Available online at www.sciencedirect.com
ScienceDirect
Computer Communications 30 (2007) 1396–1405
computer communications
www.elsevier.com/locate/comcom

Two-dimensional packet classification algorithm using a quad-tree

Hyesook Lim [a,*], Min Young Kang [a], Changhoon Yim [b]

[a] Department of Information Electronics Engineering, Ewha Womans University, Seoul, Republic of Korea
[b] Department of Internet and Multimedia Engineering, Konkuk University, Seoul, Republic of Korea

Received 6 January 2006; received in revised form 9 January 2007; accepted 9 January 2007
Available online 17 January 2007

Abstract

For the last few years, there is an explosive growth in the development and the deployment of network applications that transmit and receive audio and video over the Internet. In order for such multimedia applications to function properly, networks need to provide the level of performance, which is called the quality of services (QoS). An essential element for the Internet routers to provide the QoS is the packet classification which classifies incoming packets into classified flows. Based on the pre-defined rules composed of multiple header fields, incoming packets are classified into a specific flow, and packets are treated differently according to the classified flow. Efficient packet classification algorithms have been widely studied, but none of known algorithms except the linear search considers the priority of rules in constructing the data structure of classification tables. In this paper, we propose a priority-based quad-tree (PQT) algorithm for packet classification. In constructing a quad-tree generated based on recursive space decomposition, the priority of rules is primarily considered in the proposed algorithm. In the simulation using the class-bench databases, the proposed algorithm achieves very good performance in the required memory size and reasonable performance in the classification speed. The proposed algorithm also provides good scalability toward large classifiers.

Keywords: Packet classification; Quality of services; Priority; Recursive decomposition

1. Introduction

The Internet protocol (IP) deployed today provides a best-effort service to all the datagram it carries. In order words, the Internet makes its best effort to move each datagram from a sender to a receiver as quickly as possible, but it does not make any guarantee about the end-to-end delay or inter-packet delay. Due to the lack of effort to deliver packets in a timely manner, it is an extremely challenging problem to develop successful multimedia networking applications for the Internet. In order for the multimedia applications to be successful in the Internet, the Internet routers need to provide the different quality levels to different applications [2]. To meet these requirements, the routers should be able to perform resource reservation, admission control, and output scheduling, and a pre-requisite to deploy these mechanisms is the packet classification [3]. Flow-aware routers capable of performing the packet classification distinguish and classify the incoming traffic into different flows and apply the different class of services to each flow.

Class is assigned by the rules composed of several fields in a packet header, and the set of rules is called classification table. Packet classification is the process of determining the class for an incoming packet by referring the classification table in the routers. Rules matching to the incoming packet headers are searched, and the rule with the highest priority is selected as the final match. Flow-aware routers with the packet classification capability can provide packet filtering, traffic rate limiting, class-based routing, and accounting and billing [3].

As the same as the IP address lookup in the Internet routers, the packet classification needs to be performed for all incoming packets in real-time, but the packet

* Corresponding author. Tel.: +82232773403; fax: +82232773494.
E-mail address: hlim@ewha.ac.kr (H. Lim).

0140-3664/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2007.01.004

classification has much more complex data structures and operations than the IP address lookup. It has to process layer-4 fields in addition to layer-3 fields and needs to perform different types of operations for each field.

There are many recent contributions to packet classification [4]. The search time, the required memory size, and the update performance are three major metrics in evaluating packet classification algorithms. The packet classification algorithms focus on optimizing some of the three requirements and allow tradeoffs amongst the metrics depending on the needs of various applications. Area-based quad-tree (AQT) [5] is considered one of the most prominent packet classification algorithms which achieve good search time and the small memory size compared to other algorithms [6].

In this paper, we propose an efficient packet classification algorithm which improves the search time and the required memory size compared to the AQT. Based on the recursive space decomposition of the AQT, the proposed algorithm additionally utilizes the priority of rules in reducing the number of tree levels and in removing empty nodes.

The organization of this paper is as follows: Section 2 describes the related works of packet classification algorithms. Section 3 describes the concept of recursive space decomposition which is the basic construction of the AQT algorithm and our proposed algorithm. In Section 4, we propose the priority-based recursive space decomposition for the packet classification. Section 5 presents simulation results and performance comparison using class-bench databases [1], and Section 6 briefly concludes the paper.

2. Related works

The basic structure in storing classification table is to arrange the rules linearly in the order of priorities and then to search from the highest priority rule toward the lower priority rules. The search finishes when it either finds the matched rule or reaches the end of the list (default case). Even though the linear search is the only algorithm which considers the priority of rules in a search process and requires a small amount of memory of $O(N)$, it does not provide good scalability because of the $O(N)$ search time. To reduce the search time, ternary content addressable memory (TCAM) is widely used for packet classification in various router equipment companies [7,8]. TCAM provides very good search performance which is a single memory access. However, it consumes large power and about six times larger area than ordinary memories, and it is rather expensive. Hence packet classification algorithms using ordinary memories instead of TCAM should be studied. Packet classification algorithms can be categorized as trie-based algorithms, heuristics-based algorithms, and geometric algorithms [3,4].

