Research activity of Fabio Vitucci

PhD – first year

f.vitucci@netserv.iet.unipi.it

TLC NetGroup

Department of Information Engineering

University of Pisa

Index

1. Introduction

My activity, supported by the TLCNetGroup of University of Pisa, focuses on research and development of Network Processors. The laboratories of Telecommunication Networks of Department of Information Engineering dispose of four RadiSys ENP-2611 cards [1], provided by Intel [2] and equipped with the Network Processor Intel IXP2400 [3]. Therefore our activities pertain only to this architecture, even though we continue to analyze alternative solutions. The main ideas that lead the fundamental choices about algorithms and programming in our studies are common to Network Processors of different vendors.

For the research activity, a test-bed (shown in fig. 1) is provided to test working and performance of programmed modules. In particular, the Spirent ADTech AX4000, a traffic generator and analyzer, allows us to create the suitable traffic flows for testing and to observe their short-range and long-range statistical properties.

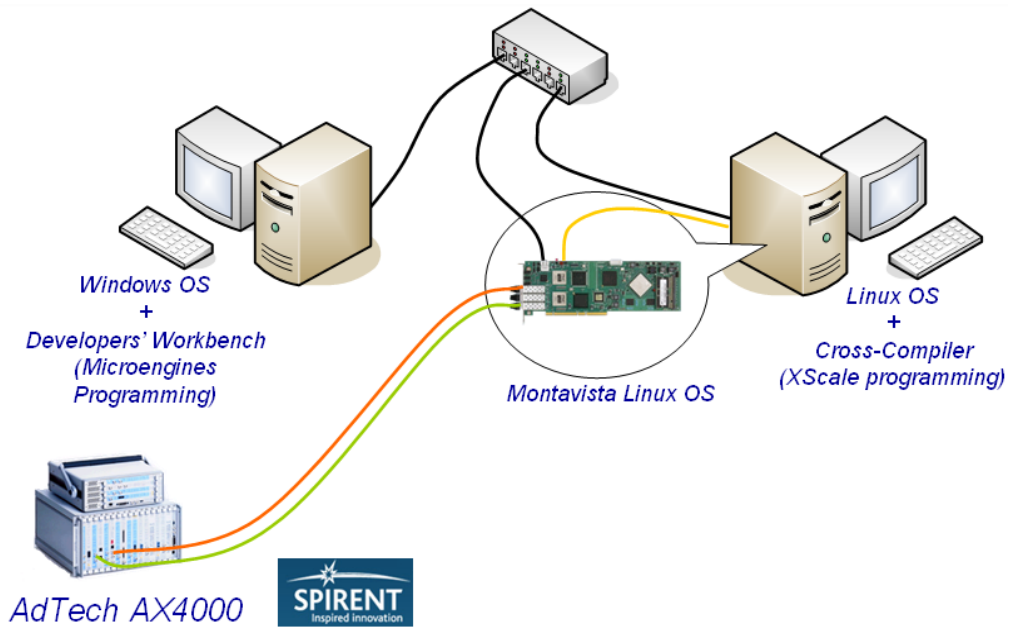

Fig. 1: Test-bed

2. Packet classification

Nowadays packet classification is a fundamental task for network devices such as edge routers, firewalls and intrusion detection systems. Determining which flow packets belong to is important for many applications, and it is necessary, for example, to provide differentiated services, to detect anomalous traffic and to sort attack patterns. Therefore packet classification is becoming more and more complex, with more flexibility and higher performance requirements. Complexity, flexibility and performance are just the main targets for NPs, indeed a packet classifier has been chosen as the first application to be implemented on the IXP2400.
The main target of my research is to propose a rigorous methodology for designing a network component, the packet classifier, which takes into account both the functional specifications of the component itself and the specific features of the candidate hardware for its actual implementation.

Since multidimensional classification is a complex problem, many researchers have explored and proposed a wide variety of algorithms. In my research, all these solutions have been investigated to find the most suitable for our hardware platform. In the following, the main theoretical principles of these algorithms are illustrated (for details see references [5-13]), divided into four big categories [4].
The first category consists of the algorithms that use basic data structures. The simplest data structure is a linked-list of rules stored with decreasing priority: a packet is compared to each rule sequentially until a rule is found to match all fields; clearly, such a linear search exhibits poor scaling properties. An advanced data structure is a *d*-dimensional hierarchical trie, recursively constructed as follows: we first build a *1*-dimensional trie according to the first dimension of rules. Then, for each prefix in the first trie, we recursively construct a (*d-1*)-dimensional hierarchical trie according to rules that specify exactly that prefix in the first dimension. Other advanced data structures proposed for packet classification are Set-Pruning trie [5] and Grid of Tries [6].
The second category is composed of Geometric Algorithms. Each field of a rule can be specified either through a prefix/length pair or in terms of an operator/number. From a geometric point of view, both these specifications could be interpreted as intervals on a line. Thus, a rule with *d* fields represents a *d*-dimensional hyper-rectangle and a classifier is a set of such hyper-rectangles with the associated priorities. In this context, a packet header (*d*-tuple) represents a point P in the *d*-dimensional space and the packet classification problem consists of finding the highest priority hyper-rectangle that encloses P (see fig. 2). This category includes the Cross-Producting algorithm [6], a solution for 2-D classifiers proposed by Lakshman et al. [7], the Area-Based Quadtree (AQT) structure [8] and the Fat Inverted Segment Tree (FIS-tree) [9].
The third category is that of Heuristic Algorithms. Such algorithms are based on the evidence that, due to the considerable redundancy of classification rules, the size of data structures of real classifiers is generally much smaller than that theoretically calculated in worst-cases. Recursive Flow Classification (RFC) [10], Hierarchical Intelligent Cuttings (HiCuts) [11], Tuple Space Search [12] are classification algorithms that, in different manners, perform a heuristic pre-partitioning of the problem space first, and then rearrange the decision data structure in order to take advantage of the above mentioned redundancy. This way, they reduce considerably the time for packet classification while saving memory.

