A FPGA-based scalable architecture for URL legal filtering in 100GbE networks

Jaime J. Garnica, Sergio Lopez-Buedo, Victor Lopez, Javier Aracil
High Performance Computing and Networking Group
Universidad Autonoma de Madrid
28049 Madrid, Spain
Email: jaime.garnica@uam.es

Jose Maria Gomez Hidalgo
Research and Development Department
Optenet
28230 Las Rozas, Madrid, Spain
jgomez@optenet.com

***Abstract*—Legal filtering is common practice in many countries to avoid access to websites with criminal or violent content. This kind of filtering is typically implemented at the edge routers of ISP's core networks, so it is mandatory to support very high bit rates. This paper proposes a hardware-software solution based on FPGAs, which scales up to 100 Gbps Ethernet. A FPGA-based PCIe board equipped with two network interfaces is used to intercept ISP traffic. The FPGA performs an initial filtering of the packets whose destination is potentially forbidden, based on a hash of the destination IP address. Filtered packets are sent to the software application, which inspects them and decides if the URL is actually forbidden or not. This two-level filtering allows for the scalability of the proposed solution to very high bit rates, not only because it simplifies FPGA design, but also because it significantly reduces software load, since potentially forbidden destinations are few. Additionally, this solution adds a minimal latency to most of the packets, and also allows for updating filtering rules without interrupting ISP traffic. The paper presents a proof-of-concept 10GbE implementation of the proposed architecture, as well as an analysis of its scalability up to 100GbE.**

I. Introduction

In most countries, ISP's (Internet Service Providers) are requested to provide legal URL filtering so that certain webpages are not available to Internet users. Even in those countries with strong civil liberties, URL filtering is used to prevent access to webpages with criminal content: Child abuse, terrorism apology, fraudulent activities, etc. When such criminal content is detected, a legal process is started which ends by a judge ruling that a given page is against the law and should not be available to Internet users in the country. There are also non-governmental organizations in charge of monitoring the Internet and detecting pages with illegal contents. A good example of such organizations is the Internet Watch Foundation (IWF), a UK foundation in charge of detecting child abuse content in the Internet. Lists provided by the IWF are used in many countries to avoid users from accidentally stumbling into such criminal images.

Legal URL filtering is usually implemented by ISP's at edge routers, where their backbone networks connects to the Internet exchange points. Such location simplifies the network architecture, since legal filtering is centralized in one point. However, line rates at the backbone are very high, in the order of Gbps. Packet filtering at such high rates is not an easy task [1]. As it will be explained in Section II, software-only solutions using commodity hardware are not capable of achieving multi-gigabit per second rates. It is therefore mandatory to explore other solutions, such as network processors or FPGAs. In any case, the solution must have low-cost and reconfigurability characteristics. The latter is especially important, since illegal content is continually being discovered. Another relevant feature to be taken into account when selecting the target technology is latency. In an ideal scenario, the filter should be transparent to non-filtered traffic, adding a negligible latency.

Fortunately, the number of illegal URLs is very low, in the order of thousands. Therefore, the fraction of traffic being filtered will be negligible for a typical ISP. Taking advantage of this fact, what we propose in this paper is a two-level filtering. The first level of filtering is based on the destination IP address. While this filtering is very easy to implement, it does not completely solve the problem. For example, imagine a shared-server web hosting service, where the same server stores many perfectly legal pages as well as one with criminal content. The destination IP will be the same for all pages, legal an illegal, but only the illegal one should be filtered. Therefore, URL filtering requires deep packet inspection to check what is the requested URL in the GET HTTP command. However, we can consider this deep packet inspection as a second level of filtering. In the first level, based on the destination IP address, we will have several false positives and no false negatives, since we have a complete list of potentially forbidden IP addresses. All packets whose destination IP address is potentially forbidden, should go through a deep packet inspection procedure in order to extract the requested URL. But the number of packets going through this deep packet inspection is very small, since the number of illegal URLs is also very low. What we propose here is to implement on a FPGA the first level of filtering, based on destination IP address, and leave for the software the more complex second level, which implies deep packet inspection. This approach has two benefits: First, deep packet inspection is avoided at the FPGA level, so the design is kept simple, favoring high rates and the use of low-cost devices. Second, the amount of traffic being processed by software is significantly reduced, so it can be handled by a low-cost commodity microprocessor. Furthermore, a FPGA-based solution has minimal latency,

