

IP Packet Forwarding Based on Comb Extraction Scheme

Zhen Xu^[1], Ioannis Lambadaris^[2], Yiqiang Q. Zhao^[1], and Gerard Damm^[3]

^[1] School of Mathematics and Statistics, Carleton University, Ottawa, Ontario, Canada, {zxu, zhao}@math.carleton.ca

^[2] Department of System and Engineering, Carleton University, Ottawa, Ontario, Canada, ioannis@sce.carleton.ca

^[3] Alcatel, Ottawa, Ontario, Canada, Gerard.Damm@alcatel.com

Abstract-In this paper, we present an efficient IP packet forwarding technique and its architecture. One forwarding table is decomposed into two balanced smaller sub-forwarding tables by a novel splitting rule. Therefore, an IP lookup can be converted into two small sub-lookups. After comparing the information attached by matching sub-prefixes from both sub-lookups, the output of an incoming packet can be determined. The sub-lookups and information comparison can perform in parallel, so our approach speeds up the Best Matching Prefix search effectively and it can reduce storage space at the same time.

Introduction

The major function of a router is to forward packets. Since the Internet traffic is increasing rapidly, speeding up the link rate is required in order to provide good service [1]. It is difficult to make the performance of a router keep up with this increasing demand. In particular, the address lookup operation is a major problem.

Many lookup algorithms create a data structure that takes advantage of the binary search tree method, which is among the mature search algorithm [2]. The binary trie[3] method and its variations including Patricia trie[4], multibit trie[5] and LC trie[6] have been presented in the literature. Some heuristic approaches were designed to facilitate the use of binary trees [2], such as search on prefix range [7,8] and search on prefix length [9]. Unfortunately, these approaches usually suffer from large storage requirement or poor updating features. In addition, some hardware-based solutions are proposed by using large DRAM

for entire forwarding table [10]. Using CAM is also presented in [11]. A good survey of these methods can be found in [12].

The main contribution of this paper is two-fold. First, a forwarding table is decomposed into two balanced sub-tables by using the Comb Extraction Scheme (CES). It can parallelize two independent search processes. In order to determine which output port the packet should be sent to finally, the comparison of information pairs, attached by those matching sub-prefixes in both sub-tables, needs to be executed at the end. Some good characteristics of the two sub-tables result from CES, as compared to the original forwarding table. For example, it can reduce storage space, speed up search time, alleviate distribution dependent problems, and minimize information comparison load. Secondly, we propose an efficient architecture to realize this methodology. The flexibility of this architecture allows IP address lookup to be easily integrated within routing SoCs and generic network packet processing units.

In this paper, we focus on the unicast (single-source, single-destination) routing of backbone routers. In an IP forwarding table, an entry usually has this structure: $\langle DesPrx, PrxLen, ForInf \rangle$. *DesPrx*, Destination Prefix, represents the network part of an IP address, *PrxLen* stands for the length of the prefix, and *ForInf*, Forwarding Information, is usually a next hop address or an output port number, respectively. Suppose

Index	Prefix	Len	Port
1	11000110100110*	14	1
2	11000110011*	11	1
3	11000110010*	11	1
4	1101011100001011*	16	1
1	1100011001*	10	2
2	011110000011*	12	2
3	11000110100*	11	2
4	10001010110011*	14	2
5	100010101011*	12	2
6	1110111101*	10	2
7	01111000101*	11	2
8	0111100011100*	13	2
1	1100011010011010*	16	3
2	1100011001110*	13	3
3	1000101011001*	13	3
4	01111000001110*	14	3

Table 1: A Sample Forwarding Table

$P = \{p_1, p_2, \dots, p_M\}$ is the set of M prefixes recognized by a router. Let's assume there are N outputs. When examining the forwarding tables we can get from [13] carefully, we can find that the number of distinct

next hops in a routing table is very small, comparing with the tens of thousands of prefixes. It is shown clearly in sample Table 1. There are $M=16$ prefixes and only $N=3$ output ports. All the entries are sorted in terms of ports, and the index is a number based on the same port.

New Data Structure

For IPv4, an IP address A is 32-bit long. It can be decomposed into two 16-bit long sub-sequences by the following strategy: From the left-most bit to the right-most bit, all the bits in odd positions are extracted to form sub-sequence a , and all the bits in even positions are extracted to form sub-sequence b . We call this splitting approach the Comb Extraction Scheme (CES). For example, let us consider the IP address in binary bits, **10100001 00110110 11010000 11101001**, in which bits in odd positions are bold. After the decomposition, a and b will be 11000101 10001110 and 00010110 11001001, respectively.

