Space and Speed Tradeoffs in TCAM Hierarchical Packet Classification

Alex Kesselman*, Kirill Kogan†, Sergey Nemzer‡ and Michael Segal§
*Google, Inc.
Email: alx@google.com
†Cisco Systems, Netanya, Israel
and
Communication Systems Engineering Dept., Ben Gurion University, Beer-Sheva, Israel
Email: kkogan@cisco.com
‡School of Computer Science, Tel Aviv University, Israel
and
Compugen Ltd., Tel Aviv, Israel
Email: sergey.nemzer@cgen.com
§Communication Systems Engineering Dept., Ben Gurion University, Beer-Sheva, Israel
Email: segal@cse.bgu.ac.il

***Abstract*— Hierarchical packet classification is a crucial mechanism necessary to support many Internet services such as Quality of Service (QoS) provisioning, traffic policing, and network intrusion detection. Using Ternary Content Addressable Memories (TCAMs) to perform high-speed packet classification has become the de facto standard in industry. TCAMs compare packet headers against all rules in a classification database concurrently and thus provide high throughput unparalleled by software-based solutions. However, the complexity of packet classification policies have been growing rapidly as number of services deployed on the Internet continues to increase. High TCAM memory requirement for complex hierarchical policies is a major issue as TCAMs have very limited capacity. In this paper we consider two optimization problems of dual nature: the first problem is to minimize the number of TCAM entries subject to the constraint on the maximum number of levels in the policy hierarchy; the second problem is to minimize the number of levels in the policy hierarchy subject to the constraint on the maximum number of TCAM entries. We propose efficient dynamic programming algorithms for these problems, which reduce the TCAM memory requirement. To the best of our knowledge, this is the first work to study the fundamental tradeoff between the TCAM space and the number of lookups for hierarchical packet classification. Our algorithms do not require any modifications to existing TCAMs and are thus relatively easy to deploy.**

I. Introduction

Growing usage and diversity of applications and attacks on the Internet makes fine-grained traffic classification the key critical issue. As a result, high-speed algorithms that scale to large multi-field databases have become a widespread requirement for a variety of network services including QoS bandwidth management, firewalls and intrusion detection. Many complicated classification policies are naturally represented in a hierarchical fashion. For instance, the top level of a hierarchical policy of an Internet Service Provider (ISP) can match the customer company, the secondary level can match the department of this company, and the third level can match specific applications. In a nutshell, a router maintains a classification policy under which incoming or outgoing packets are classified by matching against a set of rules. In addition, each rule can also specify a set of actions to be taken on packets matching this rule. Supporting hierarchical packet classification is a challenging task as it requires to perform matching at multiple levels of hierarchy in the line rate.

In this work we explore hierarchical classification with *Ternary Content-Addressable Memory* (TCAM). A TCAM is a memory device that stores data as a massive array of fixed-width ternary entries. A ternary entry is a string of bits where each bit is either 0, 1 or $\times$ ("don't care"). The TCAM searches the packet in parallel against all the ternary entries stored in the memory and produces the first rule that matches the packet. Remarkably, TCAM guarantees that each lookup is done in constant time. Usually each TCAM entry is wide enough to contain the concatenation of all the packet fields to be matched, possibly having room for some extra bits. If a matching rule consists solely of fields that specify exact or prefix matches, then it can be represented by a TCAM entry in a straightforward manner (a prefix match field is padded with the appropriate number of $\times$'s in the least significant bits). A range value may be converted to multiple prefixes or exact entries to fit the TCAM format. However, TCAMs have some limitations. Current TCAMs can support up to 133 million searches per second for 144-bit wide keys, and can store $128K$ ternary entries in a single device. TCAMs can also be configured as 72-bit and 288-bit width.

