High-Speed Packet Classification Using Binary Search on Length

Hyesook Lim
Information Electronics Engineering
Ewha Womans University
11-1 Daehyun-dong, Seodaemun-gu
Seoul, Korea
82-2-3277-3403
hlim@ewha.ac.kr

Ju Hyoung Mun
Telecommunication R&D Center
Samsung Electronics
Meatan-dong, Yungtong-gu
Suwon-city, Korea
82-31-279-7216
Juhyoung.mun@samsung.com

ABSTRACT
Packet classification is one of the major challenges for next generation routers since it involves complicated multi-dimensional search as well as it should be performed in wire-speed for all incoming packets. Area-based quad-trie is an excellent algorithm in the sense that it constructs a two-dimensional trie using source and destination prefix fields for packet classification. However, it does not achieve good search performance since search is linearly performed for prefix length. In this paper, we propose a new packet classification algorithm which applies binary search on prefix length to the area-based quad-trie. In order to avoid the pre-computation required in the binary search on length, the proposed algorithm constructs multiple disjoint tries depending on relative levels in rule hierarchy. We also propose two new optimization techniques considering rule priorities. For different types of rule sets having about 5000 rules, performance evaluation result shows that the average number of memory accesses is 18 to 67 and the memory consumption is 22 to 41 bytes per rule.

Categories and Subject Descriptors
C.2.6 [**Computer-Communication Networks**]: Internetworking - *routers*

General Terms
Algorithms, Performance, Design

Keywords
Packet classification, Hashing, Area-based quad-trie, Binary search on length, Best matching rule

This research was supported by the Ministry of Information and Communications under a HNRC-ITRC support program supervised by IITA.

ANCS'07, December 3–4, 2007, Orlando, Florida, USA.

1. INTRODUCTION
In order to provide the various levels of service qualities in the current best-effort network, packet classification is an essential pre-requisite. Packet classification is to classify incoming packets into classes based on pre-defined rules so that routers can treat each packet according to the service defined in the class that the packet belongs to [1].

Classes are defined by rules composed of multiple header fields, mainly source and destination IP addresses, source and destination port numbers, and a protocol type. While layer 2 MAC address lookup performs exact matching and layer 3 IP address lookup performs the longest prefix matching, the packet classification requires multi-dimensional matching operation using various header fields. The operation involves exact matching for protocol field, prefix matching for IP addresses, and range matching for port numbers. The packet classification is one of the major challenges in the Internet routers since it involves complicated multi-dimensional operations as well as it should be performed in wire-speed for every incoming packet which could be several hundred-million packets per second.

Recently, researches on efficient packet classification algorithms and architectures have been widely carried out. Most of packet classification algorithms have focused on developing data structures with respect to source and destination prefix fields. It is based on the analysis that collecting candidate matches using two IP address fields is much more effective than using other fields [2]. Packet classification algorithms can be categorized as heuristics-based algorithms [3]-[6] and trie-based algorithms [7]-[9].

Heuristics-based algorithms such as HiCuts [4] and HyperCuts [5] use cuttings to eliminate the largest possible number of irrelevant rules. The cutting dimension and the number of cuts are locally determined based on the heuristic characteristics of classifiers. The number of cuts is inversely related to the depth of the decision tree and directly related to the number of duplicated rules, and hence it should be properly determined to have the optimum performance in terms of the search speed and the required memory size. Tuple space search [6] algorithm decomposes a classification query into a number of exact match queries. Utilizing a heuristic characteristic that the number of tuples which is composed of the fixed lengths of the source and the destination prefixes is small, the algorithm splits a classification table into multiple tuples. The set of rules mapped to the same tuple is of a

fixed and known length and is stored into a hash table. Each tuple is searched sequentially by hashing, and hence the classification speed depends on the number of tuples.

For a multi-dimensional search, most of trie-based algorithms use hierarchical approach which performs packet classification by connecting each one-dimensional trie sequentially [3]. The hierarchical binary search tree (HBST) [7] is a recent packet classification algorithm which performs binary search for the source prefix and the destination prefix fields hierarchically. On the other hand, an area-based quad-trie (AQT) algorithm extends one-dimensional binary trie into two-dimensional quad-trie by considering source and destination prefixes at the same time [8]. However, AQT algorithm includes empty internal nodes as well as it performs linear search on prefix length as usual trie-based algorithms do, and hence it does not provide good search performance. Recently, a priority-based quad-trie (PQT) algorithm [9] is proposed. The PQT algorithm completely removes empty internal nodes in the quad-trie by relocating the highest priority rule node belonged to the sub-trie rooted by an empty node into the empty node.

