Ordered Lookup with Bypass Matching for Scalable Per-flow Classification in Layer 4 Routers

Ying-Dar Lin   Huan-Yun Wei   Kuo-Jui Wu

Department of Computer and Information Science
National Chiao-Tung University
Hsinchu, Taiwan

Tel: +886-3-5731899
Fax: +886-3-5721490
Email: ydlin@cis.nctu.edu.tw

Abstract

In order to provide different service treatments to individual or aggregated flows, layer 4 routers in Integrated Services networks need to classify packets into different queues. The classification module of layer 4 routers must be fast enough to support gigabit links at a rate of millions of packets per second. In this work, we present a new software method OLBM to lookup multiple fields of a packet, in a dynamically pre-defined order, against the classification database. This algorithm can classify packets at a rate of well over 1 million packets per second while scaling to support more than 300K flows. Complexity analysis and experiment measurements are also presented in this study.

Index Terms: classification, layer 4 router, packet filtering, lookup, match, scalability

1 Introduction

In order to support QoS in the Integrated Services (IntServ) [1][2][3] networks, several traffic control modules need to be added into the layer 4 routers which examine not only IP headers but also transport-layer headers. The admission control, in the control-plane, and the classifier, scheduler, in the user-plane, are three basic modules for QoS traffic control. The classifier, which *distinguishes* incoming packet into different flows, becomes essential. Besides QoS processing, firewall and VPN [4] for example, also need the classifier to classify packets based on multiple fields. In this work, we focus on the classification for per-flow QoS processing concerning five fields: src/dest port, protocol ID, and src/dest IP.

There are three key components in the classification module: the filter database, the classification database, and the classifier. The filter database consists of filtering rules updated by the admission control module at run-time. Then the filter database inserts its information to the classification database as search indexes for the classifier to refer. Figure 1 shows the role of the classification module and its process.

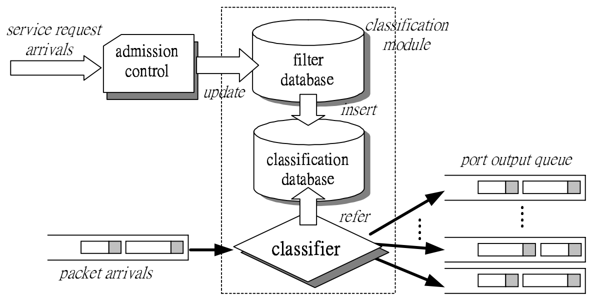

Figure 1: Classification module and its process.

Two methods have been proposed for fast classification. Table 1 is a summary comparing these two methods. Both these methods lookup *all* the five fields of a packet against *each* filtering rule. In addition, they do not seem to be scalable enough to meet the high scalability requirement. Thus, we provide a scalable method :*Ordered Lookup* with *Bypass Matching* (OLBM). Ordered Lookup (OL) may save unnecessary work without looking up all the five fields. Bypass Matching (BM) can help to finish the OL more quickly.

	High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching [5]	**Fast and Scalable Layer 4 Switching [6]**
Method	Multi-dimension matching Bit-parallelism	Dest-src tries Cross-producting
Implement	Hardware	Software
Scalability	Thousands to tens of thousands of filters	Tens of thousands of fileters
Throughput	About 1 Mpps	At least 1 Mpps
Features	Using special hardware Low memory space	Using general processor High memory space

Table 1: Summary of two published methods.

The rest of this work is organized as follows. We give our design objectives and motivation in section 2. Section 3 presents the OLBM algorithm. Section 4 draws the analytical results of the worst case. Experimental performance studies are given in section 5. Our conclusion is given in section 6.

2 Objectives and Motivation

2.1 The Design Objectives OLBM Algorithm

There are three objectives for designing a classification algorithm:

1. Throughput: The algorithm must be able to process at least 1 million packets per second. For an OC-3 link of 155 Mbps, considering that all incoming packets are as small as 64 bytes, the classifier must process 317,440 packets in one second. Thus, for a router with multiple interfaces, the processing rate of over 1 million packets per second is required.
2. Scalability: The algorithm must be scalable. Recent studies have shown that an OC-3 link might have an average of 240K flows [7], which implies that there would be as many as 240k filtering rules in the data structures of a classifier.
3. Extensibility: The algorithm must be flexible enough to be extended to lookup using more fields, or even payload, against even IP prefix type filters.