To implement a hierarchical policy, the classifier needs to access TCAM for each level of hierarchy. However, the number of TCAM lookups that can be done in the line rate is very limited. To address this bottleneck, a hierarchical policy can be converted to an equivalent policy with less levels of hierarchy through the process of flattening. Unfortunately, flattening may significantly increase the number of TCAM entries. Thus, there arises an interesting tradeoff between the

depth (number of levels in the hierarchy) of a hierarchical policy and the required TCAM space. On the one hand, the router may not be able perform d consecutive TCAM lookups for a hierarchical policy of depth d in the line rate. On the other hand, the policy flattened to a single level may not fit the TCAM memory as such a conversion may result in the TCAM space requirement exponential on d.

A. Our Results

In this paper we study two major problems dealing with hierarchical packet classification using TCAM. We are given a policy P with d levels of hierarchy and the goal is to convert P to an equivalent policy P' that needs to fulfill certain constraints minimizing the number of lookups or TCAM entries required. The proposed algorithms are very efficient as the running time depends only on the policy depth rather than the number of TCAM entries, which can be orders of magnitude larger. Moreover, our algorithms do not require any modifications to existing TCAMs being very easy to deploy. Due to lack of space the simulation results are omitted from this extended abstract and presented in the full version of the paper.

In the space optimization problem, we minimize the number of TCAM entries required by P' subject to a limit on the maximum number of hierarchy levels. The motivation behind this problem is to reduce the required TCAM space allowing packet classification in wire speed without incurring a possibly huge memory blowup if P is flattened to a single level. We propose two dynamic programming algorithms for this problem: the first algorithm is more efficient but is restricted only to policies that match disjoint fields in a packet header at different levels of the hierarchy; the second algorithm is slightly more complicated and can process general policies. The running time of the first and the second algorithm is at most $O(d^3)$ times the number of terminal (leaf) classes in P.

In the speed optimization problem, we minimize the number of hierarchy levels in P' subject to a limit on the maximum number of TCAM entries. The rationale of this problem is to utilize the available TCAM capacity as efficiently as possible to reduce the number of lookups. We propose an algorithms that applies one of our speed optimization algorithms as a subroutine. This algorithm adds a factor of $O(\log d)$ to the running time of the corresponding speed optimization algorithm.

B. Related Work

Designing algorithms that scale to millions of rules and millions of searches per second has been and continues to be an important line of research. Many software-based sophisticated approaches have been proposed in the past few years including Recursive Flow Classification [6], Crossproducting [14], [15], HyperCuts [13], Extended Grid-of-Tries [1] and Aggregated Bit Vector [2], to name just a few. Comprehensive surveys on this subject can be found in [4], [8], [16]. The complexity bounds derived by means of computational geometry imply that any software-based packet classifier with N rules and $k > 2$ fields, uses either $O(Nk)$ space and $O(\log N)$ time or $O(N)$ space and $O(\log^{k-1} N)$ time [12]. Thus, many software-based approaches are either too slow or too memory intensive for $k > 2$. Though packet classification algorithms using decision trees achieve better time-space tradeoffs [7], [17], they exploit statistical characteristics that are not reliable in general.

Due to the inherent limitations of software-based approaches, industry has increasingly employed hardware-based *Ternary Content Addressable Memory* (TCAM) for performing packet classification making it the dominant method [18], [19], [20]. A large class of packet classification systems that require up to a few hundred thousand rules have adopted TCAM for packet classification at multigigabit speeds [3], [5]. Several schemes for converting ranges to TCAM rules have been proposed in [9], [10], [11].

C. Paper Organization

The rest of the paper is organized as follows. The model description appears in Section II. The algorithms for space and speed optimization are presented in Section III and Section IV, respectively. We conclude with Section V.

II. Model Description

In this section we introduce the formal notation and define the Hierarchical Speed and Space Optimization problems.

A. Notation

A packet header contains k fields, where a field H_i ($1 \leq i \leq k$) is a string of W_i bits. In an IPv4 packet, classifiers usually check the following six fields: the Type of Service (8 bits), the Destination Address (32 bits), the Source Address (32 bits), the Destination Port (16 bits), the Source Port (16 bits), and the Protocol Type (8 bits). Note that classifiers may access other fields besides TCP/IP header such as MAC or application headers. Packets are matched against classification rules stored in a classification database.