Rule	F1	F2
R1	00*	00*
R2	0*	01*
R3	1*	0*
R4	00*	0*
R5	0*	1*
R6	*	1*

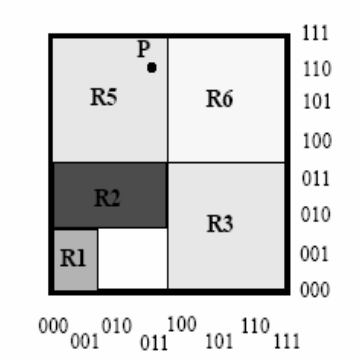

Fig. 2: Packet classification in a geometric representation

The last group is composed of Hardware-Based Algorithms, such as Bitmap Intersection [13] and Ternary Content-Addressable Memories (T-CAMs) (see fig. 3). These algorithms usually split the multidimensional classification problem into several one-dimensional searches to be performed by means of specific hardware-based solutions (for example, the use of T-CAMs allows achieving 100 million queries per second [14]).

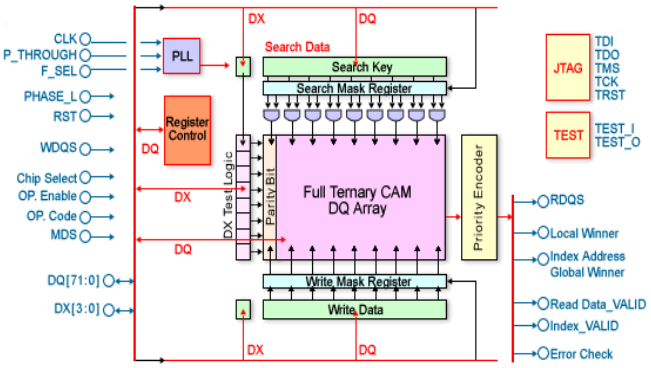

Fig. 3: An example of Ternary-CAM

3. Classification Algorithm Selection

As will be shown in the following, the classification algorithms of the first category prove to be well tailored for implementation into the network processor architecture. In particular, I select the very complex *multidimensional multibit trie* data structure because of its low search time and

small number of memory accesses. The reason of this choice is tightly related to the specific characteristics of the Intel® IXP2400 hardware architecture.

To choose an algorithm that exploits the capabilities of the Radisys® ENP-2611 board, all the classification algorithms described in previous section have been compared. Classic performance metrics are analyzed, as well as specific parameters to investigate the compatibility with NPs. The classic parameters analyzed are:

1) *Search Speed*: the increase of link speeds requires faster and faster classification;
2) *Memory requirements*: small storage requirements determine high memory access speed and, in turn, low power consumption (both features are very relevant for NPs);
3) *Scalability in classifier table size*: the size of a classification table depends on the applications; the capability of handling a large number of rules makes a classification algorithm appealing for many applications;
4) *Scalability in the number of header fields*: the number of header fields to be processed increases with the number of services provided;
5) *Update time*: the data structure needs to be updated upon insertion and removal of rules in the classifier; some applications require strictly short updating time;
6) *Flexibility in rule specification*: the ability of an algorithm to handle a wide range of rule specifications, such as prefix/length, operator/number, and wildcards, makes it reusable for many applications.

Performance metrics of classification algorithms belonging to the four categories (as described in section II-A) have been evaluated and compared. For the sake of conciseness, the table [4] in fig. 4 shows the results for two specific metrics only, namely the worst case complexity of *search time* and *storage requirements*. The worst case is computed considering a classifier table filled with totally different rules, without overlapping in the values of the fields. In the table, we denote the number of entries in the classifier with *N*, the maximum prefix length (in bits) of a field with *W*, the number of fields with *D*, and the total number of data structure levels with *L*.