2.2 The Design Motivation of Ordered Lookup

The motivation to design OL is that the classifier could use fewer fields of the packet header to find the matched filtering rule, if any. It needs fewer memory references and CPU instructions, which means that the classifier can achieve the same function faster.

This algorithm is designed for software implementation. Since the classifier has to sequentially compare all the five fields of the packet header with the filtering rules, it can pre-define an order for these fields to lookup, and may find the matched filtering rule before all the five fields are looked up.

Now how to design this pre-defined lookup order to minimize the classification time is an important issue. We propose that the classifier selects the field that the filtering rules are distributed most evenly, and compares the packet against that field first. We will describe the strategies to determine the lookup order according to the distribution of the filtering rules in section 3.3.

3 Ordered Lookup with Bypass Matching

3.1 Data Structures of the Classification Database

The date structures of our classification database are constructed by two primitive tables, named 64k-table and 256-table, as shown in Figure 2. It takes one 64k, one 64k, one 256, two 64k, and two 64k tables for the src/dest port, protocol ID, and src/dest IP fields, respectively. The index of the tables corresponds to the value of the field. Each table entry stores a list of 3-byte pointers to the filtering rules in the filter database. The data structures for the fields of src/dest IP address require more explanations. The index value of each entry in the first 64k-table represents a 16-bit IP prefix. The index value of each entry in the second 64k-table represents a 16-bit IP suffix.

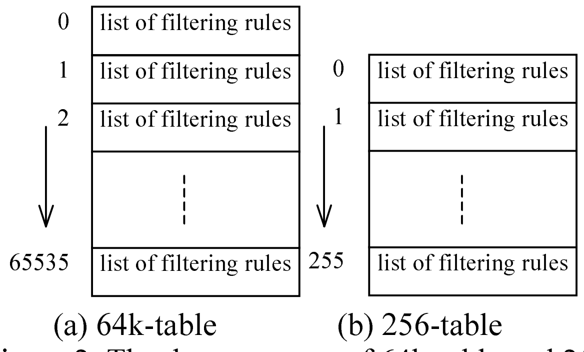

Figure 2: The data structure of 64k-table and 256-table.

3.1.1 Insertion Operations

A filtering rule is inserted into an entry of the table for a specific field, according to its value for that field. The insertion operation for src/dest IP address field needs more explanations. When a filtering rule is to be inserted into the classification database, it is inserted into the first and the second 64k-table, using its 16-bit IP prefix part and the 16-bit IP suffix, respectively. The advantage of this scheme is that we do not have to use a 32-bit flat table but needs merely two 64k-table to index an IP address field.

3.1.2 Lookup Operations

To clarify the description of the lookup operation, we first define some terms.

Definition: field-matched filtering rule

A *field-matched filtering rule* is a filter retrieved by looking up only one field of the packet header.

Definition: matched filtering rule

A *matched filtering rule* is a filter containing exactly the same values for the five fields as those of the incoming packet at the end of the classification process.

Definition: partially matched filtering rule

A filter is said to be a *partially matched filtering rule* if some of its fields are the same as those in the incoming packet.

When the classifier wants to get the field-matched filtering rules for src/dest IP address, it uses the 16-bit prefix part and 16-bit suffix part of the IP address in the packet header as indexes to the two basic tables, respectively, to retrieve the corresponding filtering rules. If the sets of filtering rules found in the first 64k-table and the second 64k-table are denoted as FR1 and FR2, respectively, the classifier performs (FR1 ∩ FR2) to retrieve field-matched filtering rules for the IP address field.

3.2 Ordered Lookup Algorithm

We introduce the OL algorithm here. First, we define some terms to clarify the algorithm.

Definition: ordered lookup
Ordered lookup is the process to perform *lookup* operation with the five fields of a packet, one by one, in a pre-defined order.
Definition: lookup operation
A *lookup* operation is to take the value of one field of a packet header as the index into the tabular data structures for the field.
Definition: match operation
A *match* operation is to compare all the five fields of a packet with the corresponding fields of one filtering rule.

In the following we describe the OL algorithm. The classifier *lookups* the first field of a packet in the pre-defined order, against the tabular data structures and obtains a set of partially matched filtering rules, which forms a *candidate* set of *matched filtering rules*. If the size of the set is 0, surely the packet does not match any filtering rules; if the size of the set is 1, the classifier has to perform a *match* operation with the packet and that filtering rule to see whether the packet really matches the filtering rule; if the number of the set is more than 1, the classifier continues to *lookup* the second field and obtains another set of filtering rules. If the number of this set is more than 1, the classifier intersects this set with the candidate set and result in a new candidate set of filtering rules. If the new candidate set contains only one filtering rule, a direct *match* is performed.; if there are more than one filtering rule, subsequent lookups and intersections are performed. Table 2 describes the required functions and variables in our pseudo code. Figure 3 shows the pseudo code of this algorithm.

