International Journal of Computer Applications (0975 8887)
Volume 89 - No. 3, March 2014

A Load Balancer for a Multi-Stage Router Architecture

Shadi Atalla, Andrea Bianco
Dip. di Elettronica.
Politecnico di Torino, Italy

Robert Birke, Luca Giraudo
Dip. di Elettronica.
Politecnico di Torino, Italy

ABSTRACT

Multi-stage software router architectures permit to overcome several limitations inherent to single stage software routers. One of the key elements of the multi-stage architecture under study are the load balancers, which are used to distribute the load among back-end routers. However, using a PC (Personal Computer) as a load balancer could create a performance bottleneck in the overall architecture. Since the operations performed by the load balancer are simple, we explore the possibility of an hardware-based implementation of load balancing functionality with the goal of improving its performance. In this paper, we describe the architecture of an FPGA-based load balancer and we present some performance results of its prototype implementation. ■

1. INTRODUCTION

Software routers (SRs) based on open-source software and PC-architecture are a valid alternative to expensive commercial routers because of their extensibility, flexibility and low cost. Today, software routers implement a set of functionalities comparable to those of commercial routers. However, they are not yet able to keep up with routing performance and interface scalability of commercial routers based on Application-Specific Integrated Circuits (ASIC), mainly due to the general purpose PC-architecture.
For these reasons, the research community has proposed many improvements for software routers through software and hardware optimization [13, 15, 14, 5] or through aggregation to increase SRs scalability [18, 16, 23]. A multi-stage router architecture was presented in [11, 12, 10, 8, 19, 9], based on the idea of building a router through aggregation of multiple software routers. More precisely, the multi-stage architecture considered in this paper (shown in Fig. 1) is organized in three stages:

—**Front-end stage**: PC-based (or FPGA-based) layer 2 load balancers (LBs) are used to balance the incoming data traffic among back-end routers;
—**Back-end stage**: PC-based routers which implement the forwarding/routing/management functions;
—**Interconnection stage**: one or more Ethernet switches used to interconnect the above two stages.

The existing prototype of the multi-stage router is software-based only and it relays on well-known technologies like Linux, Click Modular Router and XORP (or Quagga). It supports multiple LBs and routers as presented in [10, 8, 19, 9] and it features the DIST control plane running in a logically centralized entity named Virtual-CP. The Virtual-CP is responsible mainly for i) internal element identification, ii) router management, iii) LB configuration,

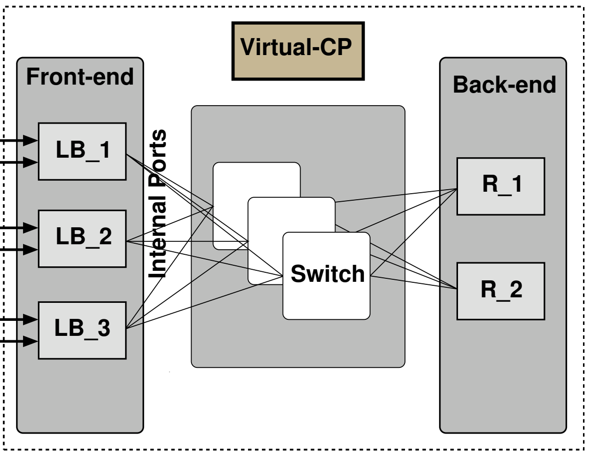

Fig. 1. Multi-stage router architecture.

iv) automatic fault recovery mechanism management for resilience, v) route distribution among back-end routers.
Whereas a software-only solution is most suitable for fast development and deployment of new features, it may provide poor packet forwarding performance due to single element limitations (routers and LBs). On the one hand, recent advances in research [5] show that the routing performance of single software routers can be increased up to 4.4 Gigabit/s (considering minimum size packets) by kernel optimization. On the LBs side, being most of the required functionalities relatively simple to implement, an hardware based approach may help in scaling performance. Furthermore, layer 2 software based forwarding performance are often surprisingly unsatisfactory [20]. Finally, the presented hardware solution based on a FPGA permits to extend the open-source approach to the hardware domain [2], without incurring in the classical drawbacks of custom hardware development, such as high cost, lack of reprogrammability and flexibility.
Thus, we explore the possibility of an hardware-only LB solution with the main goal of improving performance while maintaining extensibility and flexibility. In this paper, we present a FPGA-based implementation of load-balancer functionalities for the multi-stage router architecture. The project is based on the NetFPGA platform [4], a development PCI board equipped with four Ethernet 1Gbit/s ports and a Xilinx FPGA (Field Programmable Gate Array) used to deploy a custom-made forwarding logic.
It is worth emphasizing that it is possible to support even higher line rates by switching to faster hardware. As far as the last point

is concerned, Application-Specific Integrated Circuits (ASICs) are usually 4 times faster than FPGA (with a 30 times smaller area) [21, 7, 6].
The main contributions of this paper are: i) design, implementation and test of advanced low-level packet manipulation mechanisms, ii) integration of an hardware solution into an existing distributed router architecture obtaining performance improvements and iii) identification of a design issue in the NetFPGA board (e.g. limited access bandwidth to the external SRAM).

