Control and Management Plane in a Multi-stage Software Router Architecture

A.Bianco *, R.Birke *, J.Finochietto †, L.Giraudo *, F.Marenco *, M.Mellia *, A.Khan ‡, D.Manjunath ‡
* Dip. di Elettronica, Politecnico di Torino, Italy, Email: {firstname.lastname}@polito.it
† CONICET - Universidad Nacional de Cordoba, Argentina, Email: {jorge.finochietto}@ieee.org
‡ IIT Bombay, India, Email: azeem@it.iitb.ac.in

Abstract— **Software routers based on Personal Computer (PC) architectures are receiving increasing attention in the research community. However, a router based on a single PC suffers from limited bus and Central Processing Unit (CPU) bandwidth, high memory access latency, limited scalability in terms of number of network interface cards, and lack of resilience mechanisms. Multi-stage architectures created by interconnecting several PCs are an interesting alternative since they allow to i) increase the performance of single-software routers, ii) scale router size, iii) distribute packet-forwarding and control functionalities, iv) recover from single-component failures, and v) incrementally upgrade router performance. However, a crucial issue is to hide the internal details of the interconnected architecture so that the architecture behaves externally as a single router, especially when considering the control and the management plane.**

In this paper, we describe a control protocol for a previously proposed multi-stage architecture based on PC interconnection. The protocol permits information exchange among internal PCs to support: i) configuration of the interconnected architecture, ii) packet forwarding, iii) routing table distribution, iv) management of the internal devices. The protocol is operating system independent, since it interacts with software routing suites such as Quagga and Xorp, and it is under test in our labs on a small-scale prototype of the multi-stage router.

I. Introduction

The request for high-performance switching and transmission equipment in modern routers keeps growing, due to the continuous increase in the diffusion of information and communications technologies and new multimedia and/or bandwidth-hungry applications and services like VoIP, IPTV and file sharing. Routers are able to support the performance growth by offering an ever-increasing transmission and switching speed, mostly due to the technological advances in microelectronics.

However, contrary to what occurred for PCs – where standards were defined, allowing the development of an open, multi-vendor market – the field of networking equipment in general, and of routers in particular, has always been characterized by the development of proprietary architectures. This means incompatible equipments and architectures, especially in terms of configuration and management procedures, as well as the requirement to train network administrators to handle several proprietary architectures or to be limited to a single vendor. Thus, the final cost of equipments is high with respect to performance and equipment complexity. Software routers based on off-the-shelf PC hardware and open-source software are appealing alternatives to proprietary network devices because of the wide availability of multi-vendor hardware, the low cost, the continuous performance evolution driven by the PC-market economy of scale and the large availability of open-source software for networking applications, such as Linux, the Berkeley Software Distribution (BSD) derivatives, Click [1], XORP [2], and Quagga [3].

Indeed, despite the limitations of bus bandwidth and central processing unit (CPU) and memory-access speed, current PC-based routers have a traffic-switching capability in the range of a few gigabits per second, which is more than enough for a large number of applications. Moreover, performance limitations may be compensated by the natural PC architecture evolution, driven by Moore's law. However, high-end performance and large size devices cannot be obtained easily today with routers based on a single PC. In addition to performance limitations, several other objections can be raised to PC-based routers such as: software limitations, scalability problems, lack of advanced functionalities, inability to support a large number of network interfaces, as well as the inability to deal with resilience issues to match the performance of carrier-grade devices.

To overcome some of the limitations of software routers based on a single PC, we proposed to create a large router exploiting multi-stage-switching architectures [4]. However, this device needs an internal control protocol to coordinate the routing process among the several PCs of the multi-stage router, while maintaining a high level of independence from the used software (kernel and control plane).

Due to the need of high number of input/output ports and high bandwidth, the number of PCs in the multi-stage architecture could be large. Thus, many internal elements need to be controlled and configured. Furthermore, to limit management costs and complexity, the interconnected structure must appear to other network routers as a single entity router, which implies special coordination requirements. In this paper, we describe a control protocol for the previously proposed multi-stage architecture based on PC interconnection. The protocol permits information exchange among internal elements to support: i) configuration of the interconnected architecture, ii) packet forwarding, iii) routing table distribution, iv) management of the internal devices.