Algorithm	Worst-Case Search Time	Worst-Case Storage
Linear search	O(N)	O(N)
Hierarchical Tries	$O(W^D)$	O(NDW)
Set-Pruning Trie	$O(W^D)$	$O(N^D)$
Grid of Tries	$O(W^{D-1})$	O(NDW)
Cross-Producting	O(DW)	$O(N^D)$
FIS-Tree	O((L+1)W)	$O(LN^{1+1/L})$
AQT	O(NW)	O(W)
HiCuts	O(D)	$O(N^D)$
RFC	O(D)	$O(N^D)$
Bitmap Intersection	O(DW+N/W)	$O(DN^2)$
Ternary CAMs	O(1)	O(N)

Fig. 4: Comparison among different algorithms

In addiction to the above listed metrics, for a correct selection of the packet classification algorithm, all the possible bottlenecks of NPs have to be taken into account. In particular, the number of memory accesses, the amount of memory consumption, and the size of the instruction store have to be considered. In other words, the key challenge is designing a packet classification algorithm that requires both low memory space and low access overhead: this would provide a proper scaling with respect to high bandwidth networks and large databases of classification rules.

4. The multidimensional multibit trie

The analysis of results (especially the ones associated with search speed and *number of accesses* – not shown in the previous table) led to identify a classification algorithm based on a multidimensional multibit trie structure as the most appropriate for IXP2400. Such an algorithm has, in the worst-case, a research speed of the order of $O(W/K)$, a fix number of memory accesses (L) and a storage complexity of the order of $O(2^{(K-1)} \times N \times W/K)$, where K is the average length of strides.
In my implementation, the classifier processes the five main header fields, as typically adopted in the literature [14]: the IP Destination Address, IP Source Address, Layer 4 Destination Port, Layer 4 Source Port, and Layer 4 Protocol Type. Hence, we have a 5-dimensional classifier, which uses a 5-stage hierarchical search trie. In each stage, the classification algorithm processes a single packet header field; the analysis of each field is divided into several steps, performed by using a specific number of bits (named *stride*), instead of 1 bit at a time, to decrease the number of memory accesses (see fig. 5).
The choice of the strides lengths is based on a theoretical method proposed by Srinivasan [17] and on the analysis of the actual distributions of rules in real classifiers [14]. Thus, according to the classic IP classful addressing structure (in [14], it has been shown that the most of classification rules ignore subnetting) 32 bits long IP addresses are divided into three strides of 16-8-8 bits respectively, while the 16 bits long TCP/UDP ports are divided in two strides of 6-10 (because most of the port numbers in use are usually below 1024). Finally, a single stride of 8 bits is used for the protocol field. A packet may match more than one rule, thus each rule in the database is associated with a field that identifies its priority.

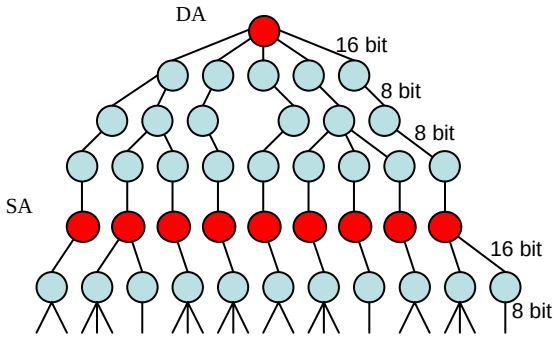

Fig. 5: Beginning of a multidimensional multibit trie

5. A modified version of the algorithm

The main issue of classical multidimensional multibit trie is memory consumption. Indeed, to handle rules with non-specified parts (e.g. 121.132.*.*) and to have a single memory access in each stride of trie, 2^s nodes for level are necessary (where s = stride length) though only a few of them correspond to actual existing rules and the remaining nodes are virtual. This requires large memory storage, and in our scenario would force to put the classification table in DRAM, the memory with the highest access latency.
In order to decrease memory requirements, when there is a rule with non-specified parts (use of non-contiguous bit-masks is very frequent in real classifiers [10]), a *level crossing* is performed, which consists of jumping to the next level without need for nodes explosion. This operation, regardless to the specific sizes of available memories, drastically reduces memory consumption as required by NPs; this permits a feasible integration of different complex applications into the device.
I developed an apposite simulator in C language to compute the number of memory accesses and the memory size of the classifier. In simulation runs, we used classification tables very similar to the ones used in some actual firewalls and edge routers; such tables contain both fully specified rules and partially specified rules (that is, several fields are not specified). Fig. 6 shows the comparison, in terms of storage requirements, between the original classification algorithm and our modified version. The results evidently justify our choices and modifications.