The classification database of a router consists of a finite set of n rules, $R_1 \ldots R_n$. Each rule R specifies matchings for one or more (up to k) fields. For each header field H_i, a rule can specify a filter F_i of length $|F_i|$ ($|F_i| \leq |H_i|$), which can be any of two kinds of matches: exact match or prefix match.

1) A packet header field H_i exactly matches the filter F_i if and only if $H_i = F_i$.
2) A packet header field H_i is a prefix match for the filter F_i if and only if the $|F_i|$ bits of H_i are equal to F_i.

A packet p matches rule R if each of p's header fields matches the corresponding filter of R if any. The header fields for which filters are not specified by the rule are matched in TCAM by a wildcard filter ("don't care"). Since a packet may match multiple rules, the classification problem is to determine the first matching rule in an ordered sequence of rules.

There is also a third type of matching, so called range matching, where the header value should fall into a contiguous interval specified by the filter. In typical packet classifiers, such fields as the source and destination port numbers are

represented as ranges rather than prefixes. Though range rules cannot be directly stored in TCAMs, they are usually converted to a corresponding set of prefixes each of which is stored in a separate TCAM entry. In this paper we deal with classification rules that reside in a TCAM device and assume that all filters are exact match or prefix (the classification database may undergo a conversion if necessary [10], [11], [9]).

We define a *class* C to be an ordered set of rules and a set of *actions* to be taken on the packet. For instance, a QoS action may be packet marking with a pre-defined DSCP value while a security action may be packet accept or reject. We denote by $|C|$ the number of rules or *cardinality* of C. We say that the class is matched if the first matching rule belongs to this class. A *policy* P is an ordered set of classes. The last class of a policy is usually so called *default* class matching all packets that have not been matched by the other classes. Note that a global order of the rules is obtained by listing the rules in the corresponding classes.

The action of a class can also apply another policy in recursive manner, creating a *hierarchical policy*. Each recursive application creates a new *level of hierarchy*. A hierarchical policy can be viewed as an directed and acyclic graph G_P with classes acting as nodes and each edge representing a recursive policy application (see Figure 1). A directed path $\mathcal{C} = C_1 \rightarrow C_2 \rightarrow \cdots \rightarrow C_d$ from a *root* class with no incoming edges to a *terminal* class with no outgoing edges is called a *class chain*. The length of the longest class chain is defined as the *policy depth*. In order to match all rules on a class chain $\mathcal{C}$, a packet has to match rules $R_1 \in C_1$, $R_2 \in C_2, \ldots, R_d \in C_d$. Observe that the total number of class chains equals the number of terminal classes.

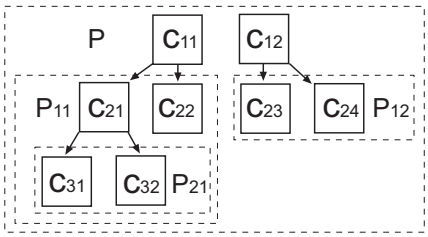

Fig. 1. An example of policy graph G_p.

In the actual implementation of a hierarchical classifier, a special header field H_0 is added to a packet at all but the first level of the hierarchy to identify the class sub-chain $\mathcal{S} = C_1 \rightarrow C_2 \rightarrow \cdots \rightarrow C_l$ $(1 < l \leq d)$ matched by the packet up to and including level l. In this way, the classifier can track the path of the packet in G_P and take the appropriate actions as matching proceeds. Furthermore, all the rules corresponding to the children of the class C_l in G_P have a common filter F_0 that identifies the class sub-chain $\mathcal{S}$. As a result, different class chains in a hierarchical classifier are independent in the sense that distinct instances of the same class are created for each node in G_P. We assume that TCAM entries are wide enough to store this extra filter.