2.1. Trie-based algorithms

Trie-based algorithms construct tries using a destination prefix and a source prefix among the fields composing classification table. A hierarchical trie (H-trie) [3,4] first constructs a trie for each dimension (field) and builds a multidimensional trie by hierarchically connecting tries in each dimension. Each node of the first trie is connected to the second trie hierarchically so that each field is to be searched consequently. The H-trie has very simple data structure, but the drawback is to require back-tracking because a packet can match multiple rules. Assuming that the maximum length is W and the number of fields is d, the H-trie requires $O(W^d)$ search time and $O(NdW)$ memory size. Set-pruning trie algorithm improves the search time to $O(dW)$ by removing the back-tracking in the hierarchical trie [3,4]. The back-tracking is removed by copying all matched rules residing in the search path into leaves. Hence it requires $O(N^d)$ memory size for the price of the improved search time. Using pre-computation of the best matching rule (BMR) of each node and switching pointers for jumping from one node to another node, grid-of-trie algorithm overcomes the both disadvantages of the back-tracking and the rule-copying [16]. The grid-of-trie requires $O(W^{d-1})$ search time while it maintains $O(NdW)$ memory size. However, the grid-of-trie does not provide incremental update because of the excessive pre-computation.

The intrinsic problem of the trie structures described above is that there would be a large number of empty nodes which is not associated with rules, and it results in large memory size and long search time. Recently, a packet classification algorithm based on two-dimensional binary prefix tree has been proposed, and the algorithm removes empty nodes in the trie structures [10]. It provides $O(N)$ memory size and $O(\log_2 N)$ search time.

2.2. Algorithms based on heuristics

HiCuts (Hierarchical Intelligent Cuttings) [11] and tuple-space search [12] algorithms are based on heuristic characteristics of real packet classifiers. HiCuts partitions the multidimensional search space based on heuristics that exploit the structure of classifiers. The characteristics of a decision tree such as its depth, the degree of each node, and the local search decision to be made at each node are chosen in pre-processing classifiers. A small number of rules are stored into each leaf nodes of HiCuts tree, and they are searched linearly. The HiCuts has some issues that the search time highly depends on the characteristics of classifiers and it requires excessive pre-processing time.

Tuple-space search algorithm decomposes a classification query into a number of exact match queries [12]. Utilizing the fact that the number of tuples which is composed of the fixed lengths of the source and the destination prefix 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.

2.3. Geometric algorithms

The geometric interpretation of the packet classification is based on that a d-dimensional rule represents a hyper-rectangle in a d-dimensional space while a prefix represents a contiguous interval on a number line. A packet header represents a point in the d-dimensional space. Classifying a packet is to find the highest-priority hyper-rectangle that contains the point representing the packet.

In cross-producting algorithm [9], for each dimension in classification table, every possible case is primarily listed, and a cross-product table is constructed. For each entry of the cross-product table, the best matching rule is pre-computed and stored. In search, the results of each one-dimensional search are combined and used as a pointer in approaching the cross-product table. This algorithm provides fast classification since packet classification is achieved by multiple one-dimensional searches (which can be performed in parallel) and a lookup to the cross-product table, but it has disadvantages that it requires the huge size of memory, $O(N^d)$, because of multiple one-dimensional tables and the cross-product table and it has difficulty in table update because of the pre-computation in building the cross-product table.

There are several algorithms based on multiple one-dimensional searches for packet classification [13–15]. In these algorithms, each rule corresponds to a bit position. From each one-dimensional search, multiple bit vectors, in which matched rules are set in the corresponding bit position, are generated, and they are logically *AND*ed to conclude the matched rules in all fields. If more than one bit are set in the final bit vector, the rule corresponding to the left most bit position is the best match or the highest priority match.

In area-based quad-tree (AQT) algorithm [5], the two-dimensional search space using the source and the destination prefix fields is recursively partitioned into four equal sized spaces. If the partitions are repeated by L times, 4^L equal sized spaces are obtained, and each space is represented by L-bit prefix pairs. Each rectangular search space is mapped into a node in a quad-tree. In other words, the entire space is mapped into the root node of the quad-tree, and four equal-sized quadrants which partition the entire space are mapped into four children of the root node, and so on. The AQT defines a crossing filter as a rule which spans at least one dimension of the rectangular space. The rules included in crossing filter set (CFS) are stored into the corresponding quad-tree node. The AQT is a very efficient data structure in the sense that it defines the geometrical representation of rules and maps each rule into a quad-tree node. However, it has some problems which prevent from being practically used. The classification speed is directly related to the depth of the quad-tree, and the depth of the quad-tree in the AQT is usually W, where the W is the maximum length of prefixes. Moreover, the constructed quad-tree in the AQT is very sparse, and this means that the quad-tree has a lot of empty internal nodes which waste the memory space.

In recent research, by analyzing the characteristics of real classifiers, Baboescu et al. have reported that efficient two-dimensional algorithms can be effectively used for multidimensional packet classification [16]. According to their observation on real classifiers, if we filter rules considering the source and the destination prefixes at the same time, less than five rules are remained as candidate matches for more than 95% of incoming packets.

Based on Baboescu's approach, in this paper, we have proposed an efficient two-dimensional algorithm which filters rules using the source and the destination prefixes and performs the linear search for the candidate rules. Our proposed algorithm is based on the recursive space decomposition in the AQT, but our approach primarily considers the priorities of rules in building the quad-tree. In other words, in each rectangular space, among the rules belonged to a rectangular space, the highest priority rule is stored in the first place into the corresponding quad-tree node and then the rules included in the CFS are stored. In this way, higher priority rules are stored in a higher level of the quad-tree, and hence they are compared earlier. Empty internal nodes in the AQT are completely removed in our proposed tree since each node includes at least a rule which is the priority rule, and hence the depth of the quad-tree is reduced in our proposed algorithm compared with the AQT.

3. Space decomposition

3.1. Space decomposition and quad-tree

Among the fields which compose rules, each of the source and the destination prefixes can be considered as a range in the geometric point of view. Hence two fields can be expressed as a rectangle in a two-dimensional (2D) space, and d fields can be expressed as a hyper-rectangle in a d-dimensional space. In this paper, a rule is considered as a rectangle in a 2D space defined by the prefix pair.