2. RELATED WORKS

The NetFPGA [4] was developed by Stanford University as a platform for both research and network experimentation. It consists of a single Xilinx Virtex-II Pro 50 FPGA, two 2 MB (512kx36) SRAMs, a quad-port Gigabit Phy, on a PCI card, with the PCI interface implemented in a Spartan II FPGA.
The availability of powerful programmable logic permits to extend the open software paradigm to the hardware domain. The logic circuitry developed for FPGAs can be made public, reused and improved by the research community. This open hardware approach can open the door to low-cost hardware implementations of performance-critical functional blocks. In this paper, we present a detailed description of a Hardware loadbalancer based on a FPGA implementation, whose performance is partly assessed in this paper. An important motivation to develop the core is providing the research community with an open-source VHDL core implementing the fast loadbalancer capable of communicating with all back-routers and VC. However, due to the clear needs of performance, we concentrate on using hardware based loadbalancer for Software Router. Particularly, we propose a reconfigurable architecture for load balancing of high-speed networks, and implement this design using FPGAs. By making use of the inherent parallelism of FPGA hardware, we are able to speed up our application by a considerable amount as compared to an equivalent software implementation. Our architecture is pipelined to achieve a high throughput, making it suitable for multi-gigabit networks.
It is common for a large web sites to balance load over many HTTP servers, and there exist commercial products to do this [1, 3]. Load-balancing may be oblivious (e.g.,spreading the requests equally over all servers, without regard for their load), or stateful (e.g., sending requests to the least-loaded server). In a data-center or a dedicated web-hosting service, the HTTP servers are connected by a regular, over-provisioned network; the load-balancer usually does not consider the network state when load-balancing across servers. Authors in [17] demonstrates a NetFPGA based load-balancer, called Plug-n-Serve, that load-balances over arbitrary unstructured networks, and tries to minimize the average response time. The system allows operators to increase the capacity of the web service by simply plugging in computing resources and switches in an arbitrary manner. Unlike the Plug-n-Serve, the proposed loadbalancer cannot precisely route individual connections, only hash-based or round robin aggregates of connections to balance flows and prevent congesting any of the back end routers.

3. ARCHITECTURE OVERVIEW

Load Balancers in the Front-end stage of the multistage router are responsible for forwarding each packet in two possible directions of data flow.
The main task of the LB in the multi-stage architecture is to balance the load among back-end routers to obtain a larger forwarding rate than in a single SR thanks to data traffic distribution.

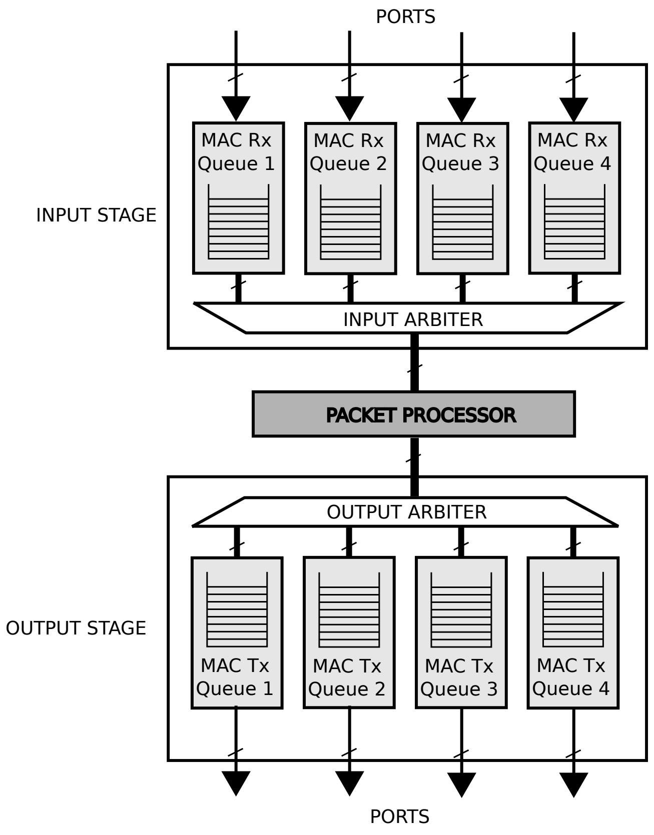

Fig. 2. LB architecture overview.