Hierarchical policies allow a high degree of flexibility and modularity in policy definition. However, a separate TCAM lookup needs to be done for each level of the hierarchy in the process of classification, which may incur large delays for policies with high depth. To speed up the classification process, a hierarchical policy can be converted to an equivalent policy of lower depth. At the same time, such a conversion can significantly increase the number of TCAM entries required to store the intersected rules. We define the *TCAM space* M of a policy as the total number of rules in all levels of the classification hierarchy.

The operations defined below deal with policy flattening. We define the intersection of two classes $C \times C'$ as a class consisting of all possible intersections of rules $R \times R'$, where $R \in C$ and $R' \in C'$ that sequentially applies actions of C and C'. The intersection of rules $R \times R'$ is defined as the union of filters for disjoint header fields and the intersection of filters for common header fields in R and R'. Next we will describe how to intersect two filters F_i and F'_i specified for a common header field H_i. Suppose without loss of generality that $|F_i| \leq |F'_i|$. If F_i is a prefix of F'_i, then $R \times R'$ will contain filter F'_i. Otherwise, $R \times R' = R_\emptyset$ is a special empty rule that does not match any packet.

Observe that $|C \times C'| \leq |C| \cdot |C'|$. For example, consider the subsequent classification policy. Let C consist of three rules $[ToS = 1]$, $[ToS = 2]$ and $[ToS = 3]$ and C' consist of two rules $[DstPort = 21]$ and $[DstPort = 80]$. In this case, $|C \times C'| = 6$ as the filters are specified for disjoint header fields and $C \times C'$ contains the following rules: $[ToS = 1, DstPort = 21]$, $[ToS = 2, DstPort = 21]$, $[ToS = 3, DstPort = 21]$, $[ToS = 1, DstPort = 80]$, $[ToS = 2, DstPort = 80]$, $[ToS = 3, DstPort = 80]$. At the same time, the TCAM space required to represent the intersection of classes may be smaller than the cardinality of the classes themselves if filters are specified for common header fields. For instance, suppose that C contains three rules $[ToS = 1]$, $[ToS = 2]$ and $[ToS = 3]$ and C' contains two rules $[ToS = 3]$ and $[ToS = 4]$. We have that $|C \times C'| = 1$ and $C \times C'$ includes just one rule $[ToS = 3]$.

B. Problem Statement

We assume that it is possible to extract all relevant packet fields in a single step according to the classification format. We say that two packet classifiers are (semantically) equivalent if and only if they apply the same actions on each packet. Next we define the optimization problems studied in this paper.

Definition 2.1: **Hierarchical Space Optimization Problem:** Given a hierarchical policy P with depth $d > 1$, the goal is to convert P to an equivalent policy P' with depth of at most l $(l < d)$ that minimizes the required TCAM space.

Definition 2.2: **Hierarchical Speed Optimization Problem:** Given a hierarchical policy P that requires TCAM space of M, the aim is to convert P to an equivalent policy P' that minimizes the policy depth subject to the constraint that the TCAM space cannot exceed the target TCAM space A $(A > M)$.

III. Space Optimization

In this section we consider the problem of minimizing the TCAM space subject to the constraint on the policy depth. First we present an algorithm for the natural case where rules in any class chain apply only to disjoint header fields. Then we propose an algorithm for the general case where we impose no restrictions on the policy whatsoever.

A. Well-Structured Hierarchies

In this section we study *well-structured* hierarchical policies in which rules in any class chain apply only to disjoint header fields. Such hierarchies have the following important property, which allows us to use a fast dynamic programming algorithm operating merely with cardinalities of intersected classes without actually intersecting the rules themselves until the final stage.

Observation 1: In a well-structured hierarchy, for any class chain $\mathcal{C} = C_1 \rightarrow C_2 \rightarrow \cdots \rightarrow C_d$ we have that the cardinality of the intersection of all classes in $\mathcal{C}$, that is $|C_1 \times C_2 \times \cdots \times C_d|$, equals the product of the cardinalities of the individual classes $|C_1| \cdot |C_2| \cdots |C_d|$.