If W is the maximum prefix length, the size of entire search space is the size of $2^W \times 2^W$. Assuming that the prefix pair of a rule is (S, D) and the length of prefix S and prefix D is i and j, respectively, the rule can be expressed as a rectangle with the size of $2^{W-i} \times 2^{W-j}$.

An incoming packet which has the source and the destination IP addresses can be represented by a point (the smallest rectangle with size $2^0 \times 2^0$) in the 2D space, and the rectangles which contain the point are the matched rules. Since rectangles can be overlapped in the 2D space, a packet which is represented by a point can be matched by multiple rules. In this case, the rule with the highest priority is selected as the final match.

Each rectangular 2D space is mapped into a node in a quad-tree, i.e., the entire space is mapped into the root node of the quad-tree, and four equal-sized quadrants

which partition the entire space are mapped into four children of the root node, and so on. Figs. 1 and 2 represent the correspondence between the recursive space decomposition and the quad-tree. Fig. 1 shows the recursive decomposition of 2D space while Fig. 2 shows the corresponding quad-tree. The first rectangle in Fig. 1 represents the entire search space and corresponds to the root node in Fig. 2. The second four quadrants by first decomposition in Fig. 1 correspond to the four child nodes of the root node in Fig. 2. While the entire search space is represented by the wildcard prefix pair (*, *), the first decomposed quadrants are represented by the prefix pairs (0*, 0*), (0*, 1*), (1*, 0*), and (1*, 1*), respectively.

3.2. Crossing filter (CF)

Fig. 3 shows the relationship between the search space and a rule rectangle which can be positioned in the 2D space. A rule can be called a filter in 2D space since it allows candidate matches to get through. In Fig. 3, a space A has the size of $2^k \times 2^k$, and a prefix pair (S_i, D_j) with lengths i and j bits is represented by a filter F. If the space A and the filter F is not related as in Fig. 3(a), they are disjoint. If A is identical to F as in Fig. 3(b), A does not need to be further decomposed for F. If both prefix length of F is longer than k ($i > k, j > k$), then A need to be decomposed for F to be stored at a quad-tree node in a lower level. If one of prefix lengths is equal to k and the other prefix length is longer than k ($i > k, j = k$ or $i = k, j > k$), then filter F crosses one of two dimensions as in Figs. 3(d) and (e). Crossing filters (CF) are defined as the filters which cross a space in at least a dimension, and the crossing filters are stored at a quad-tree node corresponding to the space A. Since a space is decomposed recursively by half in each dimension, the case when any filter crosses over the boundary of A as shown in Fig. 3(f) would not happen.

4. Proposed algorithm

4.1. Proposed priority-based quad-tree (PQT)

The AQT maps a search space into a quad-tree node using the recursive space decomposition and stores the crossing filters into the quad-tree node. However, the AQT does not consider the priorities of rules in building the quad-tree. The priority information of rules can be effectively used in building the efficient quad-tree. In this paper, we propose a priority-based quad-tree (PQT) for packet classification.

4.1.1. Building procedure

The basic concept in building the proposed PQT is to store the highest priority rule among the rules belonged to a rectangular space into the corresponding node. In order words, from the root node which corresponds to the entire space, the rule with the highest priority is stored into the root node. If there are crossing filters for the entire space which are the rules with the length of the source prefix or the destination prefix 0, then they are additionally stored in the root node. The space is decomposed into four equal-sized spaces, and the rule with the highest priority among the rules included in the decomposed space is stored in the node corresponding to the space. Crossing filters, if exists, are also stored in the node. If multiple rules are stored in a node, then these rules are connected by a linked list in the order of the priority. The space decomposition is repeated until every rule is stored.

Since the AQT only stores crossing filters at each node, a rule is stored at the node in the level of the minimum length

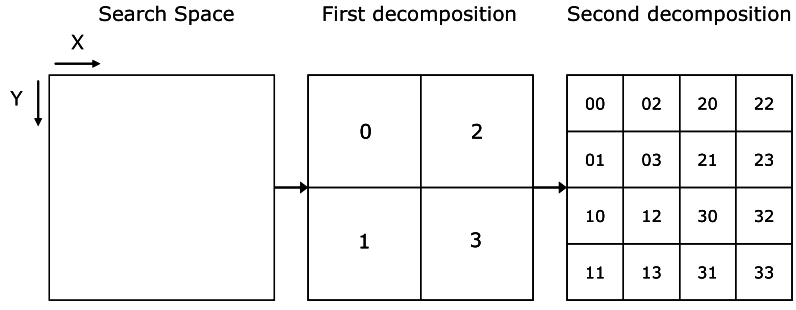

Fig. 1. Recursive decomposition of the 2D space.

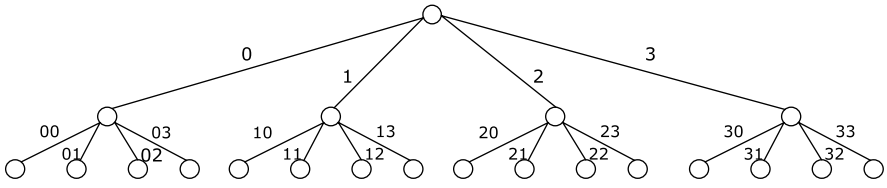

Fig. 2. Quad-tree.