Function	Description
lookup(PF, FF)	Lookup the field PF of a packet against the tabular data structures for the field FF
match(PKT, FR)	Match a packet PKT with a set of filtering rules FR
Variable	Description
Packet	The incoming packet
LO[0-4]	The lookup order of the five fields
CFR[ ]	The candidate set of filters rules for a packet
FR[ ]	One-field lookup resulting set of filters for a packet
Destination_port	The tabular data structures for the dest port number
Destination_IP	The tabular data structures for the dest IP address
Source_port	The tabular data structures for the src port number
Source_IP	The tabular data structures for the src IP address
Protocol_id	The tabular data structures for the protocol

Table 3: Function and variable descriptions of OL.

```
Algorithm: Ordered_Lookup
Input: packet, LO[0~4]     /* lookup order */
Output: CFR[ ]        /* candidate set of filtering rules */

CFR[ ] = NULL
for l = 0 to 4
  switch LO[l]   /* lookup the 5 fields by the given order */
    case Destination_Port:
      FR[ ] = lookup(packet.destination_port, Destination_port)
      goto Match
    case Destination_IP:
      FR[ ] = lookup(packet.dstination_IP, Destination_IP)
      goto Match
    case Source_Port:
      FR[ ] = lookup(packet.source_port, Source_port)
      goto Match
    case Source_IP:
      FR[ ] = lookup(packet.source_IP, Source_IP)
      goto Match
    case Protocol_Identifier:
      FR[ ] = lookup (packet.protocol_id, Protocol_id)
      goto Match
    Match:
      if sizeof FR[ ] = 0, return NULL
      if sizeof FR[ ] = 1 and match(packet, FR[ ]) = TRUE, return FR[ ]
      else CFR[ ] = CFR[ ] ∩ FR[ ]
      if l=4, return CFR[ ]    /* last lookup order */
      if sizeof CFR[ ] = 1 and match(packet, CFR[ ]) = TRUE, return CFR[ ]
  end switch
end for
```

Figure 3: Pseudo code of OL algorithm.

3.3 Decision Strategies for Lookup Order

It may happen that different lookup order will result in different classification speed. The best lookup order is the one that uses the least lookups on the average to find the matched filtering rule, if any. To minimize the number of lookups, it is straightforward that the classifier should first lookup the field in which the average number of field-matched filtering rules per table entry is the least among all the five fields. Because the classifier might get the minimal number of field-matched filtering rules on the average after each lookup, the lookup sequence is more likely to terminate in the midway.

There are other ways to determine a good lookup order. We define the average number of field-matched filtering rules per entry as *avg*, and the standard deviation of the number of field-matched filtering rules per entry as *sdv*. We provide the following three strategies for the classifier to determine the lookup order. The lookup order is according to the sorted results, from minimum to maximum, of the five fields.

MAF: Minimum Average-length First

The classifier first lookups the field that has the minimal *avg*. Thus the classifier is likely to get the least field-matched filtering rules on the average after each lookup. However, in the worst case the classifier may index to the entry that has the maximum number of filtering rules among all entries. This maximum number could be large if *sdv* of the field is large.

MSF: Minimum Standard deviation First

The classifier first lookups the tabular data structures for the field that has the minimal *sdv*. Thus the classifier is likely to get the similar number of field-matched filtering rules after each lookup.

MASF: Minimum *avg* and *sdv* first

The classifier first lookups the field that has the minimal *avg*sdv*. Thus the classifier may not only on the average case but also on the worst case get the least field-matched filtering rules after each lookup.

We compare the throughput of different lookup orders determined by these three strategies in section 5.