The space optimization algorithm for well-structured hierarchies ($SOAW$) is presented on Figure 2. $SOAW$ proceeds by first merging all chains of length greater than l into a single *virtual* class without actually intersecting the rules. Then each virtual merged class is splitted $l-1$ times in an optimal way using a level splitting algorithm based on the dynamic programming technique. Finally, for each virtual class the rules of the corresponding intersected classes are intersected to produce the output policy. Observe that class chains of length smaller than l are left untouched by $SOAW$. The running time of $SOAW$ is dominated by the running time of the level splitting algorithm, which is applied to all long chains in P. Recall that the total number of class chains equals the number of terminal classes.

Input: policy P of depth d, integer l ($l < d$)
Output: policy P' equivalent to P of depth l

- **Step 1: Merging Long Chains.** For each class chain $\mathcal{C} = C_1 \rightarrow C_2 \rightarrow \cdots \rightarrow C_d$ in P such that $d > l$ create a virtual class $C = C_1 \times C_2 \times \cdots \times C_d$ that represents the intersection of all classes in $\mathcal{C}$ (without intersecting the rules).
- **Step 2: Splitting Merged Chains.** Split each merged virtual class $l-1$ times using the level splitting algorithm.
- **Step 3: Intersecting Rules.** For each virtual class, intersect the rules of the corresponding intersected classes
- **Step 4: Creating Output Policy.** Output P' as a union of all the original chains of length at most l and the converted long chains.

Fig. 2. Space optimization algorithm ($SOAW$) for well-structured hierarchies.

Now we describe how to divide the level splitting problem into two sub-problems and combine solutions to these sub-problems into a solution to the original problem. For an intersected class $C = C_1 \times C_2 \times \cdots \times C_d$, we denote by $V(i,j,n)$ ($1 \leq i < d$, $i < j \leq d$) the cost of an optimal solution for the problem of n splittings in the intersected sub-class $C_{i,j} = C_i \times C_{i+1} \times \cdots \times C_j$. The cost is measured as the cumulative cardinality of the resulting $n+1$ sub-classes. We define initial values for the special case of $n = 0$ where nothing needs to be done and the special case of $n > j - i$ where no feasible solution exists:

$$V(i,j,n) = \begin{cases} |C_{i,j}| : n = 0, \\ \infty : n > j - i. \end{cases} \quad (1)$$

The main recurrence relation is defined as follows for $n > 0$ and $n \leq j - i$:

$$V(i,j,n) = \min_u (V(i,u,0) + V(u+1,j,n-1)) \quad (2)$$

for $i \leq u \leq j - n$.

Basically, in order to make n level splittings we consider all possibilities for the first splitting and perform exhaustive search over all the remained at most $n-1$ splittings in $C_{u+1,j}$ sub-class.

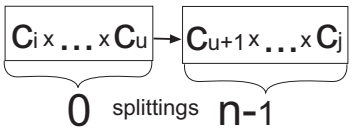

Fig. 3. Dynamic programming for well-structured policy.

Our aim is to minimize the overall cost of the produced solution. The level splitting algorithm (LSA) appears on Figure 4.

Input: class C (intersection of d classes), integer l ($l < d$)
Output: chain $\mathcal{C}'$ of length l

- **Step 1: Initialization.** Initialize $V(i,j,n)$ for $n = 0$ and $n > j - i$ according to Equation 1.
- **Step 2: Calculation.** Calculate all values $V(i,j,n)$ starting from $n = 1$ up to $n = l-1$ for $n \leq j-i$ using the recurrence Equation 2 and record the splitting of minimum cost.
- **Step 3: Reconstructing the optimal solution.** Construct the chain $\mathcal{C}'$ by splitting the classes of C with respect to the optimal solution of minimum cost $V(1,d,l-1)$.

Fig. 4. Level splitting algorithm (LSA) for well-structured hierarchies.

It is easy to see that the running time of LSA is $O(d^2 \cdot l)$ and the space complexity is $O(d^2 \cdot l)$, which is very reasonable since d and l are typically small numbers. The next theorem shows the correctness of LSA.