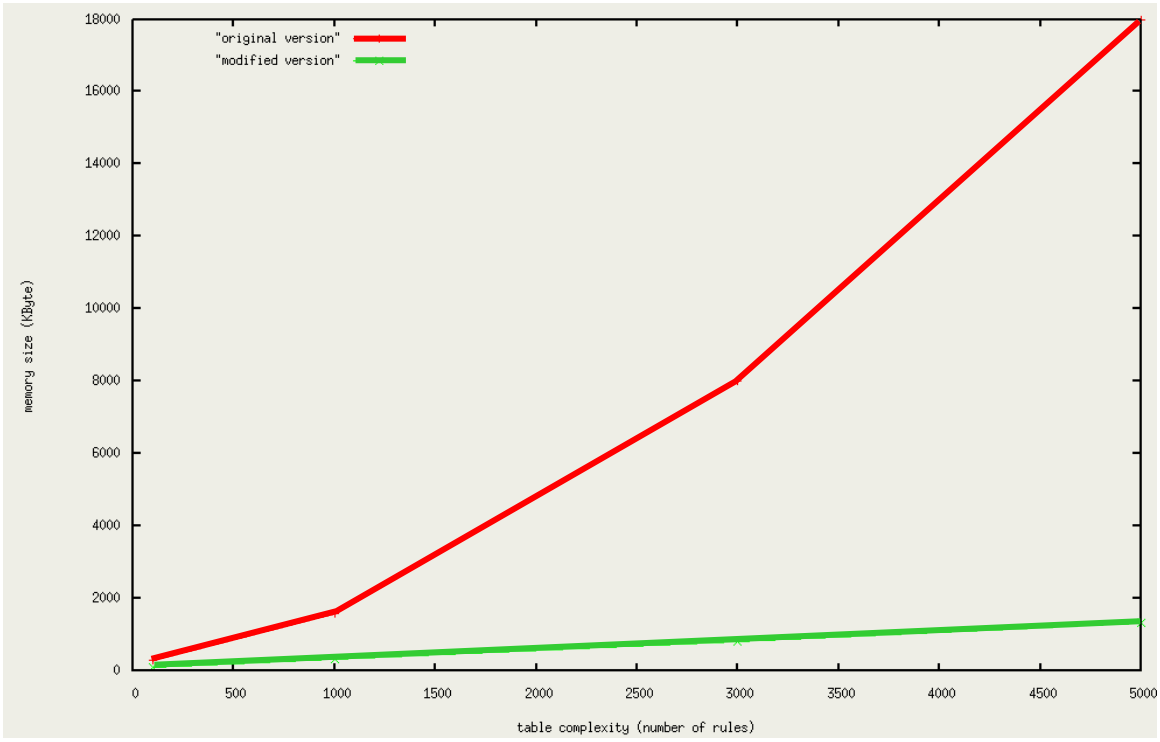

Fig. 6: Memory consumption for different versions of algorithm

Obviously, the introduction of level crossing increases the algorithm complexity, because of possible backtrackings: indeed, the use of direct crossing can create, during the matching search operation, cases of wrong paths and consequent needs for traversals (see fig. 7 for an example with one field only).
This, in turn, makes the number of processing steps variable and raises the number of instructions necessary to implement packet processing (in spite of this, the instruction store of IXP2400 microengines is large enough to easily handle the modified algorithm). This phenomenon is largely compensated by the possibility of locating the classifier table into the SRAM, a memory device with lower access delay than that of the DRAM, with the global effect of a reduction of the packet processing time.
Another improvement to the original algorithm is derived from the analysis of many real rules databases, and deals with the process of handling ranges of TCP/UDP port fields. Indeed, for these fields, it is common to find value specifications such as “greater than 1024”, “between 100 and 128”, and so on. Therefore, to cope with this issue, suitable objects are introduced in the structure of the trie nodes to identify lower and upper bounds of the range. Thus, when a packet header is processed to be classified, its port fields are no longer compared to a single value but rather to a range of them.

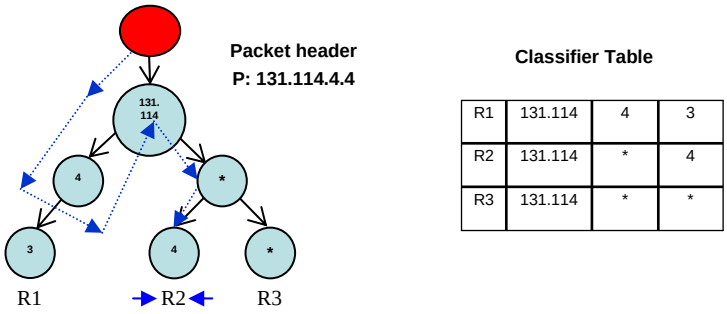

Fig. 7: A simple example of trie backtracking

6. Module implementation

This section accurately describes the functions implemented for the classification module. The XScale processor and the micro-engines perform distinct operations: the first one builds the decision-making data structure, the second ones process packet headers and “classify” them.
At first the XScale functions (implemented in C-language) will be illustrated. A PHP interface (shown in fig. 8) is created, which permits the network system administrator to insert the classification rules. Thus, the module generates a proper file to be passed to XScale, which calculates the number of necessary nodes for each level and the SRAM addresses of nodes levels. Then it initializes the data-structure, using the calculated addresses.

The microengines perform the real process of packet classification. They receive packets, retrieve the proper fields from packet headers (IP addresses, Layer 4 ports and Protocol field) and look for the exact rule according to the algorithm and using the data structure in SRAM. If a rule is matched, the TOS field is modified according to the matched rule. The functionalities of microengines are implemented in micro-code assembler, a specific assembler for the Intel microengines.
Implementing these functionalities, the multithreaded programming is accurately investigated [16], which allows to hide memory accesses latency (see fig. 9). Microengines of the IXP2400 support software-controlled multithreaded operations, by providing eight hardware-assisted threads of execution (i.e. zero-overhead context switch [16]).