978-1-4244-1982-1/08/$25.00 ©2008 IEEE

II. Related Work

Multi-stage switching architectures were previously proposed in different research areas to overcome single-machine limitations [5]. Initially studied in the context of circuit-oriented networks to build large-size telephone switches through simple elementary switching devices, multi-stage architectures were traditionally used in the design of parallel computer systems and, more recently, considered as a viable mean to build large packet-switching architectures.

Indeed, the major router producers have proposed proprietary multi-stage architectures for their larger routers [6], [7] that follow traditional multi-stage, telephony-derived switching architectures. In most cases, the routing functionality is distributed among cards installed in different interconnected racks. Such systems target high-end routers, with performance and costs that are not comparable with those of PC-based router architectures.

High-end routers typically are based on a synchronous behavior, whereas our proposed solution is fully asynchronous. Therefore, our solution does not require a global clock distribution, a complex task that requires large power consumption and limits router scalability. Furthermore, the synchronous behavior often brings about the internal switching of fixed-size data units, which implies segmentation of the variable-sizes IP packet at inputs and especially, re-assembly procedures at outputs, another complex and power-consuming task.

Similarly to our approach, the IETF is interested in distributed router architecture. The Forwarding and Control Element Separation (ForCES) Working Group [8] aims at proposing standards for the exchange of information between the control plane and the forwarding plane, when the control and forwarding elements are either in the range of a small number of hops or even in the same box. A distributed implementation of the control plane on software router architectures was proposed in [9], where the focus is mainly on control plane issues. The ForCES approach is rather general and does not contain features which are fundamental and peculiar of our architecture, such as device identification, device role as balancer, master or slave, etc.

A. Multi-stage architecture

Our reference multi-stage architecture is shown in Fig.1. The key advantages of the proposed architecture are the ability to:

- overcome performance limitations of a single–PC–based router by offering multiple, parallel data paths to packets.
- upgrade router performance by incrementally adding more switching elements or incrementally upgrading each switching element.
- scale the total number of interfaces the node can host, and, as a consequence, the router capacity.
- automatically recover from faults of any PC/element.
- support a fully asynchronous behavior.
- provide functional distribution, to overcome single–PC CPU limitations, for example allowing the offloading of CPU intensive tasks, such as filtering/cryptography, to dedicated PC.

The distributed router is composed by three different stages:

- *first stage*: an array of Load Balancers (LB) used to distribute the load across the routers at the back-end stage. This stage supplies the input/output ports of the multi-stage router. Since the task is simple, they can also be easily implemented in hardware using FPGAs, although standard PCs can be employed.
- *interconnection network*: this is a standard Ethernet network, eventually with multiple paths between the LBs and back-end forwarding engines (FE) to support fault recovery and to upgrade the switching capability of the router.
- *back-end*: an array of PCs running a software routing program to act as a FE.

In addition to these elements, the multi-stage architecture needs a control element, named *Virtual Control Processor* (CP), which controls and configures all the elements of the router, such as to present the whole architecture as a single entity to external agents.

III. Control and Management Plane

The multi-stage architecture requires a control protocol to coordinate the behavior of the different elements. The main issues to consider in the design of such a protocol are:

- *virtual CP definition and identification*
- *route distribution*
- *fault recovery*
- *security*

A. Virtual CP

The task of the virtual CP is to mask the internal architecture of the multi-stage router and let it appear as a single entity to the outside entities considering both human users and other routers. Therefore, the virtual CP must manage the routing protocols and interfaces. These tasks need the ability to redirect control messages to the correct destination and to execute configuration tasks from a centralized interface.

B. Route distribution

Every FE must have the same routing table, but only the virtual CP has the needed information to build it acting as the public end point for the routing protocols. Thus, the control protocol must distribute the routing information to all the FE at the back-end stage. The protocol must include packet recovery mechanisms to prevent inconsistencies in the routing tables.