The balancing function is done by addressing data packets at the MAC layer to specific back-end routers without considering upper layer protocol headers. More precisely, at every packet reception from an external host the LB chooses an internal destination back-end router from a balancing table according to two different schemes: *Weighted Round Robin* or *Hash-based* balancing. In the first case, the balancing table is explored sequentially, meanwhile in the second case a hash function (computed on packet headers) is used to access the table. The hash based approach avoids packet reordering by sending all packets belonging to the same *IP flow* along the same internal path. In both cases weights are simulated in the balancing table by repeating each entry proportionally to its weight (e.g. MAC destination address to forward the packet to and output port).
The load balancer is not only a packet forwarder, but it implements also filtering, classification and modification functions. Furthermore, the LB acts as an interface among the internal multi-stage router elements, the *internal network*, and the external devices, the *external network*. This implies that packets received from the two different types of networks must be managed using different forwarding rules: for instance, data packets must be balanced only if coming from the external network. Some special packets (e.g. ARP, routing protocols) may not be balanced, but managed in a peculiar way (more details are given in Sec. 4). Finally, physical ports of the LB may be connected to external or internal networks: thus, the LB must be able to detect autonomously the physical configuration (by sensing DIST control plane packets) to correctly apply forwarding rules. Since the NetFPGA platform hosts four ports, the allowed port configurations are *1ext-3int*, *2ext-2int* and *3ext-1int*.

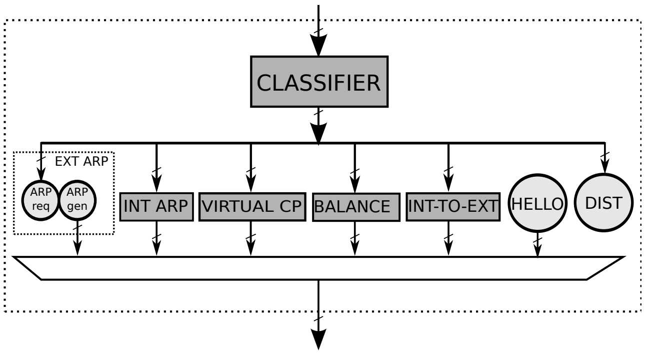

Fig. 3. Packet processor module architecture. Two types of elements are used: filters (boxes) and receivers/transmitters (circles). The EXT ARP module is composed of two elements, because a new ARP reply packet is generated when an ARP request is received from the external network.

The logical structure of the LB is modular and based on an *input stage*, composed of input queues and an input arbiter, a *packet processor* and an *output stage*, composed of an output arbiter and output queues, as reported in Fig. 2. The role of the input and of the output arbiter is to guarantee fairness among ports and to support dynamic assignment of ports to internal or external networks, because they decouple the central packet processing module from input and output queues allowing the maximum flexibility. The central packet processor implements all forwarding rules, as explained in details in Sec. 4.
Internal components are connected using buses (64-bit wide, at 125 MHz for a peak rate of 8 Gbit/s) able to sustain the maximum load of all four ports (which sums up to 4 Gbit/s in both directions). For this reason we do not use a complex switching fabric (e.g. crossbar, Clos network, etc.) among input and output queues. Furthermore, the shared bus is simpler to implement and occupies less resources in the FPGA, leaving room for the implementation of more forwarding rules.

4. PACKET PROCESSOR STAGE

Internal congestion is handled using a back pressure mechanism among all the internal stages (packets are pushed forward only if the receiving module is ready) to avoid packet losses in this stage.
From a performance point of view the packet processor is the most critical element because it receives the aggregated traffic from all the input ports. In the worst case, with minimum size packets (64 bytes, 8 words, 1 clock cycle per word), it receives a packet every 16 cycles. However, we experimentally verified that the current implementation is able to deal with this worst-case data rate. Thus, the packet processor stage is not a bottleneck even in the worst case, showing that NetFPGA internal resources are well designed to support wire-rate loads.
The packet processor stage consists of a classifier module and several processing modules, as shown in Fig. 3. A shared bus connects the classifier with the processing modules because the available bandwidth is enough to support the aggregated traffic from all four input ports, as previously explained. The output of the processing modules are queued and then multiplexed toward the output arbiter.