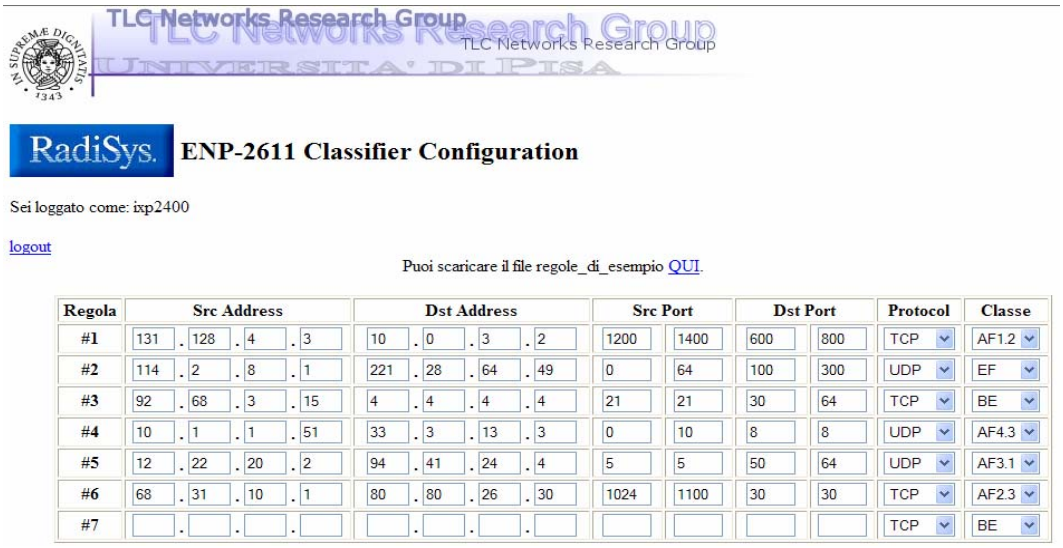

Fig. 8: A PHP interface for the rules insertion

In order to obtain an optimized code, which allows each microengines to be always busy, some important features are analyzed, e.g. stalling and filling. The main reasons for the presence of idle time in our implementation are the number and the frequency of SRAM accesses. Indeed these accesses are reduced creating a very compressed data structure, to consolidate adjacent memory accesses. This target justifies the calculation of SRAM addresses by the XScale, which subsequently is able to use only the necessary memory space, allowing less memory accesses.
Another fundamental phenomenon to be considered (and to be avoided) in the programming is *stalling*. The IXP2400 presents a finite queue of SRAM access requests: if a thread needs a memory reading but the queue is full, the thread continues to forward the request, without releasing the control of the processor and stalling the whole microengine. This clearly reduces the processing power of the microengine. In order to avoid microengine stalling, a first solution is the *instructions filling*, that is putting before a memory access request all the instructions independent by the reading, even thought they follow the memory access in the normal code flow. All these optimizations are manually performed, because the “code improvement” options provided by compilers aren’t enough efficient.

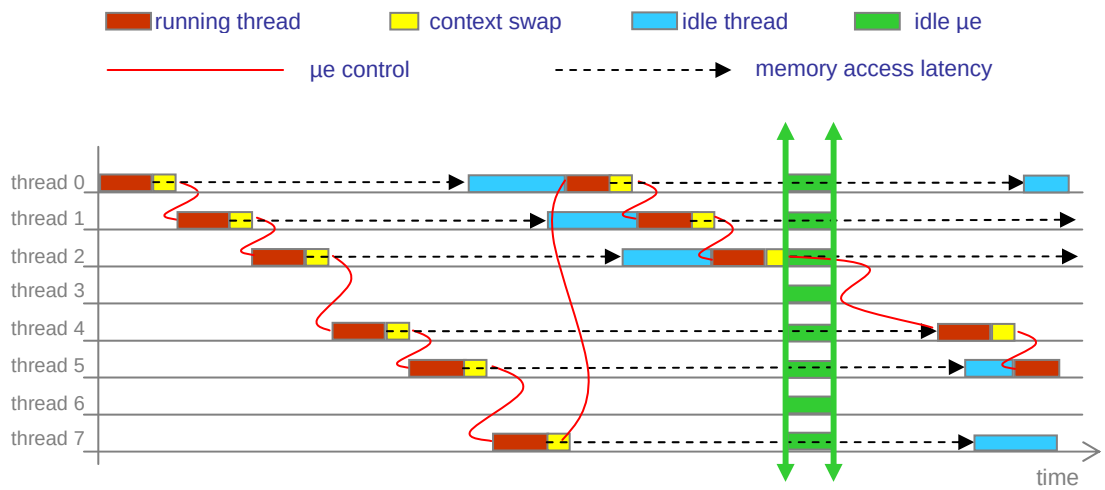

Fig. 9: A scheme of the multithreaded behaviour of a microengine

The application of packet classifier goes upon the Intel IPv4-Forwarder, which implements packet reception, header reading and transmission. This code turns out from an arrangement, made by the School of Computer Science (University of Pittsburgh), of the Intel code, properly written for the IXDP2400 card (not for the RadiSys ENP-2611 card). Actually this arrangement was incomplete and not much documented, so we had to execute many reconfiguration to use the application. Fig. 10 shows the IPv4-Forwarder application at microengine level.