IP address lookup based on binary search on prefix length [10] is known to be efficient in search performance. In this paper, we propose an algorithm which applies the binary search on prefix length into the area-based quad-trie for packet classification. Using the fact that the packet classification is to find out the highest priority matching rule, two new optimization techniques are also proposed.

The organization of this paper is as follows. In Section II, we briefly introduce the area-based quad-trie for packet classification and the binary search on prefix length for IP address lookup. Section III describes the proposed algorithm. In Section IV, two optimization techniques using the heuristic characteristics of classification tables are proposed. Section V shows the performance evaluation result of our proposed algorithm and the comparison with previous works. Section VI concludes the paper.

2. RELATED WORKS

2.1 2-dimensional Search Trie

Table 1 shows an example rule set. We will discuss the area-based quad-trie and the proposed algorithm using this example rule set.

Area-based quad-trie (AQT) is the most well known 2-dimensional search trie [3][8]-[9]. In AQT algorithm, a two-dimensional square plane of the size of 2^{32} by 2^{32} is recursively decomposed. If decomposition is repeated by n times, the number of partitioned area becomes 4^n. Each partitioned area is represented by two n-bit prefixes which are source and destination prefixes. Figure 1 shows the AQT plane corresponding to the example rule set in Table 1. Rules are represented by rectangles in the AQT plane. Assuming that lengths of source and destination prefix fields of a rule are l bits and m bits, respectively, the rule is represented by a rectangle with the size of $2^{(32-l)}$ by $2^{(32-m)}$ in the plane.

Table 1. Example Rule Set

Rule no.	Src prefix	Dst prefix	Src port (start, end)	Dst port (start, end)	Prtl
R0	0000*	1010*	53, 53	443, 443	17
R1	000111*	11110*	53, 53	25, 25	6
R2	101011*	001101*	53, 53	25,25	17
R3	00000*	10100*	67, 67	5632	6
R4	000*	1110*	1024	1024	6
R5	0011*	00*	53, 53	25, 25	4
R6	110*	01*	0, 65535	5632	6
R7	110010*	110110*	0, 65535	5632	6
R8	1010*	00110*	53, 53	25, 25	6
R9	11010*	110*	0, 15576	2783	4
R10	*	110*	*	*	*

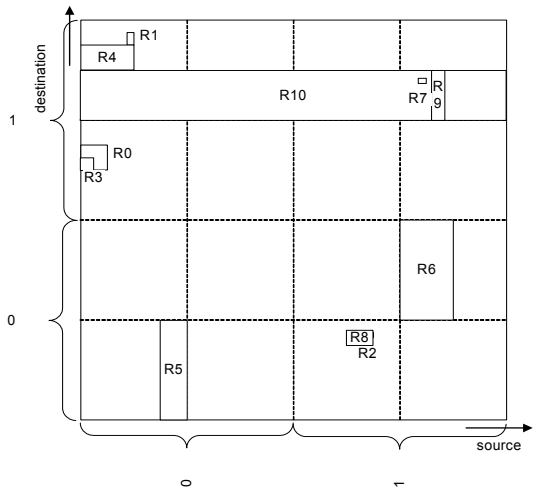

Figure 1. Rules in AQT Plane

As shown in Figure 1, each partitioned area has the same length in source and destination prefixes, but the source prefix length and the destination prefix length of a rule can be different. If any dimension of a rule has the same length as a partitioned area so that the rule completely crosses the area, it is defined as a crossing filter of the area. The area is not partitioned any more for crossing filters.

Each partitioned area is mapped into a node of a quad-trie. The entire plane is mapped to the root of the quad-trie, and 4 partitioned areas after the first partition are mapped into 4 children of the root node, and so on. Figure 2 shows the quad-trie for the rules in the AQT plane. Rules included in the crossing filter set of an area are stored into the corresponding quad-trie node, and hence the shorter length prefix of the source or the destination prefix will determine the level of the node storing the rule. If there is more than one rule stored in a node, they are assumed to be connected by a linked-list and hence searched linearly.

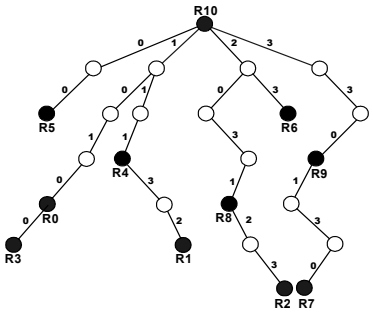

Figure 2. Area-based quad-trie (AQT)

If we have an input of (110111, 110010, 2783, 2783, 4), the search procedure in the AQT trie is as follows. Starting from the root node, the input is matched to R10, and hence R10 is remembered as the current best matching rule (BMR). Since bit pairs of the first bit to the third bit are (1, 1), (1, 1), and (0, 0), search follows edges represented by 3, 3, and 0, consequently, and encounters with a node storing R9. Since the given input is matched to R9 and R9 has higher priority than R10, the current BMR is updated as R9. The next bit pair is (1, 0), but there is no edge represented by 2. Hence the search is completed and returns the current BMR.

