A Rule Grouping Technique for Weight-Based TCAM Coprocessors

Hao Che, Yong Wang, and Zhijun Wang
The Department of Computer Science and Engineering
The University of Texas at Arlington
Arlington, TX 76019
hche@cse.uta.edu

Abstract

A crucial issue associated with a TCAM coprocessor with weights is that no more rules can be enforced if the weights are exhausted. In this paper, the problem is identified and a rule grouping technique is proposed to solve the problem. The technique allows virtually unlimited number of rules with arbitrary rule structures to be enforced. It requires no special hardware support and can be readily implemented in a fully programmable network processor and a weight-based TCAM coprocessor.

1. Introduction

A hardware-based solution for packet classification is the use of a *ternary content addressable memory* (TCAM) coprocessor to offload the *packet classification* tasks from a *network processor*. Fig. 1 shows the diagram of a typical line card with a network processor interfacing with a TCAM coprocessor. When a packet is to be classified, the network processor generates a *search key* from the header of the received packet and sends this search key to the TCAM coprocessor to find a matched *rule* in a TCAM rule table. The matched rule entry maps to a memory address in an associated memory containing the *action* associated with that rule. While waiting for the TCAM coprocessor to return an action associated with the matched rule for one thread, the network processor does a context switching to pick up a different thread without wasting clock cycles. By matching the search key against all the rule entries in parallel, the TCAM coprocessor achieves *one memory access* matching performance and therefore offers the highest possible packet classification performance [1].

There are two types of TCAM coprocessors in the market today. One requires that the rules in a TCAM table be arranged in an ordered list according to their match priorities. The action associated with the rule in the lowest memory address is returned if a search key matches multiple rules. We call this type of TCAM coprocessors the *Ordered TCAM* (OTCAM).

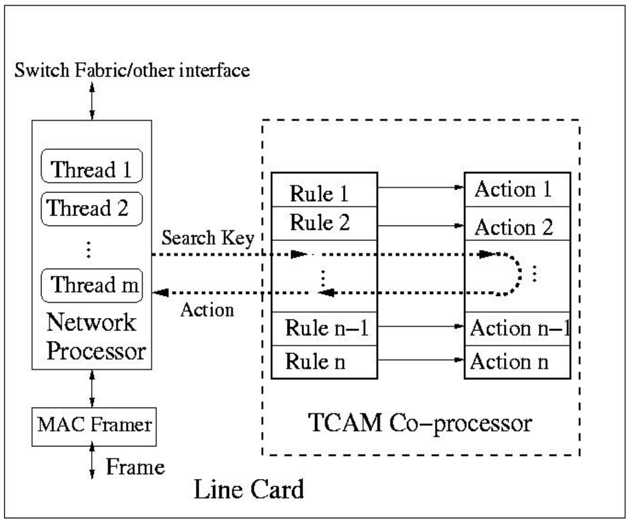


Fig 1. A network processor and its TCAM coprocessor in a line card

The other type of TCAM coprocessors has a weight sub-field of fixed size of *w* bits assigned to each and every memory slot. Here a memory slot is defined as a memory entry of size equal to the memory width. By properly assigning weight values to all the rules, each occupying certain integer number of memory slots, one can expect that each search key matching would result in a matched rule with the highest weight value among all the matched rules. We call this type of TCAM coprocessors the *WEIghted TCAM* (WEITCAM).

In principle, a WEITCAM is easier to manage than an OTCAM. For a WEITCAM, rules can be placed at any memory location as long as distinct weights are assigned to all the rules with match priority relationship.

Moreover, WEITCAMs allow the so-called zero table management for longest prefix matching (LPM) [2]. By assigning a weight value equal to the prefix length for each route, a new route can be added at any available memory address and a route can be deleted without the need to update the weights or re-arrange the existing routes in the memory. This means that w equal 5 is sufficient to support zero table management for IPv4-based LPM. In contrast, for OTCAM, an optimal algorithm in terms of the worst-case performance, as proposed in [3], requires up to 16 rule entry moves when a route is to be added or deleted from an LPM table.