Table 1. Rules for packet classification

Traffic Type	**Input Port**	**Processed by**	**Output Port**
ARP request	ext	Ext ARP	same port
ARP response	ext	Int ARP	default int
Routing protocols	ext	VirtualCP	default int
Management protocols	ext	VirtualCP	default int
Data	ext	Balance	int
DIST messages	int	DIST	none
ARP request	int	Int ARP	VLAN id
Non DIST messages	int	Int-to-ext	VLAN id

4.1 Classifier module

The *Classifier* module receives packets from any LB interface. All incoming packets are analyzed and sent to the corresponding processing module according to classification rules.
The classification rules have specific priorities: thus, if two or more classification rules match the same packet, the rule with the highest priority only is applied, and the packet is forwarded to the corresponding module only. The highest priority is assigned to the more specific rule. For instance, an incoming OSPF packet is both considered as a data packet to be balanced (being based on the IP protocol) and a control message to be forwarded to the Virtual-CP only; the second rule, being more specific, has a higher priority. If no rule is matched, the packet is dropped.
The classifier distinguishes between data (IP packets), ARP, management (e.g. SNMP) and control (e.g. RIP, OSPF, BGP) traffic. Tab. 4.1 summarizes all classification rules and the corresponding destination module: messages received from external networks and directed to the virtual CP only are forwarded to a specific internal port, named *default int*. To speed up the classification process, every incoming packet is analyzed in parallel by different FSMs (Finite State Machines), each implementing a *different* classification rule.

4.2 External ARP Module

The *External ARP* module (Ext ARP) receives ARP request messages from external networks and it replies autonomously to any request involving its external interfaces (IP configuration is done by the Virtual CP using DIST messages). Using this solution we are able to maintain as simple as possible the implementation and to solve two potential issues:

—*reply storm*: if ARP requests are sent to the internal network in broadcast, all back-end routers would reply to the external devices generating a large number of replies to be filtered eventually by the LB.
—*reverse path selection*: a LB with multiple external interfaces would be unable to forward the reply to the correct external interface unless analyzing the ARP message and comparing it to the IP configuration of external interfaces. This operation may be complex to implement, so our solution simplifies the implementation.

4.3 Internal ARP Module

The *Internal ARP* (Int ARP) module is used to manage ARP requests generated by back-end routers to external networks and the corresponding ARP replies coming back from external networks. ARP requests sent by back-end routers are updated with the source MAC address of the LB external port and forwarded. Then, any ARP reply received from external ports is forwarded to the default internal port and it is sent in broadcast to all back-end routers. This process is needed because the incoming ARP replies are sent in unicast to the MAC address of the external interface, thus the packet does not contain any information on which back-end router issued the ARP request. This solution implies a little overhead in control traffic, but it avoids a more complex stateful design. Broadcast ARP replies are acceptable, according to RFC 826 [22], even if they are not common in practice.

4.4 Virtual-CP module

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 external devices. This Virtual-CP module redirects control messages to the centralized control module (Virtual-CP) which executes the configuration tasks. Therefore control packets belonging to routing protocols (e.g. OSPF, RIP and BGP) and management packets (e.g. SNMP) are sent directly to the Virtual-CP. These packets are identified by simple filters on well-known TCP/UDP ports and on the management IP address representing the router itself. The current implementation provides filters for SNMP, BGP, OSPF and RIP. The virtual-CP module simply overwrites the packet destination MAC address with the MAC address of the back-end router where the Virtual-CP is currently hosted.

4.5 Int-to-ext module

This module handles the communication from the back-end routers to the external network. Since a LB may have more than one external port, when a packet is received from the internal network, the LB must forward the packet to the proper output port. This problem is solved assigning to each external port a unique VLAN id. Whenever a back-end router wants to send a packet to an external host, it adds to the packet a VLAN tag corresponding to the correct external port. This implies creating on each back-end router a virtual interface with the same VLAN tag as the one used for each external port of the multi-stage router and with the same IP configuration of the corresponding LB interface. This process is coordinated by the Virtual-CP. When the LB receives a VLAN tagged packet on an internal port, it removes the VLAN tag and forwards the packet to the proper external port while overwriting the source MAC address.

4.6 DIST and Hello module

The DIST module manages all the control messages of the DIST protocol related to the LB. It must decode: i) port configuration messages, ii) balancing table configuration messages, sent by the virtual CP, and iii) *hello* messages sent by all internal elements. The latter are required by the DIST protocol to advertise periodically the presence of an internal element as a distributed heart beat mechanism. The hello module is responsible for sending periodic *hello* messages using the default internal port (configurable). The *hello* packet is also used to advertise the current configuration and statistics of the LB ports. For each port, the following information is sent:

—Interface parameters: MAC and IP addresses, IP netmask, type (internal or external) and link speed;

—Traffic statistics: transmitted/received byte and packet counters.