As shown in the search example, AQT performs the linear search on prefix length as the same as all other trie-based algorithms do, and hence it does not provide good search performance. Assuming the maximum length of prefixes is *W*, the depth of AQT trie is usually *W*.

2.2 Binary Search on Length

Since our proposed work in this paper applies binary search on prefix length into a 2-dimensional trie, we describe binary search on prefix length algorithm in detail. For the binary search on length, a binary trie is primarily separated according to the level of the trie, and every node including internal nodes in each level is stored into a hash table. *Binary search is used in determining the level of the access, and hashing is used in accessing the actual prefix value of the level.*

Figure 3 shows a binary trie for the source prefix field of Table 1 and the access order performing the binary search on prefix lengths proposed by Waldvogel [10]. Waldvogel's binary search on prefix length is for the longest matching operation in IP address lookup. The algorithm has a basic assumption that there is a longer prefix if there is a match in the current level of the access and there is not a longer length prefix if there is no match in the current level of the access. In other words, starting from accessing the medium level (level 4 in Figure 3) by hashing, the search continues to the level of a longer length (level 5) for a match, and otherwise, the search continues to the level of a shorter length (level 3) for no match, and so on.

However, there could be a longer length prefix even though the input does not match in the current level as well as there may not be a longer length prefix even though the input matches in the current level. This is contradictory to the basic assumption of this algorithm. This algorithm solves the issue using pre-computed markers (denoted by *M*) and best matching prefixes (denoted by bmp). Markers are required in empty internal nodes in order to indicate the existence of prefixes in longer lengths. Denoting the current best matching prefix (BMP) with markers is also required to avoid a back-tracking in case of markers misleading the binary search. This algorithm provides excellent search performance since binary search is performed on prefix length.

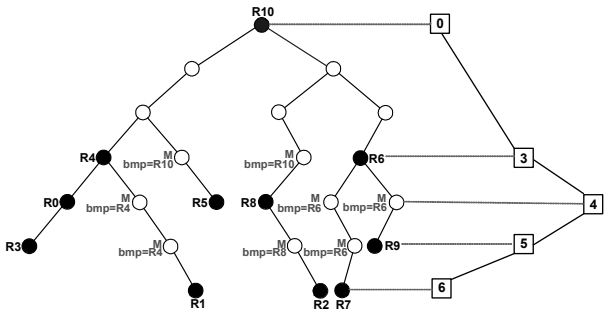

Figure 3. Waldvogel's binary search on prefix length

3. PROPOSED WORK

Rules in packet classification are composed of source and destination port numbers and a protocol type as well as source and destination prefixes. Throughout the analysis of real classification tables used in the Internet routers, it has been previously reported that 5% of inputs are matched to more than 20 rules and most of the inputs are matched to less than 5 rules [2] if we filter rules using source and destination addresses simultaneously. Using this analysis, we can conclude that efficient 2-dimensional algorithms can be used for multi-dimensional packet classification. The proposed algorithm is primarily to filter rules matched to a given input using source and destination addresses of the input and then to perform linear search on those matched rules using the remaining fields.

In filtering rules, we propose to apply binary search on prefix length which shows an excellent search performance in a 1-dimensional lookup. In the binary search on prefix length for IP address lookup, binary trie is separated according to the level of trie. Internal nodes as well as prefixes in each level are stored into the corresponding hash table, and the binary search on those hash tables is performed. In the proposed algorithm for packet classification, we propose to separate the area-based quad-trie according to the level of the trie and store rules and internal nodes of each level into the corresponding hash table and perform binary search on those hash tables.

However, we have an issue in this approach. While the IP address lookup looks for the longest prefix matched to a given input, packet classification looks for the highest priority rule among all matching rules. The rule priority is not directly related to the length of the prefix even though it shows high correlation as will be shown in a later section. In the well known binary search on length algorithm described in the previous section, because of the prefix nesting relationship, the markers and the markers' BMPs are pre-computed to help the search for the longest matched prefix [10]. However, it is not possible to use the markers and the markers' BMPs for packet classification since packet classification looks for the highest priority rule not the longest matching rule. However, if we remove them, the binary search on

length does not properly work if there is prefix nesting relationship.

If we recall the reason behind the pre-computation of the markers and the markers' BMPs, the markers should be pre-computed to indicate the existence of a longer prefix even though there is no matched prefix in the current level. The markers' BMPs should be pre-computed to avoid the back-tracking in the case that the markers mislead the search. If there is no prefix nesting relationship in which a prefix includes other prefix as its sub-string, the markers and the markers' BMPs are not necessarily pre-computed.