Similarly, a prefix also can be decomposed into two sub-sequences a and b . Both of them end by *. That

Index	Sub-Prefix	Len	Port Indicator	Forwarding Information
1	011001*	6	010	2(2)
2	0110011*	7	001	3(4)
3	011011*	6	010	2(7)
4	0110110*	7	010	2(8)
5	10010*	5	010	2(1)
6	100100*	6	100	1(3)
7	10010011*	8	100	1(4)
8	100101*	6	100	1(2)
9	1001010*	7	001	3(2)
10	100110*	6	010	2(3)
11	1001101*	7	100	1(1)
12	10011011*	8	001	3(1)
13	1011101*	7	011	2(4), 3(3)
14	101111*	6	010	2(5)
15	11110*	5	010	2(6)

Table 2: Sub-forwarding table 1 from Table 1 (extracting bits in odd positions)

Index	Sub-Prefix	Len	Port Indicator	Forwarding Information
1	000001*	6	010	2(5)
2	000010*	6	001	3(3)
3	0000101*	7	010	2(4)
4	10100*	5	010	2(3)
5	1010010*	7	100	1(1)
6	10100100*	8	001	3(1)
7	10101*	5	110	2(1), 1(2), 1(3)
8	101011*	6	010	3(2)
9	10111*	5	010	2(6)
10	11000*	5	010	2(7)
11	110001*	6	010	2(2)
12	1100010*	7	001	3(4)
13	110010*	6	010	2(8)
14	11110001*	8	100	1(4)

Table 3: Sub-forwarding table 2 from Table 1 (extracting bits in even positions)

is, they are two sub-prefixes. For example, let us consider the prefix **01101100 101***, the length of which is 11. After decomposition, a will be 011011*, the length of which is 6, and b will be 10100*, the length of which is 5.

Hence, a forwarding table can be converted into two extended sub-forwarding tables. Table 2 and Table 3 are the two examples of sub-forwarding tables when Table 1 is decomposed. Each sub-entry has this kind of structure as $\langle \text{sub-prefix, length, port-indicator, forwarding information} \rangle$.

In the **Forwarding Information** part of Table 2 and Table 3, not only is the information about the port number noted, but also the information about the corresponding index associated with that port in Table 1 is involved. The Forwarding Information is composed of the **forwarding unit** $a(b)$, which implies that, in Table 1, the original prefix of this sub-prefix is forwarded to port a , and the associated index is b . In general, the forwarding information of each sub-entry in a sub-table consists of several forwarding units. For example, the sub-prefix of the 7th entry in Table 3 is 10101*, which collects the information of all original prefixes whose bits in even positions are 10101*. It is made up of three forwarding units, 2(1), 1(2), and 1(3). The forwarding unit is the union of the port and the corresponding index. Usually, a core router has no more than 128 output ports. So the length of port can satisfy that $\text{len}(\text{port}) \leq 7$ in bits. Therefore, a 20-bit long vector is enough to represent a forwarding unit, in which $\text{len}(\text{index}) = 20 - \text{len}(\text{port})$.

In each sub-table, a N -bit **port indicator vector** is associated with every sub-entry. A bit i is set in the bit vector if and only if the i^{th} port occurs in its forwarding information. Usually the width of it is no more than 128. The total storage cost for the extra information is shown in last column in Table 4.

What is the benefit of the CES approach? We examine it from two main aspects. (1), since a forwarding table is decomposed into two sub-tables, one lookup will be divided into two parallel sub-lookups. Can CES make the two sub-lookups in balance, either in time access or memory consumption? (2), after the two parallel sub-lookups, some sub-prefixes will match each sub-search key in both sub-lookups. In order to find the BMP, we need to combine the results, comparing the information of any reasonable pair of matching sub-prefixes from both sub-lookups. Therefore, can CES cause heavy comparison loads, which need to cost extra time?

First, CES makes the entries of two sub-tables well distributed.

In comparing two bit patterns, the Hamming distance is the count of bits different in the two patterns. More generally, if two ordered lists of items are compared, the Hamming distance is the number of items that do not identically agree. Here, we give a new definition to determine the distance between two prefixes, which is similar to the Hamming distance.

