

CESNET Technical Report 8/2008
Designing a Hardware-Accelerated Firewall
with Two 10 Gbps Ports

VIKTOR PUŠ, TOMÁŠ DEDEK

Received 19.12.2008

Abstract

High-speed packet filtering should be one of the first steps in securing any modern computer network. However, solutions over 1 Gbps are practically impossible to implement in software, and must be implemented with the use of specialized hardware. This paper describes the design of a two-port firewall for 10 Gbps networks. The solution is based on hardware implementation of our classification algorithm. The firewall is designed to process data at full speed, without any packet loss. The target platform is the COMBOv2 card.

Keywords: FPGA, hardware acceleration, firewall

1 Introduction

With the increasing speeds of computer networks, tasks with extremely high demands on the computational power appeared. Examples of such tasks are pattern matching, packet classification, or even packet routing. However, implementation of these tasks as a software application for common processor could be very problematic or even impossible, due to a very high packet rate. On the 10 Gbps network, the packet rate is up to 15 millions of packet per second, so one packet is processed every 67 ns. Because speed is impossible to achieve with general CPUs, hardware implementations of these tasks are required. For the research purposes, FPGA chips are a good choice, because of the possibility to reconfigure the device many times.

1.1 Packet Classification

We focus on one of these performance-demanding tasks: the packet classification. The most common application for packet classification is a packet filtering. Packet filters are used not only as a firewalls protecting a network or computer. Another useful application is the lawful intercept, where network traffic is monitored and selected packets are sent to be further processed or stored. Classification algorithm is based on the following steps:

- The algorithm stores a set of rules ordered by priority.
- The resulting rule number is the first rule number matching the packet. Each rule has a defined action, which instructs the firewall about what to do with the packet. The basic actions are Accept and Deny for traffic filtering. Nevertheless, actions could be also more complex.

- After the packet is classified and the requested action is performed, the packet is sent to one or even more output interfaces. The selection of output interfaces could be also a part of the action.

All these steps must be performed in just several nanoseconds, so a parallel hardware implementation is a suitable solution. In the field of packet classification, many works have been published. A good overview of the current state is [7], with some recent improvements in [2] and [3]. Classification solutions may be divided into two groups:

- Technical solutions, using Ternary Content-Associative Memories (TCAMs).
- Algorithmic solutions without TCAMs.

TCAM solutions have one great advantage: the constant processing time. The property of Content-Addressable Memories is that it compares the input word to the whole content of the memory in a single access. This way, packets are classified with the constant speed. However, TCAMs have many disadvantages, namely the price and high power-consumption. That's why algorithmic solutions are a research subject. Algorithmic solutions use cheaper Random Access Memories, but their performance and memory requirements are often hard to predict. Our algorithm reaches the constant time complexity, so that it can compete with TCAM solutions.

1.2 Firewall

Current firewalls may be divided into three main groups:

- *Stateless firewalls* process every packet independently of the previous packets.
- *Stateful firewalls* store the state of every connection and process traffic based on the state of the connection.
- *Application firewalls* analyze application protocols and do application-specific filtering. Example: Email filter.

We choose to implement the stateless firewall, because of its straightforward design and clear requirements for its function.

Every stateless firewall has to perform several actions. When the packet is received, its header is parsed and important header fields are extracted. The packet is then classified on the basis of extracted header information. The requested action is known after classification, so that the firewall must perform the action (drop or forward packet, or other tasks).

Our goal is to design a hardware accelerated firewall with two 10 Gbps ports on the PC platform. We intend to use ComboV2 acceleration cards [1] for evaluation. The classification algorithm is implemented in the Virtex5 FPGA and necessary data structures are stored in the external QDR-II SRAM memory.

The rest of the report is organized as follows: in the next section we describe the overall system architecture and we give examples of its use. Section 3 describes our novel classification algorithm in detail. The firmware architecture is proposed in the section 4. Section 5 summarizes our work and we conclude this report in section 6.

