# Bitmap Intersection Lookup (BIL) : A Packet Classification 's Algorithm with Rules Updating

Akharin Khunkitti*, Nuttachot Promrit**

*Faculty of Information Technology, King Mongkut's Institute of Technology Ladkrabang, Thailand
(Tel : +66-2-737-2551; E-mail: akharin@it.kmitl.ac.th)
**Faculty of Information Technology, King Mongkut's Institute of Technology Ladkrabang, Thailand
(Tel : +66-2-737-2551; E-mail: pnuttachot@yahoo.com)

**Abstract**: The Internet is a packet switched network which offers best-effort service, but current IP network provide enhanced services such Quality of Services, Virtual Private Network (VPN) services, Distribute Firewall and IP Security Gateways. All such services need packet classification for determining the flow. The problem is performing scalable packet classification at wire speeds even as rule databases increase in size. Therefore, this research offer packet classification algorithm that increase classifier performance when working with enlarge rules database by rearrange rule structure into Bitmap Intersection Lookup (BIL) tables. It will use packet 's header field for looking up BIL tables and take the result with intersection operation by logical AND. This approach will use simple algorithm and rule structure, it make classifier have high search speed and fast updates.

**Keywords:** Packet Classification, Packet Flow, Firewall

## 1. INTRODUCTION

The packet classification nowadays becomes the important tool of many applications, such as firewall and internet router. These applications will initially search by classifying packets, detecting packets accorded to the determined condition and high priority choosing the rule. Even though the internet is a packet switched network which offers the best - effort service (the sent-received rate never guaranteed), IP network can provide many different services responding to users likes: Quality of Service, Virtual Private Network (VPN) Services, Distribute Firewall and IP Security Gateways. Rules updating can be maintained by both network administrators and Protocol, thus, the problem about packet classification is it must operate on high-speed link bandwidth along with the larger number of rules.

Many researches focus on solving packet classification algorithms problems such as [1], [2]. These researches proposed the method using High - Search Speed Complexity that defined as $O\left(D \cdot \log_2 \cdot N + \frac{N}{WORD}\right)$ and O(1) on hardware but they still encounter such problems as:

1. Bitmap-Intersection method: Storage Complexity of this method is $O(D \bullet N^2)$ which waste too much space if the packet classification algorithm have to work with many rules [1].

2. Ternary CAM method: Storage Complexity of this method is O(N) and must be developed only on hardware; nevertheless, this method use memory that has the amount of Transistor more than the other, so it wastes more power [2].

Thus, this paper introduces packet classification algorithms with high – speed searching rules: searching Speed Complexity defined as $O\left(\frac{D \cdot W}{PMX} \cdot \frac{N}{WORD}\right)$ on the appropriate software, spending appropriate time to updating rules while Storage Complexity is defined $O\left(\frac{D \cdot W}{PMX} \cdot 2^{PMX} \cdot N\right)$

## 2. RELATED WORKS

### 2.1 Meaning of Packet Classification

Packet Classification is mechanical method being used to inspect network packets. This method works by comparing data in header of packet more than 1 field. For example, compare from these fields: Source/Destination Network-layer Address (32 bit), Source/Destination Transport-layer Port Number (16 bit), Type-of-Service (8 bit), Protocol (8 bit) ,and Transport-layer Protocol Flag (8 bit) then determine packet action, let each flow packet be determined by the rule. Every each packet in the same current will perform the same action. Instead of defining this process as "Packet Filtering", this paper prefers to define this process as "Packet Classification" because the words "Packet Classification" is overall admitted and ultimately define its process, moreover the word "classify" denotes the action of every packet while the word "filtering" denotes the action of only preferred packet [3].

IP Network services examples are: 1) Packet Classification according to the rules method applied with Route Lookup considering from Destination IP Address and determine the suitable Next-hop, 2) Packet Classification according to the rules method applied with firewall considering from Source IP Address , Destination IP Address, Source Port, Destination Port or any other field and determine action likes, ACCEPT or DROP ,and 3) Classification according to the rules method applied with Quality of Service considering from Source IP Address and Source Port then determine Queue further.