Definition 1: a and b are two prefixes in one table. $|a|$ and $|b|$ represent their lengths. Let $ML = \min(|a|, |b|)$. We define **Pseudo-Hamming Distance** (PHD) between two prefixes as:

$$PHD(a, b) = \sum_{i=0}^{ML} (L - i) |a_i - b_i|, \text{ where } a_i \text{ and } b_i \text{ are the bits in the } i^{th} \text{ position, from the left-most, of } a$$

and b , L is the maximum length of sequences (In IPv4, L in the original forwarding table is 32, 16 for sub-forwarding tables). $MPHD$ is the mean of PHD of any two different prefixes in one table. PHD is affected not only by the number of bits that are not identical, but also by their positions, the left-most bits having higher weight.

For example, assuming a , b and c are 011001*, 0110010*, and 10010011* respectively. Let L be 16. $PHD(a, b) = 0$, $PHD(a, c) = 69$, and $PHD(b, c) = 79$.

Lemma: If a and b are two prefixes, and one is a prefix of the other, then $PHD(a, b)$ equals to zero.

The value of $MPHD$ can stand for the distribution of entries in a table. If $MPHD$ is a big value, it implies that, in a trie of a forwarding table, nodes spread widely, rather than just focus on several deep branches. This allows for a faster search. CES is almost the best of splitting rules to maximize the $MPHD$ of each sub-table, and there is not much divergence between the two values, which shows that CES leads to a balanced distribution of entries in the two sub-tables.

Secondly, CES also balances the sub-prefix lengths in the two sub-tables.

Definition 2: Let **Mean Prefix Length** (MPL) in any sub-table expressed by: $\frac{1}{M} \sum_{i=0}^{SM} n_{IFO}^i Len_i$, where M is the number of entries in the original forwarding table, SM is the number of sub-entries in this sub-table, n_{IFO}^i is the number of forwarding units in the i^{th} sub-entry, and Len_i is the length of the i^{th} sub-entry.

Let a be an original prefix in a forwarding table. After having been decomposed, it will be converted into two sub-prefixes, named a_1 and a_2 . By construction, the difference between the lengths of a_1 and a_2 satisfies the inequality: $0 \leq |a_1| - |a_2| \leq 1$. In fact, the difference between $MPLs$ in CES results from the number of original prefixes, whose lengths are odd values. It is a good way to prevent big difference in searching time by making the sub-prefixes in two sub-tables almost the same.

In our example, $MPL_1 = 6.5$, $MPL_2 = 6.06$, and In short, CES is an efficient way to enable the two sub-tables keep in pace.

Thirdly, CES makes the forwarding units well distributed in each sub-table.

Definition 3:

(1) **Basic load of Forwarding Information (BLFI)** of i^{th} sub-entry in each sub-table is defined as the total number of forwarding units in the i^{th} sub-entry.

(2) **Mean load of Forwarding Information (MLFI)** of sub-entries in each sub-table is defined by

$$\frac{1}{SN} \sum_{i=1}^{SN} BLFI_i, \text{ where } SN \text{ is the total number of sub-entries in this sub-table.}$$

(3) **Standard Deviation of Forwarding Information (SDFI)** of sub-entries in each sub-table is defined by

$$\sqrt{\frac{1}{SN} \sum_{i=1}^{SN} (BLFI_i - MLFI)^2}.$$

These metrics are significant to show the performance of CES. Whether the comparing time between sub-prefixes in the second phase is reasonable or not depends on these three values. In our small example, $MLFI_1 = 1.067$, $MLFI_2 = 1.143$, $SDFI_1 = 0.25$, and $SDFI_2 = 0.51$.

Fourth, CES balances the comparison cost.

Definition 4: The **comparison cost factor (CCF)** is used to judge whether the comparison load of those matching sub-prefixes in two sub-tables for an address lookup next is heavy or not. CCF is a statistical value from experiments, by counting the pairs really need to compare.

Actually, it is not necessary to compare every pair of matching sub-prefixes, for there are constraints among the matching sub-ones, once they are the final ones we are looking for. We know that if a_1 and a_2 are two the final matching sub-prefixes in the two-tables for an address, then they should satisfy the following:

(1) $|a_2|$ only can be equal to $|a_1|$ or $|a_1|-1$;

(2) In the two corresponding port indicator vectors, $\exists i, i < N, \text{PIV}_i^1 = \text{PIV}_i^2 = 1$. (PIV is the port indicator vector).