The proposed algorithm utilizes this fact. In the proposed algorithm, AQT trie is firstly decomposed into multiple tries depending on the relative level of each rule so that rules located in each decomposed trie never has nesting relationship. Hence the binary search on length can be performed without any pre-computation. The number of tries would be equal to the maximum number of prefix nesting. This approach is actually good since there are not many levels of rule nesting in classification tables which will be shown in a later section. Hence the number of decomposed tries is bounded by a small number in packet classification.

Figure 4 shows the proposed binary search on length (BSL) trie architecture for the example rule set in Table 1. Since there are maximum 3 levels of rule nesting as shown in Figure 2, we have three separated BSL tries.

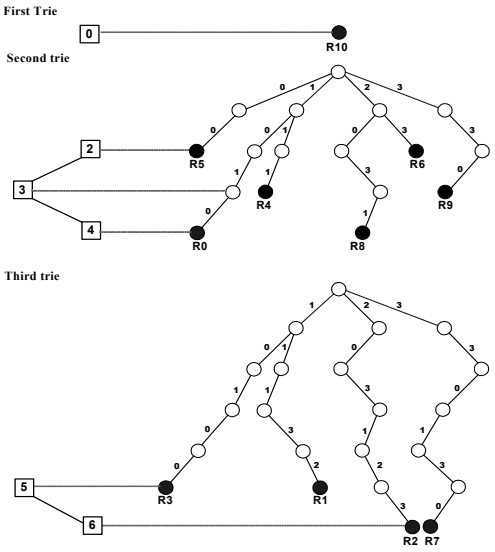

Figure 4. Proposed binary search on length in multiple tries

3.1 Building the Data Structure

The data structure is composed of two tables; a quad-trie table and a rule table. The quad-trie table is to store the data structure of an area-based quad-trie using the source and the destination prefix fields and the rule table is to store rules with the remaining fields.

Procedures to build the quad-trie table are as follows. At the first step, we need to construct an AQT trie with respect to source and destination prefix pairs of the given rule set. The second step is to decompose the AQT trie. Starting from the root node of the AQT trie, the first rule met in each path is removed from the AQT trie and the removed rules are located into the first decomposed trie. The second rule met in each path is removed from the AQT trie and located into the second trie, and so on. This process is repeated until every rule in the AQT trie is removed and located into a separate decomposed trie.

Now we have multiple decomposed tries. For each level of the trie, if there is at least one rule, a hash table is constructed for that level. Each entry of the hash table has a rule table pointer which indicates the highest priority rule among the rules mapped into the corresponding node.

The rule table is generated as follows. The number of entries of the rule table is the same as the number of rules. Each entry of the rule table has the rest of rule fields. If there is more than one rule mapped into a node, rules should be connected by a linked-list in the order of their priorities, and hence each entry of the rule table has a pointer for the next rule.

3.2 Searching

For a given input packet, each bit of source address and each bit of destination address is interleaved to make a codeword. Each digit of the codeword is 0, 1, 2, or 3. In generating a hash key for the hash table corresponding to level *i* of a trie, *i* digits of the input codeword are used. When a node is accessed using the hash key, there could be three cases.

The first case is to encounter an internal node. This case guarantees no rule in shorter lengths since rules are located only in leaves in each trie and hence search needs to go for a longer length.

The second case is to encounter an empty entry (no node). This guarantees no node in longer lengths, and hence search needs to go for a shorter length.

The third case is to meet a node with rules. In this case, if the current best matching rule (BMR) has a higher priority than the encountered rule, we do not need to compare the input with the rule. If the current BMR has a lower priority than the encountered rule, source and destination addresses of the input are compared with the source and destination prefixes of the rule. If we have a match, search needs to follow the rule table pointer, and the rest of the fields are compared with the input in the rule table. The rules linked by a linked-list pointer are only examined when the current BMR has a lower priority than the priority of the linked rules. In case that we have a complete match, the BMR is updated.

At the third case, after the comparison, search can leave the current trie since the third case guarantees no other node matched to the given input because of no rule nesting in each trie. Another condition to leave the current trie is that there is no more level to proceed. The search process is repeated for every trie.

As an example, if we have an input of (110111, 110010, 2783, 2783, 4), the search procedure in our proposed algorithm is as follows. Starting from the first trie, the input is matched to R10, and hence R10 is remembered as the current best matching rule (BMR). In the second trie, the first level of access is level 3, and hence we need to generate a hash key for level 3 using the first 3

digits of the interleaved input, which are 330. Since we have a match with R9 and R9 has a higher priority than the current BMR, we need to follow the rule table pointer and compare the rest of the fields. The search leaves the second trie since we encountered a node with rules in the second trie. In the third trie, the first level of access is level 5. The corresponding entry of 33023 is neither a rule node nor an internal node, and hence search needs to go for a shorter length. However, there is no level in a shorter length, and hence search is completed and the current BMR is returned.

