Journal of Computer Science and Cybernetics, V.32, N.4 (2017), 319–333
DOI 10.15625/1813-9663/32/4/9731

A PACKET CLASSIFICATION ALGORITHM ON MULTI-WAY PRIORITY TRIE

VU DUY NHAT[1], NGUYEN MANH HUNG[2]

[1]*Information Technology Security, MoD of Vietnam, Ha Noi, VietNam*
[2]*Post-graduate Department, Military Technical Academy;* [1]*nhatbest@gmail.com*

Abstract. Packet classification is a vital function of network devices such as routers, firewalls, IPS, IDS, etc. Speed of packet classification is a key factor that decides to bandwidth of a network equipment. The field of packet classification speed enhancement has attracted a significant number of researchers. In this paper, we propose a packet classification algorithm based on the idea of priority trie and multi-way trie. The accuracy and efficiency of the proposed algorithm are both theoretically and experimentally proved.

Keywords. Firewall, packet classification, prefix, best matching prefix, rule.

1. INTRODUCTION

Today, with the development of science and technology, bandwidth of network systems has been significantly expanded. Network devices such as routers, firewalls, IPS, IDS are required to improve their bandwidths in order to not affect the operation of the network. The devices are deployed at the network core where a large number of packets are passed. This makes the need of packet classification speed enhancement become more urgent. Currently there are two research trends to improve packet classification speed: A research bases on hardware and an other bases on software.

The first trend - Ternary Content Addressable Memory (TCAM) - is a fairly common technique for storing and processing information with three states (0, 1 and *) [1, 2, 3]. TCAM is deployed on core network devices such as routers or switches. However, high cost, large power consumption and low flexibility are limitations for TCAM to be widely deployed. On recent days, a new branch of this trend has been emerged. Several authors have used GPUs for packet classification by taking advantage of GPU performance [4, 5, 6]. Similar to TCAM, the downside of this technique is the large power consumption and high heat is generated.

The second trend is based on data structures and searching operations on them. These techniques can be divided into many forms such as trie-based structure (Trie base), decision trie (Decision Trie) or hash-based (Hash base).

The trie based on algorithm requires O(NW) of memory storage and ($2W$-1) times of memory access per lookup (where N is the number of rules, W is the length of an IP address) [7]. Quad trie structure (AQT - Area based Quad Trie) [8] is recommended for 2-dimension rules. This proposal has reduced time of consumption for updating trie. In [9], trie priority is used to find the longest prefix match (BMP - Best matching prefix) in the minimum amount of time. In [10], JA-trie structure is built based on multi-bit trie structure [11] and the concept of entropy is used in the process of building trie to reduce the tries height. It results in reducing the number of memory accesses per lookup.

The algorithms which use decision trie such as Hi-Cuts [12], or Hyper-Cuts [13] are proposed to classify packets on 2-dimension. However, these techniques require a large memory for storing rules.

The hash-based algorithms [14, 15] are proposed to seek the longest prefix that matches with a specific input address. In [15], the authors propose the construction of hash tables with different lengths and searching order in each table is defined based on structure of binary search trie. Meanwhile, [16, 17] launched the idea of using the hash which is associated with bloom filter model for packets classification.

This paper proposes a packet classification algorithm based on a new data structure which is called Multi-Way Priority (MWP) trie. The MWP algorithm will return the longest prefix that matches an input address. Our algorithm is based on two key points. Firstly, in the MWP trie, the length of the prefix stored in a parent node is always greater than or equal to the length of prefixes that are stored in its child nodes (Priority [9]). Secondly, our algorithm uses multi-way trie data structure as the one in [10] but it differs from [10] on branching. The article is organized as following. Section 2 is an introduction of related algorithms such as Priority trie algorithm, JA-trie. Section 3 is the proposal of packet classification algorithm based Multi-Way Priority trie. Section 4 presents an experiment and evaluation of our proposed algorithm. Finally, section 5 is conclusions and recommendations for future work.

2. RELATED KNOWLEDGE

2.1. Priority trie

2.1.1. Introduce

Priority trie is proposed by Hyesook Lim, Changhoon Yim and Earl E. Swartzlander [9]. They have used modified binary trie structure to store and find out the longest prefix match with an input address. The main idea of priority trie is: In the binary searching trie, prefixes with greater lengths are located in higher position nodes compared to prefixes with shorter lengths. Consequently, the search process will end as soon as an appropriate prefix is identified without further search as conventional binary searching algorithms.

2.1.2. Data structures

The prefixes are sorted from largest to smallest according to their lengths. A prefix with the greatest length is stored in the root node and is marked as a priority node. At level j, depending on bit j^{th}, prefix P will be stored on the left or right node of the parent node and marked as a priority node: If the j^{th} bit of P is '1' then P is stored on the right node; otherwise, it will be stored on the left node. This process is repeated for all prefixes. Assuming that the length of P is n_i and P is stored in L level($L \leq n_i$), if $L = n_i$ then (P)-node is marked as a non-priority node.

