A Multidimensional Packet Classification Algorithm Based on Network Processors

YECHANG FANG, KANG YEN, AMAURY CABALLERO, NANSONG WU
Dept. of Electrical & Computer Engineering
Florida International University
10555 W. Flagler Street, Miami, Florida 33174
USA
yfang003@fiu.edu, yenk@fiu.edu, caballer@fiu.edu, nwu001@fiu.edu

Abstract: Packet classification algorithms have been studied by many authors. Most of these algorithms are developed based on tree structure, rather than linear data structure which is appropriate for network processors (NPs). With the development of networks, more and more NPs are used to execute various tasks in the network. In this paper, a multidimensional packet classification algorithm called classification based on network processors (CBNPs) is put forward. This algorithm uses parallel tuple search and multi-threaded concurrency of NPs to enhance the classification speed, and also gets much less space and time complexity than other parallel algorithms. The simulation shows that CBNPs is high-speed, multidimensional and scalable, and its comprehensive performance is much better than that of the existing packet classification algorithms.

Key-Words: Network Processor, Packet Classification, Classifier, Regulator.

1. Introduction
Packet classification mechanism is mainly used in classifiers and the regulators of NPs. Its role is to ensure that the input data stream accords with the traffic conditioning agreement (TCA). This mechanism classifies the stream to certain behavior aggregate, and tags them correspondingly as packets. The classifier follows the rules specified in TCA and assigns packets to different classes in the group according to some domains in header. Then they will be handed over to the appropriate regulator module for further processing.

Generally the classification process is carried out at the communications sub-network nodes or routers. The routers are divided into either flow or non-flow identification routers. Flow identification routers track the transmission and conduct similar processing on the packets of the same flow. Non-flow identification routers process each incoming packet separately. Main processing steps on each packet in the flow identification router are: routing table lookup, packet classification, packets special treatment such as discarding unauthorized packets, and exchange scheduling.

2. Existing Packet Classification Algorithms
With in-depth research of network service traffic characteristics, the implementation of regulator has become more mature. The usual practice is to use token bucket and the leaky bucket algorithms, and other appropriate combination such as the single rate Three Color Marker (srTCM) and two rate Three Color Marker (trTCM) [1]. Recently a time sliding window based algorithm called time sliding window Three Color Marker (tswTCM) has been proposed. This algorithm uses the rate estimator to measure the average rate in the time sliding window, and then uses this rate as the basis of marker. [2]

From the implementation point of view, these algorithms can be divided into two kinds. One is hardware based algorithm that can be fully implemented by hardware, and the other is software based algorithm for software implementation. Hardware based algorithms such as ternary CAMs algorithm [3], use the parallelism feature of hardware to conduct packet classification. The classification process is fast, but the available dimension is small, the scalability is poor, and the cost is high. Also some operators cannot be directly supported. The utilization of the memory array may not be very efficient. They are not practical for
PC-based routers. On the other hand, software based algorithms generally provide specific dimension classification capability, especially the common one-dimensional and two-dimensional classification algorithms. For example, set pruning tries, grid of tree [4], and Area-based quad-tree (AQT) algorithms [5] are two-dimensional classification algorithms. However, algorithms that can provide support for high-dimensional classification may not well meet the requirements on space complexity and time complexity. For example, hierarchical cuttings algorithm [6] require large space in the worst case, and for some other algorithms, such as bit vector search algorithm [7] and back tracking algorithm, the classifying speed is slower. Cache-based [8] approach also does not work very well in practice due to the low hit rate and smaller flow duration.

3. The CBNPs Algorithm
Most of the current packet classification algorithms are based on recursive structure, but the programming of NPs does not support recursive calls, and access of the tree structure is very complex. So we propose a multidimensional packet classification algorithm called CBNPs which uses linear data structure.

3.1 Algorithm Description
The CBNPs algorithm includes two parts: preprocessing and classification. The preprocessing is mainly preparations for simplifying and accelerating the classification operation. The main function of classification is fast index operation on multidimensional packet classification algorithm, assuming it has \( r \) rules in the worst case, and for some other algorithms, such as bit vector search algorithm [7] and back tracking algorithm, the classifying speed is slower. Cache-based [8] approach also does not work very well in practice due to the low hit rate and smaller flow duration.