### 2.2 Packet Classification Requirements

Packet Classification according to the rules method is the significant tool that used to determine the flow of packets on Internet that must operate on the high - speed link bandwidth with the large rule. So Packet Classification applications must show the best perform. The working procedure is then determined by following conditions:

1. High search speed: Packet Classification according to the rules method must operate on the high – speed link bandwidth.

2. Low storage space : Good Packet Classification according to the rules method must be capable to cooperate with larger rule but use the low hard disk space.

3. Less updating time Packet Classification according to the rules method: While augmenting or decreasing rule, the application that has high frequency of rule updating must spend the high search speed also. The quickness of rule updating shouldn't be lower than the quickness of searching.

### 2.3 Convert rules to Prefix

Many Packet Classification according to the rules methods must convert rule to Prefix Sets, determine Prefix size equal to header field (W) of packet. Each set number is not more than 2W – 2 [4]. For examples, determining each Prefix size equal to 4 bits, rule no. 1 is 4-7, thus, set of Prefix is {01**}, or rule no. 2 equals to 1- 14 , thus, set of Prefix is {0001, 001*, 01**, 10**, 110*, 1110}

## 3. BITMAP INTERSECTION LOOKUP (BIL)

From the problems stated in the previous section, we realize the importance of packet classification algorithms. Then our approach offers simple packet classification algorithms which can be used in multi - field searching , moreover, this method can also work well on software and hardware.

Figure 1 below shows an architecture of BIL method which consists of following components: 1) BIL table with D tables , 2) each table includes $2^{PMX}$ addresses, each address includes N bits (0 bits ups to N-1 bits called Bitvector), 3) each position on Bivector is identical with rule contained in rule table, 4) arrange rule by priority (descending), 5) to augment rules, rule will be defined as Mask or Range, and 6) convert rule into Prefix in order to use for Set Bitvector in BIL table and store every action of each rule into rule table further.

BIL method divides header of packets into smaller block then take the value of each block to look up in BIL table. The result from looking up called Bitvector, take that Bivector to be intersected with AND logic, and then the algorithm priority chooses the most important rule as the arrows stated in Fig. 1. Classify Algorithm and Update Algorithm will show in Fig. 2 and 3, and example of Classify in Fig.4

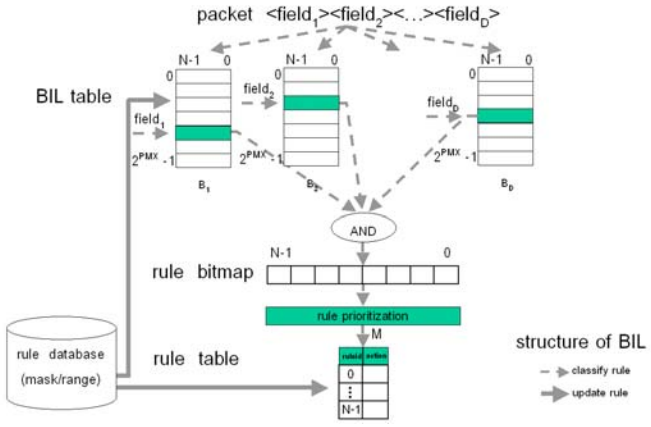


Fig. 1 An architecture of BIL

In defining this algorithm, we determine:

N is number of rules

D is number of header field

$F_i$ is number of bits in each divided header filed:

$1 <= i <= D$

WORD is bits size used in each processing

If $F_i$ has the same size in every field, each field is W bits , since all bit size equal to D•W bit

Divide $F_i$ into $P_{i,j}$ bit, determine

$M_i$ is Block divided in $F_i$ where $\sum_{j=1}^{M_i} P_{i,j} = F_i$

Consider any $F_i$ in only 1 field, where $P_{i,j}$ of $F_i$ equal to $PM_i$ bits, thence $M_i = \frac{F_i}{PM_i}$