C. Fault recovery

The control protocol must have the ability to react to failures occurred to any element of the router, activating procedures to substitute broken elements or to reconfigure the network. The most critical point of failure is the virtual CP, so the control protocol must monitor its functions and eventually substitute the virtual CP.

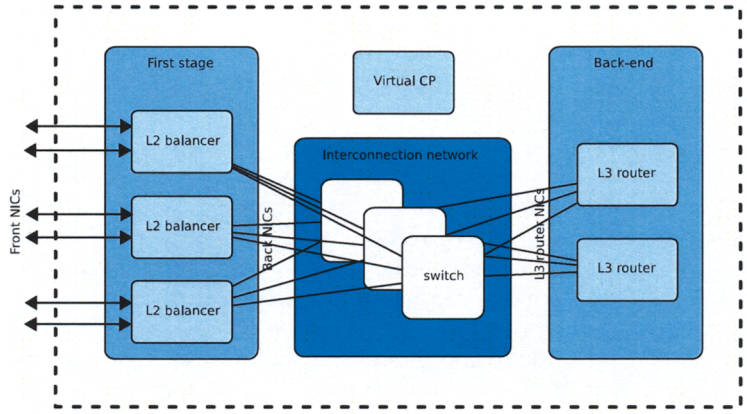

Fig. 1. Multi-stage architecture: the router is composed by three stages and by a control element to mask the internal architecture to an external observer.

D. Security

Security mechanisms are necessary to prevent malicious router manipulation like injection of false routes.

IV. The DIST Protocol

DIST is the protocol designed and developed in our labs to implement the control and management plane requirements in the multi-stage router architecture. The protocol is managed by demons running on every FE, together with traditional routing software. Each demon cooperates strictly with the routing software working as a "plugin". The current implementation supports both Xorp and Quagga.

We chose to create DIST as a plugin instead of directly interfacing with the kernel to be more operating system independent. Furthermore, it would be out of our objectives to write our own implementation of the routing protocols.

The virtual CP is implemented by choosing one FE as master node. This node runs all the routing protocols; LBs redirect all the routing protocol traffic to the master node. The master communicates with the other FE (slaves) using a master-slave communication paradigm. Every FE is identified with a 16 bit number (randomly selected at start-up) included in all generated messages, and the multicast group to join to receive and send control information is well-known. The element ID can be separated in two parts, element type (2 bit) and element id (14 bit); the node type could be *master*, *slave* or *balancer*.

Since the number of active elements in the multi-stage architecture could be large due to the need of supporting high switching capability, significant signaling information may be exchanged between the FEs. To limit the amount of bandwidth allocated to signaling and the resources devoted to this task, the DIST protocol exploits IPv4 multicast transmissions. A single copy of every signaling message can be send to all other elements, without knowing the details of other nodes, thus reducing the amount of generated traffic. The choice of using multicast addresses forces the use of UDP as the transport protocol, which does not supply any reliability mechanism: the control protocol must supply custom procedures to detect and resolve errors or missing packets.

A. Route distribution

The master receives routing table updates from the control plane software via LBs and must distribute the information to the slaves using specific messages of the signaling protocol (route add/delete messages). The routing information is described in an independent format to allow the interoperability of different routing softwares (i.e. Quagga or XORP). When an update is send, the master increments the sequence number used to identify the different routing table updates messages.

When a slave receives an update message (or an hello packet), the header field is parsed to establish the sequence number of the message; the current message can be accepted only if all the previous updates have been received and accepted. In this case, the packet is processed and sent to the routing software which updates the local routing table. No acknowledgment is sent to the master, to prevent scalability problems due to ack storm.

If the sequence number is ahead of the expected one, the slave sends a negative acknowledgment (NACK) to the master with the expected sequence number, triggering a retransmission of all packets starting from the lost packet. This procedure was defined to avoid sending many NACKs when a burst of packet is lost. The master serves only one NACK in every hello interval, to prevent multiple retransmissions of messages.