3.4 Bypass Matching

As mentioned in section 3.2, the classifier terminates the lookup process by direct match when the number of field-matched or partially-matched filtering rules is one. In fact, the lookup process might be terminated more quickly. Assume that the number of field-matched or partially-matched filtering rules is *k,* then the classifier matches the packet directly with the *k* field-matched or partially-matched filtering rules if the cost of *k* matches is less than that of the remaining lookups. We call this operation *bypass matching*, and define the maximum value of k that satisfies the above criteria for direct matching as K. Figure 4 is the modified pseudo code for the OLBM algorithm.

```
Match:
  if sizeof FR[ ] = 0 return NULL
  if sizeof FR[ ] <= K
        for each J in FR[ ]
                if match(packet, J) = FALSE
                        remove (J, FR[ ])
        return FR[ ]
  else CFR[ ] = CFR[ ] ∩ FR[ ]
  if I=4, return CFR[ ]    /* last lookup order */
  if sizeof CFR[ ] <= K
        for each J in CFR[ ]
                if match(packet, J) = FALSE
                        remove (J, CFR[ ])
        return CFR[ ]
```

Figure 4: Modified pseudo code for OLBM

The threshold K for bypass matching is a machine dependent threshold. We now describe how to determine the bypass matching threshold. There are two major kinds of CPU instructions in our algorithm. One is *memory reference*, and the other is *compare*, with their costs *Cm* and *Cc*, respectively. If there are k filters left after lookup some fields, whether to directly match k filters or continue to lookup the remaining fields is up to the following inequality:

$$k(Cm + 5Cc) < 2 \cdot Cm + k \cdot \frac{N}{E} \cdot Cc + (Cm + 5Cc)$$

where *E* is the number of entries in the tables for a field and *N* is the number of filters.

The left part of the inequality is the cost of k direct matching, each of which contains one memory reference to retrieve the filter and five comparisons to compare the five fields. The right part of the inequality is the least cost to lookup the remaining fields, which happens in the following situation: the next field lookup first use two memory reference to take the next field of the packet to index to the corresponding table, and retrieves (N/E) field-matched filtering rules, in the average case. Then the (N/E) filters intersect with the k filters to form a new candidate set of matched filtering rules, which costs $(k \cdot \frac{N}{E} \cdot Cc)$. The least cost situation happens in that the resulting candidate set contains only one filter, and its remaining cost is simply a match operation of that filter, which costs $(Cm + 5Cc)$.

Thus,

$$k < \frac{3Cm + 5Cc}{Cm + (5 - \frac{N}{E})Cc}$$

So the maximum number of k, denoted by K, equals to $\left\lfloor \frac{3Cm + 5Cc}{Cm + (5 - \frac{N}{E})Cc} \right\rfloor$.

In brief, once you have decided to run this algorithm on some machine, given the costs of memory reference and compare instructions, with the knowledge of current number of filtering rules N and current number of entries in the table, the machine dependent threshold K can be determined.

4 Complexity Analysis: Time and Space

Our OLBM algorithm is concerned with five fields. It has at most five lookups in the classification process. Because each field is basically the same, so we show the time and space complexity of this algorithm by analyzing a single field first.

The worst case happens when all of the filtering rules for this field are stored in only one table entry. After retrieving the field-matched filtering rules, the classifier either lookups into the next field and then intersects the resulting field-matched filtering rules with those from the previous lookup, if any, or matches the packet with the currently candidate set of matched filtering rules. Both of them take *O(N)* time in the worst case.

As we have mentioned that there are at most five lookups and four matches in our method, so the time complexity of lookups and matches for *F* fields are

*O(N*F)* and *O(N*(F-1))*. And the time complexity of our algorithm is *O(N*F)*. In IntServ networks filters are always in per-flow type. Thus the space complexity for our classification database is *O(N*F)*.

Table 3 shows the time and space complexity of three classification methods. The space complexity of ours is much lower than [5]. [6] needs a huge memory space and cannot scale to 300K filtering rules.

Description	Time Complexity	Space Complexity
Ordered Lookup with Bypass Matching	$O(N*F)$	$O(N*F)$
High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching [5]	$O(N*F)$	$O(N^2*F^2)$
Fast and Scalable Layer 4 Switching [6]	$O(\log W + \frac{W}{k})$	$O(N*W)$

Table 3: Time and space comparison of the methods.
(W denotes the maximum bit length of any destination or source prefix, k denotes the number of fields to be checked by the classifier.)

5 Performance Study

We have implemented and experimented our algorithm on two platforms. One is the Intel Pentium-II 350 Mhz CPU platform and the other is the Sun UltraSparc 300Mhz CPU platform. The hit ratio for arriving packets is 80%, i.e. only 80% of arriving packets will hit one filtering rule. The default strategy for deciding lookup order is MAF.

5.1 Memory Usage

Figure 5 shows the memory usage of our algorithm. Our classification algorithm uses quite reasonable size of memory, e.g. 10 MB for 300K filtering rules, while [6] uses 7.489 MB memory for only 20K filtering rules.