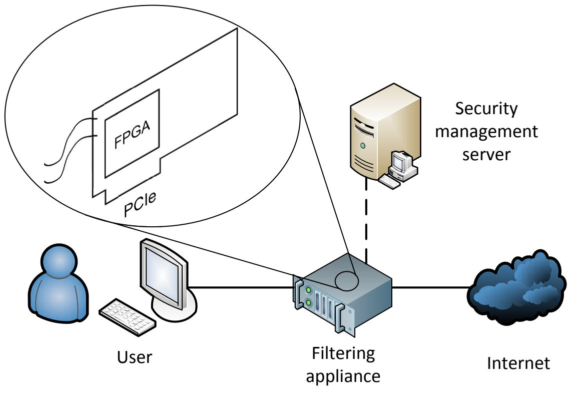

Fig. 1. Solution overview

which cannot be attained with other technologies such as network processors.

Fig. 1 shows a diagram of the proposed solution. It is based on a FPGA board with two network interfaces, which intercepts all traffic going from the ISP to the Internet. The board has a memory where the list of potentially forbidden IP addresses is stored. Actually, what is stored in the memory is hash of the IP addresses, as it will justified in Section II. Packets with a potentially forbidden destination are sent to the host via the PCIe interface of the FPGA board. The host inspects the destination URL of these packets in order to decide if they have to be filtered or not. The host connects to the security management application, which is running in another server. This security management application is in charge of updating the list of forbidden URLs. Section III further details the architecture of the proposed solution. A 10 Gbps implementation is presented in Section IV, and Section V presents its results as well as an analysis of its scalability to 100 Gbps.

II. Design decisions

A. URL filtering methods

URL filtering methods are traditionally done by software-only solutions based on commodity hardware [2], [3], [4]. The main advantages of these solutions are their flexibility and ability to implement complex URL lookup methods. Such methods are not only used by URL filtering applications, but also by search engines [2], web caching [3] or content distribution networks [4]. However, the response time of software methods is in the order of milliseconds [5]. To improve the performance of URL filtering methods, there are four approaches: (1) improve the URL lookup algorithm [5], (2) implement parallel algorithms to take advantage of multi-core architectures [6], [7], (3) split the incoming traffic among multiple machines and (4) reduce CPU consumption in the packet capturing process with dedicated hardware [8].

The latter option, dedicated hardware, is gaining momentum since it is very difficult to scale solutions based on commodity hardware to multi-gigabit rates. Such dedicated hardware solutions can be for example based on Network Processors, which are many-core architectures specially designed for networking applications (e.g. Cavium Octeon), or they can also be based on FPGAs. The main advantages of FPGAs are its ability to work at very high rates, reaching 100 Gbps, and a total control over the latency, since designers exactly know the number of cycles that it takes to process a packet.

B. FPGA-based solutions

Different FPGA-based solutions for packet classification at very high rates have already been described. Usually, these solutions are standalone designs working on a known set of rules. This implies that to update the rule list it is necessary to reconfigure the FPGA, thus interrupting network traffic. Designs are usually based on Bloom Filters an Binary Tries. Bloom Filters [9] use hash functions extracted from the set of rules, which allow a for a very efficient $O(1)$ lookup, but with false positives. Binary Tries [1], [10] create binary search trees from a set of rules using different types of memories. Memories such as TCAMs have been used in several approaches, but these memories are costly [11].

Our solution follows a different approach. Instead of creating a custom FPGA design based on a given set of rules, the design is generic: Filtering is based on a memory, where a bit is used per IP address to indicate whether it is “suspicious” or not. Therefore, changes in the ruleset are made by just updating this memory, so there is no need for FPGA reconfiguration and thus no traffic interruptions. As it was stated in the introduction, IP address alone is not enough to decide wether the packet should be filtered or not. This is another distinguishing feature of our approach. We consider a hardware-software solution, where a first level of filtering is done in the FPGA, and the more complex deep packet inspection is done in software. This is opposed to other solutions where all processing is done at the FPGA.