3.3 Table Update

The classification table update could be a rule deletion or a rule insertion. If a rule needs to be deleted, the rule is firstly searched and then deleted. If the deleted rule has a linked-list pointer, the linked-list pointer of a rule which points the deleted rule should be replaced by the linked-list pointer of the deleted rule. This process causes a one step back-tracking, but it is not difficult because the rule entry has been visited during the search of the deleted rule anyway.

If a rule needs to be inserted, the rule is firstly searched and the proper node storing the new rule is determined. If the new rule does not match any existing rule in the current trie, this means that the new rule does not have any nesting relationship with other existing rules. Hence the rule can be stored into the trie by either creating a new node or storing it to an empty node. The remaining fields are stored into the rule table.

If there is a matched rule and the shorter length prefix of the new rule is shorter than the matched rule, the new rule should be located into the current trie since the new rule should be in a higher level than the existing rule in the AQT trie. For the rule of which its position is taken away, the same processing is repeated in the next trie, and so on. If the new rule is to be stored in a node with existing rules, depending on the priority of new rule, the linked-list pointer of an existing rule in the rule table should be modified. Therefore, the maximum number of rules affected by a rule insertion is limited by the number of tries plus 1 which is a small number.

4. OPTIMIZATION TECHNIQUES

If rule priorities are considered in the building procedure of packet classification, performance improvement is expected. In other words, depending on which trie is search first, the search performance could be different. If a trie with many low priority rules is searched first, the search performance would be lowered, and if a trie with many high priority rules is searched first, search performance would be improved. Utilizing the fact, we introduce two new optimization techniques in improving the search performance of the proposed algorithm in this section.

4.1 Optimization Technique 1 (Search Lower Priority Rules Later)

From the analysis of rule distribution of class-bench databases [11] which is known to be similar to classification tables in real routers, we found out that rules with a wild-card in any of prefix fields have low priorities in general. Wild-card rules are stored in the root node of the AQT trie. In the proposed algorithm, since we construct the first trie using the first rule met in the path from the root node of the AQT trie, if there exist at least a rule in the root node, the root node itself is the first trie. Since we search from the first trie, the search performance is not good if there are many rules linked together in the root node and their priorities are low.

The rule distribution of the class-bench databases with the size of 5000 rules shows that 10 to 600 rules are located in a root node depending on the type of classification tables, and priorities of those rules are relatively low. The first optimization technique is to search the trie composed of wild-card rules at the latest. If search in the root node encounters a rule with a lower priority than the current BMR, the search process is immediately finished since rules are connected in the descending order of priorities. This avoids long linear search in the root node. Figure 5 shows the proposed algorithm using the optimization technique 1 which just moves the root node from the first trie to the last trie.

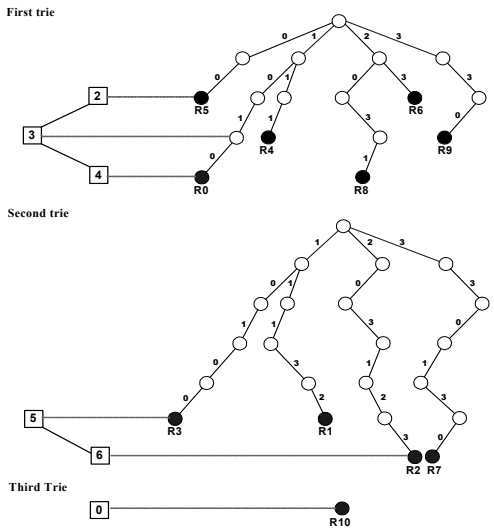

Figure 5. Proposed algorithm with optimization technique 1

4.2 Optimization Technique 2 (Search Higher Priority Rules Earlier)

If we search higher priority rules earlier, then better search performance is expected. From the further analysis of class-bench databases, we found out that there is some correlation between the length of prefixes and the priority of rules. The relationship is that rules with longer prefixes tend to have higher priorities than rules with shorter prefixes. In other words, rules located in lower levels of AQT trie tend to have relatively higher priorities than rules located in higher levels. Therefore, if we search rules in lower levels first, the better search performance is expected since we can avoid linear searches in many nodes. The second optimization technique of our proposed algorithm is to decompose the AQT trie from the bottom to the top of the trie.