Define $PM_i$ as PMX bit in every field and MX is Block amount in each field, thence $MX = \frac{W}{PMX}$ and the number of the table is D•MX

```
1. Read header field of packet and keep into buffer
2. for (i=0; i<D; i++ ) {
3.       for (j=0; j<MX; j++ ) {
4.             Divide header field of packet i into small block,
               shift bit right had side bit by PMX bit,
               record into memory
5.       }
6. }
7. for (i=0; i<N/WORD; i++ ) {
8.       for (j=0; j< D•MX; j++ ) {
9.             take the header field to Lookup in BIL table and
               take the Bitvector at position WORD i from BIL table j
               to intersect by logic AND
10.      }
11.      if (readout intersected by logic AND != 0) {
12.            calculate rule number of the first 1 bit of
               Bitvector and Return Action value from rule table
13.      }
14. }
15. Return Default Action
```

Fig. 2 Classify Algorithm

```
1. Read header field of packet and keep into buffer
2. translate the instruction of rule and record value of
   each field into memory
3. record ruleid, Action and instruction into rule table,
   define tmp = 0 and d = 0
4. for (i=0;i<D; i++) {
5.      tmp +=d, then convert rule value to Prefix (value/mask)
6.      While (current != NULL) {
7.        d=temp
8.        for (k=0;k < MX; k++) {
9.             let dividing Prefix value (value/mask) be j member
               of field i by shift bit right had side bit by PMX bit,
               record the readout as value and mask
10.            for (l=value AND mask ;l<=( value AND mask)+~mask; l++) {
11.                 set Bitvector at address no.l, bit no. ruleid in BIL table d
12.            }
13.            d++
14.       }
15.  }
16.}
```

Fig. 3 Update Algorithm

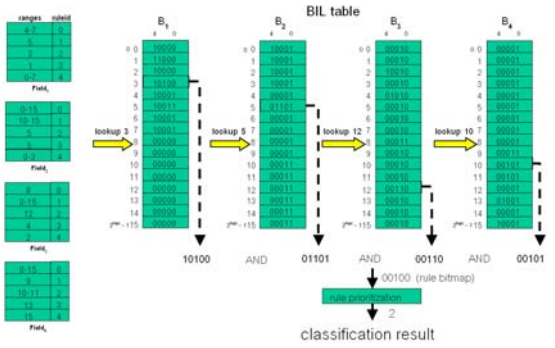


Fig. 4 example of Classify

Our research shows that the quickness of searching, the quickness of rule updating and managing storage space of BIL program depend on specific Prefix size, so we concentrate on the study of Prefix size that allow the program work best, considering from the product of working performance which are: 1) Product of updating time and storage space 2) Product of searching time and storage space 3) Product of searching time and updating time, and 4) Product of searching time, updating time and storage space. The objective is to study the influence each product on the highest and the lowest of the graph that well affected towards the program's working performance.

## 4. EXPERIMENTAL RESULTS

According to the hypothesis mentioned earlier, we had developed BIL program with gcc complier on Linux Kernel 2.4 operating system and had created rules' database and classifying packets which are suitable for evaluating the program performances. These rule databases stored in script file named "rules.db" and packet databases stored in file name "dump.db" because it's necessary to control an interfere, which could happen from network traffic and network application or even from operating system's Kernel, not to affect the working performance of program. This procedure allow us not to mistakenly measure searching time and updating rule besides that we can test and compare working performance of Linear Search program [4] and Bitmap-Intersection program [1], we need to develop these programs in the same environment.

We had designed the experiments into 2 sections which are: Section 1 is to divide Prefix size into small Block (such as dividing Source IP Address into 16+16 bits) to calculate the precise value that allow BIL program work with the best performance. Section 2 is to compare working performance of Linear search program, Bitmap Intersection program and BIL program.

The database used in section 1 is 1 field rule database and let it be Source IP Address (32 bits) and Source MAC Address (48 bits), The database used in section 2 is 1 field rule database and let it be Source IP Address (32 bits). We specially create database cases for measuring searching time and updating time are: 1) Best Case 2) Average Case ,and 3) Worst Case

**Experiment 4.1: Dividing Prefix sizes to calculate the most effective value**

**Objective:** To calculate the stable prefix size and changeable prefix size which allow BIL program spend less time in searching and updating process, moreover it effectively use storage space for Source MAC Address (48 bits) where number of rules are 4, 096 .

