility and the isolation between slivers required by the PlanetLab design guides.

The three modules that compose the multihoming solution for OneLab will be set up using configuration files that will define settings like, for example, the addresses available for the peers in a communication, the timer configuration or the port to listen Probes.

The architecture designed for the multihoming service at the highest level will be made of a daemon which controls all the rest of

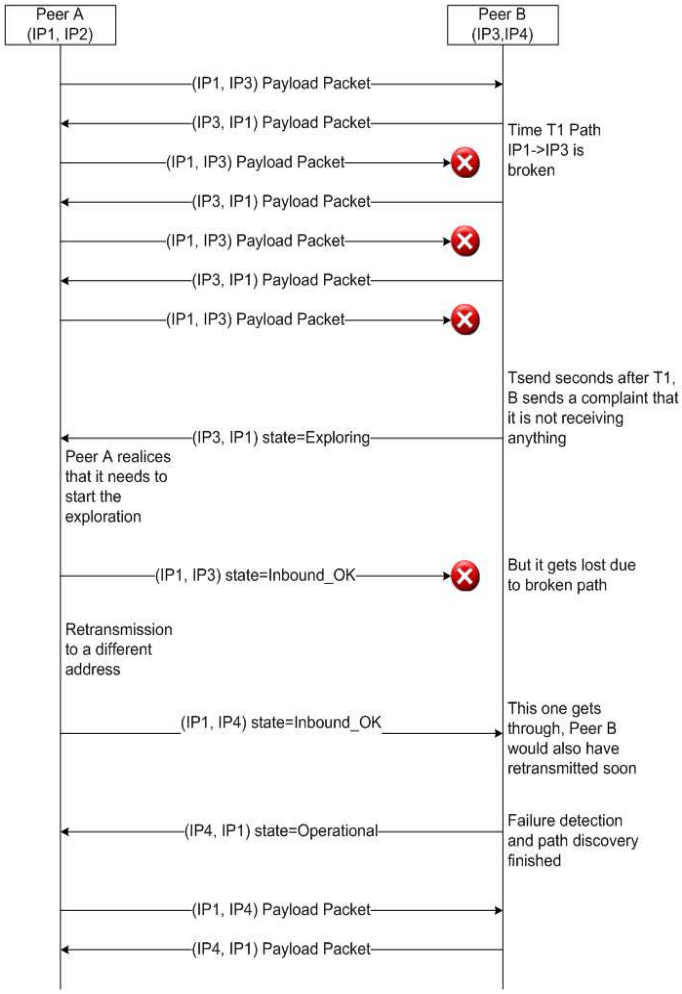

Figure 2: REAP Example

the processes. This daemon is in charge of parsing the configuration files and of running the different REAP instances, the Link Failure Simulator and, when needed, the mechanism for path modifications. The global configuration file will define the behavior of the various modules by providing a set of variables affecting the modules behavior.

Each of the modules will provide information regarding its operation by a file system structure. Each process will create a folder in which all the state variables are stored and accessible by user space applications. Examples of stored variables are address information, timer configuration of REAP, and failure characteristics.

Apart from the information accessible through the files, we propose an API for accessing these variables and modifying the running behavior of the processes. Through this API, for instance, the timers used by REAP can be changed on the fly.

4.1 Failure Detection and Path Exploration Method

As explained in section 2.2.1 the protocol selected to detect failures and find a new pair of candidate locators is REAP.