Figure 6 shows the proposed algorithm using the optimization technique 2 which searches rules with longer prefixes earlier. In the first step, leaves of the AQT trie are firstly removed to construct the first trie. In the second step, empty leaves left by removing rule leaves are discarded from the AQT trie. These two steps are repeated until all the rule nodes are removed from the AQT trie. The optimization technique 2 achieves better search

performance as will be shown in a later section since the higher priority rules are compared earlier and it avoids a lot of linear searches in other tries.

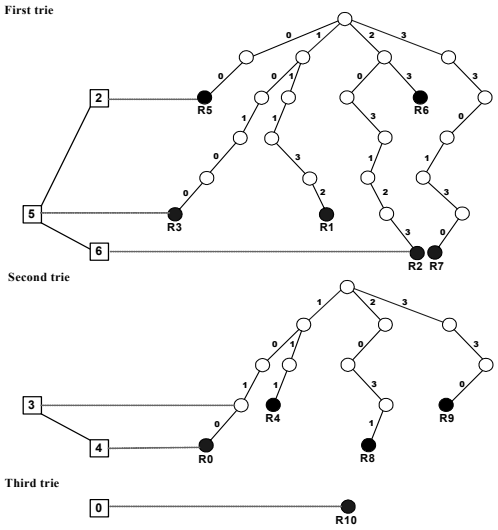

Figure 6. Proposed algorithm with optimization technique 2

5. SIMULATION RESULTS

The performance of the proposed algorithm and the proposed optimization techniques was evaluated using the class-bench rule tables [11] which are known to have close characteristics of real classification tables. Three different types of rule sets, access control lists (ACL), firewalls (FW), and IP chains (IPC), were generated.

The number of rules in a set is about 1000 or 5000 rules. We show the performance evaluation result in terms of the number of rules (N), the number of BSL tries (N_t), the worst-case number of memory accesses (T_{wst}), the average number of memory accesses (T_{avg}), the required memory size in storing BSL tries (M_{trie}), and the required memory size in storing a rule table (M_{rule}). We also show the average memory consumption required in storing a rule (M/rule), which is the total memory size divided by the number of rules.

Table 2 shows the performance of the proposed algorithm. The number of memory accesses of the proposed algorithm is more than expected since the algorithm firstly performs the linear search in the root node.

Table 2. Performance evaluation result of proposed algorithm

	N	N_t	T_{wst}	T_{avg}	M_{trie} (Kbytes)	M_{rule} (Kbytes)	M/rule (bytes)
ACL1k	958	5	47	21.3	23	20.2	45
ACL5k	4659	6	78	34.7	84.3	97.9	39
FW1k	870	3	293	192.7	3.5	18.3	25
FW5k	4343	3	1002	560.7	4.2	91.4	22
IPC1k	988	5	76	63.2	32.3	20.7	54
IPC5k	4467	5	263	182.3	25.5	93.8	27

Table 3 shows the performance result of the optimization technique 1 which performs the root node search at the latest. It is shown that the performance improvement in the worst-case and the average number of memory accesses is significant with this simple modification. Performance in all other metrics is the same as the original proposed algorithm since the optimization technique 1 only changes the search order.

Table 3. Performance evaluation result on the proposed optimization technique 1

	N	N_t	T_{wst}	T_{avg}	M_{trie} (Kbytes)	M_{rule} (Kbytes)	M/rule (bytes)
ACL1k	958	5	43	17.34	23	20.2	45
ACL5k	4659	6	59	20.10	84.3	97.9	39
FW1k	870	3	286	115.95	3.5	18.3	25
FW5k	4343	3	971	392.54	4.2	91.4	22
IPC1k	988	5	71	33.38	32.3	20.7	54
IPC5k	4467	5	233	65.18	25.5	93.8	27

Table 4 shows the performance evaluation results of the proposed optimization technique 2 which searches higher priority rules earlier. As shown, the average number of memory accesses was reduced even further compared with the proposed optimization technique 1. The performance of other metrics is slightly different since BSL tries in different shapes were constructed.

Table 4. Performance evaluation result on the proposed optimization technique 2

	N	N_t	T_{wst}	T_{avg}	M_{trie} (Kbytes)	M_{rule} (Kbytes)	M/rule (bytes)
ACL1k	958	5	21	12.54	23.2	20.2	45
ACL5k	4659	6	39	17.91	93.8	97.9	41
FW1k	870	3	378	19.31	3.8	18.3	25
FW5k	4343	3	804	67.02	4.5	91.4	22
IPC1k	988	5	69	21.89	33.9	20.7	55
IPC5k	4467	5	234	32.30	24.5	93.8	27

In Table 2 to Table 4, it is shown that the number of levels of rule nesting in classification tables is 3 to 6, and hence the number of tries constructed by the proposed algorithm is limited by a small number.