Despite the merits mentioned above, a WEITCAM may suffer from **weight depletion**. Namely, no more than 2^w rules with match priority relationship can be enforced. Weight depletion is a critical issue for WEITCAMs simply because new rules may not be enforced due to weight depletion, even if the TCAM coprocessor has enough memory resource to accommodate those rules. Unfortunately, commercially available WEITCAM coprocessors generally have limited number of bits assigned to the weight sub-field, e.g., 7 bits for AMCC nPC2110 [4] and 10 bits for Netlogic CFP3256 [5], supporting up to 128 and 1024 weight values, respectively. As we shall see shortly, the number of rules with priority relationships can be potentially very large, causing weight depletion.

In view of the criticality for solving the weight depletion issue for WEITCAMs, in this paper, we propose a rule grouping technique to deal with the issue. We are able to show that with this technique, virtually unlimited number of rules with arbitrary rule structures can be enforced, at the expense of a second TCAM lookup for oversized rule groups with match priority relationship. The technique can be readily implemented in a fully programmable network processor and the associated TCAM coprocessor without additional hardware support.

The rest of the paper is organized as follows. Section 2 identifies the weight depletion problem. Section 3 describes the rule grouping technique to solve the problem. Finally, Section 4 concludes the paper.

2. Weight Depletion Problem

To facilitate the discussion in this as well as the following sections, we first define two terms, i.e., *Connected Rule Group* (CRG) and *Multiple Match Group* (MMG). A CRG is a set of rules which form a connected graph. A rule table is composed of one or multiple CRGs. A CRG is further composed of one or multiple MMGs. Here an MMG is a group of rules that have match priority relationship. The following example, which will be reused throughout the rest of the paper, illustrates how CRGs and MMGs are identified.

Assume there is a policy filtering (PF) table composed of 14 rules: {A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10}, A_{11}, A_{12}, A_{13}, A_{14}}. The rule structure is depicted in Fig. 2. The area each rule covers is represented by a solid line. The integer numbers at different levels represent different match priorities or weight values. An arrow connects two rules overlapping with each other and it points from the lower priority rule with a smaller weight value towards the higher priority rule with a higher weight value.

An MMG is composed of all the rules along an arrow chain. Hence, three MMGs can be identified from Fig. 3. They are:

MMG-1 = {A_1, A_9, A_{11}, A_{12}, A_{13}};
MMG-2 = {A_2, A_3, A_5, A_6, A_7, A_8, A_{10}, A_{14}}
MMG-3 = {A_2, A_3, A_4, A_7, A_8, A_{10}, A_{14}}

The rules in MMG-1 are chained together. MMG-2 and MMG-3 share most of the rules. By definition, MMG-1 itself forms a CRG and MMG-2 and MMG-3 form another one.

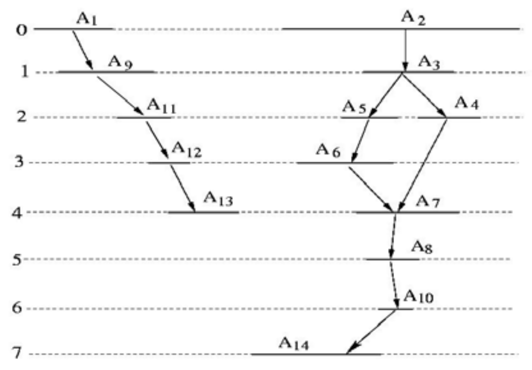


Fig. 2 An example PF rule structure

The above example shows that a rule can belong to one or multiple MMGs in a CRG. Obviously, *different rules in an MMG need to be assigned distinct weight values.*

Now, we can make the following two important observations: (1) **The weight assignment can be localized to individual CRG, i.e., the whole weight space can be reused by different CRGs;** (2) **In each CRG, the required number of weight values is equal to the size of the largest MMG in that CRG.** These two observations immediately lead to the following conclusion: **Weight depletion occurs if at least one CRG has an oversized MMG in it, i.e., the size of the MMG exceeds the weight space size.**

The question to be answered is then how large an MMG can be for a general PF rule table, e.g., rules in the form of 104-bit five-tuples: *{source IP address, destination IP address, Source Port, Destination Port, Protocol Number}*. In other words, we want to know whether or not the WEITCAM weight space size, e.g., 128, is big enough to support the largest possible MMG in practice. The following example attempts to answer this question.

In general, any bit in a rule can take on any of the three possible values, i.e., 0, 1, or x, where 'x' represents a wildcarded or "don't care" bit. Fig. 3 shows all possible candidate rules and the rule structure, generated from just three bits. A solid line links between a rule and one of its subset rules and a dotted line connects two rules, which intersect with each other.