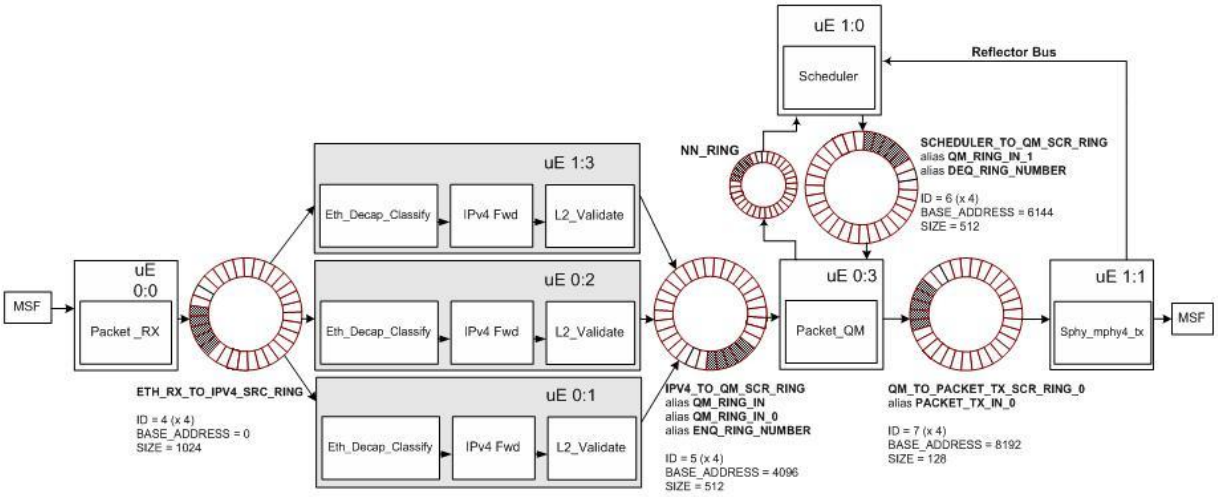

Figura 10: IPv4 Forwarder architecture

The red circles represent the rings, which are on-chip circular memories. The more external rectangles represent the microengines, the more internal the microblocks (pieces of code that implement a specific function). The white microengines contain the *driver-blocks*, directly provided by the Intel, strictly dependent on hardware, dealing with low level functionalities, (e.g. transmission and reception mechanisms). The grey microengines contain the *user-written microblocks*, concerning the packet process functionalities and resulting the real target of developers.
The classification module is inserted into the IPv4 forwarder (in the pipeline that processes packets), in order to obtain a layer 3-4 packet classification (see fig. 11).

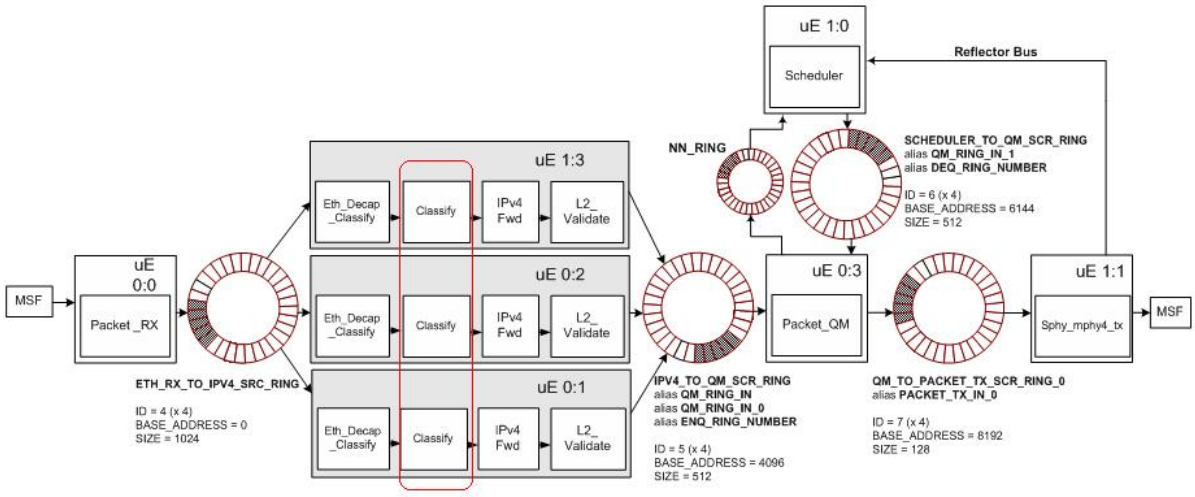

Fig. 11: IPv4 Forwarder with the classification module

7. Measurements

The last step of my research about the packet classifier was a series of measurements to test its working and performance. The classifier provides a maximum packet rate of 2.033.000 pkt/sec, without lost packets. Beyond, the lost packets are only the packets exceeding this threshold (see fig. 12) and the maximum supported packet rate notes only a little decrease (see fig. 13). Indeed, robustness in case of congestion is one of the most important characteristics of the classifier.
The reason of packet loss is that the packet processing pipeline, with the additional cycles of classification block, is not able to sustain the packet arrival speed in the first reception ring. Table in fig. 13 presents the cycles for the different operations performed by the classification module. The first column represents the “plain” instruction cycles for each operation, the second one also contains the “theoretical” cycles for expected memory readings and the third one shows the experienced values (minimum and maximum values). The results remark the complexity of network processors programming, due to extreme variability of memory access latencies.

