A novel ultra high-speed multi-layer table lookup method using TCAM for differentiated services in the Internet

Masanori Uga Kohei Shiomoto
NTT Network Service Systems Laboratories
3-9-11 Midori, Musashino, Tokyo 180-8585, Japan
Phone: +81 422 59 4402/ Fax: +81 422 59 4549
E-mail: {Uga.Masanori | Shiomoto.Kohei}@lab.ntt.co.jp

Abstract

This paper proposes ultra high-speed multi-layer table lookup method using ternary content-addressable memory (TCAM) for an ultra high-speed policy-based packet forwarding engine. It can store very wide policy rules in TCAM, whose width is limited. For IP version 6, policy rules could be 304 bits wide. This method enables us to use commercially available TCAM for multi-layer table lookups, and thus build an ultra high-speed packet forwarding engine for differentiated services in the Internet.

I. Introduction

The Internet is expanding more and more in physical size and traffic volume and many applications have been developed for the Internet. Although the Internet was based on a simple best-effort packet forwarding paradigm, demand for differentiated services is arising. A differentiated services (Diffserv) model for possible next-generation IP packet forwarding has been discussed in the IETF. A typical diffserv router looks like Fig. 1. A key component in the diffserv router [1, 2, 3] is a packet classifier . It is located on the line card and classifies each incoming packet by searching a policy table, which is a set of policy rules. Each entry in the policy table consists of a six-tuple: IP source address (SA), IP destination address (DA), TCP/UDP source port number (SP), TCP/UDP destination port number (DP), protocol identifier (PID), and differentiated service code point (DSCP). The packet classifier examines the policy table to see whether there is a policy rule that matches the arriving packet. A typical policy table contains hundreds or thousands of policy rules, so packet classification requires a huge amount of processing power. Packet classification must be done for every packet because every packet might match a policy rule in the table. The throughput requirement for packet classification is very tight to avoid a bottleneck in wire-speed packet forwarding. Packet classification is such a complicated task that it is one of most challenging issues in the design of a high-speed diffserv router.

Ternary Content-addressable memory (TCAM) has become commercially available recently. TCAM is a promising device to build a wire-speed table lookup engine at such ultra high-speed, because it returns the matching entry with a single memory access. However there is an issue with TCAM when it is applied to the policy table lookup problem. The limited width of TCAM is not resolved yet, when it is applied to the policy table lookup problem,

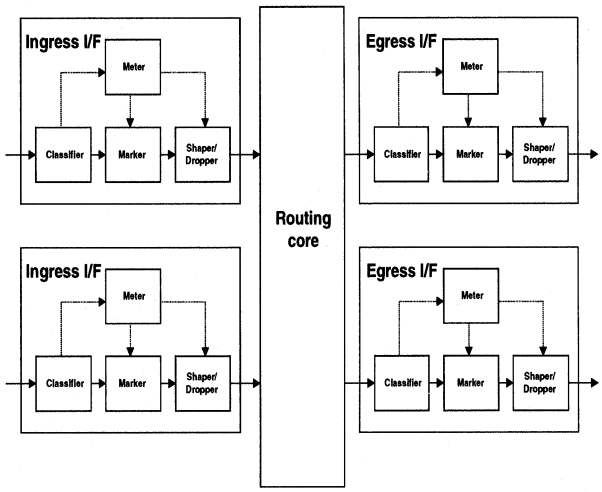

Figure 1: Diffserv router architecture.

This paper proposes a novel high-speed multi-layer table lookup method using TCAM. Chapter II describes the problem with TCAM in more detail. Chapter III proposes a novel packet classification method using TCAM for an ultra high-speed policy-based packet forwarding. Chapter IV shows the implementation and the evaluation of our method. Chapter V summarizes the main point of the paper.