**Experiment 4.1.1:** Dividing stable Prefix size by using 1 field rule database as Source MAC Address

**The result shows:**

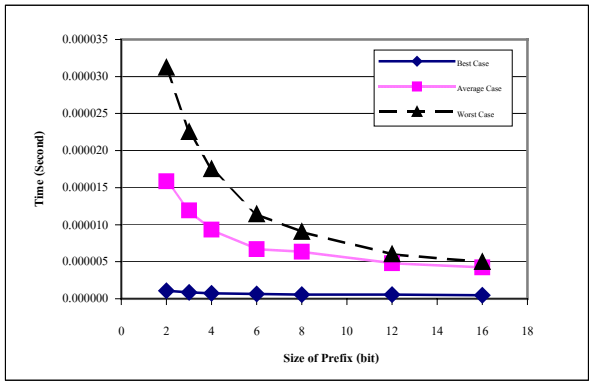


Fig. 5 Average searching time per packet (second of time) for Source MAC Address

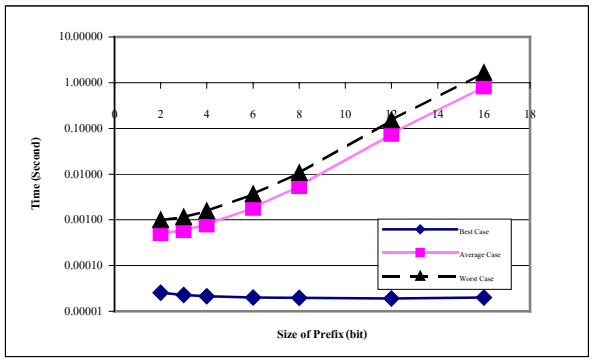


Fig. 6 Average inserting time per rule (second of time) for Source MAC Address with logarithm scale

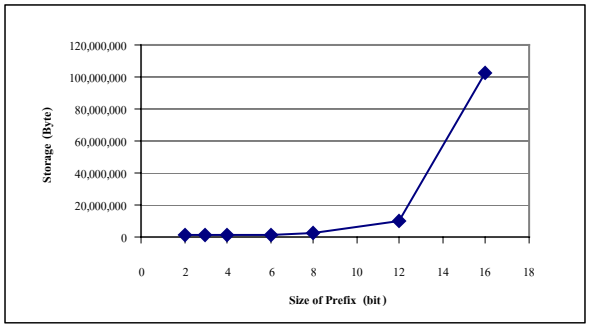


Fig. 7 Storage space (bytes) for Source MAC Address

It appears in Fig.5 that the quickness of searching time relates to PMX size. The bigger PMX is determined; BIL program spends lesser searching time in every case. When PMX is increased to 16 bits, BIL program spends the less searching time.

From Best Case studied in Fig.6, the bigger PMX is determined; BIL program spends lesser updating time. When PMX is increased to 16 bits, BIL program spends the less updating time. Moreover, when PMX is increased in Average and Worst Case, BIL program spends more updating time as well. When PMX is increased up to 16 bits, BIL program spends the most inserting time.

Fig.7 shows that BIL program uses more storage space when PMX extend to 16 bits

Table 1 Shows Prefix sizes suitable for Source MAC Address considering from the lowest point of products

| Products | Cases | | |
|---|---|---|---|
| | Best Case | Average Case | Worst Case |
| 1. updating time x storage space | 4 Bits | 2 Bits | 2 Bits |
| 2. searching time x storage space | 6 Bits | 6 Bits | 6 Bits |
| 3. searching time x updating time | 16 Bits | 3 Bits | 3 Bits |
| 4. searching time x updating time x storage space | 6 Bits | 3 Bits | 3 Bits |

Considering from 4 products presented in section 3, the result concluded from Table 1 shows that Worst Case that let PMX is 2 bits, 3 bits, and 6 bits: is the most suitable Prefix size for Source MAC Address. We had brought these products to use in experiment 4.1.2 further.