4.7 Balance Module

This module performs the layer 2 balancing function on the incoming data traffic from the external ports. A balancing table is used to select the proper back-end router. The packet destination MAC address of the LB interface is replaced by the MAC address of one of the back-end routers. Since a LB may have more than one internal port, the balancing table stores both the back-end router MAC address and the internal port on which to forward the packet. Currently, the load balancing scheme can be *weighted round-robin-* or *hash*-based:

—**Weighted round-robin** is designed to handle back-end routers with different forwarding capacities. Each back-end router can be assigned a weight proportional to its capacity by the Virtual-CP, receiving a proportional amount of load. A pure round-robin balancing is obtained using equal weights for all back-end routers.

—**Hash** assigns packets to back-end routers by evaluating a simple hashing function on a set of defined protocol fields to index the table contents.

Weighted round-robin is the easiest solution to implement but it has three drawbacks: i) the granularity of the weights as well as balancing accuracy is limited because the size of the balancing table is finite; ii) packets from the same flow may be processed by different back-end routers (out-of-order delivery); iii) entries related to the same back end-router may be placed in successive locations inside the table, thus packet bursts may be directed to the same bakc-end router disrupting the balancing mechanism on short time period.
On the other hand, hashing is more complex but it allows to overcome the packet reordering problem, because packets with the same header fields are sent to the same back-end router. However, hashing can not guarantee a balanced traffic distribution among back-end routers, because the function outcome depends on input packet headers distribution leading to potential performance problems too.
The load-balancing table is computed by the Virtual-CP, which has a global view of the multi-stage router, and it is sent to LBs by using the *table update* DIST message, composed of the standard DIST header [8], an additional header (number of entries and *Flush* flag) and a list of table entries. Each entry comprises a back-end router MAC address, a LB output port and an integer weight w which indicates the number of repetitions of the entry in the final table. Upon reception of the *table update* message, the LB reconstructs the final table adding each entry w-times. The table is stored in the FPGA memory, being it faster than the on-board SRAM. Furthermore, this choice avoids contentions for the SRAM, which is already used to store queued packets. The table contains up to 1000 entries. The *flush* flag, if set, instructs the LB to overwrite the existing table instead of updating it.
Table entries are removed based on timeouts. Each time a DIST *hello* message is received all table entries related to the sending device are refreshed (using a binary flag). The LB scans periodically the table to delete out-of-date entries. Using a per-table timeout permits to significantly reduce the timer handling complexity. Furthermore, using local timeouts permits to shorten the detection time of unavailable back-end routers, reducing the possibility of losing packets due to balancing toward a back-end router which is not active any more.

5. PACKET FORWARDING EXAMPLE

To give major insight on how the LB balancer works, we describe the step-by-step operations performed to route a packet between two external hosts A and B residing on different networks (NET_A and NET_B) interconnected by the multistage router. A and B have respectively as next hop EIP_A and EIP_B (the router external interfaces). To be able to work properly, all the external interfaces are mirrored on each back-end router using logical network interfaces.

Table 2. Routing table of back-end routers

Destination	Gateway	Output Interface	VLAN_ID
MGMT	-	eth0	None
NET_A	-	eth0.1	VID_A
NET_B	-	eth0.2	VID_B

The routing table of each back-end router is shown in Table 2. Finally we assume that the ARP tables are initially empty. Figure 4 summarizes the scenario.
A initiates the communication sending an ARP request in broadcast, asking for EIP_A. LB_A checks if EIP_A corresponds to one of its external IP addresses and answers with an ARP reply containing EIP_A and EMAC_A. A then sends the data packet to EMAC_A. LB_A receives the packet and sends it to R1 by changing the destination MAC from EMAC_A to MAC_R1 (the choice of the MAC depends on the load balancing scheduler). R1 receives the packet and computes the next hop, which is B located on NET_B. B is connected to the external interface having VLAN tag VID_B. R1 asks for the MAC address of B by sending an ARP request using VLAN tag VID_B. Based on the VLAN tag, the packet is sent to LB_B and out from interface EVID_B. When the packet is sent out, the LB overwrites the MAC address MAC_R1 with EMAC_B. B receives the ARP request and answers with an ARP reply sent to EMAC_B, subsequently received by LB_B. Unfortunately at this point there is no way looking at the packet to determine who among the back-end routers sent the ARP request. Therefore, the ARP reply is sent in broadcast to the internal network to reach all the back-end routers. Since this frame does not contain any VLAN tag, it is discarded by the other LB preventing this spurious ARP reply to propagate to the external subnets. The data packet is sent out using VLAN tag VID_B and destination address MAC_B through LB_B which overwrites the source MAC address. Finally, B receives the packet with the correct destination MAC address and the source MAC address of the LB. External devices only detect the external interface addresses and no internal host information is exposed.