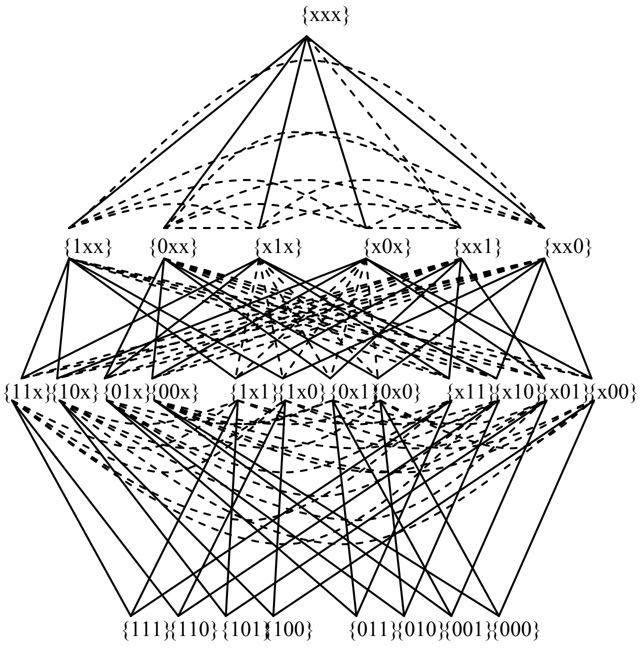


Fig. 3 A complete diagram for 3-bit candidate rules and the rule structure

First of all, the all-wildcarded candidate rule {xxx} is a superset of all the other candidate rules. It is a default match rule and is solely used as a reference point to build the rule structure. Hence, it does not appear in any PF table. If a search key does not find a match in the PF table, it implies that the search key matches this rule.

All kinds of MMGs can be built by picking up rules with increasing match priority in Fig. 3, along the solid lines *downward* and dotted lines *in either direction*. The rules can be picked up only along a solid line in downward direction because a rule, which is a subset of the other rule, must have higher match priority than that rule. On the other hand, rules can be picked up in either direction along a dotted line because either of the two intersecting rules may have higher match priority than the other one.

Allowing rules to be picked up along dotted lines is the root causing weight depletion. For example, let's first restrict ourselves by picking up rules only along solid lines downward. Obviously, by doing so, one can have up to only 3 rules in an MMG. In other words, the number of rules in an MMG cannot exceed the rule length, i.e., the number of bits in a rule. This is exactly why the zero table management for an LPM table using WEITCAM is possible.

Now, if we remove the above constraint and allow rules to be picked up along both solid lines and dotted lines, we can see the number of rules in an MMG can then be very large. This is because the rules not only can be picked up in downward direction but also in horizontal and upward directions. For example, rules can be picked up starting from {1xx} to {x11}, and then from {x11} back to {0xx}, and so forth.

Theoretically, we have the following results:

$$N_k = 2^k n!/k!(n-k)!\ ,$$

where N_k is the number of candidate rules with k number of unwildcarded bits and n is the length of the rule in the unit of bits. The summation of N_k, for k =0, 1, 2, …, n, gives the total number of possible candidate rules, which equals 3^n. As an example, for n = 32, in principle, an MMG of size equal to N_k, where N_k = 3,141,447,188,480, at k = 10 alone, can be built. It becomes clear that for general PF tables, e.g., 104-bit five-tuple PF table, the size of an MMG that one can build is virtually unbounded.

Now, let's look at a more practical example on how a large MMG can be formed by simply adding one rule. Fig. 4 (a) gives a policy table with 104-bit five-tuple rules. For all the rules, only the source IP address is specified and all other fields are wildcarded. Two CRGs with one MMG in each, i.e., ***B*** and ***C***, are identified. Hence, the same weight values can be assigned to rules in ***B*** and ***C*** as shown in Fig. 4 (a). Now, suppose a rule *D,* which only specifies the protocol field and all other fields are wildcarded, is added to the policy table as shown in

Fig. 4 (b). Note that rule *D* partially overlaps with all the rules in both ***B*** and ***C***. Assume rule *D* has higher priority than B_4 and lower priority than C_1. As a result, each rule in ***C*** must have a weight value larger than that of *D* and each rule in ***B*** must have a weight value smaller than that of *D*. Hence, adding rule *D* leads to the merge of ***B*** and ***C*** into a larger MMG. This example demonstrates that large MMGs can be formed as a result of introducing partially overlapping rules, even though the number of rules that can match with a search key simultaneously may be small.