Only if the matching sub-prefixes a_1 and a_2 , which come from different sub-tables, meet the demands above, comparison is needed. CCF is a parameter to observe the number of pairs, which satisfy the conditions, and need to do real comparison. Anyway, CCF has its upper bound. Let $\text{MinNum} = \min(\text{Num}_1, \text{Num}_2)$, where Num_1 and Num_2 are the numbers of matching sub-prefixes from the two sub-tables. $\text{CCF} \leq 2 \times \text{MinNum}$.

	Entries		Sub-entries	MPHD	MPL	Max(BLFI)	MLFI	SDFI	CCF	Storage Cost (in Byte)
Mae-east	47206	Sub-table 1	4026	55.22	11.18	93	11.73	13.41	8	186.06K
		Sub-table 2	5341	56.56	11.18	86	8.84	9.71		209.15K
Mae-west	77002	Sub-table 1	5703	56.47	11.22	100	13.05	15.49	8	270.81K
		Sub-table 2	6989	57.84	11.22	78	11.02	12.57		241.95K
Aads	63980	Sub-table 1	5689	56.80	11.35	110	11.25	14.28	8	245.14K
		Sub-table 2	6735	57.45	11.35	89	9.50	10.84		261.44K
Paix	22116	Sub-table 1	4077	54.59	11.15	40	5.42	5.35	7	117.65K
		Sub-table 2	4704	55.67	11.15	28	4.70	4.23		127.48K

Table 4: Performance of sub-tables by using the CES

	Entries		Sub-entries	MPHD	MPL	Max(BLFI)	MLFI	SDFI
Mae-east	47206	Sub-table 1	6939	59.85	16.00	280	6.80	13.63
		Sub-table 2	1349	43.24	6.47	2735	34.99	93.62
Mae-west	77002	Sub-table 1	10794	23.32	16.00	253	7.13	15.33
		Sub-table 2	1692	51.63	4.79	5939	45.50	164.45
Aads	63980	Sub-table 1	8314	61.54	16.00	465	7.69	16.65
		Sub-table 2	3540	49.03	9.94	3385	18.07	73.38
Paix	22116	Sub-table 1	4540	59.68	16.00	128	4.87	7.98
		Sub-table 2	1238	48.85	5.03	1406	17.86	49.95

Table 5: Performance of sub-tables by such a splitting rule: extracting the higher 16 bits to form sub-table 1 and extracting the lower 16 bits to form sub-table 2

Table 4 and Table 5 give us the performances of sub-tables by using different splitting rules respectively. It is clear that the CES is much better than the other one (extracting the higher 16 bits to form sub-table 1 and extracting the lower 16 bits to form sub-table 2).

Comparison Set

In this section, we are going to analyze the matching sub-prefixes from two sub-tables, in order to find the common matching prefix. This part can be implemented in an ASIC.

The first step is to decide whether further comparing is necessary, which is pointed out before.

The second step is to compare the forwarding units, only when the first step succeeds.

If $P1_i$ and $P2_j$ are two matching sub-prefixes coming from each sub-table, each of them contains one forwarding information set, which is a collection of forwarding units. We need to compare every unit in a set with the units in another set. Let $Info1_i$ and $Info2_j$ be the information sets of $P1_i$ and $P2_j$, which are composed of $M1$ and $M2$ information units. So, for each comparison, $M1 \times M2$ pairs of comparison units are needed.

In the comparison between $Info1_i$ and $Info2_j$, if there exists an exact match in one comparison unit, it implicates that $P1_i$ and $P2_j$ are the right decomposition parts of an original prefix in a forwarding table.

Lemma: In the comparison between $Info1_i$ and $Info2_j$, there at most exists one exact match in all pairs of comparison units.

Proof: Assume that there exists two pairs of units, $(Info1_{i,k}, Info2_{j,m})$ and $(Info1_{i,l}, Info2_{j,n})$, match exactly. That is, $Info1_{i,k} = Info2_{j,m}$, and $Info1_{i,l} = Info2_{j,n}$. It means that in the original forwarding table, there are two entries, which have the same prefix, but will be forwarded to different ports. It is impossible

	Entries	Average delay (ns)	Delay(80% of comparisons) (ns)
Mae-east	47206	2.24	<4.59
Mae-west	77002	3.36	<7.87
Aads	63980	1.96	<4.72
Paix	22116	0.56	<1.21

Table 6 Cost for comparison/matching sub-prefix

for unicast. So the assumption is not right. Then there at most exists one exact match in all pairs of comparison units.