2 System Architecture

The firewall consists of several layers (see Figure 1). The hardware layer is the COM-BOv2 card [1]. It is responsible for physical and electrical environment. Firmware layer is the configuration of the Virtex5 FPGA. This is where the accelerated part of the firewall is placed. Packets are received and classified in the Firmware layer. Also the requested action is performed directly in the FPGA. The Kernel layer is a Linux kernel module, which is responsible for firmware configuration and communication with the userspace. And finally, the Userspace layer is a set of software tools and libraries for system management and possibly packet processing.

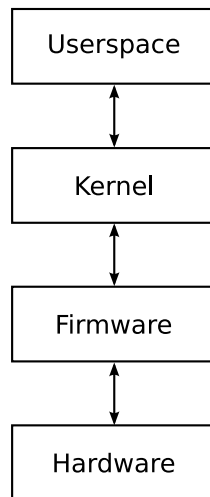


Figure 1. Layer structure of the firewall

We propose two basic scenarios for the firewall configuration:

2.1 Local Configuration

In this simpler option, the firewall is configured from a configuration file containing the ruleset. A set of command-line tools is ready for firmware initialization, rule-set loading, statistics gathering etc. Figure 2 shows the nificgend daemon, which prepares all necessary data structures and configures the classification core of the firewall. The second tool in Figure 2 is the nific-config, which is used by user to load ruleset into the nificgend.

2.2 Remote Configuration

This option may be useful for automated and remote control over the firewall. It is especially well suited for the legal intercept applications. Figure 3 shows the system architecture. The nificgend daemon is also used here, but it receives its configuration data from the remote Control Center over the Netconf protocol [8]. The nificexp daemons are responsible for forwarding of selected packets to the remote Monitoring Center. The nificd daemon is a mediator of configuration data among all parts of the system.

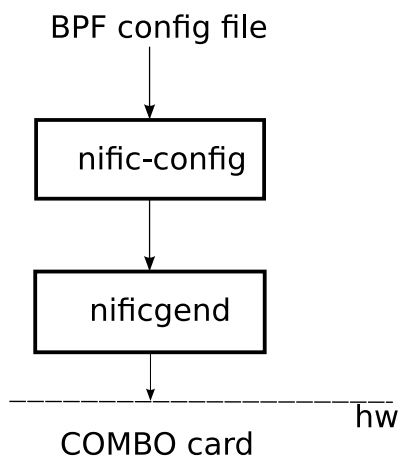


Figure 2. Local configuration of the firewall

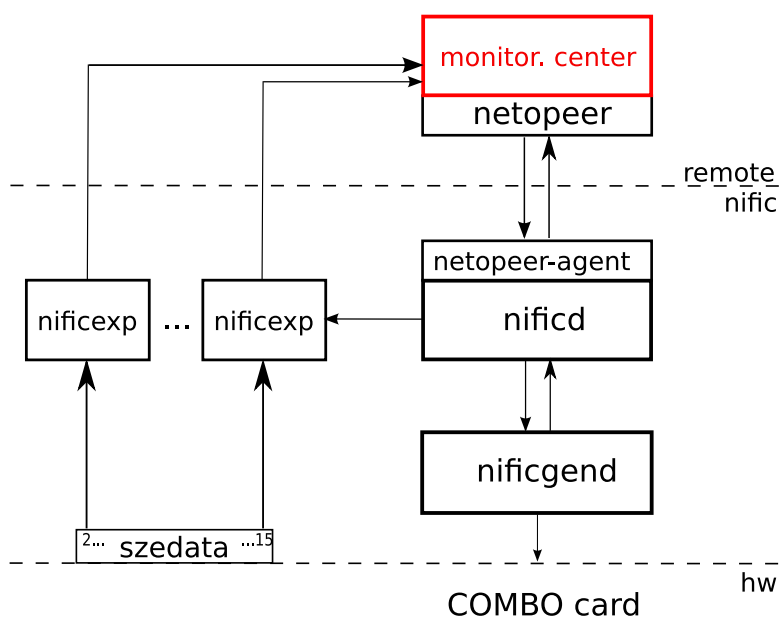


Figure 3. Remote configuration of the firewall

2.3 Examples of Use

The proposed system may be used in several scenarios, we discuss three of them here. The most simple is the firewall, filtering traffic on the edge of the network (Figure 4). Rules are written in a format similar to BSD Packet Filter, which is widely accepted in the industry.