A ring buffer is used by the master to store the last N update packets sent. The size N of the ring buffer is a key parameter of the protocol, because it determines how far in the past a packet can be lost and still recovered. If the sequence number of the lost packet is less than the actual sequence number minus N, this error recovery scheme fails.

To overcome this limitation, we introduce an alternative method to synchronize the routing table; if the slave detects more than N consecutive missing updates, it sends a full routing table download request to the master. This download is done using a reliable TCP connection. This also solve the issue of synchronizing the routing table of newly connected

FEs e.g. to recover from a previous fault or to upgrade the multi-stage router capability.

The fact that the routing information can be acquired trough two concurrent methods may lead to inconsistencies. To avoid this, while the master is making a full routing table transfer, updates of the routing table are put temporarily on hold, storing updates coming from routing software. When the transfer ends, both master and slave resume the normal operations by processing the stored updates.

B. Hello packets

A lost packet is identified only when the slave receives an update containing a sequence number larger than the one it is expecting. If no updates are sent, the packet loss remains undetected. To avoid this and to manage FEs ups and downs, the protocol uses hello packets. Every node sends periodically a hello packet, once packet in every period named hello time interval. The hello packet coming from the master contains the sequence number of the last message, so that slaves can detect a possible packet loss.

The hello packets are also used to advertise the presence of an element allowing automatic (re)configurations and as a heartbeat mechanism for fault recovery. Finally, they are used to distribute the initial sequence number of update packets.

C. Automatic configuration

One important configuration task is to select who FE acts as the master. This is done during the *master election phase*: when a new FE joins, it chooses a random ID and sets the candidate master ID to its own ID. Then, for three hello time periods, it only sends and listens to hello packets, updating the list of alive FEs.

At every received hello packet, the FE verifies the sender type. If the type is set, to master the multi-stage router has already a master and the *master election phase* ends. If the sender type is set to slave, the sender ID and the candidate master ID are compared. If the sender ID is smaller than the candidate master ID, the sender becomes the new candidate master. At the end of the master election phase, every element knows the master ID and it will accept route updates only if generated by the master.

Since the initial ID is chosen at random, a problem that may occur is the presence of two elements with the same ID. To solve this problem, a new command message is introduced. When an element receives an hello packet with the sender ID equal to its own ID, it sends immediately a packet which signals the problem to the sender. When the sender receives a notification of duplication, it changes its ID choosing randomly a new ID and it postpones the end of the master election phase. This insures that at the end of the *master election phase* every element has an unique ID.

Another automatic configuration procedure is needed for the LBs. When a LB sees a packet with type set to "master", it updates the filters for the routing protocols so as to redirect the traffic to the proper address of the master FE. The load balancing is done according to the values contained in a table of active slave FEs.

D. Fault Recovery

One of the requirements for the distributed control plane is fault recovery. If a node fails, an automatic reconfiguration of the system must be started to minimize the effect on the whole system. The failures are detected verifying the table of active FEs. The table contains the ID of all active FEs (i.e. FEs that sent a packet "recently") and the time at which the last packet was received from the corresponding FE. This table is checked periodically. Any entry which is older then five hello time periods is considered unavailable, is dropped and a fault recovery mechanism is started.

The action taken depends on the node type. The most critical failure is the failure of the master node. In this case the slaves start a slightly modified *master election phase* to speed up the process. During normal operation, at every hello packet reception, a FE keeps track of the smallest slave sender ID and selects it as backup master. Again, since the hello packets are received by all FEs, every FE makes the same choice. When the master is dropped from the list of active elements, the backup master becomes immediately the new master and sends an hello packet to advertise other nodes and to stop their master election phases.

If a slave fails, it is simply dropped from the table of active nodes. Therefore, the LB will not send any traffic to it anymore.

A failure of a LB is considered equivalent to a link/interface failure. No special recovery mechanism is implemented in this case.

E. Security

In the first version of the protocol, security was not considered one of the main issues. Therefore, the only security mechanism implemented is a filter at the LBs, dropping all messages coming from outside and using the internal DIST multicast address.