	Rule	Weight
B_3	2.2.2.x—x.x.x.x—x.x—x.x—x	2
	Empty slot	
C_4	1.1.1.1—x.x.x.x—x.x—x.x—x	3
C_3	1.1.1.x—x.x.x.x—x.x—x.x—x	2
B_2	2.2.x.x—x.x.x.x—x.x—x.x—x	1
C_2	1.1.x.x—x.x.x.x—x.x—x.x—x	1
C_1	1.x.x.x—x.x.x.x—x.x—x.x—x	0
B_4	2.2.2.2—x.x.x.x—x.x—x.x—x	3
B_1	2.x.x.x—x.x.x.x—x.x—x.x—x	0
	Empty slot	

(a)

	Rule	Weight
B_3	2.2.2.x—x.x.x.x—x.x—x.x—x	2
D	x.x.x.x—x.x.x.x—x.x—x.x—6	4
C_4	1.1.1.1—x.x.x.x—x.x—x.x—x	8
C_3	1.1.1.x—x.x.x.x—x.x—x.x—x	7
B_2	2.2.x.x—x.x.x.x—x.x—x.x—x	1
C_2	1.1.x.x—x.x.x.x—x.x—x.x—x	6
C_1	1.x.x.x—x.x.x.x—x.x—x.x—x	5
B_4	2.2.2.2—x.x.x.x—x.x—x.x—x	3
B_1	2.x.x.x—x.x.x.x—x.x—x.x—x	0
	Empty slot	

(b)

Fig. 4 An example of MMG merging as a result of adding a new rule

Although no statistics were reported on the MMG size for the existing PF tables, the statistics on the number of partially overlapping rules for ISP's access control lists (ACLs) did suggest that the MMG size could be large even for a rather small ACL. For example, [6] showed that for an IPv4 source-destination-pair based ACL with only 607 rules, the observed number of partial overlaps is 2,249. This implies that potentially a large number of rules can be chained together to form a large MMG. Even for prefix based PF tables, the weight depletion problem can still occur. For example, for an IPv6 source-destination-pair based ACL, in principle, the maximum size of an MMG is equal to the rule length, i.e., 256, much larger than 128, the weight space size for one of the existing WEITCAMs.

3. A Rule Grouping Technique

Contrary to requiring that a uniquely matched rule need to be returned in the presence of multiple rule matches, this technique is based on the very idea of allowing multiple simultaneous matches of rules with the same matched weight value. It is based on the following property of the WEITCAM:

When a search key finds matches with more than one rule with the same highest weight value, the action associated with one of these matched rules is returned. However, exactly which one of the actions is returned cannot be determined.

The above property leads to a technique called *rule grouping technique*. The idea is to sequentially break the rules in an oversized MMG into up to 2^w segments with each segment having no more than 2^w number of rules and to assign the *same weight value* to all the rules in a segment but distinct, increasing weight values for rules in different segments with increasing match priorities. Hence, a match of a segment implies that the best matching rule must be one of the rules in that segment. This best matching rule is found through a second rule table matching, which is composed of all the segments containing more than one rule. Since a rule can belong to multiple MMGs in a CRG, the rule grouping needs to be done starting with the largest oversized MMG, and then the next largest, and so on, to avoid any possible conflicting assignments of rules common to multiple MMGs in a CRG.

The best way to explain the details of this technique is through an example. Consider the example in Fig. 2. Let us assume that $w = 2$, i.e., the weight space size is 4. In this case, all three MMGs are oversized. In what follows, we explain how the rule grouping technique can be used to solve the weight depletion for this PF table.

Fig. 5 (a) shows how rules are grouped into segments to form the first rule table. For MMG-1, four segments are created. They are $\{A_1\}$, $\{A_9\}$, $\{A_{11}\}$, and $\{A_{12}, A_{13}\}$. Since MMG-2 and MMG-3 belong to the same CRG and the size of MMG-2 is larger than that of MMG-3, we first group rules for MMG-2. As can be seen in Fig. 5 (a), four segments are created. They are {A_2}, {A_3}, {A_5, A_6} and {A_7, A_8, A_{10}, A_{14}}. With this assignment for MMG-2, all the rules in MMG-3 have assigned weight values except A_4. A_4 can be assigned either weight value 1 or 2. In the former case, we create a segment with two rules, i.e., {A_3, A_4}, which has to appear in the second rule table. In the latter case, however, we just create a segment with one rule, i.e., A_4 itself, and it does not appear in the second rule table. Hence we assign weight value 2 to A_4 in MMG-3.