For example, the prefixes set $\mathbf{P} = \{100, 110, 10011, 10010, 1000, 1111, 111, 101, 00, 01\}$, the priority trie is built as Figure 1.

In Figure 1, P_9, P_7, P_6 are not priority nodes.

2.1.3. Packet classification on the priority trie

With the proposed trie structure, the accuracy of finding the longest prefix that match an input address is demonstrated in Lemma [8]: Let $\mathbf{X}$ be a priority node in the level L, if an IP address matches P_m at node $\mathbf{X}$, then P_m is the longest prefix in the prefix set $\mathbf{P} = \{\ P_1, P_2, ..., P_N\ \}$ which

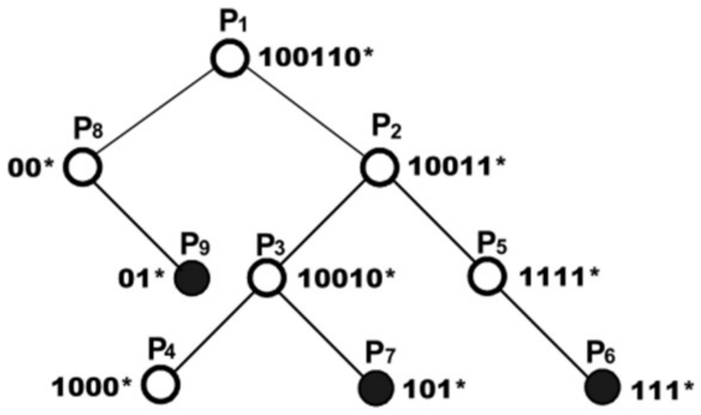

Figure 1. Example Priority trie

matches the IP address.

The searching process starts from the root node. At each node, the IP address is compared with the prefix in this node. According to the lemma proved, if the first n_i bits of the input IP address match the prefix P_i with length n_i at node **N** and **N** is the priority node, then P_i is the longest prefix which matches IP address and the searching process will end without looking at lower levels. This is a very important feature of the proposed algorithm. It reduces the number of memory accesses.

If an IP matches with a prefix in a non-priority node or does not match with the current node of level L, the bit $(L+1)^{\text{th}}$ of IP will be considered. If the bit $(L+1)^{\text{th}}$ is '0', the searching continues to the left node, and vice versa, the searching continues to the right node.

2.1.4. The weakness of priority trie

The main drawbacks of the priority trie [9] are as follows:

– In the worst case, the height of a priority trie is N, where N is the number of prefixes in the set of rules.

– In case P is a prefix of Q (Q has length greater than length of P), then there will be a node on the trie that stores P and searching process will include a number of unnecessary memory accesses. This is shown in the example Figure 1: P_2 is a prefix of P_1, the priority trie still has a node that is storing P_2 and the searching for an address with 100 111* format still has to travel through 2 nodes. This restriction will be eliminated in our trie structure.

2.2. JA-trie

2.2.1. Introduce

JA-trie [10] is proposed based on multi-bit trie structure [11] with stride of 8. The difference between JA-trie and trie in [11] is that the processing of JA-trie includes blocks "*" in each prefix and this greatly reduces memory arises as multiple bits stored on the original multi-bit trie. Another feature of the JA-trie is that the order of cutting bits from the prefix string when building the trie and bits of the IP address when classifying packets are not followed in a normal order as in the multi-bit trie. Cutting order is determined based on the entropy weighted of bit blocks. It significantly reduces amount of storage memory as well as the height of trie.

2.2.2. Data Structures

JA-trie is built in the multi-bit trie with the following features:

At each node, TB(**T**ransition **B**itmap) and RB (**R**ule **B**itmap) bit vectors are added. Each vector includes k bits (k is the number of stride).

- TB: every transition has a k-bit bitmap associated, where k is the number of strides that composes a string to match. Bit j^{th} of the bitmap is asserted if the transition represents the j^{th} stride of the string to match. It is important to note that a transition can represent more than one stride at the same time.
- RB: every rule is stored in a node which includes k-bit of bitmap. The j^{th} bit is asserted if the rule is referred to the j^{th} chunk of the tuple.

The segment "*" is used in JA-trie as an input data, but does not create a specific jump and does not generate storage nodes.

JA-trie uses entropy measure for the jumps, which aims to reduce the number of nodes in the trie storage. Entropy of a jump is calculated by the formula:

$$H(x) = \sum_{i=1}^{n} P(x_i) \times \log \frac{1}{P(x_i)} \quad (1)$$

Where $P(x_i)$ is a probability that x_i appears in the set of values in the jump being considered. According to the formula (1): A value which has high probability will have low entropy and vice versa; Block value "*" has entropy equal to 0. In the process of building trie, a segment which has low entropy will be selected in advance to reduce the number of branches of each node; thereby, it reduces storage of memory.

For example, for a set of 4 rules (Table 1), the results of entropy calculation with the jump of 8 bits are shown in Table 2.

Table 1. Set of 4 rules

R1	64.91.107.0/32
R2	95.105.142.0/32
R3	96.105.142.0/32
R4	96.10.142.0/24

The order of cutting based on entropy is 4, 3, 1, 2. JA-trie is built as shown in Figure 2.