Suppose a packet \( P \) arrives, then the classification process through the rule search is as follows:
1) First, in each dimension \( j \), we use binary search or other search algorithms to find the interval corresponding to \( P \), and thus get the bit vector \( B_j \) corresponding to \( P \) in the \( j \)-dimension.
2) For all the bit vectors \( B_1, \ldots, B_n \), we can get the tuple corresponding to the bit with the first value of 1 in \( B \). And this tuple is the tuple \( t \) with the highest priority including the packet \( P \).
3) In the hash table corresponding to the tuple \( t \), we get the rule \( r \) matched with \( P \) by a memory access.

3.2 Performance Analysis
In the preprocessing of the CBNPs algorithm, we assume that there are a total of \( n \) rules and \( m \) corresponding tuples. In a general case, \( m << n \), which means the space occupation of bit vectors is very low. Since hundreds of rules are in the rule base of NPs, if we use tuples instead of the rule bases, space occupation reduction of each bit vector is considerable.

3.2.1 Space Complexity Analysis
As discussed before, \( n \) rules projection on the \( j \)-axis will produce a maximum of \( 2n+1 \) mutual non-covered intervals. If an \( m \)-bit vector is assigned to each interval, we have the following relation

\[
C_s = K \times (2n+1) \times m
\]

where \( C_s \) is the space used by all bit vectors, and \( K \) represents the dimension. Space complexity of the CBNPs algorithm is \( O (mn) \), which is the same as that of the parallel algorithm. In the worst case, \( m = n \). But \( m << n \) on the average, this greatly reduces space complexity.

The concept of tuple space is derived through the observation of actual rule bases. It is a heuristic solution to solve packet classification issues by using the tuple space. This solution can guarantee less space complexity in the average conditions, so space complexity of the CBNPs algorithm is much less severe than the parallel algorithm.

3.2.2 Time Complexity Analysis
If the bit of a bit vector is reduced, the amount of memory access required to read the vector will also be reduced. Here we have

\[
C_t = K \times m \times w
\]

where \( C_t \) represents the required times for memory access.
accesses if all bit vectors are read, and $w$ is the word length of one memory access. In the CBNPs algorithm, vectors can be read on each dimension in parallel, the same as bit parallel algorithms. In addition, one memory access is required to access the corresponding hash tables to find the matching rules, but this access time can be ignored. Therefore, the total time complexity is $O(t + mn/w)$. Again, the worst case happens when $m = n$, and this indicates the time complexity of the CBNPs is the same as that of parallel algorithms. Since in most cases the CBNPs algorithm satisfies the condition of $m/w < n/w$, the time complexity of the CBNPs algorithm is greatly improved over parallel algorithms.

### 3.3 Implementation and Optimization

From the classification process, it can be seen that when the number of partial matching or field matching is 1, the classification process is completed. So we conduct a direct matching operation to make sure the rule $r$ really matches the packet $P$. In fact, with the assumption that the number of part matching or fields matching rules is $k$, if the cost to directly match the $k$ rules is less than the cost to search the rest of the rules, then we can conduct direct $k$ times of matching operations. In this way, the classification process can be accomplished faster. Here's a how to determine the maximum value $K$ that satisfies the above criteria $k$. Upper bound of $K$ is hardware-related. The value of $K$ may be different if the algorithm is running on different NPs. The CPU instructions used in this algorithm are mainly the following: access memory operation including SRAM access and SDRAM access where accessing time is set to $M_{sram}$ and $M_{sram}$; comparison operation where the cost is set to $C$; logic operation where the cost is set to $A$; testing instructions of location where the first bit is 1 in a 16-bit word, for which the cost is set to $findset$. If $k$ filtering rules are left after searching a number of sorting databases, whether to conduct direct matching or continue to search depends on the following inequality (3):

$$K(4M_{sram} + 5C) < M_{sram} + A + 4\text{ findset} + (4M_{sram} + 5C)$$

(3)

where $4M_{sram}$ means using four SRAM memory accesses to fetch a rule $r$, and $5C$ expresses using five times operations to compare the five fields. $M_{sram}$ uses one SDRAM access to fetch the index which matches the packet fields. $A$ is the new preparation set matching the new fields by performing bitwise operation to the previous preparation set. $4\text{ findset}$ represents using four instructions to find the first bit corresponding to the rule $r$ matched by the fields in the original rule base. $4M_{sram} + 5C$ determines whether the rule $r$ really matches the packet $P$. Here we can change (3) to (4).

$$K < \frac{M_{sram} + A + 4\text{ findset} + (4M_{sram} + 5C)}{(4M_{sram} + 5C)}$$

(4)

Once a machine is decided to run the algorithm, the cost of executing each instruction in the formula can be determined. So is the value of $K$. When the number of rules matched with the entered packets is less than or equals to $K$, the matching operation can be directly conducted without further search operation. This will accelerate the classification process.