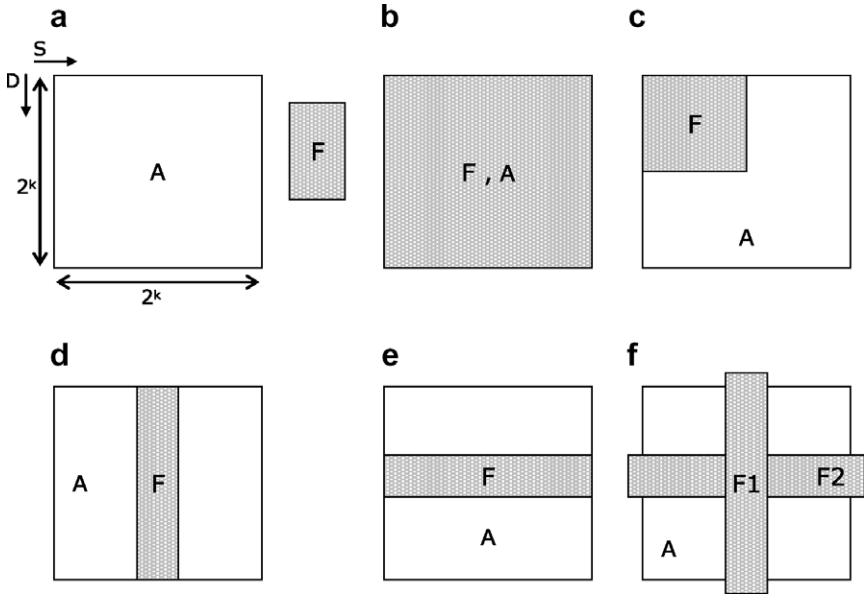

Fig. 3. Relationship between the search space and the 2D filter. (a) Filter F and Square A are disjoint. (b) Filter F = (S_i, D_j), and $i = k$, $j = k$. (c) Filter F = (S_i, D_j), and $i > k$, $j > k$. (d) Filter F = (S_i, D_j), and $i > k$, $j = k$. (e) Filter F = (S_i), D_j, and $i = k$, $j > k$. (f) Filter F1 = (S_i, D_j, and $i > k$, $j < k$; filter F2 = (S_i, D_j, and $i < k$, $j > k$.

of the two prefixes. If a prefix pair of a rule is (S_i, D_j) with the length of i bits and j bits, then the rule is stored at a node in the level of min(i, j). Hence if the maximum of min(i, j) is W, then the tree depth would be W in the AQT. Moreover, if there are rules with long prefix lengths, many empty nodes will be generated since the space decomposition needs to be performed until the crossing filter condition is satisfied.

On the other hands, in the proposed PQT, there will be no empty node and the tree depth is greatly reduced since the highest priority rule belonged to a decomposed space is stored in the corresponding node even if it does not satisfy the crossing filtering condition. According to [16], source and destination prefixes which constitute rules mostly have the lengths of 0, 16, 24, and 32 in the classification tables of real routers. If the AQT is applied for actual classification tables, the tree depth is typically greater than 30. In the PQT, the tree depth is reduced significantly, since the rules with the prefix length of greater than 16 can be stored in a much higher level by their priorities.

Table 1 shows an example classification table which is composed of 12 rules. The source prefix, the destination prefix, the source port number, the destination port number, and the protocol type are the fields that compose of the classification table. Rules with smaller rule number assume to have higher priorities. Fig. 4 represents the 2D rectangular space that the source and the destination prefixes of the rules in Table 1 correspond to. Fig. 5 shows the quad-tree built by the AQT, and Fig. 6 shows the quad-tree built by the proposed PQT. In Fig. 6, the rules represented with bold-italic such as R0, R2, R4, R5, and R9 are the rules stored by the priorities in the nodes. In other words, the rule R0 is stored into the root node because it has the highest priority, and the rule R4 is stored at a node in the level 1 since R4 has the highest priority in the lower-right quadrant which corresponds to the prefix

Table 1
An example classification table

Rule No.	Src. prefix len	Src. prefix	Dst. prefix len	Dst. prefix	Src. port No.	Dst. port No.	Protocol type
0	5	10100*	2	01*	>1024	>1024	*
1	5	11110*	0	*	80	80	TCP
2	1	0*	5	11100*	80	80	UDP
3	1	0*	5	11100*	88	88	TCP
4	3	111*	5	11110*	*	>1024	UDP
5	5	01110*	4	0111*	80	80	TCP
6	1	1*	5	10011*	21	21	TCP
7	0	*	5	10101*	80	80	*
8	4	1100*	0	*	88	88	TCP
9	4	1101*	4	1101*	*	*	*
10	3	010*	0	*	>1024	>1024	TCP
11	4	0110*	0	*	80	80	*

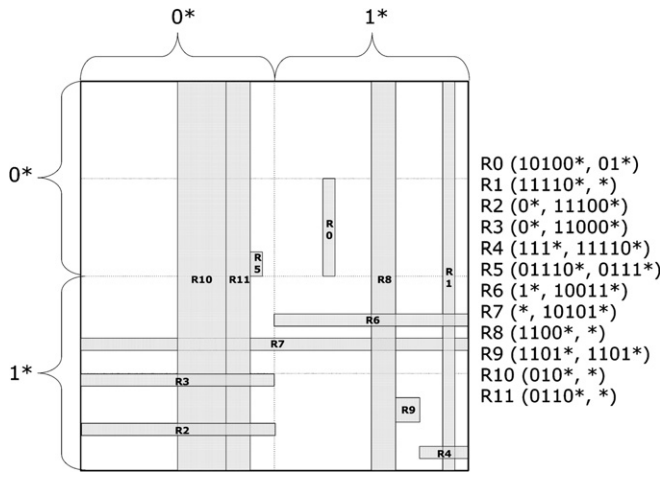

Fig. 4. Rules in a 2D space by the source and the destination prefixes.