We performed extensive simulations in order to compare our proposed algorithm with previous works. The performance was compared in terms of the average number of memory accesses (T_{avg}), the worst-case number of memory accesses (T_{wst}), and the required memory size (M) storing classification table including the trie structure and the rule table. The performance of HiCuts [4] varies depending on the pre-determined number of cuts and the pre-determined number of rules in each leaf node. The performance shown here is the case that the number of cuts and the number of rules in a leaf node are 8 and 4, respectively.

Table 5 shows the performance comparison for ACL type classifiers. Figure 7 shows the performance comparison in term of the average number of memory accesses. As shown in the table and the figure, the proposed algorithm shows much better performance than other algorithms in both of the search

performance and the memory requirement. Especially, since the ACL type classifier has a lot of rules with long prefix lengths, the optimization technique 2 of the proposed algorithm shows excellent performance.

Table 5. Performance comparison result for ACL types

	ACL1k			ACL5k		
	T_{avg}	T_{wst}	M (Kbytes)	T_{avg}	T_{wst}	M (Kbytes)
H-trie [3]	77.2	124	82.9	84.0	177	401.5
HiCuts [4]	57.1	309	178.4	93.4	443	956.3
AQT [8]	38.6	64	56.4	50.1	94	200.2
PQT [9]	35.6	75	29.9	59.6	113	145.6
BV [12]	66.0	68	153.3	64.1	76	2793
Proposed	21.3	47	43.2	34.7	78	182.2
Prop opt 1	17.3	43	43.2	20.1	59	182.2
Prop opt 2	12.5	21	43.4	17.9	39	191.7

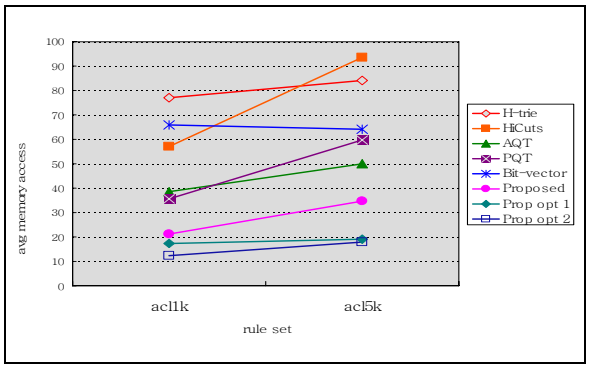

Figure 7. Average number of memory accesses for ACL types

Table 6 shows the performance comparison on FW type rule sets, and Figure 8 shows the performance comparison in the average number of memory accesses. For FW type classifiers, the hierarchical-trie (H-trie) shows good performance. In the FW type classifiers, many rules have long length source prefixes and short length destination prefixes. The proposed algorithms as well as the AQT algorithm store rules into a level which is equal to the length of a shorter prefix. Hence many different rules are mapped into a same node which causes a long linear search. However, the proposed optimization technique 2 still shows the best performance in the average number of memory accesses.

Table 6. Performance comparison result for FW types

	FW1k			FW5k		
	T_{avg}	T_{wst}	M (Kbytes)	T_{avg}	T_{wst}	M (Kbytes)
H-trie [3]	52.1	117	39.4	69.2	162	119.1
HiCuts [4]	68.0	342	1373	411.6	1691	3704
AQT [8]	369.3	444	35.2	660.5	1193	479.8
PQT [9]	197.9	293	27.2	571.1	999	136.0
BV [12]	196.6	318	111.9	738.8	1044	2340
Proposed	192.7	293	21.8	560.7	1002	95.6
Prop opt 1	115.9	286	21.8	392.5	971	95.6
Prop opt 2	19.3	378	22.1	67.0	804	95.9

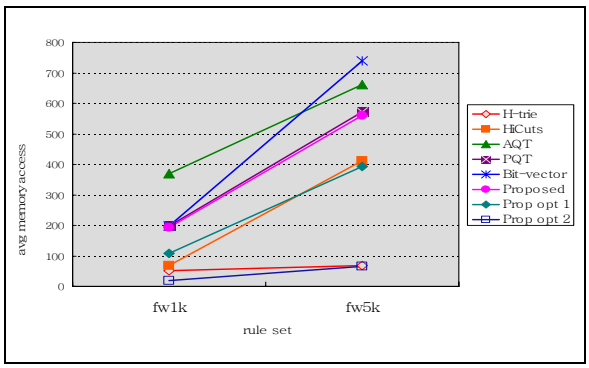

Figure 8. Average number of memory accesses for FW types

Table 7 and Figure 9 show the performance comparison on IPC type classifiers. The IPC type classifiers have the characteristics in the middle of the ACL type and the FW type. The proposed optimization technique 2 shows the best performance in every aspect. From the performance comparison, while the performance of existing algorithms heavily depends on the rule characteristics, the proposed algorithms show steady performance on every type of classification tables.