Another option is a passive network probe, see Figure 5. In this case, network interfaces are used only to receive mirrored data from any network. The filtering functionality is used to select only relevant packets. Unimportant packets are dropped in the Firmware layer, so that host system (Userspace layer) receives only fraction of the network traffic for further software analysis or storage.

The third possibility is the legal interception scenario, where the system forwards filtered packets to the remote Monitoring Centers for further analysis or storage, see Figure 6.

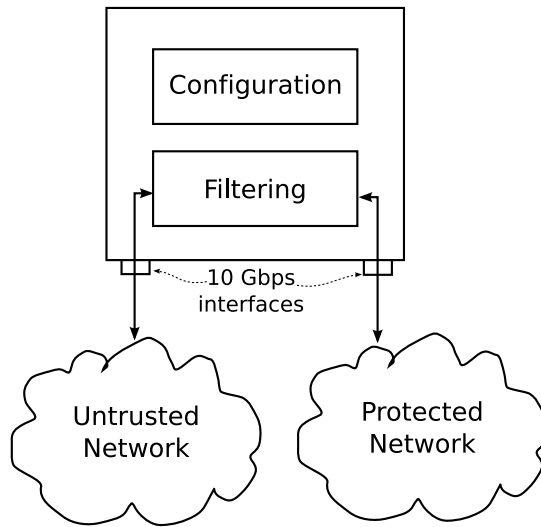


Figure 4. Device used as a firewall

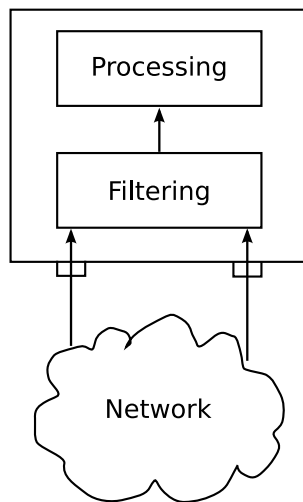


Figure 5. Device configured as a network traffic analysing probe

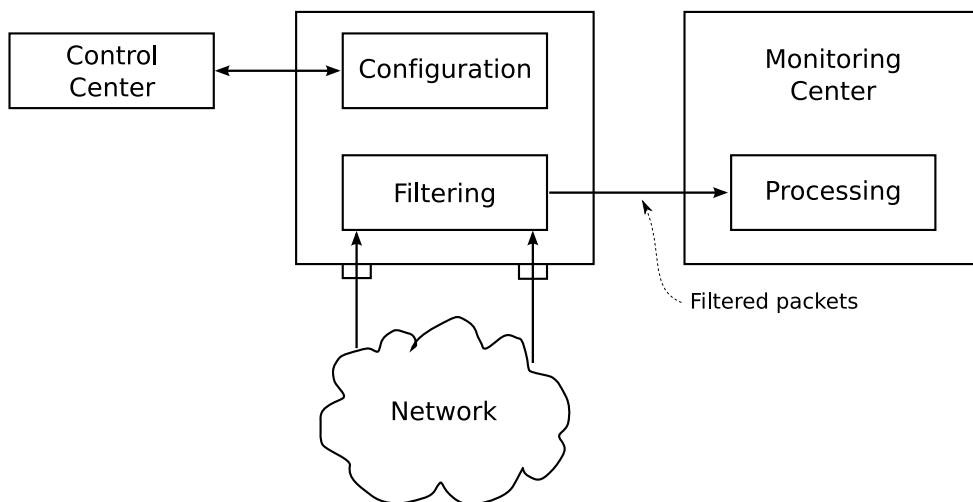


Figure 6. Device with remote configuration and monitoring centers

3 Packet Classification Algorithm

Our algorithm is based on the problem decomposition, similarly to several previously published approaches [2] and [3]. Algorithm scheme is shown in the Figure 7.