The REAP protocol relies on two timers (the Keep Alive Timer and the Send Timer), and a probe message, namely the Keep Alive message. The Keep Alive Timer (T_{KA}) is started each time a node receives a data packet from its peer, and stopped and reset, each time the node sends a packet to the peer. When the Keep Alive Timer expires, a Keep Alive message is sent to the peer. The Send Timer (T_{Send}), defined roughly as three times the Keep Alive Timer plus a deviation to accommodate the Round Trip Time, is started each time the node sends a packet and stopped each time the node receives a packet from the peer. If no answer (either a Keep Alive or data packet) is received in the Send Timer period a failure is assumed and a locator path exploration is started. Consequently, the Send timer reflects the requirement that when a node sends a payload packet there should be some return traffic within Send Timeout seconds. On the other hand, the Keep Alive timer reflects the requirement that when a node receives a payload packet there should be a similar response towards the peer within Keep Alive seconds. Note that if no traffic is exchanged, there is no Keep Alive signaling. As a consequence, there is a tight relationship between the values of the timers defined by the REAP protocol and the time required by REAP to detect a failure. The current specifications suggest a value of 3 seconds for the Keep Alive Timer, and of 10 seconds for the Send Timer, although these values are supported by neither analytical studies nor experimental data. Once a node detects a failure, it starts the path exploration mechanism. A Probe message is sent to test the current locator pair, and if no responses are obtained during a period of time called Retransmission Timer T_{RTx}, the nodes start sending Probes testing the rest of the available address pairs, using all possible source/destination address pairs. A more detailed description of the REAP protocol and an analysis of its behavior can be found on [15]. Recent work has shown that the recovery time, i.e., the time needed to get a new source/destination address pair, can be reduced by allowing REAP to send multiple probes upon failure detection [9]. Figure 2 presents a use case of the REAP protocol for better understanding. It presents a packet exchange between two end hosts, A and B. After a period of normal packet exchange the path between them fails (T_1) only in the direction from A to B. T_{Send} seconds after the failure, one of the hosts (on this case B) detects a failure on the path and starts a path exploration mechanism, which begins by sending a probe through the path being used. On this case, the probe is received by A, which answers the probe indicating that A can see B (state Inbound_OK). As this probe is being sent through the failed path, it does not reach the destination. After a Retransmission Timer, A sends another probe using other path. On this case it reaches B. After the reception of this probe, B is in operational state (it can see A and A can see B) and sends one more probe indicating its state to A. After reception of this last probe, A is in operational state too and the protocol ends.

In order to work, this protocol needs information about the incoming/outgoing application packets along with some control packets (Keep Alive, Probe). REAP is being implemented on OneLab as a service running on a sliver. The notification about the incoming/outgoing packets is obtained by the use of the PCAP library [10]. PCAP allows an application to sniff packets of a given interface and to filter these packets following selected criteria. On a OneLab sliver the PCAP library works in a similar manner as on a normal linux machine with the exception of only being able to sniffer packets which source or destination is an application running on the slice. Due to this limitation, the control packets are sent and received by standard UDP sockets.

The Failure Detection and Path Exploration module is tightly coupled to the Transparent mechanism for path modification and Link Failure Simulator. In one hand, once the module has detected a failure on the link and found a new path, it informs the Path modification module of it. On the other hand, the Link Failure simulator informs the REAP module of a failure so it starts dropping the application packets and running the path exploration mechanism defined in REAP.

4.2 Transparent mechanism for path modification

Once the Failure Detection and Path Exploration module has detected a failure and found a new path, it informs this module of the new source and destination addresses to use. This module will use these new addresses to modify the IP header of the application packets and restarting the routing process followed by the packet. This modification is performed on the reception of the application packets, so the modification of the path is transparent to the application which receives the packets exactly in the same way as before the path failure.

The modification of the packets is performed by using the Libiptc [11] library. This library allows a packet captured on an Iptables chain to be passed to user space. In user space the packet may be modified and can be incorporated to the normal packet kernel flow. On this basis, the mechanism for path modification is composed of a set of iptables rules to extract the packets from the kernel flow and a user space program which modifies the IP header of the packets and recalculates de IP checksum. The transport protocol checksum does not need to be modified since on reception the IP header is set again as the original packet and re-injected to the kernel which then computes the transport checksum with the correct values. When a packet has been modified, it is injected on the normal kernel flow, the packet is again routed according with the new IP headers and sent through the appropriate interface.

4.3 Link Failure Simulator

Based on the previous modules, the Link failure simulator must provide the mechanisms to:

- Stop the application packets flow.
- Stop the REAP control packets flow.
- Stop the sniffering of packets used on the Failure detection and Path exploration module.