**Experiment 4.1.2 : Dividing changeable Prefix patterns by using 1 field rule database as Source IP Address**

Determined 13 patterns of Prefix size:

- Pattern 1: let Prefix size be 2+2+2+2+2+2+2+2+2+2+ 2+2+2+2+2+2 bits
- Pattern 2 let Prefix size be 3+3+3+3+3+3+3+3+3+3+2 bits
- Pattern 3: let Prefix size be 4+4+4+4+4+4+4+4 bits
- Pattern 4: let Prefix size be 2+6+10+14 bits
- Pattern 5: let Prefix size be 4+8+8+12 bits
- Pattern 6: let Prefix size be 2+4+6+8+10+2 bits
- Pattern 7: let Prefix size be 6+6+6+6+6+2 bits
- Pattern 8: let Prefix size be 7+7+7+7+4 bits
- Pattern 9: let Prefix size be 8+8+8+8 bits
- Pattern 10: let Prefix size be 9+9+9+5 bits
- Pattern 11: let Prefix size be 10+10+10+2 bits
- Pattern 12: let Prefix size be 11+11+10 bits
- Pattern 13: let Prefix size be 12+12+8 bits

**The result shows:**

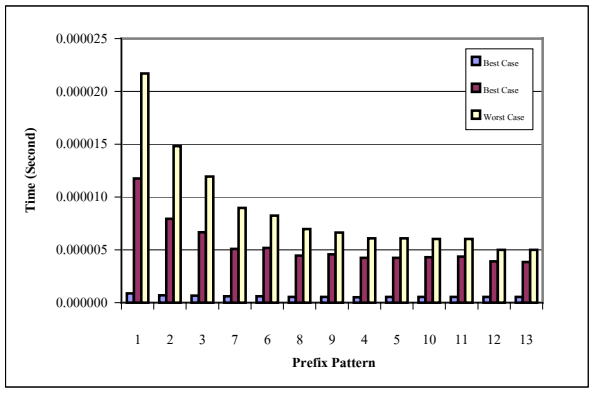


Fig. 8 Average searching time per packet (second of time) of Prefix pattern 1 to 13, ordered by the quickness of searching ascending according to Worst Case

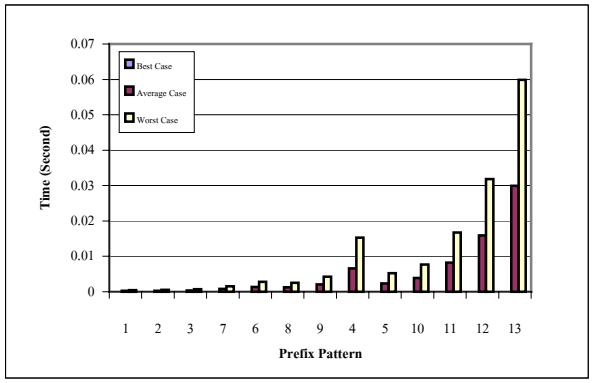


Fig. 9 Average inserting time per rule (second of time) of Prefix pattern 1 to 13, ordered by the quickness of searching ascending according to Worst Case

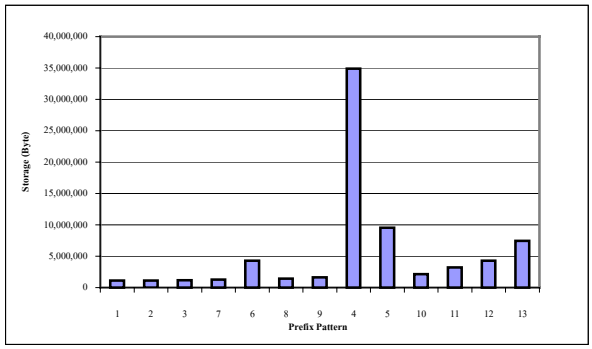


Fig. 10 Storage space (bytes) of Prefix of Prefix pattern 1 to 13,ordered by the quickness of searching ascending according to Worst Case

From the results are presented in Fig. 8, we find that Pattern 1 (2+2+2+2+2+2+2+2+2+2+2+2+2+2+2+2 bits) in Worst Case spends the most searching time and Pattern 13 (12+12+8 bits) spends the less searching time.