The first step is the parallel processing of all significant packet header fields by the LPM algorithm. From the given set of prefixes with various lengths, LPM algorithm selects the one that best matches the given full-length value. Many advanced LPM algorithms are based on trie, but use additional improvements. We use the TreeBitmap algorithm [5] to perform the Longest Prefix Match operation for IP and TCP/UDP ports. For this reason, TCP/UDP port ranges are converted to prefixes (one range is converted to one or more prefixes). Other field are processed in simple modules like tables for protocol number and input interface number processing, and CAMs for MAC addresses processing.

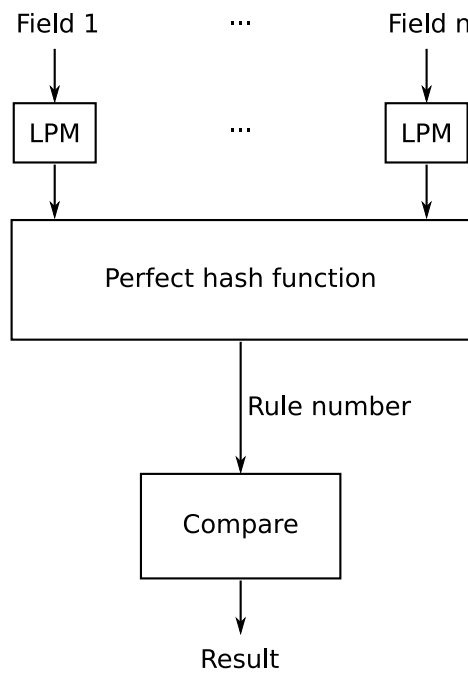


Figure 7. Scheme of the classification algorithm

All results of the first processing stage are concatenated into one 67 bit-wide word. Detailed analysis shows us that several words may correspond to one rule. These words are called *pseudorules* in the literature [2]. Figure 8 shows how pseudorules are generated.

Table 1. Three example rules

Rule	Dimension 1	Dimension 2
R1	1*	*
R2	1*	00*
R3	101*	100*

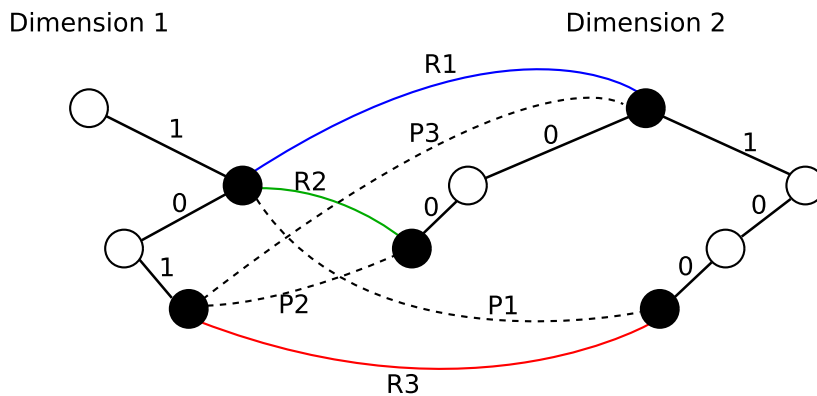


Figure 8. Pseudorules generation

We can see tries for classification in two dimensions, with three rules R1, R2, R3 (Table 1). But some combinations of LPM results are not covered, although they also must produce the correct rule number. For example, combination of LPM results 101 in Dimension 1 and 00 Dimension 2 is not in the Rule Table, but the correct result is the rule R2. That is why three pseudorules P1, P2, P3 must be added to cover the uncovered combinations. Table 2 contains these pseudorules.