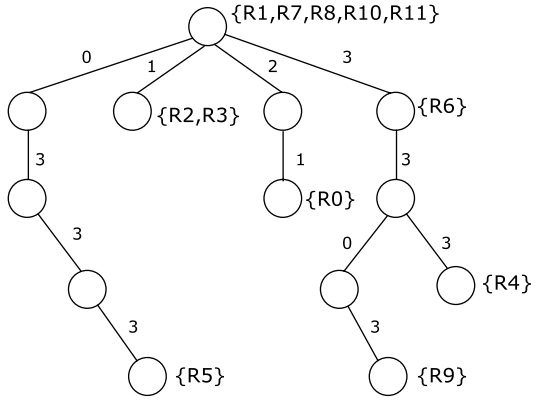

Fig. 5. The quad-tree by AQT.

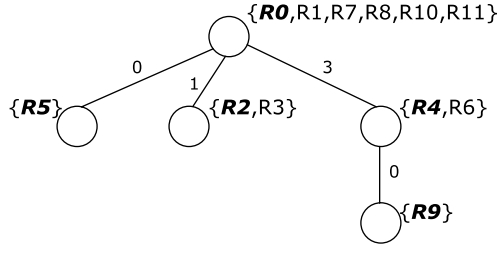

Fig. 6. The quad-tree by proposed PQT.

pair (1*, 1*). The rule R6 is stored at the same node as R4 since it satisfies the crossing filter condition. From the Fig. 6, we can see that the quad-tree by the proposed PQT has no empty node and a smaller tree depth than the quad-tree by the AQT. In Table 1, the * symbol means that the following bits are don't care.

The data structure for the proposed PQT has a tree table for storing the quad-tree structure and a rule table for storing the remaining fields. Table 2 shows the tree table for the proposed PQT for the classification table. The number of entries of the tree table is the same as the number of rules. The first five entries (rows) show the rules that are stored by priority in each node. If more than one rule is stored

Table 2
Tree table

Memory addr.	Rule No.	Stored by priority	Src. prefix len	Src. prefix	Dst. prefix len	Dst. prefix	Child 0		Child 1		Child 2		Child 3		Linked list	Rule table
							Valid	Ptr	Valid	Ptr	Valid	Ptr	Valid	Ptr	Ptr	Ptr
0	0	1	5	10100*	2	01*	1	1	1	2	0	–	1	3	5	0
1	5	1	5	01110*	4	0111*	0	–	0	–	0	–	0	–	–	5
2	2	1	1	0*	5	11100*	0	–	0	–	0	–	0	–	10	2
3	4	1	3	111*	5	11110*	0	–	0	–	0	–	1	4	11	4
4	9	1	4	1101*	4	1101*	0	–	0	–	0	–	0	–	–	9
5	1	0	5	11110*	0	*	0	–	0	–	0	–	0	–	6	1
6	7	0	0	*	5	10101*	0	–	0	–	0	–	0	–	7	7
7	8	0	4	1100*	0	*	0	–	0	–	0	–	0	–	8	8
8	10	0	3	010*	0	*	0	–	0	–	0	–	0	–	9	10
9	11	0	4	0110*	0	*	0	–	0	–	0	–	0	–	–	11
10	3	0	1	0*	5	11000*	0	–	0	–	0	–	0	–	–	3
11	6	0	1	1*	5	10011*	0	–	0	–	0	–	0	–	–	6

Rule No. 16bit	Stored by Priority 1bit	Src. Prefix Length 5bit	Src. Prefix 32bit	Dst. Prefix Length 5bit	Dst. Prefix 32bit	Valid0 1bit	Pointer0 16bit	Valid1 1bit	Pointer1 16bit	Valid2 1bit	Pointer2 16bit	Valid3 1bit	Pointer3 16bit	Linked List Pointer 16bit	Rule Table Pointer 16bit

Fig. 7. Entry structure of the tree table.

in a node, these rules are linked by a linked list in the order of priority. The entry structure of the tree table for storing quad-tree structure is shown in Fig. 7. The first bit *stored by priority* indicates whether it is stored into the node by priority or by crossing filter condition. The source prefix length and destination prefix length are stored with the source and the destination prefixes. There are four *valid* and *pointer* fields to store information about four child nodes. If the *valid* bit is 0, there is no valid child in that position. If the *valid* bit is 1, the *pointer* field represents the memory pointer of the valid child. The *linked list pointer* is used to point to the next rule that is stored in the same node. Only when the source and the destination prefixes are matched with an incoming packet in the search procedure, the rule table search is invoked to compare with remaining fields of the rule. The *rule table pointer* is used to point the corresponding rule table entry.

Table 3 shows the rule table for the proposed PQT. The number of entries in the rule table is the same as the number of rules. The rule table entry structure for storing the remaining fields of rules are shown in Fig. 8. Since port numbers can be represented by a range, the start and the end of the range should be represented. The structure of the rule table is simple, in which, the information about the source port range, the destination port range, and the protocol number is stored.

Table 3
Rule table

Rule No.	Src. port No.	Dst. port No.	Protocol type
0	>1024	>1024	*
1	80	80	TCP
2	80	80	UDP
3	88	88	TCP
4	*	>1024	UDP
5	80	80	TCP
6	21	21	TCP
7	80	80	*
8	88	88	TCP
9	*	*	*
10	>1024	>1024	TCP
11	80	80	*

Rule Number 16bit	Src. Port Wildcard 1bit	Src. Port Start 16bit	Src. Port End 16bit	Dst. Port Wildcard 1bit	Dst. Port Start 16bit	Dst. Port End 16bit	Protocol Wildcard 1bit	Protocol 8bit

Fig. 8. Entry structures of the rule table.

4.1.2. Search procedure

The search procedure to find the best match or the highest priority rule for a given packet starts at the first entry (root node) of the tree memory. As the search goes to the next lower level in the proposed PQT, the search space is reduced to 1/4. When multiple rules are stored in a node, the rules are searched linearly using the linked list in the tree table.