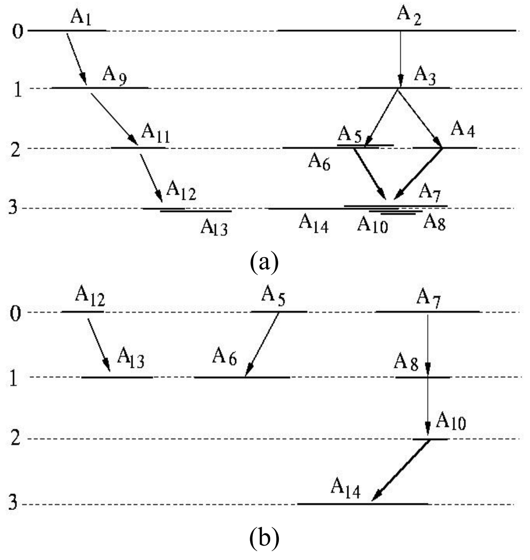
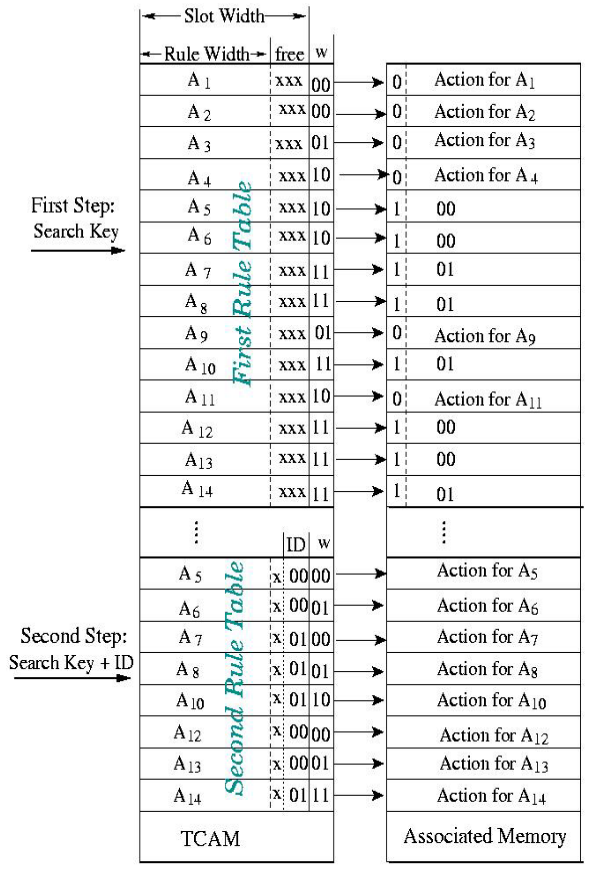


Fig. 5 Resulting rule structures as a result of the application of the rule grouping technique to the rule structure in Fig. 2

The second rule structure then stretches all the compressed rule segments {A_{12}, A_{13}}, {A_5, A_6} and {A_7, A_8, A_{10}, A_{14}} in Fig. 5 (a) back to one rule per level, as shown in Fig. 5 (b), corresponding to the second rule table. Note that all the segments in Fig. 5 (b) are allowed to use the whole weight space. This works for segments from different CRGs, such as {A_{12}, A_{13}} and {A_5, A_6}, since rules from different CRGs have no priority relationship. However, this may not work for segments from the same CRG, e.g., {A_5, A_6} and {A_7, A_8, A_{10}, A_{14}}. A key design of this technique is then to use a *segment ID* to append to each rule in the second rule table. Also a match of a rule in a segment with more than one rule in the first rule table returns the segment ID the matched rule is associated with, which is used as part of the search key for the second table match. This allows the weight space to be locally significant for individual segments.