Theorem 1: The level splitting algorithm (LSA) finds an optimal solution of minimum cost for the problem of n splittings ($n > 1$) in a class formed as intersection of d classes ($d > n$).

Proof: The proof is by induction on the number of splittings n.

Basis ($n = 0$). Clearly, LSA finds an optimal solution for $n = 0$ since it just returns the original input as no splittings are necessary.

Step ($n \rightarrow n+1$)**.** Suppose that LSA finds an optimal solution for the number of splittings of at most n and let us show that it also finds an optimal solution for $n+1$ splittings. Note that LSA considers all options for making the first splitting, one of which necessarily corresponds to an optimal solution. Having done an optimal first splitting, it must be the case that the rest of this optimal solution consists of an independent optimal solution for the right subclass with at most n splittings. By the induction hypothesis, LSA finds an optimal solution for the right sub-class with at most n splittings. Therefore, LSA finds an optimal solution for at most $n+1$ splittings. ■

B. Arbitrary Hierarchies

In this section we deal with the general case of arbitrary hierarchies. Unfortunately, such hierarchies require to calculate the actual intersection of the rules for an intersection of classes in order to obtain its cardinality as demonstrated by the next observation.

Observation 2: In an arbitrary hierarchy, for any class chain $\mathcal{C} = C_1 \rightarrow C_2 \rightarrow \cdots \rightarrow C_d$ we have that the cardinality of the intersection of all classes in $\mathcal{C}$, that is $|C_1 \times C_2 \times \cdots \times C_d|$, is bounded from above but may not be equal to the product of the cardinalities of the individual classes $|C_1| \cdot |C_2| \cdots |C_d|$.

That necessitates using a slightly more complicated dynamic programming algorithm for level merging compared to the level splitting algorithm used for well-structured hierarchies. The space optimization algorithm for general hierarchies ($SOAG$) appears on Figure 5. $SOAG$ merges all chains of length greater than l by running a level merging algorithm based on the dynamic programming technique. The dominating component of the running time of $SOAG$ is the level merging algorithm, which processes all long chains in P.

Input: policy P of depth d, integer l ($l < d$)
Output: policy P' equivalent to P of depth l

- **Step 1: Merging Long Chains.** Merge all chains of length d' greater than l using the level merging algorithm with the number of merges $m = d' - l$.
- **Step 2: Creating Output Policy.** Output P' as a union of all the original chains of length at most l and the converted long chains.

Fig. 5. Space optimization algorithm ($SOAG$) for general hierarchies.

In what follows we present a way of dividing the level merging problem into two sub-problems and combining solutions to these sub-problems into a solution to the original problem. For a class chain $\mathcal{C} = C_1 \rightarrow C_2 \rightarrow \cdots \rightarrow C_d$, define $V(i,j,n)$ ($1 \leq i < d$, $i < j \leq d$) as the cost of an optimal solution for the problem of merging n levels in the class sub-chain $\mathcal{C}_{i,j} = C_i \rightarrow C_{i+1} \rightarrow \cdots \rightarrow C_j$. We estimate the cost of a solution as the total cardinality of the produced $j-i-n+1$ sub-classes. Initial values are set for the special case of $n = 0$ where no merges have to be performed, the special case of $n > j-i$ which permits no feasible solution and the case of $n = j-i$ where all levels are completely merged. Note that there is no need to merge more than $d-l$ levels to obtain the desired solution. We define initial values for the special case of $n = 0$ where nothing needs to be done and the special case of $n > j-i$ where no feasible solution exists:

$$V(i,j,n) = \begin{cases} \sum_{u=i}^{j} |C_u| : n = 0, \\ \infty : n > j-i, \\ |C_i \times \cdots \times C_j| : n = j-i,\ n \leq d-l. \end{cases} \quad (3)$$

We specify the main recurrence relation for $n > 0$ and $j - i > n$ in the following way:

$$V(i,j,n) = \min_u (V(i,u,u-i) + \quad (4)$$

$$V(u+1, j, n-(u-i))) \quad (5)$$

for $i \leq u \leq i+n$.

