Fast Filter Updates for Packet Classification using TCAM

Haoyu Song, Jonathan Turner
{hs1, jst}@arl.wustl.edu
Washington University in St. Louis, USA, 63130

***Abstract*— This paper addresses the problem of efficient filter updates in TCAMs. Under realistic conditions, filter updates can lead to significant performance degradation. This paper introduces an approach to using TCAMs that encodes filter priority as a TCAM field, allowing the highest priority filter to be identified with a small number of lookups, while greatly simplifying filter set management, and reducing the impact of updates on lookup throughput. Our approach supports wire-speed processing for OC-192 links using commercially available TCAM components.**

I. Introduction

TCAMs are widely deployed in high performance network routers for packet classification because of their unmatched lookup throughput, and their generality. In TCAMs, packet filters are represented as ternary bit strings and stored in decreasing priority order. Given a packet header, the search for the best matched filter with the highest priority is performed on all the entries in parallel. The index of the first matched filter is then used to access a memory to retrieve the associated data. This elegant architecture supports classification of a packet in just a single clock cycle, allowing a state-of-art TCAM chip to support a sustained search rate of 250 million packets per second. Even for OC-192 networks, the peak packet rate in the worst case is no more than 30M packets per second, far less than the search capability that a TCAM provides. While TCAMs remain the most popular choice for high performance packet classification, they do have several drawbacks.

- *Low Density & High Cost.* A TCAM requires up to 16 transistors for a bit while SRAM requires just six and SDRAM just one. Consequently, the storage density of a TCAM is significantly lower than that of commodity memory technologies. Moreover, the relatively small market for TCAMs makes them expensive, with a cost per bit that is roughly 20 times that of SRAM and hundreds of times that of DRAM.
- *High Power Consumption.* Because they search all entries in parallel on every packet, TCAMs consume a lot of power. 25 Watts per TCAM component is a fairly typical power budget, adding significantly to the cooling requirements for a backbone router. Modern TCAMs allow entries to be grouped into segments, that can be selectively searched in order to reduce power usage. When filters are partitioned among segments appropriately, this can significantly reduce power consumption [9], [5], [10].
- *Arbitrary Range Support.* TCAMs naturally support searches on ternary bit strings. This is not ideal for packet filters that include arbitrary ranges for some of their fields. The naive way to solve this problem can lead to significant expansion in the space required to represent a filter [5]. This observation has triggered the development of new methods that combine single field searches with encoded range values [2], [1], [3] and proposals for direct hardware support of range lookups [5].
- *Multiple-Match Support.* Recent network security applications require all the matching filters to be reported, not just the first one. Conventional TCAMs only output the matching filter with the smallest index. It seems likely that future TCAMs, driven by new requirements, will be designed to support multiple matches efficiently. Algorithmic approaches for dealing with this problem are discussed in [8], [1].

In this paper, we focus on TCAM filter set management, a problem that has received relatively little attention in the research literature. In an operating router, filter sets update over time in response to changes in network management policies and link availability. New filters may be inserted and existing ones deleted or modified. Because TCAMs return the first matching filter, based on storage position within the TCAM, insertion of a new filter can require many other filters to be moved in order to place the new filter at the appropriate position. In the worst case, a large fraction of the filters in a filter set may need to be moved for each insertion. While many filter deletion and modification operations can be done without moving filters (using "lazy deletion" and in-place modification), in the worst case these operations can also require large numbers of filters to be moved.

While the rate at which filters are updated is much smaller than the rate at which lookups are processed, filter updates can have a significant impact on lookup rate, since updates must be suspended while a control processor makes the changes needed to complete a TCAM update. Wang et. al. show that the movement of just 16 TCAM entries in an OC-192 router can trigger the dropping of 18 packets [7]. As applications requiring more frequent updates emerge, the impact of updates on lookup performance may becomes much worse.