V. Testbed Setup

In this section, we evaluate the time needed to distribute route information between master and slaves FEs in the multi-stage back-end. This is the only additional delay introduced in our multi-stage architecture with respect to the delays of a traditional single-stage router. Our aim is to verify if the penalty introduced by the need of internal route distribution is negligible with respect to the time needed to recompute the new routes when an update is received through a routing protocol message.

We realized a small scale prototype of the multi-stage architecture in our labs, as shown in Fig.2. Three identical PCs form the backplane of FEs. A forth PC simulates an exterior router sending route updates to the multi-stage router using the OSPF routing protocol. A fifth PC records the traffic trace from which the timing information is extracted off-line. In this scheme, the LBs are missing since they are not participating directly in the route distribution.

The use of the sniffer allows us to have a single measurement point avoiding timing synchronization problems between

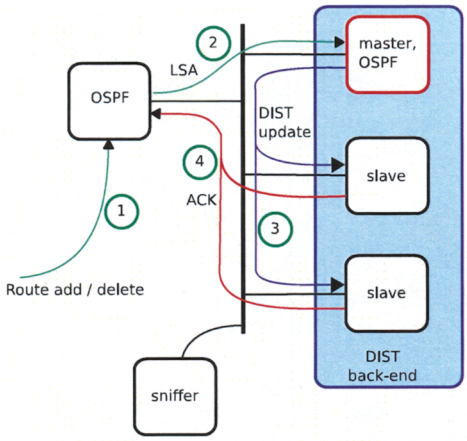

Fig. 2. Network topology used in measurements.

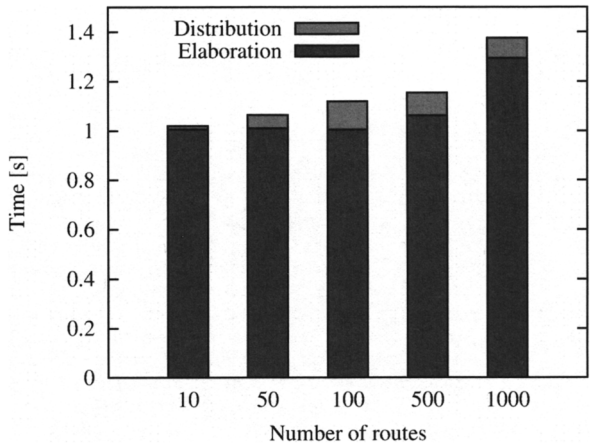

Fig. 3. Average update time versus number of routes using XORP

different PCs. However, to be able to better measure the times of each elaboration phase, the DIST daemons have been slightly modified to send explicit acks at the end of each significant phase. These acks are sent using a broadcast address to allow the sniffer to catch them, and are ignored by all other FEs.

The tests were performed using both XORP 1.5 (CVS version 2007/08/24) and Quagga 0.99.5 as routing software. The test results were averaged over five runs.

The hardware used in our experiments is:

- **OSPF Router** Pentium IV (2.4 GHz, 512 KB cache), 1GB RAM, Fast Ethernet, Linux 2.6.20-16-generic (ubuntu 7.04)
- **Back-end** Pentium III (800 MHz, 256 KB cache), 384 MB RAM, Fast Ethernet, Linux 2.6.20-16-generic (xubuntu 7.04)
- **Sniffer** Pentium IV (2.4 GHz, 512 KB cache), 512 MB RAM, Fast Ethernet, Linux 2.6.20-16-generic (ubuntu 7.04)
- **Switch** 3Com OfficeConnect Gigabit Switch 8 working at 100 Mbit/s

VI. Measurement results

The first measurements concern the route distribution. We want to determine the overhead introduced by the route distribution.

To this aim, we use the OSPF router to send OSPF LS updates to the master node. After the elaboration of a LS update, the route is passed to the DIST plugin and a route update is sent to the slaves. The slaves update the routing table and send an explicit ack to the sniffer.