Essentially, we cover all options for a leftmost merged class sub-chain $\mathcal{C}_{i,u}$ with $u-i$ mergings. Then we consider all possible sub-divisions of the remaining $n-(u-i)$ mergings of the class sub-chain $\mathcal{C}_{u+1,j}$. The goal is to minimize the total

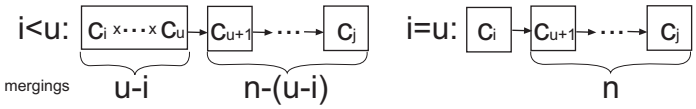

Fig. 6. Dynamic programming for a chain with arbitrary structure.

cost of the resulting solution. The level merging algorithm (LMA) can be found on Figure 7.

Input: chain $\mathcal{C}$ (of length d), integer m ($m < d$)
Output: chain $\mathcal{C}'$ of length $d-m$

- **Step 1: Initialization.** Initialize $V(i,j,n)$ for $n = 0$, $n > j-i$, and $n = j-i$ ($n \leq m$) according to Equation 3.
- **Step 2: Calculation.** Calculate all values $V(i,j,n)$ starting from $n = 1$ up to $n = m$ for $n > j-i$ using the recurrence Equation 4 and record the merging of minimum cost at each stage.
- **Step 3: Reconstructing the optimal solution.** Construct the chain $\mathcal{C}'$ by merging the classes of $\mathcal{C}'$ with respect to the optimal solution of minimum cost $V(1,d,m)$.

Fig. 7. Level merging algorithm (LMA) for general hierarchies.

We obtain that the running time of LMA is $O(d \cdot m^2)$ and the space complexity is $O(d^2 \cdot m)$. The subsequent theorem demonstrates the correctness of LMA.

Theorem 2: The level merging algorithm (LMA) finds an optimal solution of minimum cost for the problem of merging ($n > 1$) levels in a chain of length d ($d > n$).

Proof: The proof is by induction on the number of mergings n.

Basis ($n = 0$)**.** Obviously, LMA finds an optimal solution for $n = 0$ since it just returns the original input as no mergings need to be done.

Step ($n \rightarrow n+1$)**.** Assume that LMA finds an optimal solution for the number of mergings of at most n and let us prove that it finds an optimal solution for $n+1$ mergings as well.

Observe that LMA examines all possibilities for making z ($1 \leq z \leq n+1$) mergings in the leftmost class sub-chain. It must be the case that one of these mergings is a maximal merging in an optimal solution. Having done an optimal first merging, the other mergings in this optimal solution is independent optimal solution for the right class sub-chain under at most $n+1-z$ mergings. If the first merging is an empty merging (i.e. $z = 0$), then the above argument is applied to the right class sub-chain. According to the induction hypothesis, LMA finds an optimal solution in the right class sub-chain with at most $n+1-z \leq n$ mergings for the first non-empty merging (i.e. $z > 0$). Hence, LMA finds an optimal solution for at most $n+1$ mergings. ■

IV. Speed Optimization

In this section we study the problem of minimizing the number of levels in the policy hierarchy subject to the constraint on the maximum TCAM space. We utilize the space optimization algorithms from the previous section. The speed optimization algorithm is presented on Figure 8.

Input: policy P with TCAM space M of depth d, TCAM space A ($A > M$)
Output: policy P' equivalent to P with TCAM space of at most A

Perform *binary search* on the policy depth between 1 and d by applying either $SOAW$ or $SOAG$ on P depending on the structure of P's hierarchy and find the minimum depth l for which the TCAM space of the produced policy P' does not exceed A.
Output P'.

Fig. 8. Speed optimization algorithm.