C. Rule storage and lookup

According to our approach, the system requires 4 Gb (2^{32} bits) to store all the possible IPv4 addresses. Main types of memories in FPGA boards are: BlockRAM, DDR, and RLDRAM/QDR-II memories. Next, an analysis on the suitability of each type of memories is presented.

1) Memories: First, BlockRAM capacity is just in the order of MB, so it is not enough to store all IPv4 addresses. Second, the storage capacity of DDR memories is high enough to store all addresses, but they do not allow pipelining for nonconsecutive accesses [12]. Each time the DDR memory is accessed for a non-preloaded data, the whole page that contains the data is loaded, which is a slow process. If the search address order is known, delay is acceptable [10], but unfortunately the incoming order of destination IP addresses is not predictable. Finally, QDR-II and RLDRAM memories provide a pipelined access. QDR-II timing is somewhat better, with a 2-cycle read latency [13], while RLDRAM capacity is typically higher. The size of these memories is continuously increasing, reaching tens to hundreds of MB in modern boards. However it is not enough to store all IPv4 addresses.

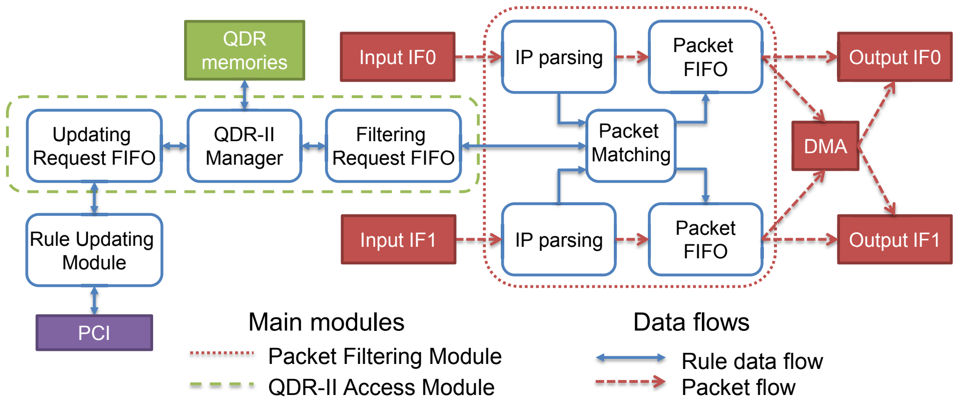

Fig. 2. Architecture design and data flows

2) Hash function: Since there are currently no memories featuring both enough capacity and pipelined access, it is necessary to find a different approach. The approach proposed in this paper is to apply a hash function to the destination IP address in order to reduce the number of bits and therefore the storage requirements. Although this approach is very efficient in terms of performance ($O(1)$ complexity), it suffers from false positives due to contention in the hash function (several IP addresses having the same hash). However, as it was stated in the introduction, our approach already suffers from false positives: It is the software who should decide if the connection should be forbidden or not by inspecting the requested URL. Therefore, the use of hashes simply increases the false positive rate, which is still very low, but allows for using pipelined memories such as BRAM or QDR-II. Actually, we found that the best tradeoff between speed and capacity (false positives) occurred for QDR-II memories, since BRAMs are too small and therefore the hash contention rate would be unacceptable.

III. FPGA DESIGN

The software application has two main tasks: (1) update the IP address list in the FPGA and (2) decide the action is applied to each "suspicious" packet. The software application keeps a list of the forbidden URLs, whose corresponding IP addresses are known. These IP addresses are sent to the FPGA via the PCIe. Let us remark that the updating process is online, so there are no traffic cuts. Regarding the hardware design, a FPGA-based board with two network interfaces is used, so traffic in each direction of the network will be filtered. From the operating system point of view, the board works like a conventional Network Interface Card. The FPGA is in charge of filtering just the "suspicious" packets to the software application. The software application receives these "suspicious" packets and it decides whether or not the packets are finally blocked. This combined architecture yields to software workload reduced, thanks to the filtering performed by the FPGA card that only forwards a small fraction of packets, which is compatible with the processing capacity of the software.