We consider the difference in the timestamp between the LS update and the DIST route update message as the elaboration time of the OSPF packet. We consider the difference between the DIST route update message and the last ack received from the slaves as the route distribution time.

Fig.3 shows the total average time needed to update a route within the multi-stage router using XORP. The total time is split into elaboration time and distribution time.

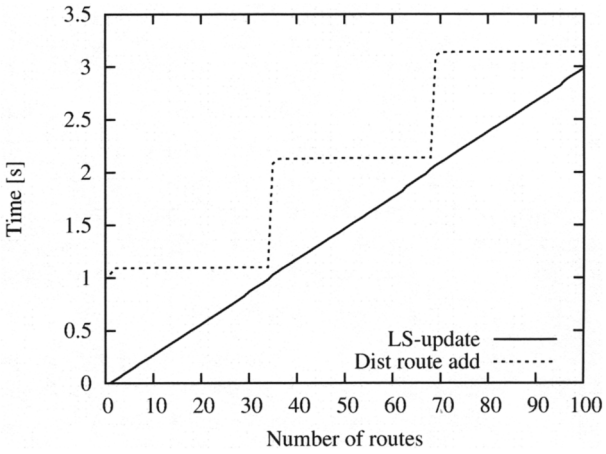

Fig. 4. XORP OSPF timeouts.

Observe that the elaboration time is about one order of magnitude larger than the distribution time. This means that the overhead introduced in the multi-stage architecture due to the route distribution process is really limited. However, the elaboration time is faily long (about 1 s). This is not dependent on our protocol, but it is a consequence of the behaviour of the XORP OSPF daemon, that starts a timer at the first packet reception to aggregate LS updates before updating the routing table. At the end of the timer, received packets are elaborated. Then, a new timer is scheduled. This is clearly shown in Fig.4.

Fig.5 shows the total average update time but when using Quagga. Again, the elaboration time is about one order of magnitude larger than the distribution time, meaning that the route distribution time introduced by our protocol is negligible. Quagga does not aggregate LS-updates; therefore, the elaboration time is much smaller. Also, the distribution time is smaller compared to XORP. This is due to a more complex internal scheduling scheme adopted in XORP with asynchronous inter-process communication based on callback

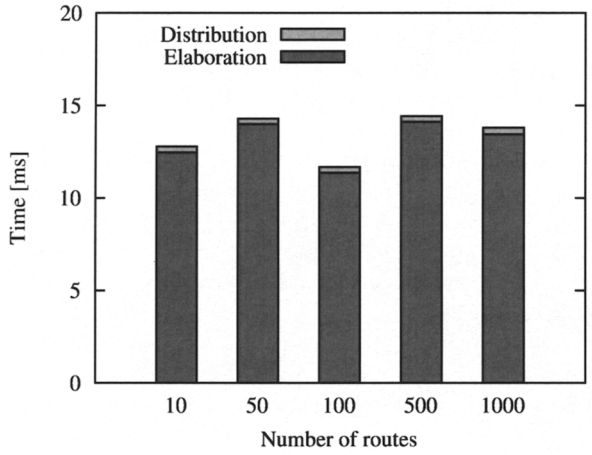

Fig. 5. Average update time versus number of routes using Quagga.

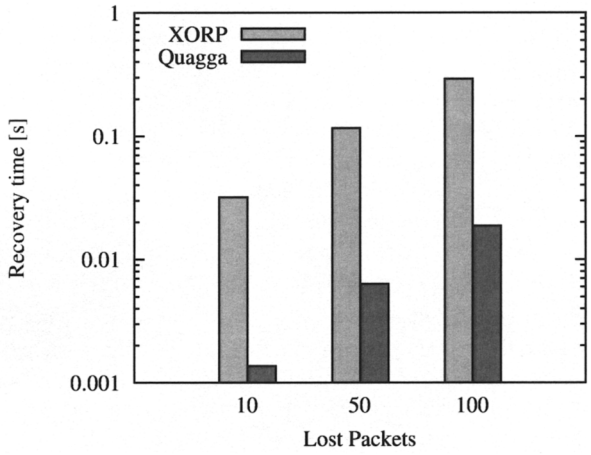

Fig. 6. Resynchronization time with N consecutive lost routes.

functions, which introduces some overheads.