### 4. Simulation Results

The micro engines of NPs provide chained SDRAM memory accesses to ensure continuous high-speed access to SDRAM. And the bus arbiter gives higher priority to memory accesses from chained SDRAM than other micro-engines to ensure that instructions are continuously sent to the SDRAM unit. SDRAM chained access is completed by adding chain-ref options to the SDRAM instructions. The instruction parameter settings will be determined based on the number of rules when the program is executed.

If the number of rules is less than or equals to 64, we can run the SDRAM instructions without parameters while taking a certain field from the fields sorting library. The format is as follows:

```
sdram [sdram_cmd, $$sdram_xfer-reg, source-opl, source-opi, chain-ref], where [sdram_cmd] means the instruction operations; [$$sdram_xfer-reg] is the name of register which stores the read data. [source-opi] and [source-opi] calculate the memory start address where SDRAM is read and written. By default, one memory word can be read each time, so 64 rules can be taken each time.
```

When the number of rules is between 65 and 256, we need to add ref-count parameter into the SDRAM instructions. This parameter represents how many 4-word length storage units are continuously accessed during each memory access. The SDRAM instructions set ref-count value between 1 and 4, so the number of rules is between 65 and 256.

When the number of rules is greater than 256, the SDRAM instructions with indirect-ref parameter will conduct continuous access of the SDRAM. In the edge routers, the rules are few in the rule base.
Only about 0.7% of the rule bases have more than 1000 rules, and the average number of rules in most cases is about 50.

In this simulation, we adopt K1297 signaling analyzer to simulate packet sending, and the operation system for K1297-G20 is Windows NT 4.0. We use 128 rules to implement this algorithm on NP Intel IXP2400, and randomly generate these rules according to the following strategies:
1) 25% of the rules belong to the same prefix. For example, 166.111.*.* is used to simulate the LAN environment.
2) 50% of the rules are based on TCP, and 45% of the rules are based on UDP, because most of the rules are for TCP/UDP protocol.
3) 40% of the rules are for port 80 (WWW), and 20% of the rules are for ports 20 and 21 (FTP).

In the simulation, we randomly produce 1000 different types of data flows with 100% hits. Each packet length is 64-byte. The rule set has a total of 397 rules, in which source or destination address field of 126 rules are non-precise bit-string. We have four defined non-precise bit strings, which are intranet, extranet and two regions. We also define 24 types of services, including a definition of the source port number, and two definitions of a particular destination port number within the range. In addition, to test the algorithm performance of cases with even larger rule set, we randomly generate 2000, 10000, 20000, 30000, 50000 and 100,000 rules. Evaluation results have been listed in Table 1, which show the average results obtained from 16 times of program running.

<table>
<thead>
<tr>
<th>Number of Rules</th>
<th>Throughput (Rule/ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>397</td>
<td>3660</td>
</tr>
<tr>
<td>2000</td>
<td>3660</td>
</tr>
<tr>
<td>10000</td>
<td>3620</td>
</tr>
<tr>
<td>20000</td>
<td>3540</td>
</tr>
<tr>
<td>30000</td>
<td>3440</td>
</tr>
<tr>
<td>50000</td>
<td>3300</td>
</tr>
<tr>
<td>100000</td>
<td>3200</td>
</tr>
</tbody>
</table>

From the results, it can be seen that the algorithm fully shows the parallel nature of NPs. It is better than the existing packet classification algorithms in code complexity and overall performance.

5. Conclusions

This paper analyzes the existing packet classification algorithms and compares their performances. Combined with the feature of multi-threaded concurrency of NPs, a multidimensional packet classification algorithm based on a linear structure called CBNPs is proposed. It accelerates the classification by reducing the number of classification rules and classification domain width. The simulation results show that CBNPs has better comprehensive performance than the current packet classification algorithms.

Packet classification searching algorithm is another research area following the routing searching algorithm. Currently a lot of packet classification algorithms are quite outstanding, but they are all based on the TRY tree structure. Because of the special code space of NPs, tree structure access is difficult. Therefore the tree structure must be converted into a linear structure to facilitate micro-engines searching. Converting various TRY tree structure into reasonable linear structures and developing the fastest access method will be our future research direction for packet classification algorithm.

References