Since each forwarding unit is 20 bit long. Based on nowadays limitation of transforming width, it is possible to input 6 forwarding units of each matching sub-prefix at the same time. Then 36 comparison units can work in parallel. Therefore, all comparison units work in serial to the end until there is a comparison unit exact match. The delay of every 36 parallel comparison is 280ps, when using VLSI feature size of $l = 0.13mm$. Table 6 shows the time cost for comparing every forwarding unit of two matching sub-prefixes. We find that varies when the forwarding table's size increases. Actually, the real cost is smaller than this, since the comparison stops when there exists an exact match.

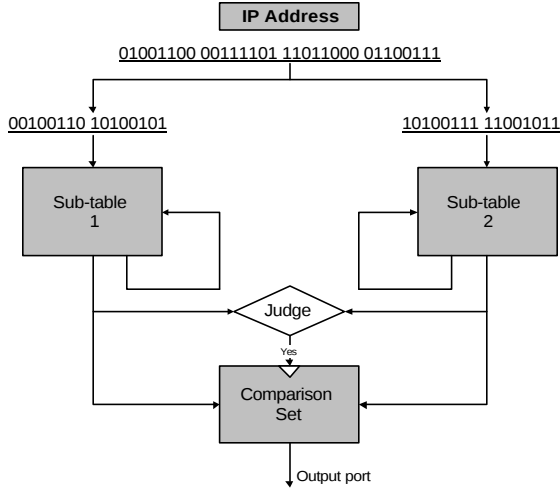


Figure 1: the architecture of the system

...	...	...	...	...	...
100110*	38912	6	010	NULL	A
...	...	6	010	NULL	A
...	...	6	010	NULL	A
1001101*	39424	7	100	38912	B
...	...	7	100	38912	B
10011011*	39680	8	001	39424	C
...	...	8	001	39424	C
10011011*	39935	8	001	39424	C
...	...	...	...	...	...
1011101*	47616	7	011	NULL	D
...	...	7	011	NULL	D
1011101*	48127	7	011	NULL	D
...	...	...	...	...	...

...	...
A	2(3)
B	1(1)
C	3(1)
...	...
D	2(4) 3(3)
...	...

Figure 2: Structure of CES + Index tables

Architecture of the New Algorithm

Figure 1 describes a rough picture on how this system works. We provide two structures based on CES.

1) CES + Index tables

The maximum length of entries is sharply reduced due to CES. The size of the array is 2^{16} for IPv4. Each entry of the array has the structure: {length[4], port-indicator[128], pParent[16], pInformation[16] }, in which, pParent is the pointer to its parent, the most specific prefix of it, and pInformation is the pointer to its forwarding information.

Each main index table consumes 1.28Mbytes, however the additional table for forwarding information is small (memory cost is shown in Table 4). The total memory consumption is about 3Mbytes. It is not scalable to IPv6, for the size of the index table is 2^{64} , which is still impossible for nowadays technology.

2) CES + Binary Trie

Binary trie is a basic structure in IP lookups. A forwarding table is decomposed into two half-level sub-tables. The storage cost for two 16-level tries is much smaller than one 32-level trie. Table 7 gives the memory cost when we use CES + Binary trie, smaller than only when binary trie used. Most memory is consumed at the nodes with forwarding information. The updating time is $O(\frac{W}{2})$, where W is the prefix length.

Storage Cost	Mae-east	Mae-west	Aads	Paix
Sub-table1	215.5K	310.2K	285.6K	147.8K
Sub-table2	247.2K	288.9K	308.3K	161.6K
Original Table	1295.3K	2003.8K	1657.8K	718.8K

Table 7 Storage cost comparison (CES+Binary trie vs Binary trie) (in Byte)

Different architecture of a sub-table will lead to a different search strategy.