The second measurements concern the error recovery mechanism based on the negative acknowledgments. To simulate packet losses, we disconnect the network cable from a slave FE. Then, we generate a number N of routes. When routes have been distributed, the isolated slave is reconnected to the multi-stage router. When the slave receives the first hello packet from the master, it synchronizes its routing table using a single cumulative NACK. The time interval between the master hello packet and the end of the route distribution to the previously isolated slave FE is extracted offline from the sniffer's packet trace.

First, we did functional tests to verify that lost packets are correctly recovered and that the routing tables are synchronized. As expected, from Sec.IV-A, this is true as long as the burst of lost packets is smaller than the size of the transmit ring buffer N. Later, we measured the time needed to recover N lost routes. This time is extracted from the sniffer packet trace using the timestamp of master hello packets and the last acknowledgement sent by the isolated slave.

To avoid recovery failures, we set the size of the transmit ring buffer to 1000 messages. Fig.6 shows the results.

As expected, the recovery time is proportional to the number of lost packets. This is true for both XORP and Quagga plugins. However, better looking at the timescales, notice that the XORP plugin is about one order of magnitude slower, although the same operations are done in both plugins. The reason is again the more complex internal scheduling scheme adopted in XORP. This is confirmed also by the average time needed to send a NACK when a hello packet is received: 30ms in the XORP plugin versus 3ms in the Quagga plugin.

VII. Conclusion

We described a control protocol needed to control the behaviour of PCs acting as forwarding elements in our proposed multi-stage router architeture. The protocol interacts directly with software routing suites such as Quagga and Xorp, to be operating system independent.

We showed that the protocol works correctly, allowing the multi-stage architetures to behaves externally as a single-stage router. The protocol is able to deal with internal elements failures and recovery. The additional delay needed to distribute routes to internal elements is negligible with respect to the time needed to recompute new routes. This shows that the delay performance penalty introduced by the multi-stage architecture is not a major issue and is largely compensated by its property of scalability and robustness to failures.

Acknowledgment

This work was done within the research project Bora-Bora (Building Open Router Architectures-Based On Router Aggregation, URL: http://www.tlc-networks.polito.it/projects/borabora/) funded by the Italian Ministry of University and Research.

References

[1] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek *The Click Modular Router*, ACM Trans. Comp. Sys., vol. 18, no. 3, Aug. 2000, pp. 263—97.

[2] M. Handley, O. Hodson, E. Kohler, *Xorp: An Open Platform for Network Research*, Proc. 1st Wksp. Hot Topics in Networks, Princeton, NJ, Oct. 28—29, 2002.

[3] GNU, *Quagga*, `http://www.quagga.net`

[4] A. Bianco, J. M. Finochietto, G. Galante, M. Mellia, F. Neri *Multi-stage Switching Architectures for Software Routers*, IEEE Network "Advances in Network Systems", ISSN: 0890-8044, July 2007

[5] J. Duato, S. Yalamanchili, L. Ni, *Interconnection Networks: An Engineering Approach*, IEEE Comp. Soc. Press, 1997.

[6] Cisco Systems, *Carrier Routing System*, `http://www.cisco.com/application/pdf/en/us/guest/products/ps5763/c1031/cdccont_0900aecd800f8118.pdf`

[7] Juniper Networks, *Routing Matrix*, `http://www.juniper.net/solutions/literature/white_papers/200099.pdf`

[8] IETF, *Forwarding and Control Element Separation (ForCES)*, `http://www.ietf.org/html.charters/forces-charter.html`

[9] H. Hagsand, M. Hidell, and P. Sjodin, *Design and Implementation of a Distributed Router*, Proc. 5th IEEE Int. Symp. Sig. Proc. Info. Tech., Athens, Greece, Dec. 18—21, 2005.