Table 7. Performance comparison result for IPC types

	IPC1k			IPC5k		
	T_{avg}	T_{wst}	M (Kbytes)	T_{avg}	T_{wst}	M (Kbytes)
H-trie [3]	71.9	128	121.6	85.6	192	224.7
HiCuts [4]	27.1	147	656.6	103.9	505	3060
AQT [8]	94.5	119	71.2	344.8	415	234.3
PQT [9]	73.6	106	30.9	202.1	295	139.9
BV [12]	63.6	80	154.3	151.9	230	2351
Proposed	63.2	76	53.0	182.3	263	119.3
Prop opt 1	33.4	71	53.0	65.2	233	119.3
Prop opt 2	21.9	69	54.6	32.3	234	118.3

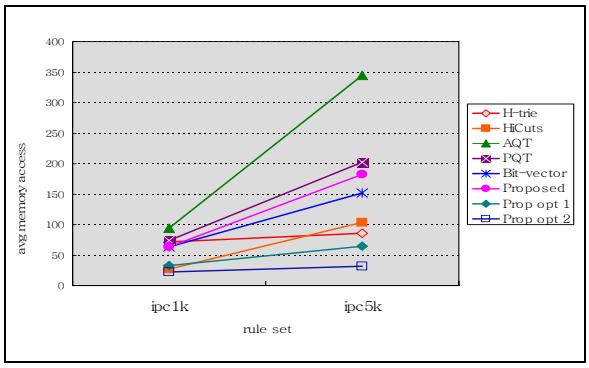

Figure 9 Performance comparison result for IPC types

6. CONCLUSION

Algorithms for efficient packet classification in the Internet routers have been widely studied, but none of those algorithms satisfies all of the performance metrics such as search speed,

required memory size, incremental update, and scalability toward large classifiers.

In this paper, we proposed an efficient packet classification algorithm which shows excellent performance in the search speed and the required memory size. By decomposing a 2-dimensional quad-trie into multiple disjoint tries, we have explored a method to apply the binary search on length for packet classification. Two optimization techniques are also proposed to improve the search performance further.

From the simulation result using class-bench databases, we found out that the number of levels of rule nesting in classification tables is 6 at the maximum, and hence the number of tries constructed by the proposed algorithm is limited by 6. The average number of memory accesses for the proposed algorithm is about 17 for a 5000 rule classifier of ACL type which is much better than related works. The average memory consumption is 22 to 55 bytes per rule. While the performance of most of the previous works heavily depends on the characteristics of classification tables, the proposed algorithm showed steady performance not much depending on table characteristics.

7. REFERENCES

[1] H. Jonathan Chao, "Next generation routers," Proceedings of the IEEE, vol. 90, no. 9, pp. 1518 – 1558, Sept. 2002.

[2] F. Baboescu, S. Singh, G. Varghese, "Packet classification for core router: is there an alternative to CAMs?," IEEE INFOCOM 2003, vol. 1, pp. 53-63, Mar. 2003.

[3] P. Gupta and N. Mckeown, "Algorithms for packet classification," IEEE Network, vol.15, no. 2, pp.24-32, Mar./Apr. 2001.

[4] P. Gupta, N. Mckeown, "Classification using hierarchical intelligent cuttings," IEEE Micro, vol.20, no. 1, pp.34-41, Jan./Feb., 2000.

[5] S. Singh, F. Baboescu, G. Varghese, J. Wang, "Packet Classification Using Multidimensional Cutting," in Proc. ACM SIGCOMM, pp. 213-224, Aug. 2003.

[6] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in Proc. ACM SIGCOMM'99, pp. 135-146, Aug. 1999.

[7] H. Lim, H. Chu, and C. Yim, "Hierarchical Binary Search Tree for Packet Classification," IEEE Communications Letters, vol. 11, no. 8, pp.689-691, Aug. 2007

[8] M. M. Buddhikot, S. Suri, and M. Waldvogel, "Space decomposition techniques for fast layer-4 switching," in Proc. Protocols for High Speed Networks, pp. 25-41 Aug. 1999.

[9] Hyesook Lim, Min Young Kang, and Changhoon Yim, "Two-dimensional packet classification algorithm using a quad-tree," Computer Communications, Elsevier Science, vol. 30, no.6 pp. 1396-1405, Mar. 2007.

[10] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in Proc. ACM SIGCOMM1997, pp. 25-35, 1997.

[11] D.E. Taylor and J.S. Turner, "ClassBench: a packet classification benchmark," INFOCOM 2005, vol.3, pp. 2068-2079, 13-17 March 2005.

[12] T.V. Lakshman and D. Stildialis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in Proc. ACM SIGCOMM1998, Sep. 1998.