Fig. 6 shows the implementation of the above example in a WEITCAM coprocessor. Note that, in general, up to 2^w segments in an oversized MMG can be supported and hence a segment ID field size of *w* bits may be required to uniquely identify each segment. Since w = 2, the segment ID field size is also set to 2. In this example, we assume that the rule length is 3 bits short of the TCAM slot width. Hence, there are 3 bits at our disposal for free. We assign the last 2 of the 3 bits as the segment ID field for each rule in the second rule table. In general, however, if the number of available free bits is not large enough to accommodate the segment ID field, an extra slot must be allocated for that field. The last 3 bits and the 3rd last bit in each rule field in the first and second tables, respectively, are not used and hence are wildcarded.



Slot Width
First Step: Search Key
First Rule Table
Second Step: Search Key + ID
Second Rule Table
TCAM
Associated Memory

Rule Width	free	w		Associated Memory
A1	xxx	00	0	Action for A1
A2	xxx	00	0	Action for A2
A3	xxx	01	0	Action for A3
A4	xxx	10	0	Action for A4
A5	xxx	10	1	00
A6	xxx	10	1	00
A7	xxx	11	1	01
A8	xxx	11	1	01
A9	xxx	01	0	Action for A9
A10	xxx	11	1	01
A11	xxx	10	0	Action for A11
A12	xxx	11	1	00
A13	xxx	11	1	00
A14	xxx	11	1	01
⋮				⋮

		ID	w	
A5	x	00	00	Action for A5
A6	x	00	01	Action for A6
A7	x	01	00	Action for A7
A8	x	01	01	Action for A8
A10	x	01	10	Action for A10
A12	x	00	00	Action for A12
A13	x	00	01	Action for A13
A14	x	01	11	Action for A14

Fig. 6 Table formats in the WEITCAM for the rule structures in Fig. 5

In this example, rules in segments {A_{12}, A_{13}}, {A_5, A_6} and {A_7, A_8, A_{10}, A_{14}} are assigned segment ID values 0, 0, 1, respectively. Also note that in each action field associated with the first rule table, the first bit is used to indicate whether a second table match is needed. If it is, i.e., the bit set to 1, instead of an action, a segment ID is given in the associated memory.

The procedure for a rule table matching is as follows. The network processor generates a search key and passes the search key to the WEITCAM to match against the first rule table. Upon receiving the returned result from the WEITCAM, the network processor first checks the first bit in the returned result. If it is 0, the network processor starts to process the action that follows it. Otherwise, it appends the segment ID following that bit to the search key, and passes it to the WEITCAM to match against the second rule table to find a final match.

The rule grouping technique can allow the support of up to 2^{2w} rules in an MMG, at the expense of the possible need for a second WEITCAM search and the memory space expansion to accommodate the second rule table. For instance, with this technique, an MMG with size as large as 16,384 can be supported for a WEITCAM with $w = 7$. Obviously, this technique can be easily generalized to allow the support of 2^{kw} rules in an MMG, at the expense of k WEITCAM searches in the worst case and the adding of k more rule tables. This scaling feature allows virtually unlimited number of rules in an MMG to be supported. In practice, however, $k = 2$ should be more than enough to eliminate weight depletion problem, given that today's PF table sizes are smaller than 10 K and $w \geq 7$ for all the existing WEITCAMs.

Finally, note that the rule grouping technique takes effect if and only if there is at least one oversized MMG in the PF table. In other words, if all the MMGs in the PF table are not oversized, applying the rule grouping technique to the PF table does not result in the adding of a second rule table and a second rule table matching.

4. Conclusions

In this paper, the weight depletion problem for WEITCAM coprocessors was identified. A rule grouping technique was proposed to solve the weight depletion problem. The technique allows virtually unlimited number of rules to be enforced. The implementation of the technique requires no specially hardware support and can be readily implemented in a fully programmable network processor and its WEITCAM coprocessor.

References

[1] P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE Network*, March 2001.

[2]http://www.netlogicmicro.com/library/zerotable.html

[3] D. Shah and P. Gupta, "Fast Updating Algorithms for TCAMs," *IEEE Micro*, p. 36, Jan.-Feb 2001.

[4] http://www.amcc.com

[5]http://www.netlogicmicro.com/datasheets/cfp3256.html

[6] M. E. Kounavis, A. Kumar, H. Vin, R. Yavatkar and A. T. Campbell, "Directions in Packet Classification for Network Processors", *Second Workshop on Network Processors (NP2),* February 8-9, 2003.