2.2.3. Packet classification on the JA-trie

With an input IP address, the segment is cut in order as one in building trie and a bit vector BM$_{class}$ is used. The searching process starts from the root node. The jumping from node **N** to node **M** is permitted only when IP's segment i^{th} is equal to the value stored in the **M** and **M**'s transition

Table 2. Results cut and entropy of the segments

Rule	Seg 1	Seg 2	Seg 3	Seg 4
R1	64	91	107	0
R2	95	105	142	0
R3	96	105	142	0
R4	96	10	142	*
Entropy	1.5	1.5	0.81	0

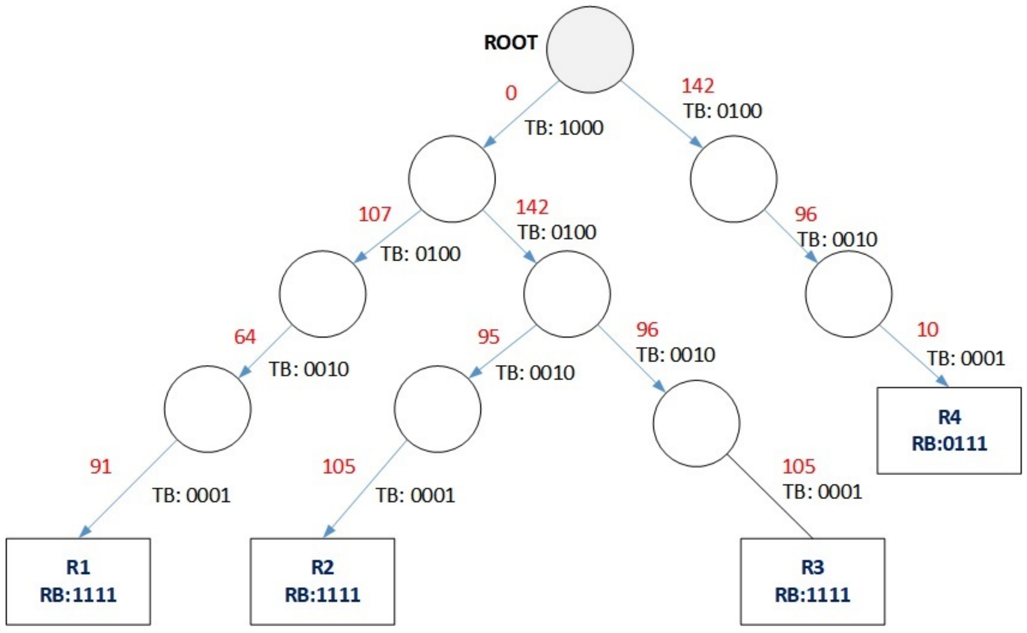

Figure 2. JA-trie

bitmap has bit i^{th} being turned on. In this case, the bit i^{th} of BM$_{class}$ is set. The searching process will end when no more jumps can be done. The rule which matches the input IP address is the one that (the rule of the list of rules in final node) has RB equal to BM$_{class}$.

For example, the IP address 96.10.142.70 is classified on the JA-trie Fig 2 as follows:

- The segments cut in order to build the trie are 70, 142, 96, 10. BM$_{class}$ is initialized to '0000'.
- 1st jump(70): no jump from ROOT.
- 2nd jump(142): ROOT has child (142) and the 2nd bit of TB of node(142) is 1; jump to the node (142) and the BM$_{class}$ is '0100'.
- 3rd jump(96): (142) has child(96) and the 3rd bit of TB of node(96) is 1; jump to the node (96) and the BM$_{class}$ is '0110'.
- 4th jump(10): (96) has child(10) and the 4th bit of TB of node(10) is 1; jump to the node(10) and the BM$_{class}$ is '0111'.

The classifying process ends, BM$_{class}$ is equal to R4's RB, so the IP matches R4.

2.2.4. The weakness of the JA-trie

The main drawbacks of the JA-trie are as follows:

– Just considering the case when the length of prefix is a multiple of k. For example, if $k = 8$ then the prefixes, with length 23 or 25, are not a recommended solution to deal with.

– In the case of large rule sets, each node has multiple RB and process of finding appropriate rule will require a long time.

– There is not any recommended solution of getting stride k to obtain optimum results in storing the prefixes and packet classification.

3. PROPOSED PACKET CLASSIFICATION ALGORITHM BASED ON MULTI-WAY PRIORITY

3.1. The basic concepts

Definitions 1. Degree of a prefix.

Consider prefixes P and Q; length of P is l; length of Q is t. Q is called n degree prefix of P if and only if the following three conditions are satisfied:

– $t \leq l$;

– The first n bits of Q coincide with the first n bits of P;

– The $(n+1)^{\text{th}}$ bit of Q is different from $(n+1)^{\text{th}}$ bit of P

Denote $Q = L_n(P)$.

Definitions 2. Degree of a set of prefixes.