II. Ternary CAM

A. Ternary CAM principle

Ternary CAM is a memory device, in which each entry is searched by the associated content. The data is injected into the TACM. All entries are compared with the injected data. If at least one entry matches it, the data associated with the entry is returned. Otherwise no data is returned. The data at each entry is ternary: it takes either "one", "zero", or "don't care" as shown in Fig.2.

Parallel comparison and ternary data is implemented with hardware circuitry as shown in Fig.3. Considering the TCAM circuitry principle, it is not difficult to recognize that TCAM memory size cannot be expanded with respect

0-7803-6711-1/01/$10.00 (C) 2001 IEEE

to data width direction. [1]

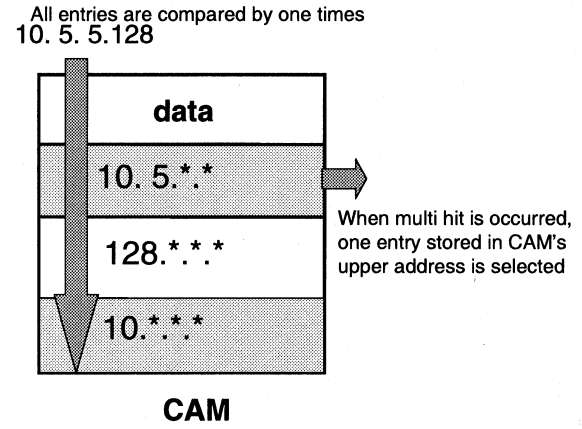

Figure 2: Principle of TCAM.

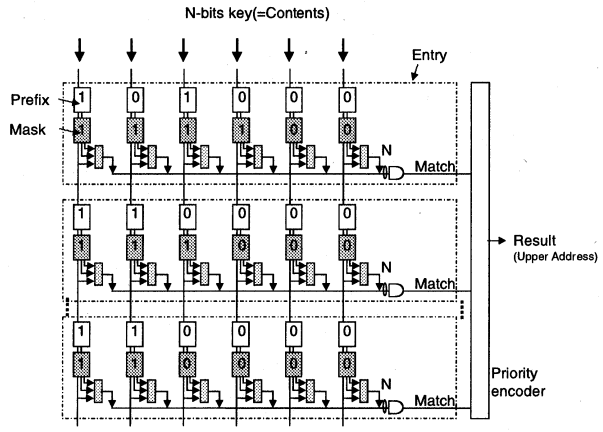

Figure 3: Hardware circuitry of TCAM.

B. *Problem with policy table look-up*

Six-tuple is used to classify flows. Six-tuple consists of source and destination IP addresses, source and destination port, protocol identifier, and differentiated service code point. For IP version 4, six-tuple is 120-bit width: SA (32 bits), DA(32 bits), SP(16bits), DP(16bits), PID(16bits), DSCP(8 bits). For IP version 6, it is 304-bit width. The width of table cannot be expanded if we use TCAM device as mentioned above. We need to develop a mechanism to get such a long policy entry fitted in TCAM with limited width.

III. Proposed method: Policy table lookups

A. *Detecting ambiguous rules*

We assume that ambiguous policy problem is resolved in policy table. That is there a now ambiguous policy rules which match the same packet header simultaneously. The ambiguous policy is detected as explained below [4].

[1]It might be expanded with respect to address space direction by cascading them.

Let's restrict to the case of SA and DA matching problem. Suppose that we have two policy rules: rule-A (SA, DA)=(129.60.83.1/32,129.70/16) and rule-B (SA, DA)=(129.60.83/24,129.70.115.4/32). When we have the packet whose SA and DA are 129.60.83.1 and 129.0.115.4, it matches both rules. If the rule-A comes first in the policy table, rule-A is selected. Otherwise the rule-B is. Those rules are ambiguous. Drawing a two-dimension graph whose x-axis is SA and y-axis is DA for rule, we can identify whether two rules are ambiguous or not as explained below. Regarding the rule-A, x-axis is covered on a spot (129.60.83.1) and y-axis is covered over a range from 129.70.0.0 to 129.70.255.255. Regarding the rule-B, x-axis is covered over a range from 129.60.83.0 to 129.60.83.255 and y-axis is covered over a spot of 129.70.115.4. In this way, the rule-A and the rule-B have an overlap region. For a packet whose SA and DA fall into the overlap region, we can not determine which rule should be applied. In this way, if there is an overlap region, two rules are ambiguous (See Fig. 4.)

When two rules intersect each other, the overlapped area is ambiguous. The ambiguity should be detected and the appropriate action should be determined by network operators.

For example, when there are two policy rules A=(*,01*,reject) and B= (10*,01*,permit), overlapping area (10*,01*) is ambiguous.

This ambiguous problem can be resolved by making no ambiguous rules from two ambigous rule. Figure 6 shows that rule-B is divided into two parts B1 and B2 in Fig. 5. There are no overlapped area among A, B1, and B2.

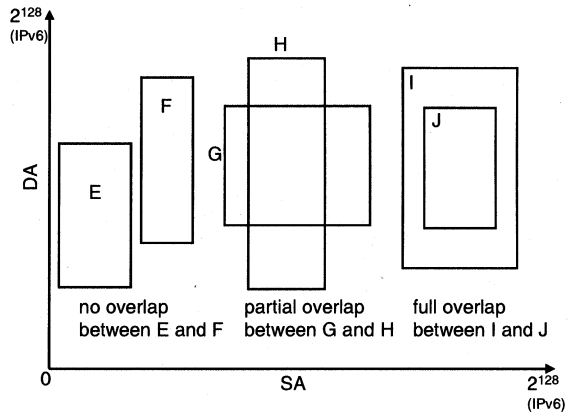

Figure 4: Ambiguous rules.

B. *Idea: divide and conquer*

The problem with TCAM is its width limitation for long policy rule, which could be 304-bit width for policy rule with IP version 6. To get around the problem, we take an approach to divide the policy rule into two pieces: first and second halves. First half of the policy rule and second half can be both stored in the same TCAM (See Fig. 7). We should note that when we divide the policy

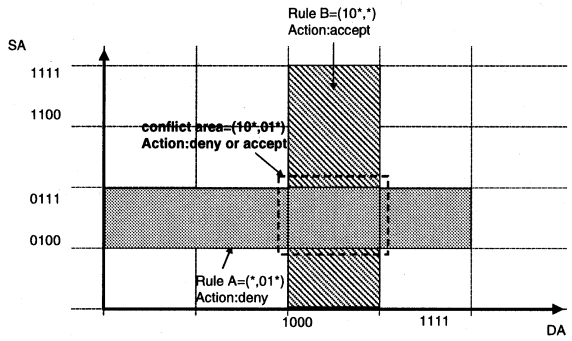

Figure 5: Intersected rules.

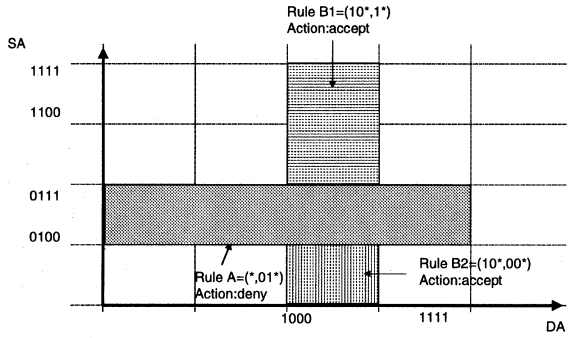

Figure 6: Division of intersected rules into non-intersected rules.

rule into two pieces, we may face ambiguity problem even if the original rule is not ambiguous (See Fig. 8). Suppose that the rule-A is (SA,DA)=(1100,0101*) and the rule-B is (SA,DA)=(010*,010*). If we divide the rule into the SA part (first half) and the DA part (second part), the second part for those rules are ambiguous (0101* and 010*). To solve this ambiguity problem with the second part, we introduce the *rule-id* in the TCAM. The rule-id is stored with the first part of the rule. The rule-id is used to distinguish the first part of the policy rule. Distinct rule-id is assigned to each entry for the first half of policy rule. When we are doing the first part matching (*round-id* is also introduced to distinguish the round of the search: first or second), the rule-id is determined. After doing the first part matching, we use the rule-id to do the second part matching. The *second search flag* is stored with the first half of the rule to indicate whether the second half for the rule exists or not. If the second half does not exist, the second half can be skipped.

When the packet whose SA and DA are 0100 and 0101 comes,

The above-mentioned solution, which introduced rule-id, is incomplete because it cannot avoid the ambiguity of the first part of the rule. In the first half matching phase, the rule comes first (say, the rule-X) is selected if multiple rules match the packet header. The rule-id is used in the second half matching phase and so the possible candidates are excluded in the second half

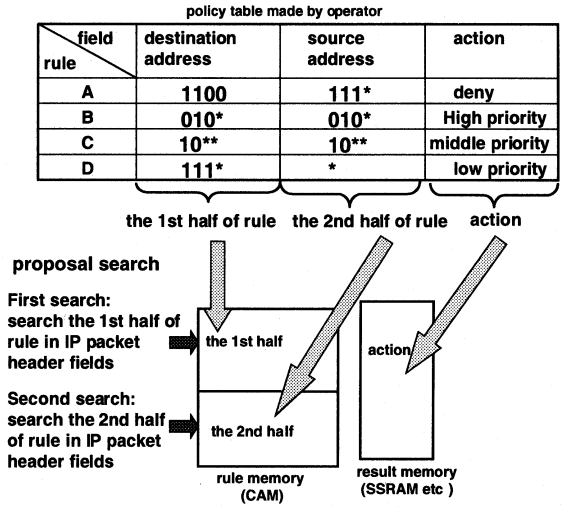

Figure 7: ProposedMethod.

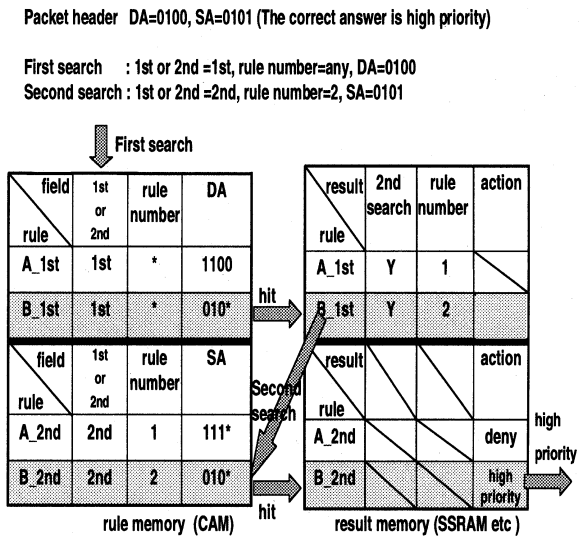

Figure 8: Problem with second half ambiguity.

matching phase. It follows that the correct matching rule (say, the rule-Y) is missed (See Fig. 9. In Fig. 9 the rule-X is (SA,DA,action)=(11**,01**,deny) and the rule-Y is (SA,DA,action)=(1***,11**,accept). Suppose that a packet whose SA and DA are 1100 and 1100 arrives. It matches the rule-X in the first half matching phase, because the rule-X comes first. We, however, failed to find the policy rule whose rule-id is the same as the rule-Xs that the arriving packet matches in the second half matching phase. In this case the packet classifier should find the policy rule-Y (SA,DA,action)=(1***,11**,accept) for the packet. Because the second half of the rule-Y is marked another rule-id different with the one for the rule-X, we failed to find it.

To avoid the second half ambiguity problem, we insert an extra rule (say, the rule-Z) such that it is the part of the rule-Y and its rule-id is the same as the rule-Xs

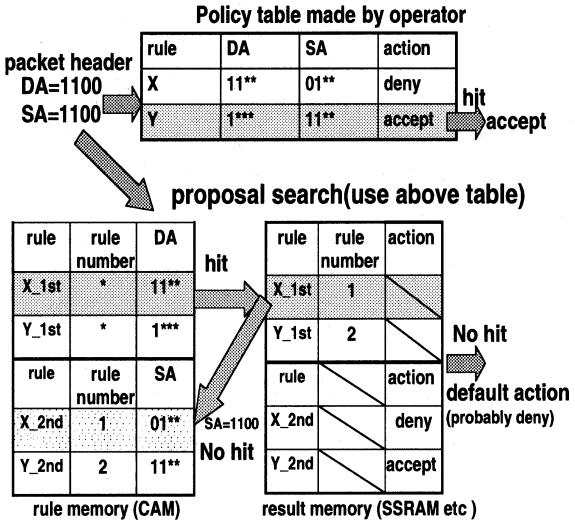

Figure 9: Problem with first half ambiguity.

(See Fig. 10. In this case the correct answer is the rule-Y. The above-mentioned problem with the first half ambiguity stems from the fact that the we fall into incorrect rule (the rule-X) during the first half matching phase. When we fall into the rule-X, we use the rule-id associated with rule-X in the second half matching phase. To recover from the rule-X in the second phase, we add an extra rule that is equivalent to the correct answer (the rule-Y) whose rule-id is the same as the rule-Xs in the original policy table. The extra inserted rule is broken into two pieces: first and second halves, each of which is stored in the same TCAM.

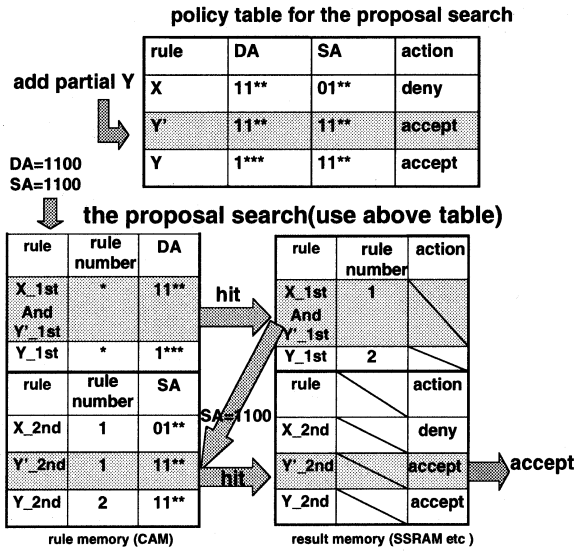

Figure 10: Extra rule insertion to avoid the second half ambiguity.

Which rule should be used as the extra inserted rule? In the example in Fig. 10 we used rule-Y as the extra inserted rule. For every pair of policy rules whose first half share the same prefix, we insert extra copy of the policy rule whose first halfs prefix is shorter with the rule-id being the same as the others.

In the above-mentioned example we explained the problem for two-tuple case. The same idea can be applied for more complicated case such as six-tuple case.

IV. Implementation and Evaluation

The proposed algorithm can be implemented by three devices ,TCAM which stores policy table, SSRAM which stores action table and a search engine(Fig. 11). The search engine extracts the header fields needed to search from a packet and performs the first search by TCAM. After the first search, the search engine extracts Assoc.Tag from SSRAM and performs the second search. If necessary. this process is continued. Finally, the search engine performs the action based the result from SSRAM to the packet.

We evaluate a search speed of the proposed algorism. This evaluation is based on using TCAM which is now commercially available [5]. This CAM can search 128 bit words at 33 ns. 304 bits search is needed for IPv6 packet classification. Third search is required by using above TCAM in worst case. This search engine has to perform 6 process which are the first search, the extracting Assoc.Tag for the second search, the second search, extracting Assoc.Tag for the third search, the third search and extracting the action. If each search stage uses one TCAM, faster search could be performed by pipeline process. Because Extracting Assoc.Tag and action is performed by reading SSRAM, this process needs less than 33 ns by using fast SSRAM. 33 ns is assigned each of these process for pipeline process. Figure 12 shows this pipeline process. This pipeline process provides 33 million/sec packet classification. In other words, the proposed algorism can provide 10 Gb/s packet classification [2].

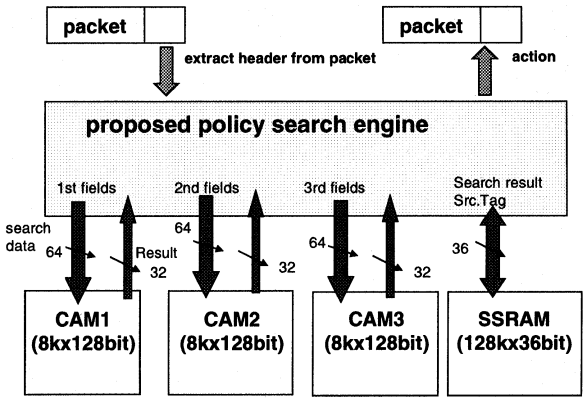

Figure 11: The proposed search engine

[2] when 1 packet length is 40 byte

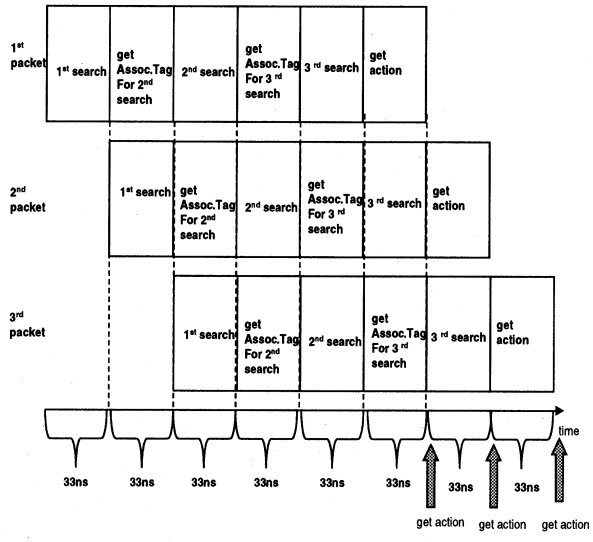

Figure 12: Evaluation

V. Conclusions

Contribution of this paper is summarized as follows. For policy table lookup, we break each policy rule into several pieces and store them in the same TCAM to get around the width limitation of TCAM. We can store long policy rule (policy rule is 304-bit width for IP version 6) in commercially available TCAM. We introduced the rule-id to avoid the second half ambiguity problem. We came up with inserting a copy of rule to avoid the first half ambiguity problem.

By applying above-mentioned idea, we can use commercially available TCAM for multi-layer table lookup, which enable us to build ultra high-speed packet forwarding engine for differentiated services in the Internet.

VI. References

[1] V. Srinivasan, G. Varghese, S. Suri, and M Waldvogel, "Fast and scalable layer four switching," in *Sigcomm '98*, Sept. 1998.

[2] T. V. Laksman and D. Stidialis, "High speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Sigcomm '98*, Sept. 1998.

[3] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *Sigcomm '99*, Aug. 1999.

[4] A. Hari, S. Suri, and G. Parulkar, "Packet filter management for layer4 switching," http://www.ccrc.wustl.edu/ hari/infocom.ps.

[5] "1M(128x8k) TCAM by Netlogic microsystems," *http://www.netlogicmicro.com/products/ipcam /NL85721.html.*