The architecture of the design is divided into different modules, as it can be seen in the Fig. 2. First, the main one is the Packet Filtering Module, which inspects the fields of every incoming packet and requests the rule list. After memory modules answer the request, a forwarding action will be applied to the packet. The forwarding of "suspicious" packets to the software application is done via DMA. Second, the Rule Updater Module carries out the rule updating requests that are received from the software application. These updates are sent via PCIe. The Rule Updating Module writes the updates in the FPGA memories and reads the information that the software application needs from the rule list status. Finally, the QDR-II Access Module manages the access to the memories in order to maintain the traffic processing when an update request arrives. By keeping pending requests in FIFO modules, it allows to access to both the filtering and updating processes without traffic cuts. The following subsections describe in detail each part of the architecture as well as other important issues such us rule storing and memory access.

A. Packet Filtering Module

Filtering is the essential task of the design and it is carried out by this module. The Packet Filtering Module contains the Packet Matching Module, which receives the packet IP addresses when incoming packets are parsed into the IP Parsing Module. These IP addresses are hashed and then sent as a query address to the QDR-II Manager Module. After receiving an answer from the QDR-II, an output must be selected depending on the matching result. The Packet Filtering Module implements a pipeline to avoid packet dropping caused by the memory access delay. After IP parsing inside the corresponding module, Packet FIFO stores the packets that are being checked. When the Packet Matching Module obtains the matching result, it decides what is the correct output for the packet (network or DMA).

Let us remark that the filtering process is done in both network directions using two interfaces, which simultaneously access the same rule tables. Therefore, the system must support not only 149 Mpps for 100 Gbps Ethernet, but twice. For this reason, the Packet Matching Module alternates and controls

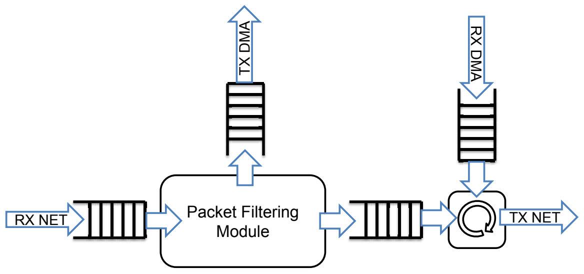

Fig. 3. Packet flow contention

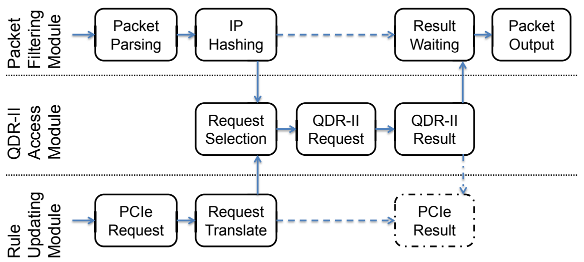

Fig. 4. Pipeline schema

the searches for each one of the interfaces, while keeping consistency in the pipeline. There are several solutions for this contention problem, the simplest one is to create time slots between the two network interfaces. However, other solutions may be valid such as a priority request control. Since the maximum clock frequency in QDR-II memories is very high, up to 333 MHz, the same number of time slots can be reserved for each interface. Equal time slots implies no more than 298 MHz or requests per second.

System delays must not stop the IP packet stream. To avoid it, a packet FIFO system is implemented for frame contention during the filtering process (Packet FIFO modules). As can be seen in Fig. 3, the Packet FIFO system is used in the Packet Filtering Module for each network interface, i.e. for each direction. Packet drop occurs when the delay added by the FPGA is higher than the packet reception interval. There are three critical points in terms of delay. Packet filtering modules present the highest delay, because they have to wait to the memory response. Secondly, the DMA transmission rate is lower than the network interface input rate. Thus, packets must be stored temporally not to interrupt the pipeline. Finally, the process of aggregating received packets from the DMA and forwarded packets from the Packet Filtering Module also adds a delay. To summarize, there are four packet FIFOs for each network interface in the contention system. Let us remark that there is a fifth packet FIFO inside the Packet Filtering Module for each network interface (Fig. 2).

B. Rule Updating Module

The Rule Updating Module is in charge of updating the rule list stored in the FPGA. It receives requests from the application software via PCIe. Not only updating requests may be received but also read requests of the current memory status in order to maintain coherency between the software and the hardware rule tables. As it happened with filtering requests, update is managed by the QDR-II Access Module.