To stop the application packets and control messages a rule in Iptables can be used. A simple rule on the INPUT chain of Iptables dropping the application packets and control messages is used to stop all the traffic exchange on the slice.

However the PCAP library sniffs the packets directly from the interface before these packets are processed by the kernel and the Iptables rules does not apply over it. To overcome this issue the Link Failure simulator sends an inter-process signal to the Failure detection and Path exploration module which bypass the processing of the messages sniffered by pcap, actively triggering the failure detection mechanism.

5. Use of the Multihoming Functionalities

We envision several application scenarios for the platform described above. For instance, a video content distribution service might test, on the OneLab testbed, how the video service is impacted by a link failure and how multihoming can protect the content delivery. Further, a study of the parameter configuration related to the recovery can be done in a large-scale environment.

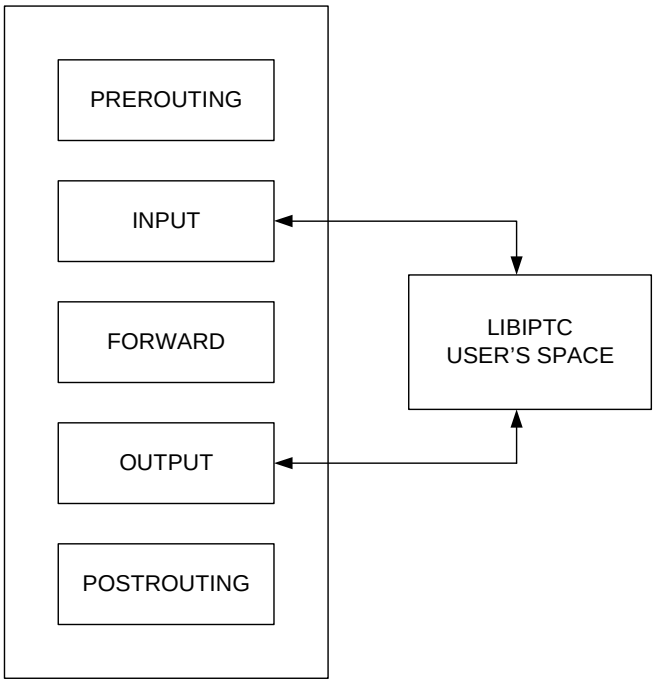

Figure 3: Transparent mechanism for path modification

One can also imagine that REAP can be extended in order to provide additional path characteristics. Indeed, with the growing importance of the Internet combined with the growth of multihoming [8], more and more Internet users needs to better select the paths taken by their packets. Today, although many of measurements techniques have been developed within the IPPM working group of the IETF, an application that needs to select a path must implement its own measurement system to obtain data for performing its selection. This implies that several applications running on the same host or in the same campus will probably perform almost the same kind of measurements. In the long term, duplicating those measurements is not the appropriate solution. A better solution would be to permit the REAP component to query a service that is run continuously. This service would provide information about:

- the topology of the visible part of the Internet. See Donnet and Friedman [12] for further details on topology discovery.
- the delay. See Wang et al. [13] for details on delay measurements.
- the bandwidth. See Prasad et al. [14] for details about the bandwidth estimation.

Researchers can use this platform to investigate the effects of multihoming and failure recovery mechanisms on their applications, including the study of parameter configurations related with the recovery. If REAP is extended to provide information about some characteristics of the paths, researchers can study the impact of these characteristics on their applications.

Besides, there is an ongoing discussion about what information about path conditions and path changes would be useful for the upper layers.

Such upper layers include transport layers with congestion control procedures and applications that are sensitive to path conditions. This mechanism would allow researchers to explore how upper layers could use a more detailed network layer information and conduct experiments with this.

Finally, this service could be used as a starting point to build more complex multihoming solutions. For example, the module that manages different addresses in the node assumes static configuration, but researchers could use this basic functionality to test a protocol for exchanging dynamically and in a secure way end-point addresses information.