Generally speaking, we need to find the optimal value l of the policy depth and then optimize the TCAM space of the transformed policy P' for depth l. The binary search is performed because this value is not known in advance. Once we have found the optimal depth, the space optimization algorithm guarantees minimization of the TCAM space. The running time of the speed optimization algorithm is $\log d$ times the running time of $SOAW$ or $SOAG$, that is at most $O(d^3)$ times the number of terminal classes in P, respectively. Remarkably, the running time of the speed optimization algorithm does not depend on A, which can be by orders of magnitude larger than d.

V. Conclusion

Hierarchical packet classification is a key operation needed in provisioning of many crucial network services. One of the major challenges in design of the next generation high-speed switches is to deliver wire-speed packet classification. TCAMs are the dominant industry standard used for multi-gigabit classifiers. However, as packet classification policies grow in depth and complexity, there arises a fundamental tradeoff between the TCAM space and the number of lookups for hierarchical policies.

In this paper we propose novel algorithms based on dynamic programming for solving two important problems concerned with hierarchical packet classification. The algorithms for the first problem minimize the TCAM space given a constraint on the policy depth while the algorithm for the second problem minimizes the policy depth subject to the constraint on the maximum TCAM space. Our algorithms do not require any modification to existing packet classification systems and can be easily deployed. As far as we know, this is the first work to study TCAM speed and space optimization for hierarchical packet classification.

References

[1] F. Baboescu, S. Singh, and G. Varghese, "Packet Classification for Core Routers: Is there an Alternative to CAMs?", *Proc. of IEEE INFOCOM*, 2003.

[2] F. Baboescu and G. Varghese, "Scalable Packet Classification", *Proc. of ACM SIGCOMM*, 2001.

[3] J. Bolaria and L. Gwennap, "A guide to search engines and networking memory", $http : //www.linleygroup.com/reports/memory\_guide.html$, 2004.

[4] H. J. Chao, "Next Generation Routers", *Proceedings of the IEEE*, Vol. 90, No. 9, pp. 1518-1558, 2002.

[5] P. Gupta, K. Etzel and J. Bolaria, "A Scalable and Cost-Optimized Search Subsystem for IPv4 and IPv6", *EE Times NetSeminar*, $http : //www.eetimes.com/netseminar.html$, 28 June 2004.

[6] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields", *Proc. of ACM SIGCOMM*, pp. 147-160 , 1999.

[7] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings", *Proc. of Hot Interconnects VII*, Aug. 1999.

[8] P. Gupta and N. McKeown, "Algorithms for Packet Classification", *IEEE Network*, pp. 24-32, March/April 2001.

[9] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary cams", *Proc. of ACM SIGCOMM*, pp. 193204, Aug. 2005.

[10] H. Liu, "Efficient mapping of range classifier into ternary-cam". *Proc. of the Hot Interconnects*, pp. 95100, 2002.

[11] J. van Lunteren and T. Engbersen, "Fast and scalable packet classification", *IEEE Journals on Selected Areas in Communications*, Vol. 21, No. 4, pp. 560571, 2003.

[12] M. H. Overmars and A. F. van der Stappen, "Range searching and point location among fat objects", *Journal of Algorithms*, Vol. 21, No. 3, pp. 629656, 1996.

[13] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Classification using Multidimensional Cutting", *Proc. of ACM SIGCOMM*, 2003.

[14] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and Scalable Layer Four Switching", *Proc. of ACM SIGCOMM*, 1998.

[15] D. E. Taylor and J. Turner, "Scalable Packet Classification using Distributed Crossproducting of Field Labels", *Technical Report WUCSE-2004-38 Washington Univ., St. Louis*, 2004.

[16] G. Varghese, "Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices", *Morgan Kaufmann Publishers Inc.*, 2004.

[17] T. Y. C. Woo, "A modular approach to packet classification: Algorithms and results", *Proc. of IEEE INFOCOM*, pp. 12131222, 2000.

[18] "Cypress Semiconductor Corp. Content addressable memory", $http : //www.cypress.com/$.

[19] "Integrated Device Technology, Inc. Content addressable memory", $http : //www.idt.com/$".

[20] "Netlogic Microsystems. Content addressable memory", $http : //www.netlogicmicro.com/$.