C. QDR-II Access Module

This module allows dynamically updating the list of rules without interrupting the filtering process. QDR-II memories are the only communication between the Rule Updating Module and the Packet Filtering Module. To allow a shared access, the QDR-II Manager Module decides when each module can access the memories. Request FIFO modules store the requests to allow the pipeline to work correctly. For each requester module a Request FIFO system is integrated between the requester and the QDR-II Manager Module.

The QDR-II memory access protocol is the Xilinx' Memory Interface Generator (MIG) protocol [14]. The MIG protocol requires a full FIFO control and independent signals for reading and writing. To keep the pipeline, a couple of Request FIFO modules that store addresses and data for QDR-II requests and responses are used. Requests received by these FIFO modules use a proprietary Local Bus protocol because of simplicity. Then, a translation is done by the Request FIFO modules to the MIG protocol. Memory access clock is usually higher than the system clock in order to achieve a higher access rate. These clock domain exchange is also done by the Request FIFO modules.

D. Pipelined module integration

QDR-II requests present a delay of tenths of nanoseconds due to clock domain exchange and module communication. Then, a pipeline design is required to achieve high rates with no packet lost. Packet filtering and rule table updating must have their own pipeline. Then, the QDR-II Access Module must have another one to support them both. Fig. 4 shows the pipeline design for the system. First, packet filtering pipeline stages are: (1) to parse IP addresses from incoming packets, (2) to hash the IP address and send the request to the Request FIFO module, (3) to wait for the memory to response and (4) to forward the frame to the network or the DMA. Second, rule updating pipeline is organized in the following steps: (1) to read the request from the PCIe, (2) to request the Request FIFO module and (3, reading requests only) to wait the response from the QDR-II memory. This third step is only carried out when the rule updating system checks the rule list status. Let us remark that the QDR-II Access Module pipeline is shared by previous processes. QDR-II Access Module pipeline is divided into: (1) selecting the proper requester input, (2) sending the request to the memory and (3, read only) returning the returned data to the respective requester.

IV. 10 Gbps implementation

Nowadays, FPGA-based PCIe development boards provide interfaces up to 10 Gbps. Therefore, the proof of concept has been done at this rate. The reference system is the NetCope Server from INVEA-Tech [15]. It is based on 64-bit CentOS 5.3 distribution of GNU/Linux. The FPGA board is a combo of two cards connected to the server via PCIe. The first card is equipped with a Xilinx FPGA XC5VLX155T Virtex-5. QDR-II memory is divided into two CY7C1513AV18 devices, with a size of 72 Mb each. The second card is equipped with two XFP cages for the 10 Gbps network interfaces. NetCope Server development provides a firmware that implements the network interface management, PCIe access to INVEA's proprietary Local Bus and memory controllers (MIG). Our architecture is placed on the customer application core.

In the one hand, the clock frequency at which the design works is 125 MHz. The frames arrives at 128 bits per cycle. The IP addresses are obtained at the second cycle. On the other hand, the QDR-II memory works at 250 MHz. So it it is necessary to change the clock domain of the requests by means of the Request FIFO modules. Clock frequencies allow simultaneous access from the updating and filtering processes so that a memory access overflow is avoided. Since, the QDR-II memories have a storage capacity of 72 Mbits (2^{26} bits), a 26-bit addressing has been used. So that the hash used in this project is a 32-bit CRC from which the least significant 26 bits have been used. Therefore, there are up to $2^{32-26} = 2^6 = 64$ collisions per address because of the 6 bits of the hash that are ignored.

The software that runs on the server communicates with the card by means a driver that uses ioctls. Since the QDR-II capacity is less than the 4 Gb required to store the whole IPv4 addressing space, the driver maintains the whole blacklist in software. Maintaining this whole list is necessary because it might happen that two forbidden IP addresses share the same hash. In that case, if one IP address is removed the corresponding bit should continue to one in the QDR-II memory, because there is still the other IP address forbidden. In order to maintain this coherency, the whole IP address blacklist must be stored in the driver.