6. TESTBED DESCRIPTION

Each test is based on synthetic traffic generated using an Agilent N2X router tester equipped with Gigabit Ethernet modules. The LB is inserted into a PCI slot of the host PC, which is needed only to power up the board. Board configuration is done emulating configuration packets directly sent from the router tester itself. We set the timeout to values large enough to avoid timeout expiration during the experiment. Thus, no further configuration messages are sent during measurements. Results are averaged over three runs lasting 30 seconds each. In all experiments the variance between different runs is small, thus no error bars are reported in the plots.
Among the two currently available NetFPGA platforms, we use the Gigabit version, which is a PCI board equipped with four Ethernet ports, 4.5 Mbytes of SRAM and 64 Mbytes of DRAM. The main FPGA is a Xilinx Virtex2-Pro 50 running at 125 MHz and offering about 50K logic elements. This FPGA holds all the user-defined logic, as well as the MAC part of the Gigabit ports (the PHY part resides on a dedicated chip). The DRAM uses both edges of a 200 MHz 32-bit bus to transfer data to/from the FPGA, while the SRAM is connected by a 36-bit bus running synchronously with the main FPGA.
The NetFPGA proved to be well-suited for our project since less than 50% of FPGA logic resources are exploited leaving room for future improvements (e.g. bigger tables, more filtering rules, etc.).

7. PERFORMANCE EVALUATION

We assess the performance achieved by the LB in different configurations. We measure both throughput and latency in both the *ext-to-int* and *int-to-ext* directions (unidirectional and bidirectional) for different frame sizes, ranging from the minimum of 64 bytes to the maximum of 1518 bytes. In the *int-to-ext* path, packet sizes are increased by four bytes, because the VLAN header needed by our internal forwarding scheme is added.

7.1 2-port Load Balancer

Tab. 5 shows the offered load O_L, throughput T and latency L for a LB with one external and one internal port. In the ext-to-int case, the LB is always able to process the offered load. In the int-to-ext case, as well as in the bidirectional case, a small discrepancy (e.g. up to 6% in the worst case) between input and output throughput is visible. This is not due to an overload of the LB, but to the removal of the VLAN tag which is used only internally and accounts for an amount of bandwidth depending on the average packet rate. At the same input load, the smaller the frame size, the larger the packet rate, thus the throughput drop (4 bytes lost per packet). We only report the latency at the maximum input load, being the latency constant with respect to the offered load. This means that internal queues are almost empty and that the LB works at wire-speed. Indeed, the latency is low (few μs) and it only depends on the frame size, due to the store and forward operation.

7.2 3-port Load Balancer

We add a third port to the previous scenario to test the load balancing and VLAN tagging functionalities. We start configuring the third port as internal and send different balancing table configurations to the LB. The relative normalized theoretical throughput towards back-end router i can be computed using:

$$T_i = \frac{w_i}{\sum_{k=1}^{n} w_k}$$

where T_i is the percentage of traffic handled by router i having weight w_i, and n is the total number of entries in the table. The LB is always able to match the theoretical throughputs with no losses in all the considered scenarios.
Next we use the same 3-port scenario to test VLAN-based output port selection needed on the int-to-ext path, as explained in Sec. 4.5. In this case, the LB is configured with two external ports and one internal port. The router tester generates two flows with different VLAN tags and sends them to the internal port. Our tests show that the LB is able to untag the packets and to forward them to the correct external interface at wire speed.

7.3 4-port Load Balancer

When using four ports and unidirectional traffic, the LB is still able to process the full wire speed. However with bidirectional traffic (i.e. 2 Gbit/s from ext-to-int and 2 Gbit/s from int-to-ext) the LB is overloaded, as shown in Fig. 5 and 6. Notice that in the ext-to-int case we are not able to match the input load, while in the int-to-ext case with small packets the input load is matched. In this case, the LB takes advantage of the small reduction in traffic due to the 4-byte VLAN tag removal. The bottleneck seems to reside in the access speed (4 Gbit/s) to the on-board SRAM, which is used to implement the queues to store packets to solve contentions. Indeed, if we remove the packet processor (Fig. 2) and we connect directly