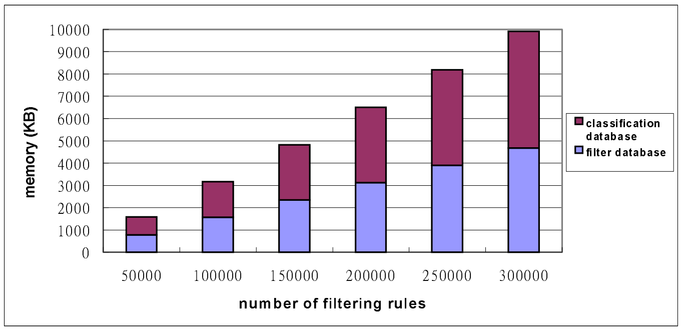

Figure 5: Memory usage of the ordered lookup algorithm.

5.2 Throughput

Figure 6 shows the throughput of OL and the improvement of BM with threshold K as 2. In this experiment the filtering rules are randomly generated. Note that without BM the throughput drops more rapidly because the number of lookups for each packet increases when the number of filtering rules increases. We can see that from Figure 7.

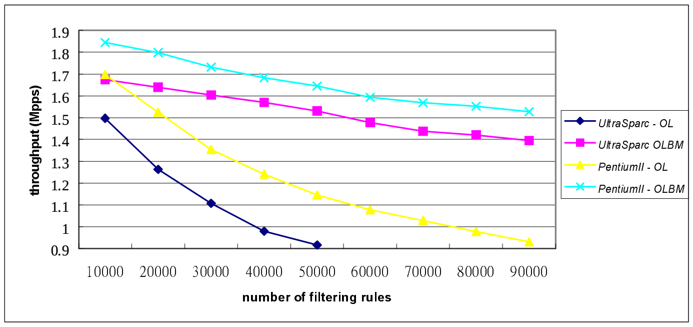

Figure 6: The effect of OL and OLBM on two platforms.

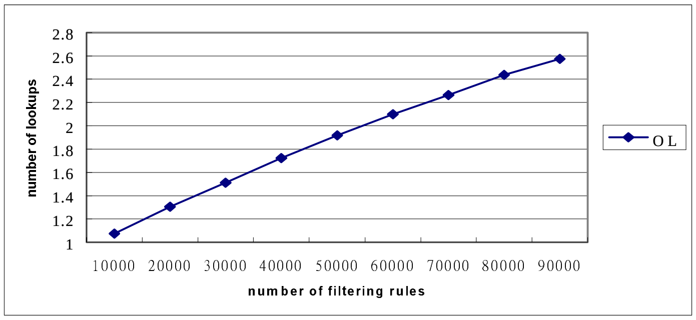

Figure 7: Number of lookups used by the OL.

5.3 Sensitivity to Locality

In routers both *incoming packets* and *filtering rules* might have some degree of address locality and could affect the performance of the classifier. We use three simple address locality models shown in the Table 4 to simulate the address patterns observed at a router.

	80% filters concentrate in	20% filters are
Model 1	30% of the address space	Randomly generated
Model 2	20% of the address space	
Model 3	10% of the address space	

Table 4: Three address locality models

From Figure 8 we can see that higher address locality leads to lower throughput, especially when the number of filtering rules is large. It is because the average number of field-matched filtering rules after each lookup increases, as both the address locality and the number of filtering rules increases. But overall, the throughput sensitivity to locality is not very high.

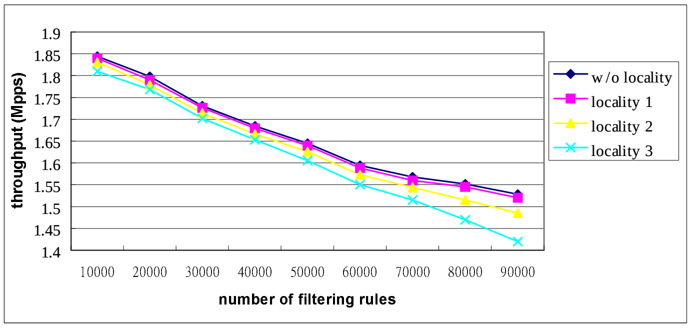

Figure 8: The effect of address locality (on P-II).

5.4 Decision Strategy for Lookup Order

We use the address locality model 1 in section 5.3 to generate the filtering rules for this experiment. In section 3.3 we provided three decision strategies for lookup order. In Figure 9 it shows that MSF and MASF result in almost the same throughput. The lookup orders decided by these two strategies are almost the same in the repeated runs. The standard deviation seems to be more dominant than the average. But the throughput of MAF is a little lower, because of the worst case we described in section 3.3. MSF and MASF turn out to be better strategies for our algorithm.