Let $\mathbf{G}$ be the set of prefixes, $\mathbf{G}$ is n^{th} degree of prefix P if and only if every prefix Q of $\mathbf{G}$ satisfies $Q = L_n(P)$.

Denote $\mathbf{G} = \mathrm{S}_n(P)$.

Definitions 3. The biggest prefix.

Let $\mathbf{G}$ be the set of prefixes, P is the biggest prefix of $\mathbf{G}$ if and only if $\forall Q \in \mathbf{G}$ $(Q \neq P)$, length of Q is less than or equal to length of P.

Theorem 1. *Let* $\mathbf{G}$ *be a set of prefixes* ($\mathbf{G}$ *does not contain two identical prefixes) and* P *is the biggest prefix of* $\mathbf{G}$*: If an IP address matches* P *then* P *will be the best matching prefix of the IP.*

Proof.

Suppose that l is the length of P. Assume that there exists prefix Q ($Q \in \mathbf{G}$ and length of Q is k) which matches the IP. Since P is the biggest prefix, so $l \geq k$. On the other hand, there do not exist two identical prefixes in $\mathbf{G}$, so $l \neq k$. This leads to $l > k$. Therefore, P is the best matching prefix of the IP. ■

Theorem 2. *We have two sets of prefixes* $\mathbf{G}_1$*,* $\mathbf{G}_2$ *and prefix P, in which* $\mathbf{G}_1 = S_i(P)$*,* $\mathbf{G}_2 = S_j(P)$ *and* $i \neq j$*: If an IP address matches with prefix* P_1 *(*$P_1 \in \mathbf{G}_1$*) then there will not exist any prefix* $P_2 \in \mathbf{G}_2$ *so that* P_2 *matches with the IP.*
Proof.
*Case $i < j$:

Suppose $P = x_1\ x_2 \ldots x_i x_{i+1} \ldots x_j x_{j+1} \ldots$

According to Definition 1, $\mathbf{G}_1$ will include the prefix of the form $x_1\ x_2 \ldots x_i \bar{x}_{i+1} \ldots$ and $\mathbf{G}_2$ will include the prefix of the form $x_1\ x_2 \ldots x_i x_{i+1} \ldots x_j \bar{x}_{j+1} \ldots$

Because the IP matches with P_1, the IP must have form: $x_1\ x_2 \ldots x_i \ldots$. Therefore, the $(i+1)^{\text{th}}$ bit of IP is different from the $(i+1)^{\text{th}}$ bit of any prefix in $\mathbf{G}_2$. This also means that there does not exist any prefix $P_2 \in \mathbf{G}_2$ so that P_2 matches with the IP.
*Case $i > j$:

Suppose $P = x_1\ x_2 \ldots x_j x_{j+1} \ldots x_i x_{i+1} \ldots$

According to Definition 1, $\mathbf{G}_1$ will include the prefix of the form $x_1\ x_2 \ldots x_j x_{j+1} \ldots x_i \bar{x}_{i+1} \ldots$ and $\mathbf{G}_2$ will include the prefix of the form $x_1\ x_2 \ldots x_j \bar{x}_{j+1} \ldots$

Because IP matches with P_1, the IP must have the form: $x_1\ x_2 \ldots x_j x_{j+1} \ldots x_i \ldots$. Therefore, the $(j+1)^{\text{th}}$ bit of IP is different from the $(j+1)^{\text{th}}$ bit of an any prefix in $\mathbf{G}_2$. This also means that there does not exist any prefix $P_2 \in \mathbf{G}_2$ so that P_2 matches with the IP. ■

3.2. The idea of the algorithm

Based on Priority trie [9] and JA-trie [10], we build multi way priority – MWP trie with the following characteristics:

– Length of the prefix which is stored at a parent node is always greater than or equal to length of prefixes which is stored in its child nodes.

– Like JA-trie, MWP trie is a multi-way trie. However, there are differences in branching to overcome the limitations of the JA-trie and traditional multibit-trie. Each node $\mathbf{N}(P)$ on the MWP trie includes k children and the i^{th} child of $\mathbf{N}$ is built from $\mathbf{G}$ ($\mathbf{G}$ is a set of i^{th} degree prefixes of P).

3.3. The structure of multi way priority trie

3.3.1. Node of MWP trie

MWP is multi way trie, each node has the following characteristics:

– Node $\mathbf{N}$ stores P prefix.

– Node $\mathbf{N}$ has a *Backtrack* field which is applied to the case that Q is a prefix of P. In this case, we will not need a node to store Q and we just set $\mathbf{N}$'s *Backtrack* to be length of Q.

– Each node has a maximum of k child nodes (k = 32 with the IPv4, 128 with IPv6).

– Length of prefix which is stored at a parent node is always greater than or equal to length of prefixes which are stored in its child nodes.

– The m^{th} child of node **N** is a node that contains the biggest prefix of m^{th} degree prefix set of P.