Table 2. Three added pseudorules

Pseudorule	Dimension 1	Dimension 2	Rule
P1	1*	100*	R1
P2	101*	00*	R2
P3	101*	*	R1

We construct a special hash function using the algorithm for the perfect hash function [9] which uses the random acyclic graph search method. This hash function has intended collisions, all pseudorules corresponding to one rule are hashed to the same output number. In our example, LPM results (1*, 00*) and (101*, 00*) are both hashed to the address of rule R2. This way, the algorithm performs rule lookup in constant time and does not need to store any pseudorules. To compute the constructed hash function, two memory accesses must be performed for every input word. The size of the perfect hash function data structure depends on the amount of pseudorules. Because it could be very large, an external memory is used.

As the hash function gives some result for each packet, even if packet matches no rule, the false positive error could occur. This issue is resolved in the third step of the algorithm, where packet header is compared to the selected rule. If packet header fits the rule, the correct rule was found. If not, packet does not match any rule.

An important feature of our approach is massive software preprocessing, which has to be done for every firewall configuration. The preprocessing includes content generation for all packet header processing components, the hash function memory, the rule table and many more. The main idea is to do as much work in software in advance as possible to save as much FPGA resources as possible.

4 Firmware Architecture

The NetCOPE platform [4], which includes Input/Output Buffers and DMA controllers, is taken as a basic platform for our design, so we have a certain level of abstraction upon interfaces. A hardware scheme of the proposed solution is in the Figure 9. The whole figure shows the packet processing pipeline. As can be observed, the system has internally three ways, instead of two. This is because of the PCIe connection – hardware acceleration card is plugged in the PC, and software applications can also receive and send packets. All three lines are internally implemented by the FrameLink protocol in the FPGA. The FrameLink protocol is a simple modification of the Xilinx LocalLink protocol¹. Each FrameLink frame encapsulates one Ethernet frame and adds 128 bits of FrameLink header and a few service signals. To transmit full 10 Gbps flow, each line is 128 data bits wide and the whole design is running at frequency of 125 MHz.