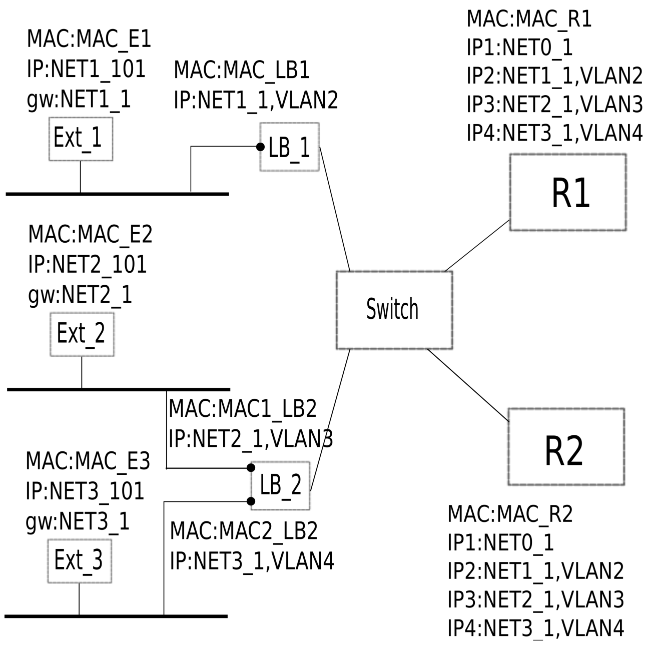

Fig. 4. Map of elements (and their addresses) involved in the packet forwarding example)

Table 3. 2-port LB performance

Frame Size [Bytes]	Unidirectional						Bidirectional		
	Ext-to-Int			Int-to-Ext			Aggregate		
	O_L [Mbit/s]	T [Mbit/s]	L [μs]	O_L [Mbit/s]	T [Mbit/s]	L [μs]	O_L [Mbit/s]	T [Mbit/s]	L [μs]
64 (68)	753	753	3.46	764	719	3.42	1517	1472	3.5
128 (132)	859	859	4.11	862	836	4.10	1721	1695	4.26
256 (260)	924	924	5.85	925	911	5.46	1849	1835	6.37
512 (516)	960	960	7.84	960	953	8.24	1920	1913	8.38
1518 (1522)	986	986	17.86	986	984	18.50	1972	1970	23.19

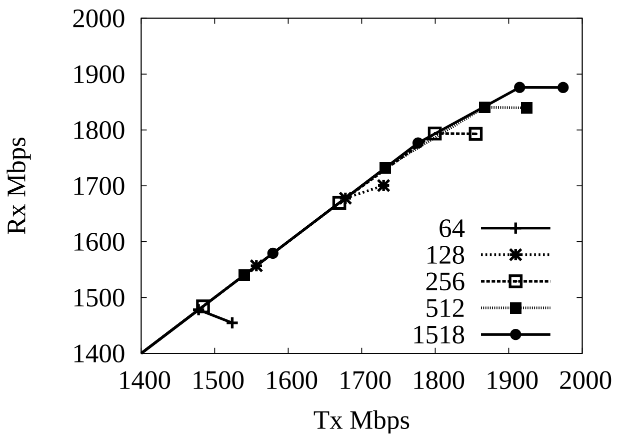

Fig. 5. Aggregate throughput for two ext-to-int flows. 4 port configuration scenario and bidirectional flows.

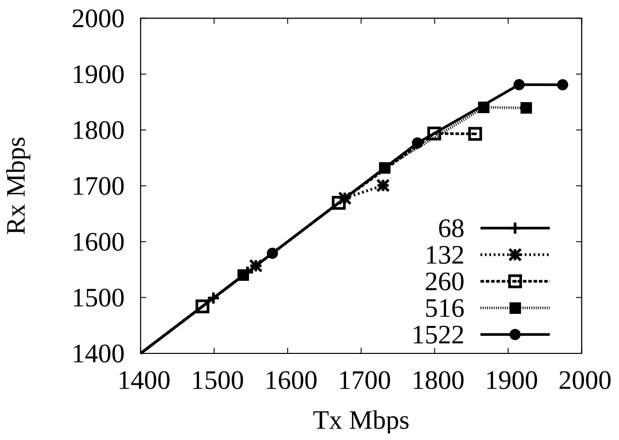

Fig. 6. Aggregate throughput for two int-to-ext flows. 4 port configuration scenario and bidirectional flows.

the queues, we still observe the same performance limit. This is the only limitation we detected in the NetFPGA board.