If CES + Index table is used, when a search starts, the first sub-prefixes we reach in two sub-tables are the longest matching sub-prefixes. Not only is the forwarding information of both of them sent to the comparison set, but also they will point to their own most specific parent rows, and output another pair of forwarding information to compare. But now the lengths of sub-prefixes are shorter than the former one. Therefore once there is an exact match in the comparison set, the search stops. The mean loop in our experiments was 1.272, so the average of total delay in comparison is not more than 8ns (if the total entries are not more than 80K).

If CES + Binary tries is used, when a search starts, the first matching sub-prefixes we reach in two sub-tables are the shortest. We need to do the comparison of their forwarding information, and on the same time, we need to continue traversing the sub-tries until they are exhausted. The last exact match is the final output port of this IP packet. The total average delay in comparison is no more than 25ns, since the CCF is less than 8 (if the total entries are not more than 80K).

No matter which architecture we use, the comparison set works when both sub-lookups are preparing for the next pair of comparing sub-prefixes. From the experiment, we can see comparison set is fast enough not to be a speed bottleneck, if the forwarding table is not too big.

Conclusion

We proposed a new methodology and architecture for IP address lookup. Our approach advocates decomposing a forwarding table into two smaller sub-forwarding tables by using CES. We compare the reasonable matching sub-prefixes from the two sub-tables. Two sub-lookups and comparison can work in parallel, which provide a new way to speed up the average search time efficiently to handle OC-192.

But unfortunately, with the size of a forwarding table increasing, the forwarding units attached by a sub-prefix increases. If the both comparing sub-prefixes carry hundreds of forwarding units, the comparison delay will affect the performance of the whole system. CES cannot improve the performance in worst cases, but can make a big improvement for the average search time. There are lots of potentials to improve performance of comparison set, when the load is heavy. The authors will focus on solving this problem in the future.

Acknowledgement

The authors would like to acknowledge that this research is an initiative of Mathematics of Information Technology and Complex Systems, MITACS and the National Capital Institute of Telecommunications, NCIT in Collaboration with Alcatel's Research and Innovation Center in Ottawa, Canada.

Reference:

- [1] C. Labovitz, "Scalable of the Internet Backbone Routing Infrastructure," Ph.D. thesis, Univ. of Michigan, 1999.
- [2] M. J. Akhbarizadeh and M. Nourani, "An IP Packet Forwarding Technique Based on Partitioned Lookup Table," ICC'02, April, NYC, 2002.

- [3] K. Sklower, "A Tree-Based Packet Routing Table for Berkeley Unix," Proc. 1991 Winter Usenix Conf., 1991, pp. 93-99.
- [4] D. Morrison, "PATRICIA-Practical Algorithm to Receive Information Coded in Alphanumeric," Journal of ACM, Oct. 1968, vol. 15, no. 4, pp. 514-534.
- [5] V. Srinivasan and G. Varghese, "Faster IP Lookups Using Controlled Prefix Expansion," IEEE Trans. on Computer Systems, Feb. 1996, vol. 17, no. 1, pp. 1-40.
- [6] S. Nilsson and G. Karlsson "IP-Address Lookup Using LC-Tries," IEEE JSAC, June 1999, vol. 17, No. 6, pp. 1083-1092.
- [7] S. Suri, G. Varghese, and P. R. Warkhede, "Multiway Range Trees: Scalable IP Lookup with Fast Updates," Tech. Rep. 99-28, Washington Univ. 1999.
- [8] B. Lanpson, V. Srinivasan, and G. Varghese, "IP Lookups Using Multiway and Multicolumn Search," Proc. IEEE INFOCOM'98, Apr. 1998, pp. 1248-1256.
- [9] M. Waldvogel, G. Varghese, J. Turner, and B. Plather, "Scalable High Speed IP Routing Lookups," Proc. ACM SIGCOMM'97, Spet. 1997, pp. 25-36.
- [10] P. Gupta, S. Lin, and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," Proc. IEEE INFOCOM'98, Apr. 1998, pp. 1-11.
- [11] A. McAuley and P. Francis, "Fast Routing Table Lookup Using CAMs," IEEE INFOCOM'93, vol. 3. March 1993, pp. 1382-1391.
- [12] M. A. Ruiz-Sanchez, E. W. Biersack and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network* 15, 2 March / April 2001, pp. 8-23.
- [13] <http://www.merit.edu/ipma/>.
- [14] F. Baboescu, and G. Varghese, "Fast and Scalable Conflict Detection for Packet Classifiers," Proc. Of the 17th IEEE International Symposium on DFT'02,