3.3.2. Trie building procedure

The Algorithm 1 implements the building a node.

```
Algorithm 1: BuildNode
Input: G is set of input prefixes
Output: node is a node of MWP being built
1: Begin
2:     If (G is empty) then return
3:     Prefixlongest = GetLongest(G)
4:     node.key = [ValueOf(Prefixlongest)] Left shift (W–Prefixlonggest.length)
5:     node.len = Prefixlonggest.length
6:     For i = (W-1) downto 1 do
7:     Begin
8:         $G_i$ = GetRules(Prefixlongest, i, G)
9:         BuildNode(node.children[i], $G_i$)
10:    End
11:    UpdateBacktrack(node)
12:    BuildNode(node.children[0], $G_0$)
13:End
```

Explain:

– Line 3: GetLongest(**G**) is a function which returns the biggest prefix of **G**.

– Line 4: Executes the converting of the prefix from string to number and left shift (*W-Prefixlongest.length*) bits. Shifting of bits is to serve packet classification. It will be presented in the next section.

– Line 6: W is length in bit of IP address ($W = 32$ with IPv4, $W = 128$ with IPv6).

– Line 8: G_n is a set of prefixes of the n^{th} degree of Prefixlongest, the function GetRules(*Prefixlongest, n,* ***G***) returns the n^{th} degree prefixes of Prefixlongest in **G**.

– Line 9: Builds the i^{th} child.

– Line 11: Updates *Backtrack* field for child node as following principles:

  – i) *Backtrack* is an only set when prefix is the other's prefix in $\mathbf{G}_n$ and the shortest prefix has length of n .

  – ii) The *Backtrack* field of right node of n^{th} child node is set to n if it has not been set.

– Line 12: Builds the 0^{th} child

The MWP construction processing is done by calling BuildNode($ROOT$, *Prefix_Set*) where $ROOT$ is the root node and *Prefix_Set* is the set of all prefixes in the rules set.

According to the Algorithm 1, the length of prefix on the parent node is always greater than or equal to length of prefixes which are stored in its child nodes.

For example, building MWP trie for the **G** rule sets of 12 rules is shown in Table 3.

Table 3. Rule set with 12 rules

Rule	**Prefix**
R1	11010000
R2	11100
R3	1111111
R4	00100
R5	10000
R6	101111
R7	00011
R8	1100000
R9	110
R10	11
R11	00
R12	1

R1(11010000) is the biggest prefix of **G** and $\mathbf{G}_i$ is built as in Table 4.

Table 4. Sets $\mathbf{G}_i$ of R1

$\mathbf{G}_3$	$\mathbf{G}_2$	$\mathbf{G}_1$	$\mathbf{G}_0$
1100000	11100	10000	00100
110	1111111	101111	00011
	11	1	00

ROOT node stores R1;

$\mathbf{G}_3$ is input data set to build 3rd child of ROOT. (R8) node is created. Because R9 is belongs to G_3 and its length is 3, (R8).*Backtrack = 3*;

$\mathbf{G}_2$ is a data input to build 2nd child of ROOT. (R3) node is created. Because length of R10 is 2, (R3).*Backtrack=2*;

$\mathbf{G}_1$ is a data input to build 1st child ROOT. (R6) node is created, because length of R12 is 1, (R6).*Backtrack=1*;

$\mathbf{G}_0$ is a data input to build child 0 of ROOT. (R4) node is created. Continuously, (R7) is 2nd child of (R4). Because length of R11 is 2, (R7).*Backtrack=2*;

MWP is built with the sets of rules in Table 3 as Figure 3.

3.3.3. Packet classification on MWP trie

Algorithm 2: implements classifing packets in MWP trie. The main idea is as follows:

– The classification process is started from the root node.

– In each node, the IP address is compared with prefix that is stored in the node:

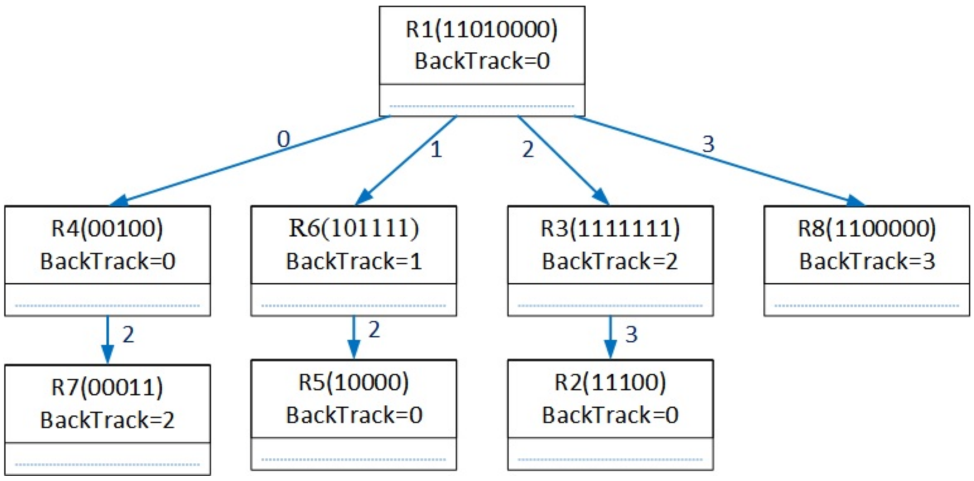

Figure 3. The example of MWP trie

– If matched, the classification process is finished.