6. Conclusion

In this paper, in the context of the OneLab project, we explained how PlanetLab-like nodes can be extended in order to support multihoming. The multihoming functionalities chosen to be part of the solution are:

- A failure detection and path exploration method.
- A transparent mechanism able to change the path used by a flow.
- A mechanism to simulate link failures.

First we have described the failure detection and path exploration method. It is based on a modified version of REAP so that it can work under an IPv4 network. We next discussed how the multihoming component modifies packet headers to change the IP addresses in case of a failure. This mechanism, based on libiptc, is transparent to applications. Finally we have described our link failure simulator which aims at creating artificial failures.

A new feature in a testbed is useless if it cannot be used by the community. We therefore described, in this paper, a bunch of application scenarios for which our multihoming module is suitable

The multihoming component described in this paper has been developed for an IPv4 environment. A next step in the multihoming development in OneLab would be to implement IPv6.

Acknowledgment

Authors of this paper are supported by the European-founded 034819 OneLab project. We would like to thank the OneLab consortium for their support and suggestions that improved this work..

References

[1] IETF Home Page: http://www.ietf.org

[2] OneLab Home Page: http://www.one-lab.org

[3] J. Arkko and I. Van Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming”, IETF draft draft-ietf-shim6-failure-detection-07 (work in progress), December 2006.

[4] L. Peterson and T. Roscoe, “The Design Principles of PlanetLab”, PlanetLab Design Note PDN-04-021, available at https://www.planet-lab.org/doc/pdn, January 2006.

[5] L. Peterson, S. Muir, T. Roscoe, and Aaron Klingaman, “PlanetLab Architecture: An Overview”, PlanetLab Design Note PDN-06-031, available at https://www.planet-lab.org/doc/pdn, May 2006.

[6] E. Nordmark and M. Bagnulo, “Level 3 multihoming shim protocol”, IETF draft-ietf-shim6-proto-08 (work in progress), May 2007.

[7] N. Montavont, R. Wakikawa, T. Ernst, C. Ng and K.. Kuladinithi, "Analysis of Multihoming in Mobile IPv6", IETF draft-ietf-monami6-mipv6-analysis-02 (work in progress), Feb. 2007.

[8] S. Agarwal, C.-N. Chuah and R. H. Katz, “OPCA: Robust Interdomain Policy Routing and Traffic Control”, in Proc. OPENARCH, Apr. 2003.

[9] S. Barré and O. Bonaventure, “Improved Path Exploration in shim6-based Multihoming”, in Proc. ACM SIGCOM IPv6 Workshop, Aug. 2007.

[10] V. Jacobson, C. Leres and S. McCanne, “tcpdump”, UNIX, man page, 1998. See also http://www.tcpdump.org

[11] M. Boucher, M. Josefsson, J. Kadlecsik, J. Morris, H. Welte and R. Russel, “Netfilter: Firewall, NAT, and Packet Mangling for Linux”, UNIX, man page, 1999. See also http://www.netfilter.org

[12] B. Donnet and T. Friedman, "Internet Topology Discovery: a Survey", In IEEE Communications Survey and Tutorials. 2007. to appear.

[13] J. Wang, M. Zhou and L. Yuxia, "Survey on the End-to-End Internet Delay Measurements", In Proc. 7th IEEE International Conference on High Speed Networks and Multimedia Communications (HSNMC), June/Jul. 2004.

[14] R. Prasad, C. Dovrolis, M. Murray and kc claffy, "Bandwidth Estimation: Metrics, Measurement Techniques and Tools", In IEEE Network, 6(17), Nov./Dec. 2003, pp. 27-35.

[15] A. de la Oliva, M. Bagnulo, A. Garcia-Martinez and I. Soto. “Performance Analysis of the REAchability Protocol for IPv6 Multihoming” Accepted for publication in NEW2AN 2007, Conference on Next Generation Teletraffic and Wired/Wireless Advanced Networking, September 2007.