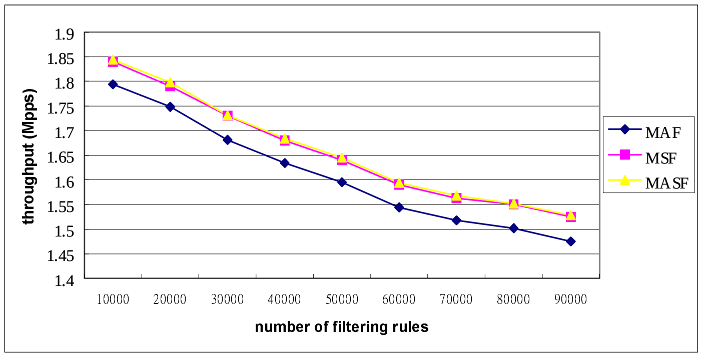

Figure 9: The effect of different lookup order decision strategies on Pentium-II platform.

5.5 Scalability

Figure 10 shows that OLBM algorithm is scalable. Even with 300K filtering rules in the classification database, the throughput is still above 1.1 Mpps, compared to 1.1 Mpps with 20K filtering rules in [6]. 300K is already larger than our design objective, 240K [7], set in section 2. The bypass matching threshold is set to 8 for this experiment. When the number of filtering rules increases, the number of filtering rules per table entry becomes larger, which increases the cost of intersection operations. The intersection operation dominates the performance of our algorithm.

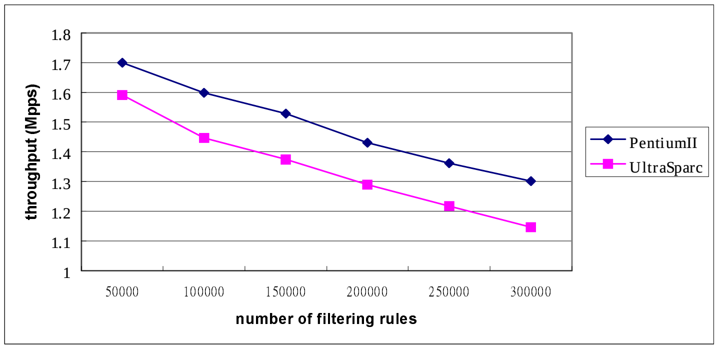

Figure 10: At least 1 million packets per second for 300K filtering rules.

5.6 Extensibility

Although our algorithm is mainly for per-flow classification, it can be extended to lookup more fields or even the payload of a packet. Once you decide to extend it to support more fields, you simply analyze the extra memory reference and compare instructions to lookup that field, and then re-compute the machine dependent threshold K as the criteria for bypass matching. In this way, the algorithm is capable of being extended to lookup various filtering rules for VPN or firewall layer 4 routers, while preserving high performance features such as scalability and throughput.

6 Conclusions

In this work we presented a new multi-field classification algorithm that scales well to support 300K filtering rules using 10 MB memory at a rate over one million packets per second, compared to 1.1 Mpps with 20K filtering rules in [6]. When there are 20K filtering rules in our classification database, our algorithm is 50% faster than the algorithm proposed by V. Srinivasan et al. [6]. [6] needs a huge memory space and cannot scale to 300K filtering rules.

Reference

[1]. J. Wroclawski, "The Use of RSVP with IETF Integrated Services," RFC 2210, September 1997.

[2]. J. Wroclawski, "Specification of the Controlled-Load Network Element Service," RFC 2211, September 1997.

[3]. S. Shenker, C. Partridge, R. Guerin, "Specification of Guaranteed Quality of Service," RFC 2212, September 1997.

[4]. B. Gleeson, A. Lin, J. Heinanen, G. Armitage and A. Malis, "A Framework for IP Based Virtual Private Networks," Internet Draft, draft-gleeson-vpn-framework-01.txt, February 1999.

[5]. T. V. Lakshman and D. Stiliadis, "High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching," *Proc. ACM Sigcomm 98*, September 1998.

[6]. V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel, "Fast and Scalable Layer Four Switching," *Proc. ACM Sigcomm 98*, September 1998.

[7]. K. Thompson, G. J. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics," *IEEE Network*, Vol. 11, No. 6, pp. 10-23, December 1997.