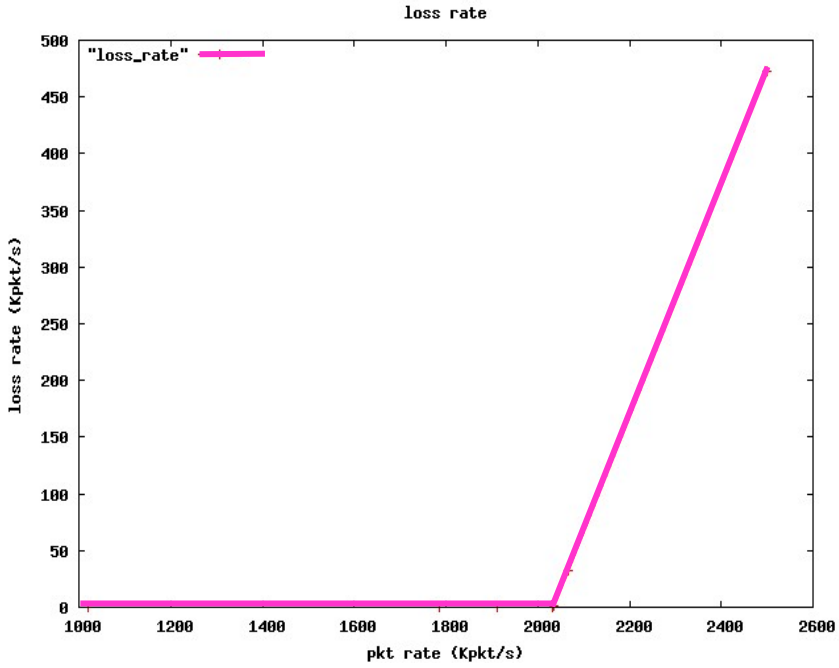

Fig. 12: Loss rate

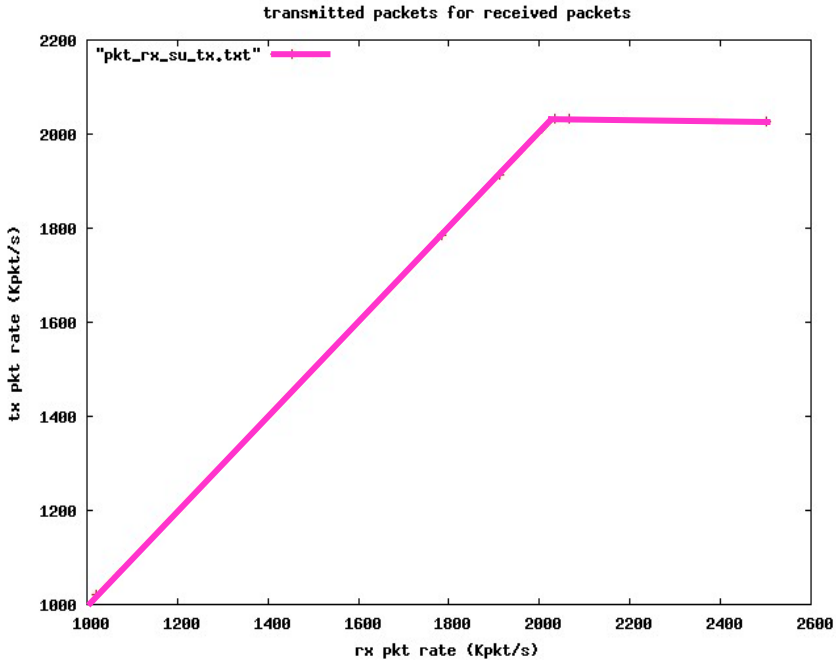

Fig. 13: Maximum supported packet rate

OPERATION	CYCLES OF INSTRUCTIONS	+ MEMORY ACCESSES DELAY	MEASURED CYCLES
Registers inizialization	**23**	**23**	**23**
Reading first node	**24**	**24 + 180**	**250:300**
Reading other nodes (7)	**15**	**15 + 120**	**190:650**
Reading ports nodes (2)	**19**	**19 + 120**	**194:654**
Writing TOS	**18**	**18 + 30**	**60:400**
TOT.	**208**	**1498**	**2051:6581**

Fig. 14: Instruction cycles for the classification module

Moreover, the classifier supports until 10000 rules. The bound is not given by intrinsic restrictions of the classifier module, but only by the size of the available memories in the RadiSys cards. Performance (in terms of throughput and processing delay) is fully independent of the number of rules: scalability towards number of rules is another fundamental feature of this classifier.
Packet delay for the whole processing pipeline differs according to traffic load (see fig. 15). Before packet loss, delays increase from 35 to 110 µsec. In case of congestion, delays reach very high values (beyond 1 msec) and become extremely variable.