The results are presented in Fig. 9 appear that pattern 13 (12+12+8 bits) in Best Case spends the less time in inserting

rule and pattern 1 (2+2+2+2+2+2+2+2+2+2+2+2+2+2+2+2 bits) spends the most time in inserting rule. While Prefix pattern 1 (2+2+2+2+2+2+2+2+2+2+2+2+2+2+2+2bits) in Average and Worst Case spends the less time in inserting rule and Pattern 13 (12+12+8 bits) spends the most time in inserting rule.

It appears in Fig. 10 that pattern 1 (2+2+2+2+2+2+2+2+2+2+2+2+2+2+2+2 bits) uses the smallest storage space while pattern 4 (2+6+10+14 bits) uses the largest storage space.

Table 2 The suitable Prefix of Prefix pattern 1 to pattern 13 considered by the lowest point of the product.

| Products | Case | | |
|---|---|---|---|
| | Best Case | Average Case | Worst Case |
| 1. updating time x storage space | Pattern 3 | Pattern 1 | Pattern 1 |
| 2. searching time x storage space | Pattern 3 | Pattern 8 | Pattern 8 |
| 3. searching time x updating time | Pattern 13 | Pattern 2 | Pattern 2 |
| 4. searching time x updating time x storage space | Pattern 8 | Pattern 2 | Pattern 2 |

Table.2 shows that Worst Cases at Prefix pattern 1 (2+2+2+2+2+2+2+2+2+2+2+2+2+2+2+2 bits), pattern 2 (3+3+3+3+3+3+3+3+3+3+2 bits), and pattern 8 (7+7+7+7+4 bits) are the most suitable pattern considering from all 4 products presented in section 3. In order to compare working performance of another program in the 4.2.1~4.2.3 experiments, we choose Prefix pattern 2 (3+3+3+3+3+3+3+3+3+3+2 bits) because pattern 2 is utmost well balance.

**Experiment 4.2: The comparison of working performance of Linear Search program, Bitmap Intersection program , and BIL program**

**Objective:** To compare working performance between Linear Search program, Bitmap Intersection program and BIL program using 1,000-10,000 rules for Source IP Address

**Experiment 4.2.1: The comparison of time spent in searching rules of Linear Search program, Bitmap-intersection program and BIL program**

**The result shows:**

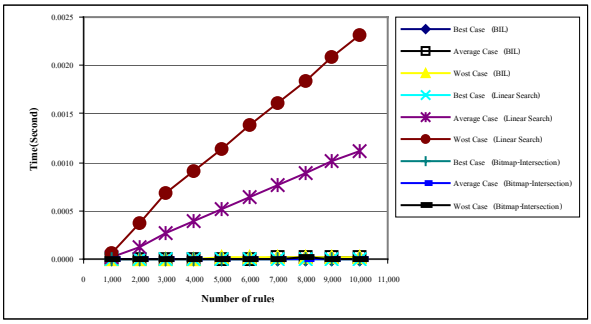


Fig. 11-A Average searching time per packet (second of time) of Linear Search program, Bitmap-intersection program, and BIL program.

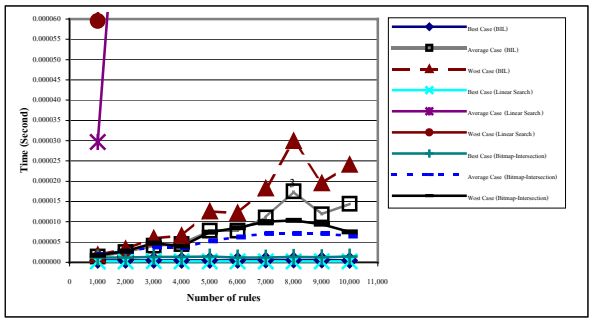


Fig. 11-B Average searching time per packet (second of time) of Linear Search program, Bitmap-intersection program, and BIL program operated with zooming in.