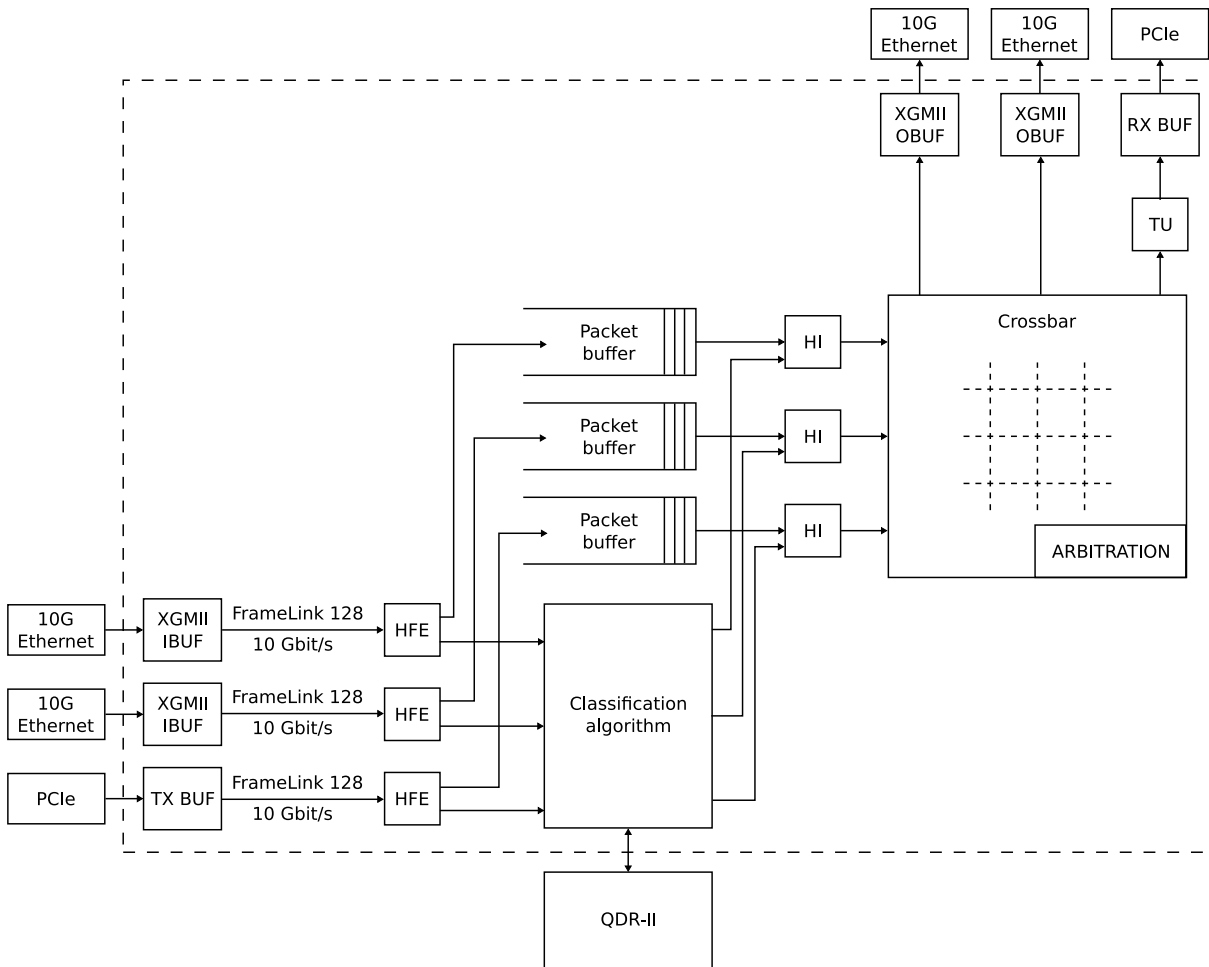


Figure 9. Firewall architecture

Packets from the network are received by XGMII IBUF modules and converted into the FrameLink format. Packets sent from software are received in the DMA TX Buffer (part of the NetCOPE platform, [4]) and also converted into FrameLink.

¹ http://www.xilinx.com/products/ipcenter/LocalLink_UserInterface.htm

Three FrameLink flows then continue to three instances of the Header Field Extractor engines, which performs packet header parsing. Further, each flow is forked into two parts: The first part is the original packet encapsulated in the FrameLink protocol. The second part is also FrameLink protocol, but it contains only parsed header fields.

The original packets are stored in three on-chip FIFO queues, implemented by Virtex5 BlockRAMs. Parsed packet header data are sent to the Classification module. The module is a straightforward implementation of the algorithm described in the previous section. The outputs of the Classification module are results of the classification algorithm: number of the correct matching rule and its associated actions. The Header Insert module inserts rule number and actions into the FrameLink header.

According to the action in the frame header, the Crossbar module switches each frame from the input line to the correct output lines. Each frame can be switched to zero (packet is dropped) or more interfaces. Even when the Crossbar has buffers in each intersection (nine buffers), the output blocking could occur when two or three input lines are switched into one output line. For that cases, there is Deficit Round Robin algorithm [10] to perform fair queuing in the Crossbar. In the output flow which will be transmitted into software, there is the Trimming Unit in addition. This module shortens packets according to the action. This trimming may be useful for saving the PCIe throughput. The flow to software is then stored in the DMA RX Buffer, from where it is then transmitted to the software processing. Two other flows use XGMII OBUF to send packets into the network.

5 Results

The whole classification core, including everything between Header Field Extractor and output interfaces, was verified by modified SystemVerilog verification environment as it was described in [6]. The input to the verification environment is the rule set. On the basis of the rule set, the set of testing packet headers is generated and is used as an input to the design. Generated packet headers are then independently classified by the verification environment and results are compared to the results from the design. This approach helped us to find a lot of tricky bugs before the design was tested in hardware. The SystemVerilog verification significantly decreased the development time of the whole system.