The search procedure is represented by a flowchart in Fig. 9. The current best match rule (BMR) is initially set to N which is the same as the lowest priority rule. The source and the destination IP addresses of the incoming packet are compared with the prefixes of the rule that is stored in the tree table. Only when they are matched, search goes to the rule table using the *rule table pointer* in order to make sure the remaining fields are also matched. If the remaining fields are also matched, the current BMR is updated with the matched rule number. If the remaining fields are not matched, the search goes back to the tree table. If the source and destination IP addresses of the incoming packet in the tree table are not matched, the *linked list pointer* is examined to see whether there exists any other rule stored in the same node. If the *linked list pointer* is valid, the comparison is repeated for the entries linked by the *linked list pointer*.

If the matched rule was stored in that node because of its priority, then it is the final BMR since it is the highest priority rule among the rules included in the search space, and hence the search is over. If the matched rule was stored because of the crossing filter condition, there could be a matched rule with a higher priority at a lower level. Hence the current BMR is remembered, and the search procedure is continued to the child node that is pointed by the *child pointer*. The child node is determined by the next bit of the source and the destination addresses of the incoming packet. Since the rule stored by the priority is firstly compared among the rules stored at each node, if the current BMR has higher priority than the priority of the rule that is stored because of its priority in the node, the current BMR becomes the final matched BMR and the search is over. If the priority of the newly matched rule is higher than the current BMR, and the BMR should be updated. If there is no more child pointer to follow, then the current BMR is the final BMR and the search is completed.

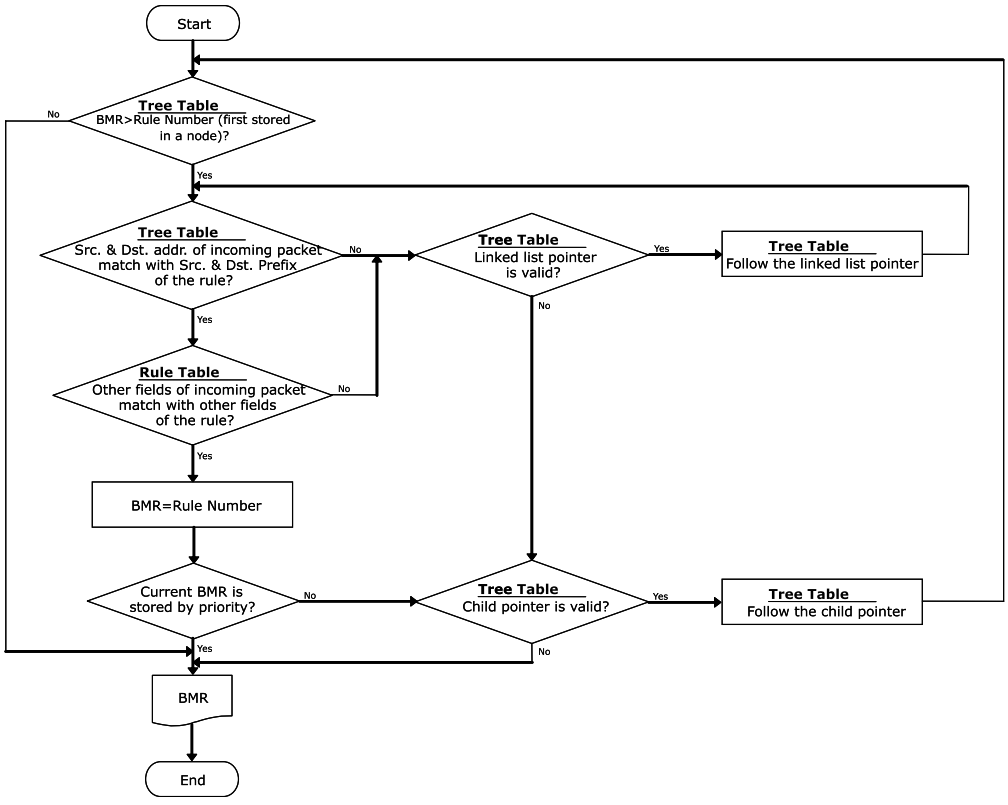

Fig. 9. Flow chart for search procedure.

4.1.3. Update procedure

The incremental update of the classification table is partially possible in our proposed algorithm. First, we consider removing an existing rule. If a rule needs to be removed, the rule is searched and the matched entry is removed except the *child pointers* and the *linked list pointer*. The *child pointers* and the *linked list pointer* of the removed rule should be kept for search even though the rule itself is removed. For dynamically changed classification tables, repeated rule deletion will cause the waste of memory space and it is not desirable, and hence the whole table should be rebuilt in a regular interval.

Now we consider inserting a new rule. By searching the tree table, we need to locate a node for the new rule to be stored and determine whether the new rule should be stored because of its priority into the node. In order words, if the new rule has a higher priority than the priority rule stored in the located node, the existing rule (stored by priority) in the located node is moved to an empty entry. The new rule is newly stored in the located node, and the *linked list pointer* needs to be updated to point the moved node. Therefore, inserting a new rule in this case results in two priority rules stored in a node, but it does not affect the search procedure. If the new rule should be stored into the node by crossing filter condition, it can be stored in an empty entry in the tree table, and the order of rules in the linked list of the node should be updated based on the priority order. The maximum number of rules affected by an insertion is limited by the maximum number of rules stored in a node.

4.1.4. Optimization of the Proposed Algorithm

Since the number of entries of the tree memory and the rule memory is the same, the tree memory and the rule memory can be combined and implemented with a single memory. The optimized structure of our proposed algorithm shows the improved search performance since two memory accesses are replaced by a single memory access.