– Else, compare the first bits of IP with the first bits of the prefix for branching.

```
Algorithm 2: ClassificationPacket
Input: $addr$ - IP address of packet;
Output: Length of Best Match Prefix
1: Begin
2:     integer $pos$
3:     integer $BMP$=0
4:     $node = ROOT$
5:     While ($node$ != NULL) do
6:     Begin
7:         $pos$ = GetMatchPrefix($addr$, $node.key$)
8:         if ($node.len$<=$pos$)then return $node.len$
9:         if ($BMP < node.Backtrack$) then $BMP = node.Backtrack$;
10:        $node = node.$mChildren[$pos$]
11:    End
12:    return $BMP$;
13:End
```

Special features in Algorithm 2 are:

– According to Theorem 1 and Theorem 2, if an IP address which matches the prefix is stored in node **N**, it is the best matching prefix and the searching will be finished without looking in other branches or the child nodes. The end of the searching is shown in Line 8.

– Line 7: The function GetMatchPrefix, Identifying coincident range (from left) of the input IP address with a prefix (according to the algorithm described in Figure 4). According to conventional theory, the finding of the left matched bits between the input IP address with a prefix will include a loop from high to low. However, in our algorithm, using *_lzcnt(packet* $\wedge$ *node.key)* function has significantly increased the speed of this operation by locating the first left bit of 1, *node.key* is built in the line 4 of Algorithm 1. If $pos \geq len$ then the IP address matches prefix being compared.

– Line 9: Marking the prefixes which match with the input IP address to avoid backtracking, in case no longer prefix is found.

– Line 10: Move to the pos^{th} child node

Examples, packet classification by Algorithm 2 is shown in Figure 3.

– The IP address is 110000000000000. We start from ROOT. IP does not match with R1 but it has 3 first bits coinciding with (R1). Therefore, classification moves to 3rd branch (R8). At the (R8) node, the checking shows that IP matches with R8.

– The IP address is 11111100000000. We start from ROOT. The input IP does not match with R1 but it has 2 first bits coinciding with (R1). Thereby, we move to 2nd branch (R3). At the (R3) node, IP does not match with R3 and *Backtrack*=2, so *BMP*=2. The IP has 3 first bits coinciding with (R3), so we move to 3rd branch (R2). At the (R2) node, the IP does not match with R2 and we could not move forward. Thus, the best matching prefix is “11” – R10.

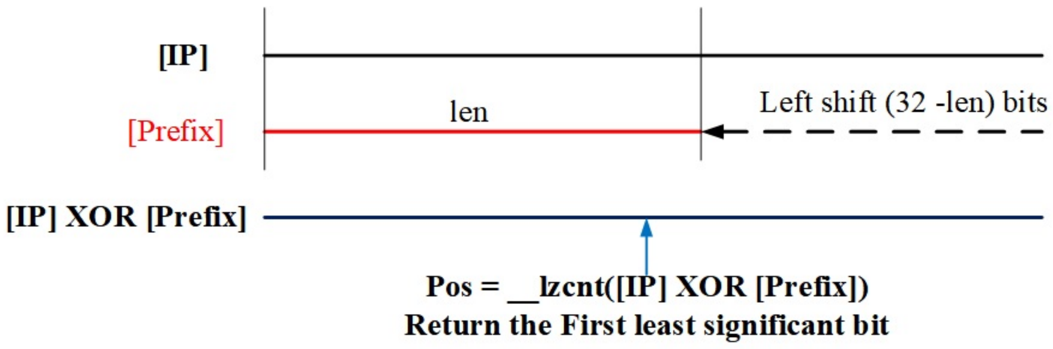
[IP]
[Prefix]
len
Left shift (32 -len) bits
[IP] XOR [Prefix]
Pos = __lzcnt([IP] XOR [Prefix])
Return the First least significant bit

Figure 4. The fast branching algorithm

4. INSTALLATION OF TESTING AND EVALUATION

With the purpose of verifying the accuracy and performance of our proposed algorithm, we install and test the classification of packets on the MWP trie structure, Priority-trie [9], JA-trie [10], Multi-bit trie [11]. The tested program is written in C language. We run the tested program on a 64-bit PC; CPU Intel Core i3 - 4010U, 1,7GHz, 2 cores; 4GB RAM.

To generate a dataset which is close to real data, we use ClassBench tool for artificial data generation. The tool is created by David E Taylor, Jonathan S. Turner of Applications Research Laboratory, Faculty of Computer Science, Washington University, Saint Louis [18]. The data sets include sets of rules and sets of parameters which are generated by above tool. Input for the tool is real data sets obtained from Internet service providers. This is the public data sets which are widely used by research community to evaluate the algorithms and packet classification devices.

4.1. Comparison in term of time consumption between the algorithms

The classification process is performed on the destination address field in 10 different data sets. Test results have shown that the MWP trie is more effective than other tries (Table 5). Where: JA - JA trie; MB - The multi bit trie with stride be 4 bits; PT – priority trie; MWP – multi way priority trie.