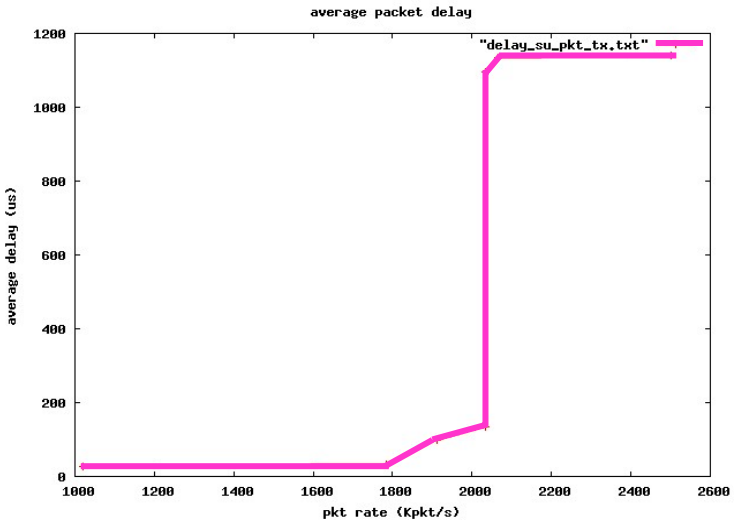

Fig. 15: Packet delay for the whole classifier

8. Future work

This research presents many open fields, which will be investigated in next mounts. The aim will be a classification module suitable for different applications and for different targets (access, edge and core network). In particular, next year the following features will be analyzed and developed:

- use of different classification algorithms;
- analysis of capabilities of Hash and CRC units, available in the IXP2400;
- real-time update of rule table;
- full flexibility in rules specification (prefix/length, operator/number, wildcards, non contiguous bit-mask, etc.).

Another scheduled activity is implementing a fully different approach to packet classification process. A dedicated microengine must analyse the first packet of each traffic flow and perform the search for the matching rule, using the data structure generated by the XScale in SRAM. Once the matching rule is found, the microengine inserts an index in a proper table put in a memory shared by all the microengines. This index is composed of an hash value of the processed header fields and an identifier of the matched rule. Once one of the microengines dedicated to packet classification analyses a packet, at first it checks in this local table: if there is an entry corresponding to the packet, the microengine immediately modifies the TOS field according to the recorded rule and passes the packet to the next module; otherwise it passes the packet to the microengine dedicated to searching the exact rule for the first packets of the flows. This way, only a microengines must access to SRAM, and only for one packet per flow. A big increase of classifier performance is expected to this new assignment of tasks among the different processors.

9. References

[1] RadiSys Corporation, "*ENP-2611 Data Sheet*", http://www.radisys.com.
[2] Intel Corporation, "*Intel IXA University Program*", http://www.ixaedu.com.
[3] Intel Corporation, Intel® IXP2400 Network Processor, http://www.intel.com/design/network/products/npfamily/ixp2400.htm.
[4] Pankaj Gupta and Nick McKeown, "Algorithms for packet classification", *IEEE/ACM Trans. Networking*, vol. 15, pp. 24-32, March-April 2001.
[5] P.Tsuchiya, "A search algorithm for table entries with non-contiguous wildcarding", Bellcore, unpublished report.
[6] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvagel, "Fast and scalable layer four switching", *Proceedings of ACM Sigcomm '98*, pp. 191-202, August 1998.
[7] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching", *Proceedings of ACM Sigcomm '98*, pp. 203-214, August 1998.
[8] M. M. Buddhikot, S. Suri, and M. Waldvogel, "Space decomposition techniques for fast layer-4 switching", *Proceedings of Conference on Protocols for High Speed Networks*, pp. 25-41, August 1999.
[9] A. Feldman and S. Muthukrishnan, "Tradeoffs for packet classification", *Proceedings of Infocom*, vol. 3, pp. 1193-1202, March 2000.
[10] Pankaj Gupta and Nick McKeown, "Packet classification on multiple fields", *Proceedings of ACM Sigcomm '99*, pp.147-160, August 1999.

[11] Pankaj Gupta and Nick McKeown, "Packet classification using hierarchical intelligent cuttings", *Proceedings of Hot Interconnects VII*, Stanford, CA, August 1999.
[12] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple search space", *Proceedings of ACM Sigcomm*, pp. 135-146, September 1999.
[13] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching", *Proceedings of ACM Sigcomm*, pp.191-202, September 1998.
[14] Michael E. Kounavis, Alok Kumar, Harrick Vin, Raj Yavatkar, and Andrew T. Campbell, "Directions in Packet Classification for Networks Processors", *Proceedings of 2nd Workshop on Network Processors*, February 2003.
[15] V. Snirivasan and G. Varghese, "Faster IP lookups using controlled prefix expansion", *Proceedings of ACM Sigmatics*, June 1998.
[16] Erik J. Johnson and Aaron R. Kunze, "*IXP2400/2800 Programming*", Intel Press, 2003.