5. Simulation and performance evaluation

Class-bench [1] is known to provide classification tables similar to real classifiers used in the Internet routers and input traces corresponding to each classification table. We have performed simulations using three different types of classification tables generated using class-bench, access control lists (ACL), firewalls (FW), and IP chains (IPC).

Since search speed is highly dependent on the number of memory accesses considering that the memory access is the most time consuming operation in search process, we have counted the number of memory accesses in searching the highest-priority matching rule. Table 4 shows the performance evaluation result in terms of the memory requirement (M), the worst-case number of memory accesses (T_{wst}), and the average number of memory accesses (T_{avg}) according to the number of rules (N) in each classification

Table 4
Performance evaluation result of the proposed algorithm

	N	M (kbyte)	T_{wst}	T_{avg}
ACL1K	958	30.7	75	35.6
ACL5K	4659	149.1	113	59.6
ACL10K	9735	311.5	127	86.3
FW1K	870	27.9	291	197.9
FW4K	4093	98.1	748	446.4
FW5K	4343	139.2	999	571.1
IPC1K	988	31.6	106	73.6
IPC5K	4467	143.0	295	202.1
IPC10K	9491	259.6	477	324.6

table. We have tried to build classification tables in sizes of 1K, 5K, and 10K, but the class-bench does not produce the classification tables in the exact size, and hence the number of rules included in each type of classification tables is a little different from the exact size. Moreover, for the FW type classifier, the class-bench does not provide large size classification tables, and hence we have used classification tables with the sizes of 1K, 4K, and 5K.

The memory requirement is the summation of the tree table and the rule table, and it is proportional to the number of rules regardless of the type of classification tables since the entry numbers of the tree and rule tables are exactly same as the rule numbers. The number of memory accesses is obtained using the optimized structure of our proposed algorithm in searching the highest-priority matching rule. The worst-case number of memory accesses is counted in order to find out the worst-case bound in search.

Table 4 shows that the performance of the ACL type is better than other types in terms of the worst-case number of memory accesses and the average number of memory accesses. Note that for the rules stored in a single node, linear search is performed in both of our proposed algorithm and the AQT algorithm. Since ACL type does not have many wild-card rules that should be stored at the root node, it requires a smaller number of memory accesses in linear search at the root node than other types. In case of FW type, it has many wild-card rules as well as many crossing filter rules that should be stored in a node, and hence a lot of memory accesses are required in the linear search. This is the reason that it shows the worst performance in the required number of memory accesses. The search performance of the IPC type is in-between. The simulation results in Table 4 show that the required memory size and the number of memory accesses in the proposed algorithm are moderately grown as the size of classification tables is grown, and hence the proposed algorithm provides the scalability toward large classifiers.

Table 5–7 show the performance comparison with other algorithms for the different types of classification tables. As shown in the tables, for small size (1K) classification tables, the proposed algorithm always requires the smallest memory size. For large size (5K and 10K) classification tables, the proposed algorithm requires the comparable size of memory with the H-trie algorithm except that H-trie requires the smallest memory for ACL10K type. The bit-vector algorithm consumes a huge amount of memory for large classifiers, and hence it does not provide a good scalability.

Table 5
Performance comparison with other algorithms for ACL type

	ACL1K			ACL10K		
	M (kbyte)	T_{wst}	T_{avg}	M (kbyte)	T_{wst}	T_{avg}
H-trie	82.9	124	77.2	156.0	152	94.9
Bit vector	153.3	68	66.0	11.6 M	73	65.3
AQT	57.7	64	38.6	552.5	116	79.1
Proposed	30.7	75	35.6	311.5	127	86.3

Table 6
Performance comparison with other algorithms for FW type

	FW1K			FW5K		
	M (kbyte)	T_{wst}	T_{avg}	M (kbyte)	T_{wst}	T_{avg}
H-trie	39.4	117	52.1	119.1	162	69.2
Bit vector	111.9	318	196.6	2.34 M	1044	738.8
AQT	36.0	444	369.3	491.3	1193	660.5
Proposed	27.9	291	197.9	139.2	999	571.1

Table 7
Performance comparison with other algorithms for IPC type

	IPC1K			IPC10K		
	M (kbyte)	T_{wst}	T_{avg}	M (kbyte)	T_{wst}	T_{avg}
H-trie	121.6	128	71.9	291.2	222	96.3
Bit vector	154.3	80	63.6	8.0 M	401	275.1
AQT	72.9	119	94.5	424.6	697	573.3
Proposed	31.6	106	73.6	259.6	477	324.6

The performance of the worst-case and the average numbers of memory accesses varies depending on classification types. For example, in the average number of memory accesses, the proposed algorithm is the best in the ACL1K type, the H-trie algorithm is the best in the FW types and the IPC10K type, and the bit-vector algorithm is the best in the ACL10K and IPC1K type.

Comparing the proposed algorithm with the AQT, the proposed algorithm requires smaller memory size than AQT in all cases as expected. In terms of the worst-case and the average number of memory accesses, the proposed algorithm shows better performance than the AQT for the FW types and the IPC types, but the worse performance for ACL10K type. It can be understood that since the proposed algorithm additionally stores a priority rule to the crossing filter rules in the AQT algorithm, if the probability that given inputs do not match to the priority rule is high, the extra memory access for the priority rule deteriorates the search performance.

Summarizing the simulation result, the proposed algorithm and the hierarchical trie (H-trie) algorithm can be the solutions for the packet classification in overall performance.