Table 5. The time of packet classification with different trie structures

Data set (Number of rules/Number of packets)	**Classifing time (ms)**			
	JA	**MB**	**PT**	**MWP**
Acl1 (833/4,913,520)	305	342	540	251
Acl2 (248/4,534,320)	170	199	273	140
Acl3 (505/5,188,400)	240	268	279	203
Acl4 (895/5,344,080)	255	270	287	240
Acl5 (985/4,254,320)	260	271	290	203
FW1 (132/4,780,440)	168	190	221	109
FW2 (588/3,648,400)	219	270	290	140
FW3 (132/4,495,120)	156	227	180	109
FW4 (5568/4,919,600)	450	439	652	406
FW5 (256/4,446,400)	172	266	275	156

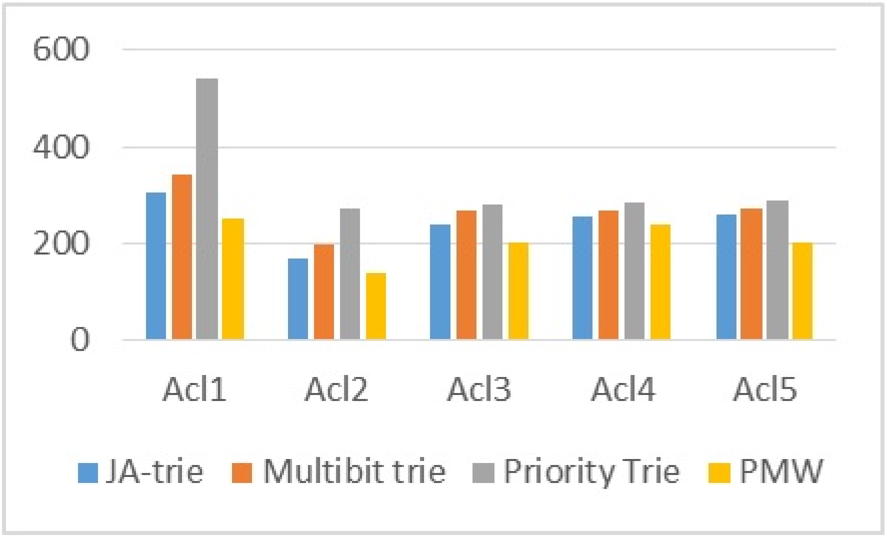

Figure 5. Comparison of classifing time (with 5 datasets)

4.2. Performance evaluation of our algorithm when the number of rules is changed

To compare the performance of the MWP trie structure with other trie structures, we fixed the number of input packets of 8 million and changed the number of input rules (the cases with large rulesets). Experiment results have shown that MWP structure is more efficient than other structures (Figure 6 - Time for classification with changing the number of input rules).

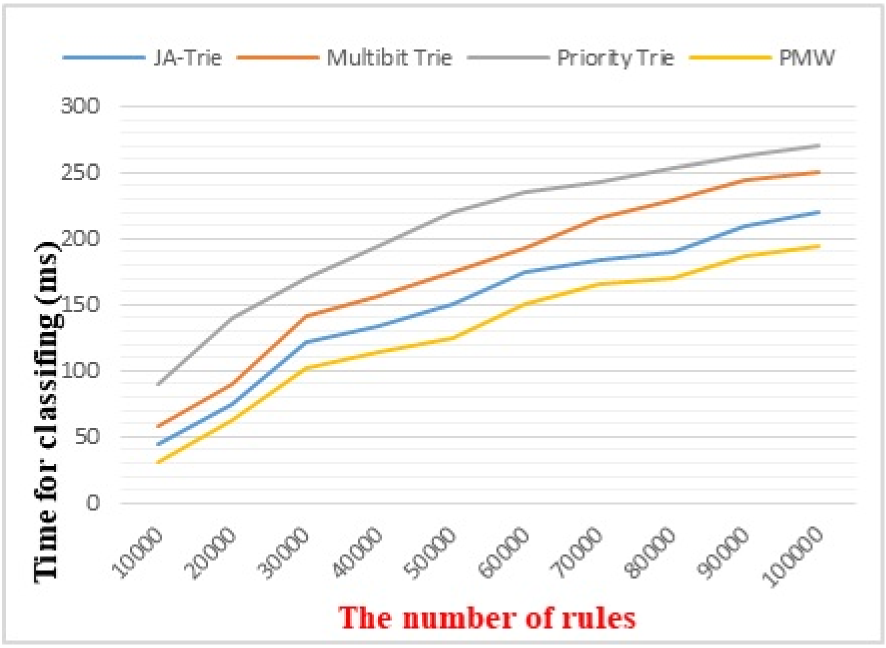

Figure 6. Time for classification with changing the number of input rules

4.3. Complexity analysis

With the structure of MWP, the first node at the i^{th} level has at most (W - $1 - i$) child nodes (W is length in bit of IP address: W=32 with IPv4, W=128 with IPv6). Therefore, the maximum height of the trie would be W and in the worst case the complexity of the search would be O(W).