Our implementation classifies packets according to these nine header fields:

- Source MAC address
- Destination MAC address
- Source IPv4 address
- Destination IPv4 address
- Protocol
- Source Port
- Destination Port
- TCP Flags
- Input Interface number

Our tests with synthetic and real-life rulesets from university campus network show that the device is able to support up to 1000 rules. There are several factors limiting the number of rules, the most severe limitation is the perfect hash table. Even when it is stored in external QDR-II memory, the number of pseudorules may be too high.

We have tested prototype of our design with 1 Gbps only, because the 10 Gbps NetCOPE platform on the COMBOv2 card is still under development, so we don't have results regarding throughput of the system. However, the number of FPGA resources is known (at least approximately), because 10 Gbps and 1 Gbps versions are very similar. The whole design in the Virtex5 LX110T FPGA consumes 16,872 (97%) slices and 126 (85%) BlockRAMs

6 Conclusion

Our results show that programmable hardware is capable of packet classification even for 10 Gbps networks, due to the possibility of exploiting the hardware parallelism (parallel execution units, pipelined processing). Our implementation on the COMBOv2 card was verified in SystemVerilog and it passed basic functionality tests.

We continue to improve the classification algorithm itself to reduce the number of pseudorules. We also seek implementation optimizations in order to support more rules with the same hardware. Example of these optimizations may be Rule Table compression or more memory-efficient LPM algorithm than TreeBitmap. We also plan to support the IPv6 protocol.

References

- [1] NOVOTNÝ, J.; ŽÁDNÍK M. *COMBOv2 – Hardware Accelerators for High-Speed Networking*. XILINX Academic Forum, San Jose, November 2008. Available online².
- [2] DHARMAPURIKAR, S.; SONG, H.; TURNER, J.; LOCKWOOD, J. Fast packet classification using bloom filters. In *Proceedings of the 2006 ACM/IEEE Symposium on Architecture For Networking and Communications Systems*, San Jose (CA), December 03 - 05, 2006. New York: ACM, 2006.
- [3] SRINIVASAN, V.; VARGHESE, G.; SURI, S.; WALDVOGEL, M. Fast and scalable layer four switching. *SIGCOMM Comput. Commun. Rev.* vol. 28, no. 4, 1998.
- [4] MARTÍNEK, T.; KOŠEK, M. NetCOPE: Platform for Rapid Development of Network Applications. In *11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, 16-18 April 2008, p. 1–6.
- [5] EATHERTON W.; VARGHESE G.; DITTIA Z. Tree bitmap: hardware/software IP lookups with incremental updates. *SIGCOMM Computer Communication Review*, vol 34, no. 2, p. 97–122, 2004.

² http://www.liberouter.org/documents/COMBOV2-2008-02-10-Academic_Forum.pdf

- [6] KOBIERSKÝ, P.; MÁLEK, T.; PUŠ, V.; ŠAFRÁNEK, D. *SystemVerilog verification of VHDL design*. technical report 35/2007³. Praha: CESNET, 2007.
- [7] GUPTA, P; MCKEOWN, N. Algorithms for packet classification, *IEEE Network*, 2001. Available online⁴.
- [8] ENNS, R. (Editor). *NETCONF Configuration Protocol*, RFC 4741⁵, IETF, 2006.
- [9] CZECH, Z. J.; HAVAS, G.; MAJEWSKI, B. S. An optimal algorithm for generating minimal perfect hash functions. *Information Processing Letters*, vol. 43, no. 5, p. 257–264, 1992.
- [10] SHREEDHAR, M.; VARGHESE, G. Efficient fair queueing using deficit round robin. *SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 4, 231-242, 1995.

³ <http://www.cesnet.cz/doc/techzpravy/2007/systemverilog-vhdl-verification/>

⁴ <http://tiny-tera.stanford.edu/~nickm/papers/index.html>

⁵ <http://www.ietf.org/rfc/rfc4741.txt>