Fig. 11-A and 1-B prove that these programs identically spend searching time in Best Case where algorithms are 1,000-10,000 rules. Linear Search program spends 0.00000023 seconds, Bitmap-Intersection program spends 0.00000128 seconds, and BIL program spends 0.00000077 seconds.

Both figures also show that each program tends to spend more searching time when N is increased in Average and Worst Case. In both case, these programs can be ascending arranged as follow: Bitmap-intersection program, BIL program, and Linear Search program.

**Experiment 4.2.2: The comparison of time spent in inserting rules of Linear Search program, Bitmap-intersection program and BIL program**

**The result shows:**

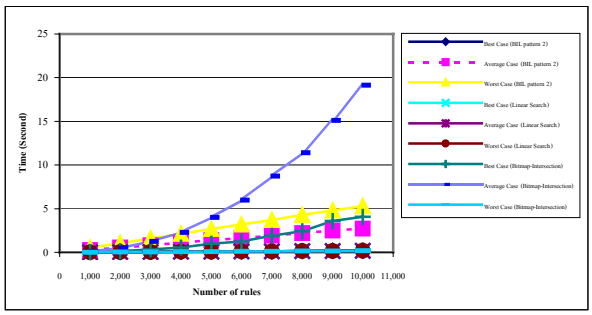


Fig. 12-A Average inserting time (second of time) of Linear Search program, Bitmap-intersection program, and BIL program.

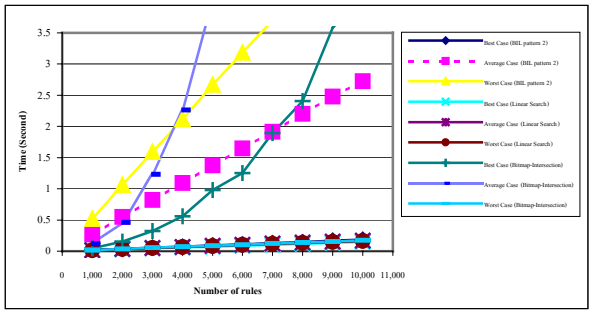


Fig. 12-B Average inserting time (second of time) of Linear Search program, Bitmap-intersection program, and BIL program operated with zooming in

It appear in Fig. 12-A and 12-B that when N is increased in Best and Average Case, each program tends to spend more time in inserting rule at the number of algorithms is 1,000-10,000 which can be ascending arranged as follow: Linear Search program, Bitmap-intersection program, and BIL program.

**Experiment 4.2.3: The Comparison of storage space of Linear Search program, Bitmap-intersection program and BIL program**

The result shows:

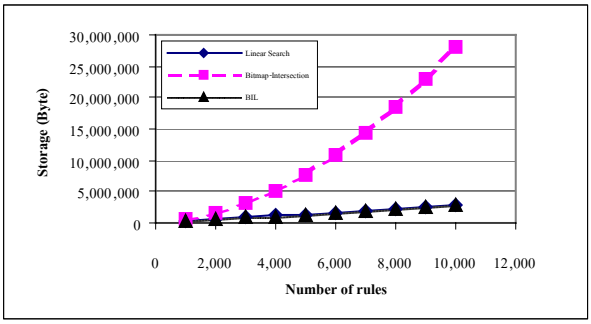


Fig. 13 Storage space (byte) of Linear Search program, Bitmap-intersection program ,and BIL program

The experiment in Fig. 13 shows that each program tends to spend more time in inserting rule when N is increased and the number of algorithms is 1,000-10,000. BIL program and Linear Search use the identical storage space but Bitmap-intersection uses the largest storage space.