6. Conclusions

The next-generation router demands packet classification capability, and this paper proposed an efficient packet classification algorithm based on recursive space decomposition. The proposed algorithm builds an efficient quad-tree by primarily considering the priority of rules in the recursive space decomposition process. The highest priority rule that is included in the search space is stored in the corresponding node of the quad-tree, even though it does not satisfy the crossing filter condition. By this way, empty nodes are completely removed and the tree depth is reduced. None of previous packet classification algorithms satisfies all the performance metrics such as search speed, required memory size, incremental update, and the scalability toward large classifiers. They make compromise among the metrics. The proposed packet classification algorithm achieves very small memory requirement. Depending on the type of the classification tables, the search speed of the proposed algorithm varies but shows reasonably good performance in all cases. The simulation results show that the required memory size and the number of memory accesses in the proposed algorithm are moderately grown as the size of classification tables is grown, and hence the proposed algorithm is also very good in the scalability.

Acknowledgement

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

References

[1] D.E. Taylor, J.S. Turner, "ClassBench: a packet classification benchmark," INFOCOM 2005, vol. 3, 13–17 March 2005, pp. 2068–2079.
[2] P. Wang, C. Chan, S. Hu, C. Lee, W. Tseng, High-speed packet classification for differentiated services in next generation networks, IEEE Trans. Multimedia 6 (6) (2004) 925–935.
[3] H. Jonathan Chao, Next generation routers, Proc. IEEE, 90(9) (2002) 1518–1558.
[4] P. Gupta, N. Mckeown, Algorithms for packet classification, IEEE Netw. 15 (2) (2001) 24–32.
[5] M.M. Buddhikot, S. Suri, M. Waldvogel, Space decomposition techniques for fast layer-4 switching, in: Proceedings of Protocols for High Speed Networks, Aug. 1999, pp. 25–41.
[6] F. Geraci, M. Pellegrini, P. Pisata, L. Rizzo, Packet classification via improved space decomposition technique, in: Proceedings of IEEE INFOCOM, vol. 1, Mar. 2005, pp. 304–312.
[7] E. Spitznagel, D. Taylor, J. Turnar, Packet classification using extended TCAMs, in: Proceedings of ICNP, 2003, pp. 1–12.
[8] D. Shah, P. Gupta, Fast updating algorithms for TCAM, IEEE Micro 21 (1) (2001) 36–47.
[9] V. Srinivasan, G. Varghese, S. Suri, M. Waldvagel, Fast and scalable layer four switching, in: Proceedings of ACM SIGCOMM, Aug. 1998, pp. 191–202.
[10] Y. Jung, H. Lim, A two-dimensional binary prefix tree for packet classification, in: Proceedings of IEEE HPSR2005, May 2005.
[11] P. Gupta, N. Mckeown, Classification using hierarchical intelligent cuttings, IEEE Micro 20 (1) (2000) 34–41.
[12] V. Srinivasan, S. Suri, G. Varghese, Packet classification using tuple space search, in: Proceedings of ACM SIGCOMM'99, Aug. 1999, pp. 135–146.
[13] J. Lunteren, T. Engberson, Fast and scalable packet classification, IEEE J. Selec. Areas Commun. 21 (4) (2003) 560–571.
[14] F. Baboescu, G. Varghese, Scalable packet classification, IEEE/ACM Trans. Netw. 13 (1) (2005) 2–14.
[15] X. Sun, S.K. Sahni, Y.Q. Zhao, Packet classification consuming small amount of memory, IEEE/ACM Trans. Netw. 13 (5) (2005) 1135–1145.
[16] F. Baboescu, S. Singh, G. Varghese, Packet classification for core router: is there an alternative to CAMs?, in: *IEEE INFOCOM 2003*, vol. 1, Mar. 2003, pp. 53–63.

Hyesook Lim received the B.S. and the M.S. degrees at the Department of Control and Instrumentation Engineering of Seoul National University, Seoul, Korea, in 1986 and 1991, respectively. She got the Ph.D. degree at the Electrical and Computer Engineering from the University of Texas at Austin, Austin, Texas, in 1996. From 1996 to 2000, she had been employed as a member of technical staff at Bell Labs in Lucent Technologies, Murray Hill, NJ. From 2000 to 2002, she had worked as a hardware engineer for Cisco Systems, San Jose, CA. She is currently an associate professor in the Department of Information Electronics Engineering, Ewha Womans University, Seoul, Korea, where she does research on router design issues such as IP address lookup and packet classification and on hardware implementation of various network protocols such as TCP/IP and Mobile IPv6.

Min Young Kang received the B.S. degree and the M.S. degree at Information Electronics Department from Ewha Womans University, Seoul, Korea, in 2005 and 2007, respectively. She is now with Samsung Electronics, Korea. Her research interests are MAC protocols for wireless LAN and VLSI design.

Changhoon Yim received the B.S. degree at the Department of Control and Instrumentation Engineering of Seoul National University, Korea, in 1986, the M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology, Korea, in 1988, and the Ph.D. degree in Electrical and Computer Engineering from the University of Texas at Austin, USA, in 1996. He was a research engineer working on HDTV at Korean Broadcasting System, Korea, from 1988 to 1991. From 1996 to 1999, he was a Member of Technical Staff in HDTV and Multimedia Division, Sarnoff Corporation, NJ, USA. From 1999 to 2000, he worked at Bell Labs, Lucent Technologies, NJ, USA. From 2000 to 2002, he was a Senior Software Engineer in KLA-Tencor Corporation, CA, USA. From 2002 to 2003, he was a Principal Engineer in Digital Media R&D Center, Samsung Electronics, Suwon, Korea. Since 2003, he has been an assistant professor in the Department of Internet and Multimedia Engineering, Konkuk University, Seoul, Korea. His current research interests include multimedia communication, video compression, and multimedia network.