8. CONCLUSIONS

We described a FPGA-based implementation of load balancing functionality, in the framework of a previously developed multi-stage router architecture, with the goal of overcoming the performance bottleneck experienced on a software-only solution based on Click Modular Router. The performance goal was achieved, since the full-feature NetFPGA implementation reaches near to wire-speed performance. The NetFPGA proved to be generally well-suited for our project (e.g. less than 50% of FPGA logic resources are exploited), but we identified as an hardware bottleneck the access bandwidth to external SRAM, which should be improved to fully support processing and forwarding needs at wire speed.

9. REFERENCES

[1] Foundry serveriron load balancer.

[2] Hardware Load Balancer for Multi-Stage Software Router. `http://opencores.org/project,loadbalancer`.

[3] Microsofts network load balancing. `http://www.foundrynet.com/products/webswitches/serveriron/`.

[4] NetFPGA. `http://www.netfpga.org/`.

[5] Katerina Argyraki, Salman Baset, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Eddie Kohler, Maziar Manesh, Sergiu Nedevschi, and Sylvia Ratnasamy. Can software routers scale? In *PRESTO*, Seattle, USA, Aug. 2008.

[6] Shadi Atalla, Andrea Bianco, Robert Birke, and Luca Giraudo. NetFPGA-based load balancer for a Multi-Stage router architecture. In *International Conference on Computer Information Systems 2014 (ICCIS-2014)*, Hammamet, Tunisia, January 2014.

[7] Shadi Atalla, Davide Cuda, Paolo Giaccone, and Marco Pretti. Belief-propagation-assisted scheduling in input-queued switches. *IEEE Transactions on Computers*, 62(10):2101–2107, 2013.

[8] A. Bianco, R. Birke, J. M. Finochietto, L. Giraudo, F. Marenco, M. Mellia, A. Khan, and D. Manjunath. Control and management plane in a multi-stage software router architecture. In *HPSR*, Shanghai, China, May 2008.

[9] A. Bianco, J.M. Finochietto, G. Galante, M. Mellia, D. Mazzucchi, and F. Neri. Scalable layer-2/layer-3 multistage switching architectures for software routers. In *IEEE GLOBECOM*, San Francisco, USA, Dec. 2006.

[10] A. Bianco, J.M. Finochietto, M. Mellia, F. Neri, and G. Galante. Multistage switching architectures for software routers. *IEEE Network*, 21(4):15–21, Jul.-Aug. 2007.

[11] Andrea Bianco, Robert Birke, Fikru Getachew Debele, and Luca Giraudo. Snmp management in a distributed software router architecture. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–5. IEEE, 2011.

[12] Andrea Bianco, Robert Birke, Luca Giraudo, and Nanfang Li. Multistage software routers in a virtual environment. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5. IEEE, 2010.

[13] Raffaele Bolla and Roberto Bruschi. RFC 2544 performance evaluation and internal measurements for a Linux-based open router. In *HPSR*, Poznan, Poland, Jun. 2006.

[14] Raffaele Bolla and Roberto Bruschi. An effective forwarding architecture for SMP Linux routers. In *IT-NEWS*, Venice, Italy, Feb. 2008.

[15] Raffaele Bolla and Roberto Bruschi. PC-based software routers: High performance and application service support. In *PRESTO*, Seattle, USA, Aug. 2008.

[16] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. RouteBricks: exploiting parallelism to scale software routers. In *SOSP*, Big Sky, USA, Oct. 2009.

[17] Nikhil Handigol, Srinivasan Seetharaman, Nick McKeown, and Ramesh Johari. Plug-n-serve: Load-balancing web traffic using openflow, 2009.

[18] IETF. Forwarding and control element separation working group (ForCES). `http://tools.ietf.org/wg/forces/`.

[19] A.J. Khan, R. Birke, D. Manjunath, A. Sahoo, and A. Bianco. Distributed PC based routers: bottleneck analysis and architecture proposal. In *HPSR*, Shanghai, China, May 2008.

[20] Marc Körner, Herbert Almus, Hagen Woesner, and Tobias Jungel. Metrics and measurement tools in openflow and the ofelia testbed. In *Measurement Methodology and Tools*, pages 127–138. Springer, 2013.

[21] Ian Kuon, Russell Tessier, and Jonathan Rose. Fpga architecture: Survey and challenges. *Foundations and Trends® in Electronic Design Automation*, 2(2):135–253, 2008.

[22] David C. Plummer. RFC 826, ethernet address resolution protocol, Nov. 1982. `http://www.ietf.org/rfc/rfc0826.txt`.

[23] Q Xu, H Rastegarfar, Y Ben M'Sallem, A Leon-Garcia, S LaRochelle, and LA Rusch. Analysis of large-scale multistage all-optical packet switching routers. *Optical Communications and Networking, IEEE/OSA Journal of*, 4(5):412–425, 2012.