Each node in the MWP stores a prefix. In case the *Backtrack* value of all nodes is zero (prefixes do not contain each other) then the memory requirement is NW. Thus, the memory requirements of MWP is O(NW).

JA-trie: if values at each octet are evenly distributed then JA-trie becomes the muiltbit trie structure with step 8. Thus, the search complexity of JA-trie is $O(W/k)$, which requires memory of $O(2^k\text{NW/k})$.

Priority trie (PT): In the worst case, the height of a priority trie is W, so the search complexity of it will be O(W). According to the structure, each node in PT always stores only a prefix, so the memory required by PT is O(NW).

Table 6. Complexity comparison of structures

Structure	**Worst-case Lookup**	**Storage**
Priority Trie	O(W)	$O(NW)$
JA-Trie	$O(W/k)$	$O(2^k NW/k)$
MWP	$O(W)$	$O(NW)$

5. CONCLUSIONS

5.1. The completed work

The article has present the classification of packets in one direction in general and with the Priority trie [9] and JA trie [10] in particular. Based on the analysis of the advantages and disadvantages of each structure, we proposed a new packet classification algorithm based on Multi Way Priority –MWP trie structure. The accuracy of the algorithm has been proved by theory and its effectiveness in performance has been demonstrated by experimental results.

5.2. The future work

MP trie structure can be implemented in practice. However, our future work will focus on optimizing the algorithm. Particularly, in the function GetLongest(**G**) (Algorithm 1), the issue of how to get the longest prefix, which the MWP trie will have the lowest height, is still open.

REFERENCES

[1] R. H. K. F. Yu, , and T. V. Lakshman, “Efficient multimatch packet classification and lookup with tcam,” *IEEE Micro*, vol. 25, no. 1, pp. 50–59, 2005.

[2] P. Z. Derek Pao *, Yiu Keung Li, “Efficient packet classification using tcams,” *Computer Networks*, vol. 50, no. 1, pp. 3523–3535, 2006.

[3] T. S. Infall Syafalni, “A tcam generator for packet classification,” *Computer Design (ICCD), 31st International Conference*, 2013.

[4] K.-C. L. H.-H. W. S.-W. G. Che-Lun Hung, Yaw-Ling Lin, “Efficient gpgpu-based parallel packet classification,” *TrustCom*, vol. 186, no. 10.1109, 2011.

[5] Y. S. D. Kang Kang, “Scalable packet classification via gpu metaprogramming,” *EDAA*, 2011.

[6] V. K. P. Shijie Zhou, Shreyas G. Singapura, “High-performance packet classification on gpu,” *High Performance Extreme Computing Conference - HPEC*, 2014.

[7] S. S. V. Srinivasan, G. Varghese and M. Waldvogel, “Fast scalable level four switching,” *in Proc. ACM SIGCOMM*, pp. 191–202, 1998.

[8] S. S. M. Buddhikot and M. Waldvogel, “Space decomposition techniques for fast layer-4 switching,” *in Proc. IFIP 6th Int. Workshop on Protocols for High Speed Networks*, pp. 25–41, 1999.

[9] E. E. S. Hyesook Lim, Changhoon Yim, “Priority tries for ip address lookup,” *IEEE Transactions on computers*, vol. 59, no. 6, 2010.

[10] A. W. M. S. G. E. A. Gianni Antichi, Christian Callegari, “Ja-trie: Entropy-based packet classification,” *IEEE 15th International Conference on High Performance Switching and Routing (HPSR)*, 2014.

[11] G. Varghese, “Network algorithms: An interdisciplinary approach to designing fast networked devices,” *Morgan Kaufmann Series in Networking*, p. 245, 2004.

[12] P. Gupta and N. McKeown, “Packet classification using hierarchical intelligent cuttings,” *Proc. Hot Interconnects*, 1999.

[13] G. V. S. Singh, F. Baboescu and J. Wang, “Packet classification using multidimensional cutting,” *ACM SIGCOMM*, 2003.

[14] J.-H. S. H. Lim and Y.-J. Jung, “High speed ip address lookup architecture using hashing,” *IEEE Comm. Letters*, vol. 7, no. 10, pp. 25–35, 2003.

[15] J. T. M. Waldvogel, G. Varghese and B. Plattner, “Scalable high speed ip routing lookups,” *Proc. ACM SIGCOMM*, pp. 25–35, 1997.

[16] P. K. S. Dharmapurikar and D. Taylor, “Longest prefix matching using bloom filters,” *IEEE/ACM Trans. Networking*, vol. 14, no. 2, pp. 397–409, 2006.

[17] K. P. K. Lim and H. Lim, “Binary search on levels using a bloom filter for ipv6 address lookup,” *Proc. ACM/IEEE Symp. Architectures for Networking and Comm. Systems (ANCS)*, pp. 185–186, 2009.

[18] D. E. Taylor and J. S. Turner, “Classbench: A packet classification benchmark,” *IEEE/ACM Trans*, vol. 15, no. 3, p. 499511, 2007.

Received on April 26 - 2017
Revised on July 04 - 2017