V. Results

100 Gbps requirements are calculated due to validate the architecture at high line rates. To do that, both clock frequencies and resource consumption are analyzed from the 10 Gbps implementation. Then, they are extrapolated and validated at 100 Gbps for a state-of-art FGPA-based board.

A. 10 Gbps implementation

1) Clock frequencies: Since a request is done for every single packet, to reach a high memory access rate is critical in order to maintain the system performance. The maximum packet rate in a 10 Gbps implementation is 14.8Mpps for minimum-size packets [16]. It means that the clock frequency for the memory access must be at least 14.8 MHz times the number of interfaces (two in our case), that is, 29.6 MHz. Rule updating rate is much lower than packet filtering requests and it can be ignored. Then, memory modules working at 250 MHz and packet modules at 125 MHz is more than enough to achieve the 10 Gbps.

TABLE I
Resource usage for 128-bit packet data width

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	25,803	97,280	26%
Number of Slice LUTs	31,330	97,280	32%
Number of bonded IOBs	487	640	76%
Number of BlockRAM/FIFO	103	212	48%
Number using BlockRAM only	53		
Number using FIFO only	50		
Total Memory used (KB)	3,438	7,632	45%

TABLE II
Resource usage for 256-bit packet data width

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	25,817	97,280	26%
Number of Slice LUTs	31,771	97,280	32%
Number of bonded IOBs	487	640	76%
Number of BlockRAM/FIFO	125	212	58%
Number using BlockRAM only	55		
Number using FIFO only	70		
Total Memory used (KB)	4,158	7,632	54%

2) Resource consumption: Since Request FIFOs are 32-bit width and just a request is stored by packet, we focus on the Packet FIFO system depicted in Fig. 3. Packet FIFOs are 128-bit width, the same as network interfaces are. As data may be kept every clock cycle, the minimum FIFO depth is equal to number of clock cycles for the QDR-II access delay. Since the access delay seen is less than 15 clock cycles, 512-word depth is more than enough. The Design Summary provides us the resource utilization as can be seen in Table I. The map report shows the BlockRAMs used by the design and FIFO cores are implemented by using BlockRAMs. In the proof of concept, 48% of the available BlockRAMs are used. Packet FIFO system used in the 10GbE implementation uses 10 packet FIFOs (5 for each interface). Each packet FIFO is $128b \times 512 = 64$ Kb. Then, total size used is $64 \times 10 = 640$ Kb of BlockRAM plus a minimal resources utilization by Request FIFOs.

B. 100 Gbps implementation

1) Clock frequencies: For a 100 Gbps network, the maximum packet rate is 149 Mpps [17]. Then, the clock frequency for the memory access must be higher than 298 MHz. Fortunately, current QDR-II memories are able to achieve frequencies up to 333 MHz. Data width provided by current network interfaces cores is up to 256 bits per cycle. Then, for the worst case of 60-byte packets, a new packet arrives every two clock cycles. To achieve 149 Mpps per interface, the system clock frequency must be also greater than 298 MHz. Virtex-7 chip family achieve higher clock frequencies [18]. In light of these results, we can conclude that in terms of clock frequencies, a 100 Gbps network could be filtered in both directions.

2) Resource consumption: FIFO data width increases to 256 bits, so the Packet FIFO system requires the double size than the 10 Gbps one ($1,280$ Kb). Table II shows the resource utilization to support 256-bit data width. This architecture is implemented for the same Virtex-5 FPGA chip. In this case, the total memory used increase to 4,158 Kb, just the 54% of the available, a lower percentage for better models such as Virtex-7. Let us remark that a 100GbE implementation increases the consumption of the FPGA resources. However, BlockRAM memories used for Packet FIFOs are the most critical limitation. Thanks to these results, we validate that the proposed architecture supports the pipeline design in 100GbE networks.

C. IPv6

For IPv6 networks, it is only necessary to change the hash function to a 128-bit input. Since a bit is used for a IP address, 2^{128} bits is the size of the entire set. It means a hash with $2^{128-30} = 2^{98}$ collisions in modern QDR-III memories with 512 Mb of storage capacity in a FPGA, what is unacceptable. Although, since the set of forbidden IP addresses is quite small, as well as the currently used IPv6 address set is, an optimized hash function could be used in order to mitigate this problem and allow for the migration of the proposed architecture to IPv6.