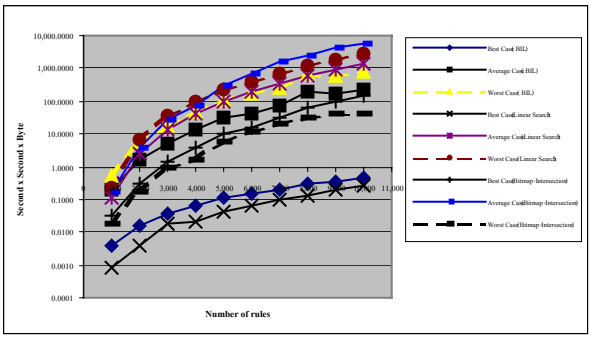


Fig. 14 products of searching time, updating time and storage space of Linear Search program, Bitmap-intersection program, and BIL program operated with logarithm scale

Figure. 11-A and 11-B shows that Linear Search program tends to spend most searching time. Figure. 12-A and 12-B shows that Bitmap-intersection program tends to spend most time for inserting rule, and Figure. 13 shows that Bitmap-intersection tends to use the largest storage space.

Table 3 The ascending results from the comparison of each product of working performance between Linear Search program, Bitmap-Intersection program, and BIL program for Source IP Address

| Case | Products | | | |
|---|---|---|---|---|
| | updating time x storage space | searching time x storage space | searching time x updating time | searching time x updating time x storage space |
| Best Case | 1.BIL<br>2.Linear Search<br>3.Bitmap-Intersection | 1.Linear Search<br>2.BIL<br>3.Bitmap-Intersection | 1.Linear Search<br>2.BIL<br>3.Bitmap-Intersection | 1.Linear Search<br>2.BIL<br>3.Bitmap-Intersection |
| Average Case | 1.Linear Search<br>2.BIL<br>3.Bitmap-Intersection | 1.BIL<br>2.Bitmap-Intersection<br>3.Linear Search | 1.BIL<br>2.Bitmap-Intersection<br>3.Linear Search | 1.BIL<br>2.Linear Search<br>3.Bitmap-Intersection |
| Worst Case | 1.Linear Search<br>2.Bitmap-Intersection<br>3.BIL | 1.BIL<br>2.Bitmap-Intersection<br>3.Linear Search | 1.Bitmap-Intersection<br>2.BIL<br>3.Linear Search | 1.Bitmap-Intersection<br>2.BIL<br>3.Linear Search |

The results from Table 3 appears that BIL program in Average Case proves more effectual than products of others which are: Product of searching time and storage space, Produce of searching time and updating time and Product of searching time, updating time and storage space.

## 5. CONCLUSIONS

This paper introduces the increasing efficiency of packet classification methods which are: 1) rearranging rules structures into Bitmap Intersection Lookup tables (BIL), 2) in searching method, dividing the header of packets into Block ,then takes each Block to look up in BIL, and 3) intersecting the result by AND logic. These methods not only simplify the working procedure of the program but also rules' structure. As a result, the program spend lesser searching time and has appropriate interval to adjust rules.

Our experiments show that BIL program can work efficiently where Prefix is 3 bits. And when compared BIL program with Linear Search program and Bitmap-intersection program, we consider that the program efficiency is related to complexity as stated previously. Furthermore, BIL program discernibly shows the best performance in Average Case between any other programs.

Results may also pave the way to the further experiment such as comparing the efficiency of other hardware internet applications like firewall or router, etc**.**

## REFERENCES

[1] T. V. Lakshman and D. Stiliadis, "High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching", *In Proc. ACM Sigcomm '98*, Sept. 1998.

[2] J. van Lunteren and A.P.J. Engbersen, "Multi-field packet classification using ternary CAM", *IEE Electronics Letters*, Vol. 38, No. 1, Jan. 2002.

[3] M.L. Bailey, B.Gopal, M.Pagels, L.L.Peterson, and P.Sarker, "PATHFINDER: A Pattern-Based Packet Classifier", *In Proc. Operating Systems Design and Implement*, Nov. 1994.

[4] P. Gupta and N. McKeown, "Algorithms for Packet Classification", *IEEE Netw*., 2001.