As we have mentioned, the lookup throughput of TCAMs actually exceed the requirements in typical applications. This suggests the possibility of trading off lookup throughput for more efficient filter updates. We show that this trade-off can be exploited to good effect, by encoding the priority as a field in the TCAM and using multiple lookups to identify the highest-priority matching filter. The resulting system can sustain worst-case lookup rates of more than 65 million per second, and average rates of more than 80 million per second.

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

II. Related Work

When TCAM is used for longest prefix matching (LPM), at most W moves are needed to insert a new prefix, where W is the number of unique prefix lengths [4]. Because for any packets, there is at most one matching prefix among prefixes of the same length, prefixes can be placed in the TCAM in decreasing order of their lengths. This ensures the correct IP lookup result and makes it relatively easy to update an entry. The update algorithm uses the property that changing the relative order of prefixes of the same length does not affect the lookup result, so one can insert a new prefix by moving just one prefix for every distinct prefix length. Since there are less than 32 distinct prefix lengths, the update time is bounded and reasonably small. One can do even better by storing prefixes in chain-ancestor order [4]. Carefully refining the memory layout can further reduce the total number of entry moves. Unfortunately, as will be explained, this approach cannot be directly used for general packet classification.

Reference [7] is one of the few prior studies of the TCAM update problem for packet classification. The authors focus on how to maintain consistent filter table throughput during the update process. They show that TCAM locking can be avoided by carefully managing the update process so that correct filter matches are ensured, even while the filter update is in progress. However, their method significantly increases the number of moves required, and while they do not lock up lookups during the update process, the filter moves do still consume TCAM bandwidth. In addition, the filter set management process is relatively complex and the batch processing introduces a significant latency, which delays the time for an update to take effect.

A typical TCAM component provides 144 bits for matching a packet header. In IPv4 applications, some of these bits are not needed because the standard 5-tuple packet header contains only 104 bits. These otherwise unused bits can be used for other purposes. The MUD algorithm uses these bits to attach a filter index to each filter in order to support multi-match classification [1]. In this algorithm, the filters are stored in incremental index order. If the first lookup returns a matching filter with index j, then in the subsequent lookups, we only need to search the filters with index greater than j. Our algorithm is similar to the MUD algorithm in the sense that it also encodes additional information in the TCAM entries. However, the information that we add is different and, we use it to improve the efficiency of filter set updates rather than to enable multi-match classification.

III. Algorithm

If enough empty entries are allocated between any two filters in a TCAM, then to insert a new filter, we can simply insert it in an appropriate empty entry, without moving any other filters. Although this is a tempting solution, there are two problems with it. First, in order to reduce TCAM power consumption, we prefer to store the filter set in as few TCAM segments as possible. Allocating empty entries between filters, makes it necessary to search more segments for a given filter set, increasing power consumption. Second, because we cannot predict future updates, we cannot guarantee that there will always be an empty position in the TCAM where we need one. When there is no empty entry, filter moves become necessary.

From this discussion, we identify two important objectives. First, we would like to store the filter set in a TCAM compactly without allocating empty entries between filters. This allows a linear growth of occupied entries and segments as the size of the filter set grows, reducing the power required for lookups. Second, we would like to minimize the movement of entries, so as to reduce the amount of work that must be done for each update and to minimize the impact of updates on lookup throughput.

A. Real Filter Priority

A filter's order in a filter set naturally reflects its priority, so the filter index can be used as its priority value. In fact, only overlapping filters need to be ordered relative to one another, in order to ensure the correctness of lookup results. Therefore, filters can be divided into groups in such a way that filters in the same group can exchange their order at will, without affecting the lookup results. The order of the groups, however, cannot be exchanged. To be specific, each group is assigned a priority value. The group of filters with a higher priority must be stored in a lower address region of a TCAM than the group of filters with a lower priority.

The algorithm for grouping the filters and assigning the priority values can be described as follows. We start from a graph in which each vertex denotes a filter. For each filter, r_i, we examine all the other filters, r_j, which overlap with r_i (i.e. $r_i \cap r_j \neq \emptyset$). If $i > j$, we create a directed edge from r_j to r_i; otherwise, we create a directed edge from r_i to r_j. This step generates a directed acyclic graph. The topological order of the vertices in this graph reflects the relative priorities of filters. In the second step, we assign priority values to filters. Each vertex with no predecessors is assigned a priority value of zero. Other vertices are assigned a priority value only after all their predecessors have been assigned a priority value. The priority value assigned to a vertex is one plus the largest priority value assigned to any of its predecessors. An example of this process is shown in Figure 1.

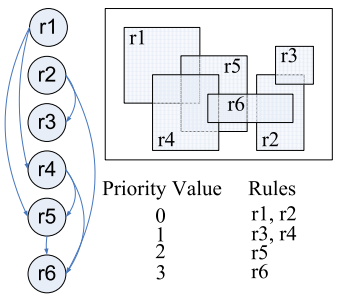

Fig. 1. Grouping and Priority Value Assignment

We evaluated real filter sets and found that the number of distinct priority values is much smaller than the number of

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

filters, as shown in Table I. We also evaluated large synthetic filter sets generated by ClassBench [6] tool and found that the number of priority levels is insensitive to the number of filters. Even for filter sets with ten thousand filters, the number of priority levels is less than 64. This property implies that if filters are updated based on their priority values, significantly less work needs to be done than if they are updated using their absolute position in the TCAM.

TABLE I
REAL PRIORITY LEVELS IN SOME FILTER SETS

filter set	filters	priorities	filter set	filters	priorities
acl1	814	40	fw1	283	53
acl2	623	35	fw2	184	55
acl3	2,400	22	fw3	160	49
acl4	3,061	22	ipc1	1,702	42
acl5	4,557	13	ipc2	192	42

The filter grouping is analogous to the prefix grouping by the prefix lengths or the chain-ancestor ordering for LPM [4], but updating filters for general packet classification is much more complex than updating prefixes for LPM. First, the number of priority levels in filter sets for the general packet classification is much more than the number of unique prefix lengths in prefix sets for the IP lookup. Second, updating a prefix in a prefix length group does not affect any other prefixes. Therefore, the number of entry moves required is bounded by the number of unique prefix lengths. However, for general packet classification, updating a filter may change multiple filters' priority values. Figure 2 illustrates the grouping result after a new filter R, which has an index between $r1$ and $r2$, is inserted into the set. Notice that $r4$, $r5$, and $r6$ all have to change their priority values. This implies that after a new filter is inserted, the priority values of several others need to be adjusted to maintain a correct topological priority order. Similar actions need to be taken after a filter is removed or modified in the filter set.

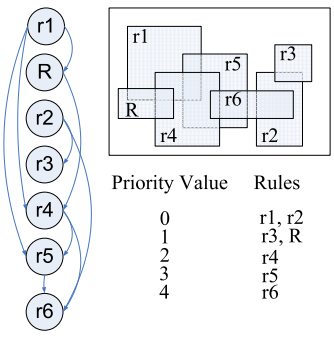

Fig. 2. Insert a New Filter R

In the worst case, all filters need to change their priority value. Fortunately, in reality this is unlikely to happen. We will show this point through analysis and the simulation.

B. Using Extended Filter

We have shown that a typical TCAM entry configuration results in some unused bits. Using a few of these unused bits, we attach the real priority value to each filter. Now if the packet classification always looks up the extended filters, the filters need not be stored in their priority order in a TCAM. Actually, a new filter can be written in any empty entry in a TCAM and no other filter needs to be moved. Clearly, now the search key has to also include the priority value. The lookup process is no longer looking for the matching filter with the minimum TCAM index but the matching filter with the minimum attached priority value. Without the prior knowledge of the priority, multiple lookup attempts are needed to figure out the best matching filter with the minimum priority value.

A linear search on the priority values does not scale to large filter sets with many priority levels. Fortunately, TCAMs have a set of reconfigurable Global Mask Registers (GMR) which can selectively mask out any bits in all entries as "don't care". Each filter has a priority value with all bits enabled. By configuring GMR bits, we can determine which bits of the priority value should be considered as if each GMR enables a range of priority values. Each TCAM lookup therefore designates one of the preset GMRs to search only a range of priority values. The result narrows down the search range for the next lookup, and eventually identifies the best match. Typically, a TCAM has up to 64 GMRs. They are more than enough for our purpose.

Figure 3 illustrates an example where there are at most 32 priority values in the filter set. The binary decision tree is traversed based on the search result of the previous lookup. If the TCAM reports a match, we follow the upper branch of the tree; otherwise, we follow the lower branch of the tree. For example, given a packet header, we first search any matching filter with the priority value between zero and 15. If the result is positive, we then search any matching filter with the priority value between zero and seven; otherwise, we search the range eight to 11, and so forth. Since each lookup step halves the searched priority range, this scheme needs only $\log N$ lookups per packet to find the best matching filter, where N is the number of unique priority values.

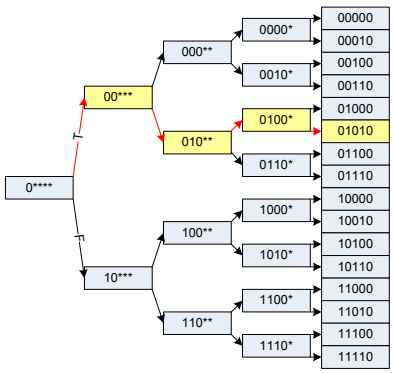

Fig. 3. Searching the Filter with the Minimum Priority Value.

Note that in this scheme, the only information used is whether or not the TCAM reports a match. If we also know the matching filter's priority value, we can accelerate the search. For example, if in the first attempt, we find a matching filter

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

with a priority value of zero, then there is no further search needed. Even when the priority value is not zero, we can use the value to advance the search range quickly in the decision tree. For example, if the first search in the range of zero to 15 returns a match with the priority value of two, then in the next step we can directly search the range zero to one rather than zero to seven.

To achieve this, we store the filter's priority value in the associated data memory as a part of the filter's associated data. Each lookup step reads this value if there is a match in the TCAM. The control logic then uses this information to choose another GMR for the next lookup or terminates the lookup. Accessing the associated data memory is pipelined with the TCAM lookups, so the TCAM throughput is unchanged.

We also use another TCAM feature to help improve the lookup performance. Along with the matching filter index, the TCAM also has a multi-match output signal indicating if there is more than one matching filter for the given key. Since our search order is in favor of the higher priority filters, during the search, if the multi-match signal shows only one single match for the given key, the filter is guaranteed to be the best matching one. In such a case, no further search is needed.

C. Lookup

The lookup of a filter (*SearchTCAM [key]*) involves a sequence of recursive calls to the sub-procedure (*SearchPriorityRange [key, low, high]*) that searches a range of priority values using a GMR. In the following pseudo code, *IsMultiMatch* is asserted when more than one filters are matched. *Priority(i)* is filter i's priority value acquired from the associated data memory. *Low* and *high* define the priority value range which can be represented with a prefix string.

SearchTCAM $[key]$
1. $low = 0$
2. $high = 2^{\lceil \log_2 MaxPriority \rceil} - 1$
3. $SearchPriorityRange\ [key, low, high]$

SearchPriorityRange $[key, low, high]$
1. $get\ filter\ index\ i$
2. $if\ (i \neq NULL)$
3. $\quad if\ (priority(i) = low\ OR\ !\ IsMultiMatch)$
4. $\quad\quad return\ i\ as\ the\ best\ match$
5. $\quad else\ if(priority(i)\ \neq low\ AND\ IsMultiMatch)$
6. $\quad\quad high = low + 2^{\lfloor \log_2(priority(i) - low) \rfloor} - 1$
7. $\quad\quad regi = i$
8. $\quad\quad SearchPriorityRange\ [key, low, high]$
9. $else$
10. $\quad if\ (is\ the\ first\ TCAM\ lookup)$
11. $\quad\quad return\ NULL$
12. $\quad else\ if(low = high\ OR\ priority(regi) = high + 1)$
13. $\quad\quad return\ regi\ as\ the\ best\ match$
14. $\quad else$
15. $\quad\quad low = high + 1$
16. $\quad\quad high = low + 2^{\lfloor \log_2(priority(regi) - low) \rfloor} - 1$
17. $\quad\quad SearchPriorityRange\ [key, low, high]$

D. Update

The update process includes inserting, deleting, and modifying filters. All of these may result in multiple filters changing their priority values. Inserting a filter implies some filters need to increase their priority value, deleting a filter implies some filters need to decrease their priority value, and modifying a filter can do both. The analysis can be done through the DAG we built in III-A.

An update involves a sequence of accesses in the TCAM and the associated memory. By performing accesses in the proper order, we can do an update using the spare TCAM cycles without blocking the normal lookups. To insert a new filter, we first get the set of filters that need to increase their priority values. We sort these filters in decreasing priority value order and then increase their priority value in turn. At last, we insert the new filter in any empty entry. For a better lookup performance, we should choose the best available entry for the new filter. Ideally, among all the filters that overlap the new filter, those with smaller priority values should be located in the small indexed entries and those with larger priority values should locate in the large indexed entries.

IV. Evaluation

A. Filter Distribution

The efficiency of our algorithm depends on the filter distributions. We have shown that even for very large filter sets, the number of unique priority values, which determines the worst-case performance bound, is small. We also examine the filter distributions in different priority value groups. An example is shown in Figure 4. The majority of filters are concentrated in groups with small priority values. This fact has two favorable implications. First, it benefits the lookup process since a packet has a higher possibility to match a filter with small priority value and our search starts from the filters with small priority values. Second, it implies the long dependent chains comprise only a few filters; hence our update process will not affect too many filters.

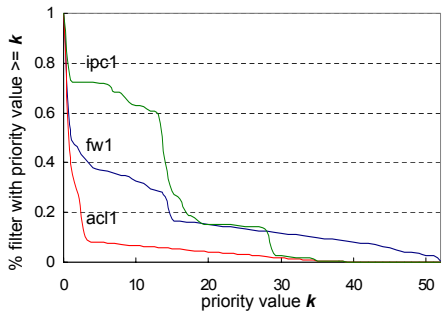

Fig. 4. The Priority Value Distribution

Indeed, the priority dependency is a result of filter overlaps. If the maximum number of overlapped filters that a packet can match is small, our lookup and update algorithms both work better. In [1], 112 real filter sets are analyzed. In only one filter set did a packet match as many as eight filters. In the majority of filter sets, no packet matched more than five filters.

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

TABLE II

LOOKUP THROUGHPUT PERFORMANCE (ASSUME TCAM RUNS AT 250MHZ CLOCK)

filter set	acl1 (814 filters)		fw1 (283 filters)		ipc1 (1,702 filters)		acl1_syn (4,415 filters)	
	# accesses	throughput	# accesses	throughput	# accesses	throughput	# accesses	throughput
the best case	1.38	181 Mpkt/s	2.18	115 Mpkt/s	3.23	77 Mpkt/s	2.08	120 Mpkt/s
the average case	2.65	94 Mpkt/s	2.68	93 Mpkt/s	4.14	60 Mpkt/s	2.97	84 Mpkt/s
the worst case	3.20	78 Mpkt/s	2.90	86 Mpkt/s	5.37	47 Mpkt/s	4.66	54 Mpkt/s

B. Lookup Throughput Performance

For each filter set, we generate a packet header trace using the ClassBench tools [6] to evaluate the lookup performance. We evaluate the worst case lookup performance by storing all the filters in a TCAM in decreasing priority value order. The best case lookup performance happens when the filters are stored in increasing priority value order and the average case performance happens when the filters are randomly permuted in TCAM entries.

The simulation results for some filter sets are shown in Table II. Note that for any case, a packet needs at most six TCAM accesses to find the best matching filter, so in the absolute worst case, the TCAM can still classify 42 million packets per second, which is sufficient for the OC-192 link speed.

C. Update Performance

The update performance is determined by the number of TCAM entry writes needed when inserting, deleting or modifying a filter. The worst case update performance happens when we insert the filters in the reversed priority order or delete the filters in priority order. To evaluate the worst case update performance, we first reverse the filters' order as they appear in the original filter set, and then we insert the filters into the TCAM one by one. After each filter is inserted, we reevaluate all the filters' current priority value and count the number of filters that need to update their priority value. This number plus one more TCAM write that actually inserts the new filter is the overall number of TCAM writes needed for an update.

In Table III, we show the average number of TCAM writes and the maximum number of TCAM writes needed after all the filters are inserted into a TCAM. We can see the average number of TCAM writes is small but the maximum number of TCAM writes can be very large. Figure 5 shows the cumulative distribution of the TCAM write numbers.

TABLE III

THE WORST CASE UPDATE PERFORMANCE

filter set	avg. TCAM writes	max. TCAM writes
acl1	3.2	110
fw1	10.8	239
ipc1	12.3	1,099

These results further affirm us that the similar update algorithm used for LPM is not applicable for the general packet classification since too large number of memory moves can be involved. On the other hand, our algorithm needs only to rewrite the portion of the extended filters that holds the priority value, which can be done very fast in general and needs not block the lookup process.

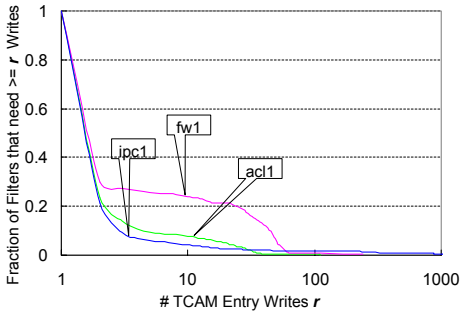

Fig. 5. The Worst Case Distribution of TCAM Accesses

V. CONCLUSIONS

In this paper we present an algorithm that trades off the surplus search capability of TCAMs for efficient filter set updates for packet classification. The real priority values of filters are derived and attached to the filters. Using the binary search on the priority values and some other features of TCAMs, the algorithm maintains a lookup throughput that is sufficient for backbone routers running at OC-192+ speeds. At the same time, that algorithm greatly reduces the work required for filter set management thus it is quite suitable for the dynamic environment where filter updates occur frequently.

REFERENCES

[1] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary. Algorithms for Advanced Packet Classification with Ternary CAMs. In *ACM SIGCOMM*, 2005.

[2] H. Liu. Efficient Mapping of Range Classifier into Ternary-CAM. In *HotI*, 2002.

[3] J. Lunteren and T. Engbersen. Fast and Scalable Packet Classification. *IEEE Journal on Selected Areas in Communications*, 21, May 2003.

[4] D. Shah and P. Gupta. Fast Incremental Updates on Ternary-CAMs for Routing Lookups and Packet Classification. In *HotI*, 2000.

[5] E. Spitznagel, D. Taylor, and J. Turner. Packet Classification Using Extended TCAMs. In *ICNP*, 2003.

[6] D. E. Taylor and J. S. Turner. Classbench: A Packet Classification Benchmark. In *IEEE INFOCOM*, 2005.

[7] Z. Wang, H. Che, M. Kumar, and S. Das. CoPTUA: Consistent Policy Table Update Algorithm for TCAM without Locking. *IEEE Transactions on Computer*, 53, Dec. 2004.

[8] F. Yu, T. Lakshman, M. A. Motoyama, and R. H. Katz. SSA: A Power and Memory Efficient Scheme to Multi-Match Packet Classification. In *ANCS*, 2005.

[9] F. Zane, G. Narlikar, and A. Basu. CoolCAMs: Power-efficient TCAMs for Forwarding Engines. In *IEEE INFOCOM*, 2003.

[10] K. Zheng, H. Che, Z. Wang, and B. Liu. TCAM-Based Distributed Parallel Packet Classification Algorithm with Range-Matching Solutin. In *IEEE INFOCOM*, 2005.

1-4244-0357-X/06/$20.00 ©2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.