VI. Conclusions

In this paper, a scalable URL legal filtering solution for 100GbE networks is proposed. Our approach is a combined architecture with the advantages of software and hardware solutions. Server workload is reduced in this combined architecture, based on the fact user's requests are not usually directed to forbidden websites. The initial FPGA design shows the modules included in the architecture and the pipeline process. QDR-II memories and hash functions allow the architecture to work with the whole set of IP addresses at high line rates. In addition, the rule updating process is done online to avoid traffic cuts. Based on this design, we have implemented a proof of concept at 10GbE. Results show the feasibility of the hardware implementation to work in 100GbE networks with state-of-art FPGAs when boards supporting interfaces at that rate will become available.

Acknowledgment

This research was carried out with support of project DEPACO (TSI-020100-2009-28) funded by the the Spanish Ministry of Industry, Tourism and Commerce.

References

[1] Y. Qi, J. Fong, W. Jiang, B. Xu, J. Li, and V. K. Prasanna, "Multi-dimensional Packet Classification on FPGA: 100 Gbps and Beyond," in *Proceedings of International Conference on on Field-Programmable Technology*. ACM, December 2010.

[2] Q. Gan and T. Suel, "Improved techniques for result caching in web search engines," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 431–440.

[3] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking (TON)*, vol. 8, no. 3, pp. 281–293, 2000.

[4] Z. Prodanoff and K. Christensen, "Managing routing tables for URL routers in content distribution networks," *International Journal of Network Management*, vol. 14, no. 3, pp. 177–192, 2004.

[5] Z. Zhou, T. Song, and Y. Jia, "A High-Performance URL Lookup Engine for URL Filtering Systems," in *Communications (ICC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–5.

[6] M. Dashtbozorgi and M. Abdollahi Azgomi, "A high-performance and scalable multi-core aware software solution for network monitoring," *The Journal of Supercomputing*, pp. 1–24, 2010.

[7] H. Yuan, B. Wun, and P. Crowley, "Software-based implementations of updateable data structures for high-speed URL matching," in *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM, 2010, p. 15.

[8] L. Degioanni and G. Varenni, "Introducing scalability in network measurement: toward 10 Gbps with commodity hardware," in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. ACM, 2004, pp. 233–238.

[9] H. Song, F. Hao, M. Kodialam, and T. Lakshman, "Ipv6 lookups using distributed and load balanced bloom filters for 100gbps core router line cards," in *INFOCOM 2009, IEEE*, april 2009, pp. 2518 –2526.

[10] M. Bando, Y.-L. Lin, and H. J. Chao, "Flashtrie: Beyond 100-gb/s ip route lookup using hash-based prefix-compressed trie," *Networking, IEEE/ACM Transactions on*, vol. PP, no. 99, p. 1, 2012.

[11] W. Jiang and V. K. Prasanna, "Field-split parallel architecture for high performance multi-match packet classification using fpgas," in *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, ser. SPAA '09. New York, NY, USA: ACM, 2009, pp. 188–196. [Online]. Available: http://doi.acm.org/10.1145/1583991.1584044

[12] General DDR SRAM functionality. (July 2001). [Online]. Available: http://download.micron.com/pdf/technotes/TN4605.pdf.

[13] QDR-II, QDR-II+, DDR-II, and DDR-II+ Design Guide. (November 2007). [Online]. Available: http://www.cypress.com/?docID=25736.

[14] Memory Interface Solutions. (September 2010). [Online]. Available: http://www.xilinx.com/support/documentation/.

[15] NetCOPE FPGA Platform for Rapid Development of Network Applications. Product Brief. [Online]. Available: http://www.invea-tech.com/data/netcope/netcope_pb.pdf.

[16] Home page of the IEEE 802.3ae 10Gb/s Ethernet Task Force. (October 2007). [Online]. Available: http://grouper.ieee.org/groups/802/3/ae/public/index.html.

[17] Home page of the IEEE 802.3ba 40Gb/s and 100Gb/s Ethernet Task Force. (January 2011). [Online]. Available: http://grouper.ieee.org/groups/802/3/ba/public/index.html.

[18] Virtex 7 family overview. (January 2010). [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm.