Fast and Scalable Packet Classification

Jan van Lunteren, *Member, IEEE*, and Ton Engbersen, *Senior Member, IEEE*

***Abstract*—Emerging Internet applications create the need for advanced packet classifiers. We propose a novel multifield classification scheme, called $\mathrm{P^2C}$, which exploits the strengths of state-of-the-art memory technologies to provide wire-speed classification performance for OC-192 and beyond, in combination with very high storage efficiency and the support of fast incremental updates. Key features of the new scheme are its ability to adapt to the complexity of a classification rule set, whereas the storage requirements and update dynamics can be tuned at the granularity of individual rules. This makes $\mathrm{P^2C}$ suitable for a broad spectrum of applications.**

***Index Terms*—Associative memories, communication system routing, communication systems, Internet, routing, search methods, table lookup, tree data structures, tree searching.**

I. Introduction

THE ROUTING and classification of Internet protocol (IP) packets is an essential element of the operation of the Internet. Evolution of the Internet into a high-speed, reliable global network infrastructure is expected to rely to a large extent on cost-efficient routing and classification of IP packets at high speeds. The peculiarities of IP operation require that for proper routing, at least the 32-bit [for IP version 4 (IPv4)] destination address of an incoming packet is looked up in the routing table and that precisely the entry in the routing table that matches the maximum number of bits from the start of the 32-bit field is used for determining the next-hop address. This is commonly referred to as longest-prefix match (LPM). Algorithms that implement this LPM search have been a topic of research for quite some time [1], [2]. The growing number of hosts on the Internet have caused a significant growth of the number of entries in the routing tables [3], and there is no sign that this growth will come to an end in the near future. On the contrary, current practices to connect hosts via multiple service providers to the Internet in order to enhance reliability is causing an exponential growth of the number of routing entries in backbone Internet routers. Approaches that minimize the use of memory for storing these routing tables have been one of the focus areas of recent research efforts [4], [5]. Initiatives such as “Integrated Services” (IntServ) and “Differentiated Services” (DiffServ) aim at introducing multiple qualities of service (QoS) in the Internet infrastructure, so that it will become possible to use the network for traffic types that have fundamentally different requirements (e.g., voice versus e-mail). These initiatives require that IP packets be classified into their appropriate class (e.g., high priority, low priority) at the edges of the network. The structure of the Internet, in which multiple Internet service providers (ISPs) interface with each other to create the larger Internet, results in edges being located at every interface point between ISPs. Furthermore, it is in the interest of every Internet user that unwanted traffic (e.g., “hackers”) be prevented from penetrating his or her computer or own network. This means that all packets need to be filtered at the entrance point to this network, commonly also referred to as the “edge.” The “firewall” technology used for this filtering is characterized by a fast dynamic change in the filter rules because for every outgoing connection the corresponding return path needs to be dynamically established, using IP source and destination addresses from a pool at the firewall’s disposal. Combining all these requirements, classification algorithms have to be optimized over three basic parameters: 1) classification performance; 2) storage requirements; and 3) update performance. Although there have been many recent contributions to classification algorithms, this contribution focuses on presenting an algorithm that optimizes all three requirements and allows tradeoffs amongst the three parameters to be made, depending on the specific application.

The remainder of this paper is organized as follows. Section II outlines the design objectives. Section III analyzes state-of-the-art classification schemes. Section IV introduces a novel scheme for packet classification, called parallel packet classification $(\mathrm{P^2C})$. The performance of $\mathrm{P^2C}$ is evaluated in Section V. Section VI presents experimental results, which are compared with the performance of existing schemes in Section VII. Section VIII discusses scalability issues and Section IX concludes the paper.

II. Design Objectives

A. Basic Requirements

The most important design requirements that new generations of packet classifiers have to meet are the following.

1) *Wire-speed classification performance*: The classifier must be able to process the highest possible packet arrival rate as determined by the link speed in combination with the minimum packet size (see Table I). Its worst case performance must be independent of the rule set and traffic characteristics [6].
2) *Storage requirements*: The amount of memory needed to store and search the classification-rule base should be small for cost reasons and to allow the use of faster but typically more expensive memory technologies for achieving higher performance (see Section II-B).
3) *Update performance*: Emerging applications such as QoS involve increasingly dynamic rule sets, in contrast to current firewall applications that have a more static nature

Manuscript received July 31, 2002; revised January 14, 2003. This work was presented in part at the IEEE Global Telecommunications Conference GLOBECOM’02, Taipei, Taiwan, R.O.C., November 2002.

The authors are with IBM Research, Zurich Research Laboratory, CH-8803 Rüschlikon, Switzerland (e-mail: jvl@zurich.ibm.com; apj@zurich.ibm.com).

Digital Object Identifier 10.1109/JSAC.2003.810527

TABLE I
WIRE-SPEED PACKET CLASSIFICATION RATES

link speed		maximum packet throughput	time available per packet
1 Gb/s	(1 GE)	3 Mpps	333 ns
2.5 Gb/s	(OC-48)	6 Mpps	167 ns
10 Gb/s	(OC-192)	26 Mpps	37 ns
40 Gb/s	(OC-768)	100 Mpps	10 ns

[7]. Next-generation classifiers must, therefore, support fast (incremental) updates of the rule base.

4) *Classification rules*: The classifier must support rule sets involving exact-, prefix-, and range-match conditions on various packet header fields, typically including source and destination addresses, as well as port numbers, protocol number, and type of service (an overview of the fields and match conditions used in actual classification rules is given in [8]). Furthermore, the classifier must be able to select the rule with the highest priority, in the case that multiple rules can match the same packet.
5) *Rule-set scalability*: The classifier must be able to scale the number of rules, the number of fields, and the field sizes that it supports in order to avoid being outdated by future Internet developments. One such development is the possible transition from IPv4 to IP version 6 (IPv6) addresses.

Besides these five requirements, further constraints can arise from implementation limitations such as maximum power dissipation and chip-area costs.

B. Overcoming the Memory-Bandwidth Bottleneck

The link speeds have increased much faster than the bandwidth of SDRAM and other memory technologies over the past few years. This poses two additional challenges to packet classifiers.

- The available *memory bandwidth* has to be used more efficiently: search and update operations must use fewer memory accesses and exploit memory system characteristics (e.g., burst modes, wide on-chip data buses) to increase bandwidth utilization.
- The available *memory capacity* has to be used more efficiently. Only in this way can faster technologies such as SRAM and on-chip DRAM, which are substantially more expensive and have significantly smaller storage capacity, be used to realize wire-speed performance for link speeds of 10 Gb/s and beyond.

C. Exploiting Rule-Set Characteristics

Analysis of several firewall rule sets has revealed some important characteristics [8].

- The number of unique match conditions for each field tends to be much smaller than the total number of rules because several rules share the same match conditions.
- The maximum number of rules that can match the same packet is very small, typically less than four.

Several emerging applications can also be expected to involve specific rule set properties. It therefore makes sense that packet classifiers exploit these characteristics to optimize their performance and improve their storage efficiency, but only to the extent that they do not lose their general applicability. This does not apply to the worst case classification performance, which should be rule-set independent as indicated above.

III. STATE-OF-THE-ART CLASSIFICATION SCHEMES

This section examines existing multifield classification schemes based on the criteria discussed above. Three categories of classification schemes are distinguished and will be discussed in the following sections. Actual performance figures will be presented and compared in Section VII.

A. Conversion Into Single-Field Search

A multifield search can be converted into a single-field search by combining the various fields into one search key. The main advantage of this concept is that well-known single-field search algorithms can be used, several of which support fast incremental updates. An important limitation, however, is that the efficiency strongly depends on how well the field conditions can be combined into match conditions on the composite search key that are supported by a single-field search algorithm. "Good" combinations that are commonly applied are the use of hashing for exact matches [9] and ternary content addressable memories (TCAMs) for exact and prefix matches [10]. Other combinations result in reduced efficiency, which can be acceptable if the number of "problematic" match conditions is small. For example, the tuple-space search scheme [11] supports prefix-match conditions using a separate hash table for each combination of prefix lengths (tuples), and TCAMs support arbitrary range-match operators by using multiple entries per range. However, this will only work well for rule sets with a relatively small number of tuples and arbitrary range-match conditions, respectively.

A major disadvantage of this concept is that the individual fields are no longer visible and that all fields must be processed using the same search algorithm. This precludes the exploitation of rule-set characteristics occurring at the granularity of individual fields. A second disadvantage is the large size of the composite search key, which renders an efficient search more difficult. For SRAM-based classifiers, it typically results in larger and less predictable latency because of the increased number of (interdependent) memory accesses needed to process the entire search key. For TCAM-based classifiers, it results in larger chip-area costs and increased power consumption owing to the wider associative cell array. Examples of other schemes applying this concept are described in [12]–[14].

B. Dependent Field Searches

Hierarchical tries, also called multilevel tries, are an example of a multifield search algorithm involving dependent field searches, i.e., the result of already searched fields affects the way subsequent fields are searched. The main advantage of this concept is that well-known and relatively simple tree-search algorithms can be used. The main disadvantage is that it only seems possible to achieve fast search times by replicating parts of the tree structure (e.g., set-pruning tries [15]), by

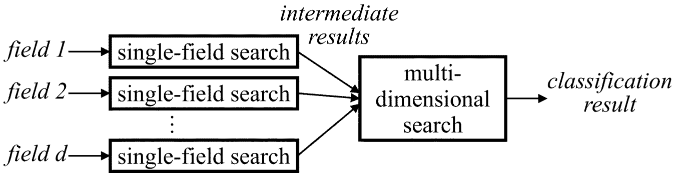

Fig. 1. Independent field searches.

embedding additional link information in the data structure (e.g., grid-of-tries [16]), and/or by careful preprocessing (e.g., HiCuts [17]), all of which result in increased storage requirements and a more complex and slow update operation. A second disadvantage is that the numerous memory accesses necessary to process all the fields are interdependent, which can result in a large and less predictable latency. Examples of other schemes applying dependent field searches can be found in [18]–[20].

C. Independent Field Searches

Fig. 1 shows the concept of a classifier in which the various fields are searched independently in a first phase, and the classification result is determined by a multidimensional search on the intermediate search results in a second phase. The rationale behind this concept is that the multidimensional search shown in Fig. 1 can be simplified and its performance improved by proper *encoding* of the intermediate search results, as compared with the multidimensional searches discussed above. The independent field searches can, in this respect, be regarded as a "preprocessing" step. Examples of schemes applying this concept are the bitmap-intersection scheme [21], the recursive flow classification (RFC) scheme [8] (here, the boundaries between the single-field and multifield searches as shown in Fig. 1 are less obvious because all searches are implemented as several stages of parallel table lookups), and the cross-production method [16]. An important advantage of the concept shown in Fig. 1 is that the field searches can be performed more efficiently because of the relatively small search keys and by exploiting specific field characteristics. Another advantage is that the memory accesses related to different field searches are independent and can be performed in parallel or in a pipelined fashion, which reduces the latency and allows a better utilization of the available memory bandwidth.

The overall performance, however, strongly depends on the encoding of the intermediate search results and the corresponding impact on the multidimensional search. The bit-map intersection and RFC schemes can be regarded as extremes in this respect: the bit-map intersection scheme encodes *all* rule information in the intermediate result vectors, whereas the RFC scheme encodes *no* rule information whatsoever. As a consequence, the bit-map intersection scheme involves only a simple "multidimensional search," basically consisting of a logical AND operation, whereas the RFC scheme involves a complex multidimensional search that needs to retrieve all rule information from its data structure. Both encoding styles, however, share the same important drawback that the storage requirements grow exponentially with the number of rules and that fast incremental updates are not supported. Other schemes applying the concept of independent field searches, the cross-production method and a modified version of the bitmap-intersection scheme presented in [22] have the same disadvantage.

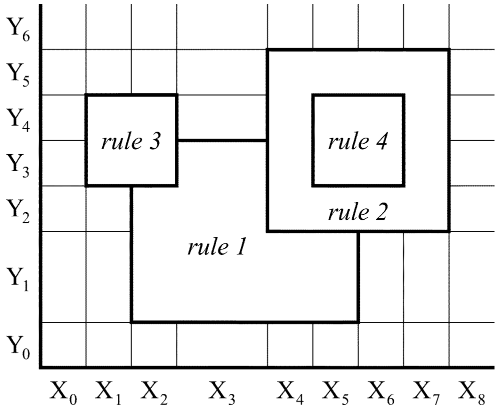

Fig. 2. Example of a two-dimensional (2-D) classifier.

IV. Parallel Packet Classification (P^2C)

The P^2C scheme employs the concept of independent field searches as shown in Fig. 1. The word *parallel* emphasizes the parallelism available between the independent field searches. The key element of the P^2C scheme is a novel encoding of the intermediate search results, which overcomes the disadvantages discussed in the previous section by significantly reducing the storage requirements and minimizing the dependencies within the search structures, thus enabling fast incremental updates. The P^2C encoding involves several styles that can be applied simultaneously and allow several performance aspects to be balanced at the granularity of individual rules. The P^2C scheme can be used with a variety of search algorithms and memory technologies. This paper will focus on a configuration in which the fields are searched using the BART scheme [5] in SRAM, and the multidimensional search is implemented using a TCAM. A configuration entirely based on SRAM technology is described in [23].

A. Primitive-Range Hierarchy

The P^2C encoding is based on the concept of so-called *primitive ranges*, which are intervals of field values for which the match conditions specified for a given field apply. The intermediate result vectors are encoded such that each primitive range can be identified by one match condition. This match condition is typically a ternary match; however, it is also possible to encode for prefix- and exact-match conditions (see [23]). Both the intermediate search results and the corresponding match conditions are generated by organizing the primitive ranges into a hierarchical structure, denoted as primitive-range hierarchy. This concept will now be explained by illustrating how it can be used to generate intermediate result vectors similar to those of the bitmap-intersection scheme (discussed in Section III-C) for the two-dimensional (2-D) classifier shown in Fig. 2, which is a variation on an example presented in [21]. Four rules are represented as rectangles in a 2-D space, where the ranges covered by the rules in each dimension result from the match conditions on the corresponding field. Higher priority rules are drawn "on

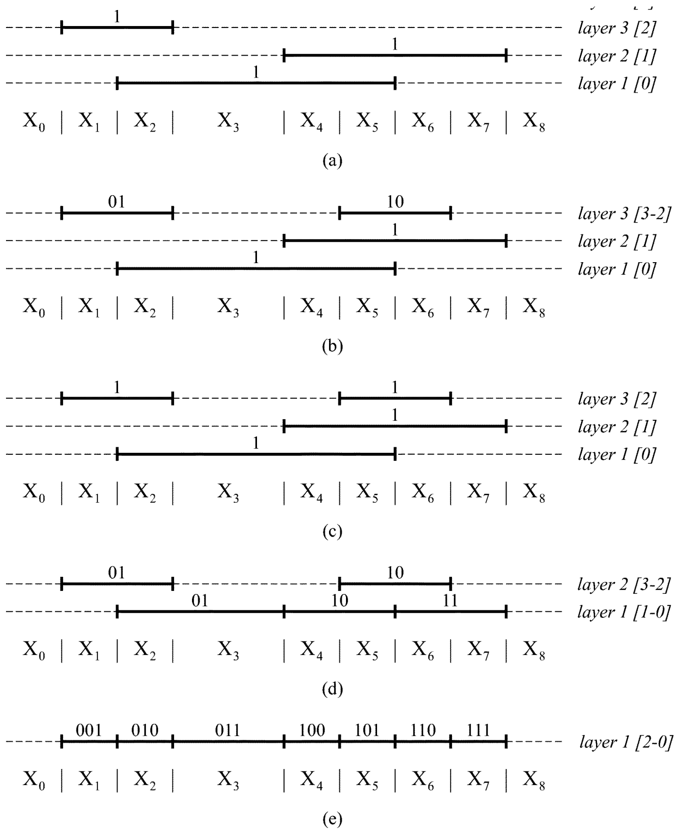

Fig. 3. Primitive-range hierarchies. (a) Encoding according to bitmap-intersection scheme. (b) $\mathrm{P}^2\mathrm{C}$-encoding style I: Independent identifiers. (c) $\mathrm{P}^2\mathrm{C}$-encoding style II: Dependent identifiers. (d) $\mathrm{P}^2\mathrm{C}$-encoding style III: Reducing the number of layers. (e) Encoding according to RFC scheme.

top" of lower priority rules. The sets of nonoverlapping intervals X_0 to X_8 and Y_0 to Y_6 are obtained by projecting the rectangle boundaries onto the corresponding axes.

Fig. 3(a) shows four primitive ranges, which equal the ranges covered in the X-dimension by the four rules in Fig. 2. The primitive ranges are organized in a hierarchical structure, comprised of four layers that each correspond to 1 bit position in the result vector (this bit position is shown between brackets to the right of each layer). Each primitive range is assigned an identifier equal to 1 (binary). The remaining part of each layer, which is not part of a primitive range, is assigned a zero identifier (not shown in Fig. 3). The intermediate result vectors are now obtained by determining the set of primitive ranges in which each interval is located, and then substituting the primitive range identifiers at the bit positions associated with the corresponding layers. For example, the result vector corresponding to interval X_2 equals 0101 because X_2 is part of the primitive range at layer 1 (resulting in the set bit at bit position 0—the rightmost bit position), as well as of the primitive range at layer 3 (the set bit at bit position 2). Table II (first row) lists all result vectors derived in this way (note that the zero result vectors corresponding to intervals X_0 and X_8 are omitted).

A set bit in the result vector indicates that the interval is part of a primitive range and that the corresponding rule applies for that interval. This can be tested using a ternary-match condition. For example, the ternary-match condition xx1x for the primitive range related to rule 2 [also at layer 2 in Fig. 3(a)] will only match the result vectors corresponding to the intervals that are part of this primitive range: X_4, X_5, X_6, and X_7. Table III lists the ternary-match conditions for all rules. Note that with the actual bit-map intersection scheme, the bits are ordered according to rule priorities in the result vectors. Although priority information could be encoded in the primitive-range hierarchy as well, the $\mathrm{P}^2\mathrm{C}$ scheme typically stores it in the multidimensional search structure.

TABLE II
INTERMEDIATE RESULT VECTORS FOR THE PRIMITIVE-RANGE HIERARCHIES IN FIG. 3

	intervals						
enc.	X_1	X_2	X_3	X_4	X_5	X_6	X_7
(a)	0100	0101	0001	0011	1011	1010	0010
(b)	0100	0101	0001	0011	1011	1010	0010
(c)	100	101	001	011	111	110	010
(d)	0100	0101	0001	0010	1010	1011	0011
(e)	001	010	011	100	101	110	111

TABLE III
TERNARY-MATCH CONDITIONS FOR THE PRIMITIVE-RANGE HIERARCHIES IN FIG. 3

enc.	rule 1	rule 2	rule 3	rule 4
(a)	xxx1	xx1x	x1xx	1xxx
(b)	xxx1	xx1x	01xx	10xx
(c)	xx1	x1x	10x	11x
(d)	xx01, xx10	xx10, xx11	01xx	10xx
(e)	010 .. 101	100 .. 111	010, 011	101, 110

B. $\mathrm{P}^2\mathrm{C}$-*Encoding Styles*

The set $\mathcal{P}$ denotes all primitive ranges in a primitive-range hierarchy, whereas $L_i \subseteq \mathcal{P}$ represents the primitive ranges located at layer i within the hierarchy.

The first $\mathrm{P}^2\mathrm{C}$-encoding style involves primitive ranges that equal the *unique* ranges corresponding to the match conditions specified by the classification rules for the dimension being encoded. For the X-dimension of the classifier in Fig. 2 this results in: $\mathcal{P} = \{r_1, r_2, r_3, r_4\}$, with $r_1 = [X_2, X_5]$, $r_2 = [X_4, X_7]$, $r_3 = [X_1, X_2]$, and $r_4 = [X_5, X_6]$. Note that no primitive range will be created for a "wildcard" on an entire field. For rules involving such a condition, a ternary-match condition is created that is entirely "don't care" $(\mathrm{xx}\ldots\mathrm{x})$.

This encoding style constructs a primitive-range hierarchy by distributing the primitive ranges over the various layers such that: 1) the primitive ranges at the same layer are nonoverlapping and 2) the total number of layers is minimized. As a consequence, the number of layers in the hierarchy will equal the maximum number of match conditions that *all* overlap each other. Fig. 3(b) shows a primitive-range hierarchy obtained by applying the first encoding style to the previous example. The hierarchy is described by $L_1 = \{r_1\}$, $L_2 = \{r_2\}$, and $L_3 = \{r_3, r_4\}$. The (minimum) number of three layers is determined by the three primitive ranges r_1, r_2, and r_4 that all overlap each other and, therefore, have to be assigned to different layers.

To allow all primitive ranges at the same layer to be assigned unique nonzero identifiers, a total of $\lceil \log(|L_i| + 1) \rceil$ bits of the intermediate result vector will be associated with each layer i, where $|L_i|$ represents the number of primitive ranges at that layer. This results in 1 bit being associated with each of layers 1 and 2, and 2 bits with layer 3 in Fig. 3(b), which also shows the identifiers assigned to the four primitive ranges. These primitive range identifiers are used to derive the result vectors and ternary-match conditions listed in Tables II and III (second row), as described in Section IV-A. The size of the result vectors and match conditions equals the total number of bits associated with all layers in the hierarchy, which is four in this example.

The first encoding style yields shorter intermediate result vectors and ternary-match conditions for rule sets with many nonoverlapping ranges, because the number of layers will decrease linearly with the number of primitive ranges "grouped" at each layer, whereas the number of result-vector bits per layer will only increase logarithmically. In the case shown, the result-vector size remained four bits, owing to the small number of rules in this example. However, as can be seen, there would be room for a fifth (nonoverlapping) primitive range at layer 3, which could be assigned identifier 11 without increasing the result-vector size. A fifth rule would always require an additional layer and result-vector bit in Fig. 3(a).

The second $\mathrm{P^2C}$-encoding style differs from the first one only in the way that the primitive-range identifiers are assigned; the primitive-range hierarchy is constructed in the same way. This encoding style further reduces the size of the intermediate result vector by exploiting relations between primitive ranges: Two primitive ranges at the same layer can be assigned a common identifier if both ranges are subsets of two disjoint primitive ranges at other layers and the (nonidentical) identifiers of the latter ranges are inserted into the corresponding ternary-match conditions in addition to the common identifier. This is illustrated in Fig. 3(c), in which the primitive ranges r_3 and r_4 at layer 3 are assigned a common identifier 1. Primitive range r_4 is a subset of primitive range r_2 at layer 2 with identifier 1, whereas range r_3 is a subset of the "empty" portion of layer 2, which will be treated as a primitive range with identifier 0. Table II (third row) lists the result vectors corresponding to Fig. 3(c), which are determined in the same way as with the first encoding style. The ternary-match condition for primitive range r_4 now contains both the common identifier 1 and the identifier 1 of primitive range r_2 (at the bit positions associated with the respective layers) resulting in a ternary vector 11x. It can be verified in Table II that this condition will only match the result vectors corresponding to intervals X_5 and X_6. In a similar way, a ternary-match condition 10x can be derived for primitive range r_3, which will only match the result vectors corresponding to intervals X_1 and X_2.

The third $\mathrm{P^2C}$-encoding style reduces the number of layers by converting overlapping primitive ranges at different layers into a larger number of nonoverlapping primitive ranges at the same layer. For example, the two overlapping primitive ranges $r_1 = [X_2, X_5]$ and $r_2 = [X_4, X_7]$ can be converted into three nonoverlapping ranges $r_a = [X_2, X_3]$, $r_b = [X_4, X_5]$, and $r_c = [X_6, X_7]$. These can be used to create a primitive-range hierarchy consisting of only two layers, $L_1 = \{r_a, r_b, r_c\}$ and $L_2 = \{r_3, r_4\}$, which is shown in Fig. 3(d) with the corresponding result vectors listed in Table II (fourth row). One aspect of this encoding style is that the two value ranges for which the match-conditions specified by rules 1 and 2 apply are now each comprised of two primitive ranges r_a and r_b, and r_b and r_c, respectively. As a result, two ternary-match conditions are needed to identify each of these value ranges from the intermediate result vectors, as shown in Table III. Repeated application of this approach will ultimately lead to a primitive-range hierarchy consisting of a single layer as shown in Fig. 3(e), which corresponds to the RFC-scheme type of encoding and clearly shows the position of the $\mathrm{P^2C}$-encoding styles between the extremes of the bitmap-intersection and RFC schemes.

$\mathrm{P^2C}$-encoding style I creates the smallest number of dependencies within the data structures and will, therefore, provide the highest update performance of the three encoding styles. $\mathrm{P^2C}$-encoding styles II and III, however, can achieve smaller intermediate result vectors and, thus, have smaller storage requirements by introducing additional dependencies at the cost of lower update performance. A key feature of the $\mathrm{P^2C}$ scheme is that all three encoding styles can be applied simultaneously for the same rule set and can be selected separately for each rule, which makes $\mathrm{P^2C}$ suitable for a broad variety of applications and implementation environments. The following use of the encoding styles is currently envisioned. $\mathrm{P^2C}$-encoding styles I and II are regarded as the standard encoding styles applied during normal operation, and can be used to balance update dynamics and storage efficiency at the granularity of individual rules. An example is a firewall involving both "long-lived" static rules, as well as dynamic rules that only exist during the lifetime of a connection. The former type of rules would then be encoded according to encoding style II, whereas the latter type of rules would be encoded using encoding style I.

Encoding style III is currently only used in exceptional cases to enforce the data structures to meet certain implementation limits, in order to facilitate implementation and to more efficiently support larger classification rules sets for which the storage capacity limits are being approached. For example, this encoding style can be used to limit the maximum number of primitive ranges that can match a given field value, in order to simplify the on-the-fly construction of the intermediate result vector (see Section IV-C). Encoding style III can also balance the TCAM width and depth requirements, in order to stay within the physical limits imposed by the TCAM implementation. It can do so by reducing the size of the intermediate result vectors and, thus reducing the TCAM width requirements at the cost of requiring more ternary-match conditions and, thus increasing the number of TCAM entries (see Section IV-D).

C. *Independent Field Searches*

The bitmap-intersection and RFC schemes search the nonoverlapping *intervals* formed by the boundaries resulting from the match conditions (e.g., X_0 to X_8, and Y_0 to Y_6 in Fig. 2) in which the field values are located, and output the corresponding intermediate result vectors, which are *precomputed* and stored separately for each interval. Significant drawback of this approach is that information related to a single rule can be distributed over many result vectors, resulting in poor update performance.

The $\mathrm{P^2C}$ scheme overcomes this problem by having the field searches determine the actual *primitive ranges* in which the field values are located, followed by an *on-the-fly* construction of

TABLE IV
INTERMEDIATE RESULT VECTOR CONSTRUCTION

on-the-fly		precomputed	
range	result	range	result
default	0000	X_0	0000
$[X_2, X_5]$	$000\underline{1}$	X_1	0100
$[X_4, X_7]$	$00\underline{1}0$	X_2	0101
$[X_1, X_2]$	$\underline{01}00$	X_3	0001
$[X_5, X_6]$	$\underline{10}00$	X_4	0011
		X_5	1011
		X_6	1010
		X_7	0010
		X_8	0000

the intermediate result vectors based on the matching primitive-range identifiers and associated bit-position information. This improves both storage efficiency and update performance, because the information related to each primitive range is typically stored only once.

The on-the-fly construction can be implemented in various ways. One possibility would be to store the primitive-range identifiers at the corresponding bit positions in vectors of the same size as the intermediate result vector, in which the remaining bits are zero. The intermediate result vector is then constructed by performing a logical OR operation on all vectors corresponding to the matching primitive ranges found. This is illustrated in Table IV for the result vectors obtained by applying the first $\mathrm{P^2C}$-encoding style on the X-dimension of Fig. 2 (Table II, second row). Note that the primitive-range identifiers are underlined in each vector. For example, if the X-field value were located within interval X_2, then two primitive ranges will be found to match: $[X_1, X_2]$ and $[X_2, X_5]$. The intermediate result vector is then constructed on-the-fly by determining the logical OR product of the corresponding vectors 0100 and 0001, resulting in 0101, which is identical to the result vector listed in Table II for interval X_2. Table IV also shows the intervals and corresponding results for the situation that precomputed result vectors would have been used, in which case a total of nine intervals would have been searched with eight different results, versus only four ranges with four different results in the case of on-the-fly construction.

The on-the-fly construction of the result vectors is slightly more complex than the concept of precomputed result vectors, because it requires the field-search algorithm to find multiple primitive ranges instead of only one interval. However, several algorithms exist that have this capability or can achieve it after small modifications. One such algorithm is the BART scheme [5], which will be discussed below.

BART is a scheme for exact- and prefix-match searches that achieves high search performance, suitable for OC-192 link speeds and beyond, by efficiently processing the search key in segments of about 8 bits, requiring in the worst case only four memory accesses to search a 32-bit IP address and two accesses for a 16-bit port number, all of which can be performed in a pipelined fashion. BART applies a novel compression technique that provides high storage efficiency in combination with fast incremental updates. The BART compression is based on a special hash function for exact- and prefix-match conditions. The hash index is formed by a subset of bits from a search key segment that are selected such that the maximum number of collisions for any hash index is limited by a configurable bound P. The value of P is based on the memory access granularity to ensure that each hash table entry, containing at most P match conditions, can be read using a single memory access. Collisions for a given hash index are then resolved by at most P parallel comparisons. For a detailed description of the BART scheme, including the incremental update function, see [5].

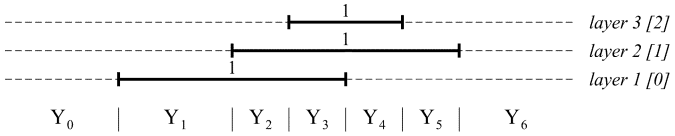

Fig. 4. Primitive-range hierarchy in Y-dimension.

The original version of BART used for routing table lookups as described in [5] needs a small modification to facilitate the on-the-fly construction of the intermediate result vectors. Instead of determining the *longest-matching* prefix condition in each search step by (at most P) parallel comparisons, and taking the one found by the last search step as the overall search result, an intermediate result vector has to be constructed based on *all* matching conditions in *all* search steps. If the on-the-fly construction is based on a logical OR operation as described above, than this modification even simplifies the BART implementation: the multiplexer function in the original version used to select the longest prefix condition from all matching conditions is replaced by a logical OR function that simply "combines" the results of all matching conditions.

For typical rule sets, most primitive ranges will relate to exact- and prefix-match conditions, which are directly supported by BART. Arbitrary range-match conditions are handled by converting them into prefix-match conditions as described, for example, in [15].

D. TCAM-Based Multidimensional Search

In order to implement the multidimensional search using a TCAM, it must be converted into a single-field search. The problems associated with this approach, described in Section III-A, are overcome by the preprocessing step comprised of the independent field searches, which results in a smaller composite search key, and, consequently, a smaller TCAM and less power consumption. The TCAM contents are obtained by concatenating all combinations of ternary-match conditions for all fields that relate to the same rule, and storing these in the TCAM ordered according to the rule priorities. The input for the TCAM search consists of the concatenated result vectors of the independent field searches. The rule corresponding to the matching TCAM entry with the highest priority becomes the classification result.

Fig. 4 shows the primitive-range hierarchy for the Y-dimension of Fig. 2 obtained by applying $\mathrm{P^2C}$-encoding style I. This hierarchy involves the following result vectors: 001 (Y_1), 011 (Y_2), 111 (Y_3), 110 (Y_4), 010 (Y_5), and the following ternary-match conditions: xx1 (rule 1), x1x (rule 2), and 1xx (rules 3 and 4). Concatenating the match conditions for the X-dimension

(Table III, second row) with those given above for the Y-dimension provides the following TCAM entries:

rule 1:	xxx1xx1
rule 2:	xx1xx1x
rule 3:	01xx1xx
rule 4:	10xx1xx

If the corresponding field values were located, for example, in intervals X_4 and Y_2, then concatenation of the result vectors would produce a vector 0011011. The TCAM entries for rules 1 and 2 match this vector, indicating that both rules apply as can be verified in Fig. 2. The rule priority then determines the classification result.

Note that $\mathrm{P^2C}$-encoding styles I and II result in exactly one TCAM entry per rule, whereas encoding style III can result in multiple entries. The latter encoding style, however, allows the size of a TCAM entry (the TCAM width) to be balanced with the total number of TCAM entries (the TCAM depth).

E. Incremental Updates

Insertion and removal of classification rules from the rule base requires modification of the primitive-range hierarchies, followed by corresponding updates of the field-search structures (based on BART) and the TCAM contents. Algorithms for fast incremental updates on the latter two search structures are described in [5] and [24], respectively. The remainder of this section will address the update of the primitive-range hierarchies.

With $\mathrm{P^2C}$-encoding style I, the insertion of a primitive range into a hierarchy consists of 1) checking whether this primitive range already exists, or, if that is not the case; 2) finding a layer with primitive ranges that do not overlap; and 3) assigning the new primitive range an identifier that is unique for that layer. The first two steps can be performed efficiently by organizing the primitive ranges of each layer in a separate search structure (based on BART) that is used to determine whether a range already exists or whether any overlap occurs. The third step is realized by maintaining a list with unused identifiers for each layer, as well as the way these identifier bits are mapped onto the TCAM-bit positions (see Fig. 3). If this list becomes empty, implying that the maximum number of primitive ranges has been reached for the given identifier size, than the identifier size is increased to create new identifiers by allocating an additional unused TCAM bit position to this layer. This is relatively simple because identifier bits do not have to be mapped onto consecutive TCAM bits. For this purpose, a list with "free" (unallocated) TCAM bit positions is maintained.

Removing a primitive range from a hierarchy consists of the following steps: 1) locating the primitive range; 2) checking whether there are still rules associated with it, and, if that is not the case; 3) removing it; and 4) returning its identifier to the "free" identifier list. The second step is necessary because a primitive range can be associated with multiple rules. This step can be implemented by registering the number of these rules in the data structure for each primitive range.

Note that primitive ranges can be organized in various ways in a hierarchy and, therefore, the *order* in which primitive ranges are inserted and deleted will influence the construction of the hierarchy and the allocation of the various resources (e.g., the TCAM bit assignment). However, this does not appear to be a problem because of the limited amount of overlapping occurring in actual rule sets (see Section II-C), which results in only a small number of layers (see Section V). Simulations have revealed that the simple strategy of locating all primitive ranges related to exact-match conditions at one layer is sufficient to limit the fluctuation in the intermediate result-vector sizes (see Section VI). If future rule sets would emerge involving increased overlapping, then this issue can be resolved by applying efficient resource management schemes similar to, for example, buffer management algorithms intended to avoid memory fragmentation.

Application of $\mathrm{P^2C}$-encoding styles II and III involve a more complex modification of the primitive-range hierarchies, because dependencies between ranges have to be found and resolved as described in Section IV-B. For these encoding styles, the same search structures are used as described above, extended with additional information related to the dependencies between primitive ranges.

V. Performance Evaluation

A. Classification Performance

The $\mathrm{P^2C}$ scheme searches all fields in parallel, resulting in a total classification time determined only by the maximum field-search time and the TCAM search time, which typically equals one clock cycle. With the BART scheme, the maximum search time will occur for the largest field, thus resulting in a worst case classification time that depends only on the maximum field size and that is independent of the number of fields and the sum of all field sizes. BART processes a search key in segments of s bits, with a typical value of $s = 8$ bits. This results in an overall classification-time complexity equal to $O(W's^{-1} + 1)$, where W' is the largest field size.

The 32-bit IPv4 source and destination addresses are typically the largest fields in current rule sets, and can be searched by BART using at most four SRAM accesses [5]. Consequently, the maximum *latency* for classifying a packet equals the time it takes to perform four SRAM accesses and one TCAM access plus some additional time used by the search logic. Because the BART scheme involves only a few memory accesses and does not use backtracking, it is very suitable for a pipelined implementation. If this is applied to all searches, then the classification *rate* will depend mainly on the longest cycle time of either the SRAM or the TCAM. This allows wire-speed classification for OC-192 and OC-768 links, which require maximum cycle times of approximately 37 and 10 ns, respectively, and hence are feasible with state-of-the-art memory technologies.

B. Storage Requirements

The low storage requirements of BART in combination with the on-the-fly construction of the intermediate result vectors make the TCAM the predominant factor determining the overall storage complexity. Worst case figures will now be derived for $\mathrm{P^2C}$-encoding style I, which has the largest storage requirements of all three encoding styles. First, an upper bound will be derived for the intermediate result-vector size I_i (in bits) in the ith dimension if the match conditions in this dimension are characterized by the following two parameters:

- the number of unique match conditions: q_i;
- the maximum number of match conditions that *all* overlap each other: k_i.

For example, for the classifier shown in Fig. 2, these parameters have the following values: $q_x = 4$, $k_x = 3$, $q_y = 3$, $k_y = 3$.

Both parameters directly impact the organization of the primitive-range hierarchy: The number of primitive ranges equals the number of unique match conditions q_i, and the number of layers equals the maximum number of match conditions that all overlap each other k_i. The latter applies, because $\mathrm{P}^2\mathrm{C}$-encoding style I groups as many nonoverlapping primitive ranges at the same layer as possible. This is verified in Figs. 3 and 4.

The result-vector size equals the total number of bits associated with all the layers in the primitive-range hierarchy. If there are n primitive ranges at a given layer, then a total of $\lceil \log(n+1) \rceil$ bits will be associated with that layer, which is necessary to assign a unique nonzero identifier to each primitive range. The largest result-vector size will be obtained when all ranges are distributed equally over all layers, resulting in a maximum intermediate result-vector size equal to

$$I_{i,\max} = k_i \left\lceil \log\left(\frac{q_i}{k_i} + 1\right) \right\rceil \text{ bits.} \tag{1}$$

Both q_i and k_i can in theory vary between one and the total number of rules, N: $1 \le q_i \le N$ and $1 \le k_i \le N$. However, analysis of actual rule sets has revealed that in practice only the value of q_i varies within this entire interval, whereas k_i is typically very small (see Section II-C). The result-vector size will be minimal if there are no overlapping primitive ranges; $k_i = 1$. This results in the following absolute lower bound:

$$I_{i,\min} = \lceil \log(q_i + 1) \rceil \text{ bits.} \tag{2}$$

$\mathrm{P}^2\mathrm{C}$-encoding style I implies that the number of TCAM entries equals the number of rules, whereas the TCAM-entry size equals the sum of all intermediate result-vector sizes. This results in worst case TCAM storage requirements equal to

$$N \sum_{i=1}^{d} I_{i,\max} = N \sum_{i=1}^{d} k_i \left\lceil \log\left(\frac{q_i}{k_i} + 1\right) \right\rceil \tag{3}$$

with N being the number of rules in the rule base, and d the number of fields.

C. Update Performance

Updating the rule base requires modification of the primitive-range hierarchies, the field-search structures, and the TCAM contents as described in Section IV-E. Because the latter data structures can be updated in parallel and each structure can sustain (incremental) update rates of the order of millions per second (see [5] and [24]), the overall update performance will depend mainly on the modification of the primitive-range hierarchies. Section IV-B indicated that the highest update performance is achieved with $\mathrm{P}^2\mathrm{C}$-encoding style I, and that the other two encoding styles are intended for more static rules. This section will, therefore, focus on the first encoding style.

Section IV-E described how the primitive-range hierarchies can be modified efficiently using a (BART) search structure for each layer. The highest update performance will be achieved if these searches are performed in parallel, and each search is performed in a pipelined fashion. However, this is generally not necessary because the update performance requirements are usually a few orders of magnitude smaller than the classification performance, and the number of layers, and consequently the number of searches, is typically very small. Therefore, sufficiently high update rates can be achieved by storing the search structures for all primitive-range hierarchies in one memory bank and searching these sequentially.

For example, if for a typical rule set based on the popular "5-tuple," the primitive-range hierarchies related to the addresses and port numbers consisted of four layers, and the hierarchy for the protocol number consisted of a single layer, then a maximum of $4^*4 + 1 = 17$ BART searches would be needed to update all these hierarchies. Because BART needs one memory access to search the protocol number and a maximum of two and four accesses to search the 16-bit port numbers and the 32-bit addresses, respectively, the 17 BART searches will require at most 49 memory accesses. If all search structures are stored in one SRAM with a cycle time of 10 ns, then the maximum time it takes to update all primitive-range hierarchies equals 490 ns, which corresponds to a worst case update rate of about 2 M updates/s. As indicated above, pipelining techniques and faster memories can be used to achieve much higher update rates. Because an update rate of 2 M updates/s already exceeds the requirements for most applications, the hierarchies can also be stored in cheaper and slower SDRAM. For example, an SDRAM having a cycle time of 50 ns achieves an update rate of approximately 400 K updates/s using the same calculation.

VI. Experimental Results

The $\mathrm{P}^2\mathrm{C}$ scheme has been validated in two ways: a software-based simulation model and a hardware prototype implemented in a field programmable gate array (FPGA). The latter prototype was realized using a commercially available platform for rapid prototyping [25] based on a PCI card, with the update function implemented in software and executed on the host computer. The small storage requirements of $\mathrm{P}^2\mathrm{C}$ allowed several memories to be implemented in the FPGA, in addition to two external SRAMs available on the PCI card. Several simulations and experiments have been performed with three commercial firewall rule sets, the largest of which contained about 1700 rules. These rule sets are proprietary, and any details have to be omitted for privacy reasons (this problem is common to most papers presenting experimental results for firewall rule sets). The match conditions related to each field were inserted into the data structures in various orders: at random, by increasing and decreasing rule priority, and by increasing and decreasing size of the intervals covered by the match conditions.

Table V shows the numbers of layers observed for the primitive-range hierarchies for the source (SA) and destination (DA) addresses and the source (SP) and destination (DP) ports, all obtained by applying $\mathrm{P}^2\mathrm{C}$-encoding style I. These numbers confirm the assumption that the hierarchies are typically comprised

TABLE V
PRIMITIVE-RANGE HIERARCHIES

rule set	rules	number of layers				result vectors	
		SA	DA	SP	DP	$\sum I_i$	$\sum I_{i,\min}$
set 1	37	3	3	1	1	22-24 bits	18 bits
set 2	138	2	2	2	2	24-27 bits	22 bits
set 3	1733	2	2	2	1	23-25 bits	22 bits

TABLE VI
P^{2}C STORAGE REQUIREMENTS

rule set	rules	SRAM	TCAM
set 1	37	0.25 kB	0.10 kB
set 2	138	0.54 kB	0.45 kB
set 3	1733	2.0 kB	5.1 kB

of only a few layers. Table V also shows the observed value ranges for the accumulated intermediate result-vector sizes $\sum I_i$ which determines the (minimum required) TCAM width. These value ranges cover the results obtained for the various rule insertion orders indicated above (see also Section IV-E). The fluctuation appears to be small. An absolute lower bound of the accumulated result-vector sizes $\sum I_{i,\min}$ was calculated using (2) after determining the number of unique match conditions in each dimension q_i. Table V shows that the lower bound of the actual value range is not far away from the absolute lower bound, especially for the largest two rule sets. This implies that for these rule sets, the storage efficiency achieved by P^{2}C-encoding style I is close to optimal, and that not much can be gained by applying the other two encoding styles. This has been confirmed by simulations with these encoding styles, which provided almost the same results. The actual SRAM and TCAM storage requirements are listed in Table VI. The TCAM requirements appear to scale almost linearly with the number of rules, whereas the SRAM requirements appear to scale even better.

The FPGA-based hardware prototype runs at a clock speed of 33 MHz, corresponding to a clock cycle of 30 ns. The prototype achieves a classification *latency* equal to five clock cycles, four of which are used for the longest field search and one for the (emulated) TCAM search, which is consistent with Section V-A. By applying pipelining, the prototype is able to achieve a classification rate of one packet per clock cycle, corresponding to 33 Mpps, which exceeds the required rate for OC-192 (see Table I).

The worst case number of BART searches needed for updating all primitive-range hierarchies that occurred during the simulations equaled 9 and involved a total of 30 memory accesses, which would enable high update performance even if the primitive-range hierarchies were stored in a single SDRAM, as discussed in Section V-C. However, it was not possible to measure a representative update rate for the hardware prototype owing to the very slow communication between the PCI card and the update function executed on the host computer.

VII. COMPARISON

Table VII compares the P^{2}C scheme with state-of-the-art classification schemes. The schemes are organized according to the three categories defined in Section III. The performance is shown for rule sets involving exact-, prefix-, and arbitrary range-match conditions (unless indicated otherwise) on d packet fields that have an average size of W bits, whereas the largest field has a size of W' bits. The maximum number of match conditions that all overlap each other for any field is represented by k. Several performance figures in Table VII are based on [15] and [26].

Table VII includes two columns related to classification performance. The first one lists the worst case time complexity related specifically to the classification latency and the second indicates whether the methods can apply pipelining to achieve a classification rate that depends mainly on the memory cycle time. Methods are only considered suitable for a pipelined implementation if they meet the following three criteria: 1) they do not apply backtracking techniques; 2) the number of memory banks is reasonably small and independent of the rule-set characteristics; and 3) updates do not require data to be copied or transferred between different memory banks (e.g., to rebalance a tree structure), because this can "remove" a significant amount of bandwidth from the search process. The next two columns in Table VII correspond to the storage requirements. The first of these columns shows the worst case storage complexity and the second indicates whether the methods exploit the occurrence of identical field conditions to optimize their storage requirements. The last column in Table VII indicates whether the methods support fast incremental updates.

Only three schemes in Table VII are able to apply pipelining techniques efficiently (see the criteria defined above) to realize a classification rate that depends only on the memory cycle time. Only these schemes will be able to achieve truly wire-speed performance for OC-192 and OC-768 link speeds by using sufficiently fast SRAMs and/or TCAMs. Of those schemes, only the TCAM-based classifier and the P^{2}C scheme support fast incremental updates. The main difference between these two schemes are their storage requirements. Although the TCAM provides worst case storage requirements that grow linearly with the number of rules if these involve exact- and prefix-match conditions only, it can require significantly more storage than the P^{2}C scheme due to 1) the large composite search key comprised of dW bits; 2) its inability to optimize based on identical field conditions; and 3) its inefficient support of arbitrary range-match conditions. This is confirmed by Table VIII, which shows the TCAM storage requirements for the same three rule sets used to obtain the P^{2}C storage requirements listed in Table VI. Both tables reveal that the TCAM storage requirements of the P^{2}C scheme are about a factor 5 smaller than those of a "pure" TCAM-based classifier, resulting in substantially smaller power consumption and chip-area costs even if the SRAM storage requirements are taken into account. Note also that P^{2}C is the only scheme listed in Table VII that is able to improve its storage efficiency by adapting to the complexity of a rule set represented by the parameter k.

Although several efforts have been initiated to define benchmarks for network processors, e.g., [27] and [28], these have not yet resulted in the establishment of reference rule sets that can be used to compare classifiers. For lack of a better solution, Table VI also lists some storage requirements published for several other schemes, but for rule sets for which often only the

TABLE VII
PERFORMANCE COMPARISON

method	classification performance		storage requirements		fast incremental updates
	worst-case time complexity (latency)	suitable for pipelining	worst-case storage complexity	exploits identical field conditions	
TCAM[1]	1	yes	NdW	no	yes
tuple space[1] [12]	W^{d-1}	no	NdW	no	yes
set-pruning tries [8]	dW	no	$N^d dW$	no	no
grid-of-tries[2] [16]	dW	no	NdW	no	no
HiCuts [17]	dW	no	N^d	no	yes
RFC [9]	???	yes	N^d	yes	no
bitmap-intersect. [21]	$W' + \frac{N}{\text{memory width}}$	no	dN^2	yes	no
cross-producting [16]	$W' + 1$	no	N^d	yes	no
P^2C	$\frac{W'}{s} + 1$	yes	$dNk \log(\frac{N}{k} + 1)$	yes	yes

[1] Storage complexity shown for exact- and prefix-match conditions only.
[2] Two-dimensional classification rules only ($d = 2$).

TABLE VIII
STORAGE REQUIREMENTS

method	rules	storage
TCAM	37	0.47 kB TCAM
	138	3.1 kB TCAM
	1733	24 kB TCAM
grid-of-tries [16]	2000	836 kB SRAM
HiCuts [17]	1733	80 kB SRAM
RFC [9]	1733	400 kB SRAM
cross-producting [16]	50	1.5 MB SRAM

TABLE IX
INTERMEDIATE RESULT VECTOR SIZE [BITS]

		$n_1 = 0.95 * q_i$.			$n_1 = 0.99 * q_i$		
	k_i	k_i			k_i		
q_i	1	2	3	4	2	3	4
32	6	7	7	8	6	6	6
64	7	8	10	9	7	8	9
128	8	10	11	13	8	9	10
256	9	12	14	17	10	12	11
1024	11	16	20	25	14	16	19
4096	13	20	26	33	18	22	24
16384	15	24	32	41	22	28	32
32768	16	26	35	45	24	31	36

number of rules was indicated. These numbers are, therefore, intended only to provide an approximate performance indication.

VIII. SCALABILITY

It is expected that several emerging applications in the near future will require classification rule sets that are significantly larger than the ones used today. Furthermore, a possible transition from IPv4 to IPv6 will introduce 128-bit IP addresses, thus significantly increasing the field sizes. This section will discuss the impact of these trends on the field searches and the TCAM search.

The storage requirements of each field-search structure directly depends on the number of unique match conditions, represented by q_i for the field in the ith dimension. For firewall rule sets, we have found that q_i is typically much smaller than the number of rules for most fields, due to the frequent sharing of match conditions by multiple rules and the relatively large number of "wildcards." Furthermore, only the numbers of unique match conditions related to the IP address fields seem to increase approximately linearly with the number of rules, whereas the "growth rate" for the other fields is less or none. Based on the high storage efficiency of BART (e.g., BART handles a 72 K entry routing table in less than 500 kB [5]), it is expected that the field searches will be able to scale to rule sets involving tens of thousands of rules with no problems.

A transition of IPv4 to IPv6 addresses will only affect the field searches but not the TCAM search. As the characteristics of IPv6 routing tables and classification rule sets are not yet entirely clear, it is difficult to estimate their impact on the storage requirements and the search performance, especially latency. However, based on the existence of many fast and efficient lookup algorithms, including BART, this is not expected to affect the overall storage efficiency and classification performance significantly.

The TCAM storage requirements involve one TCAM entry per rule (assuming encoding style I or II), having a size equal to the accumulated sizes of the intermediate result vectors corresponding to all fields. An upper bound for the intermediate result vector size is provided by (1), which is based on the worst case assumption that all primitive ranges are distributed equally over all layers. For practical rule sets, however, it appears that most primitive ranges are nonoverlapping and can be combined at a single layer, whereas the very few remaining primitive ranges are located at different layers.

This characteristic will now be taken into account to allow a more exact investigation of the scalability effects on the intermediate result vector size. This is done in the following way. If the number of primitive ranges at layer j is represented by n_j, with $\sum_{j=1}^{k_i} n_j = q_i$, then the actual intermediate result vector size equals

$$I_i = \sum_{j=1}^{k_i} \lceil \log(n_j + 1) \rceil . \qquad (4)$$

Table IX lists the value of I_i for various combinations of q_i and k_i, calculated for the situation that 95% and 99%, respectively, of the primitive ranges are located at the first layer, and the remaining primitive ranges are distributed equally over the remaining layers.

If W_i denotes the field size in the ith dimension, then $(I_i)/(W_i)$ represents the "compression" achieved for this field compared to the TCAM-based classifier discussed in Section III-A, in which the entire field is part of the TCAM entry. For example, for IPv4 and IPv6 addresses, this results in a compression equal to $11/32$ and $11/128$, respectively, for $q_i = 1024$ and $k_i = 1$. Table IX shows that the compression decreases for larger values of q_i, which is in accordance with the worst case storage complexity of a TCAM growing linearly with the number of rules N, whereas the worst case storage complexity of $\mathrm{P}^2\mathrm{C}$ grows approximately according to $N \log(N)$ (see Table VII). However, Table IX clearly shows that $\mathrm{P}^2\mathrm{C}$ provides considerable compression for 32-bit IPv4 addresses for values of q_i up to at least 4 K, and far beyond for 128-bit IPv6 addresses. As the number of unique match conditions, q_i, is often a fraction of the total number of rules (see above), this implies that $\mathrm{P}^2\mathrm{C}$ will very likely be able to scale to at least several tens of thousands of IPv4 rules and to multiple hundreds of thousands of IPv6 rules, while providing a more cost-efficient solution than a pure TCAM-based classifier.

Note that for situations in which no compression is achieved, $(I_i)/(W_i) \geq 1$, the intermediate result vector can be replaced by the original field and the original match condition used as ternary-match condition (similar to the TCAM search as described in Section III-A) to limit the size of the intermediate result vector to the field size. This can be used to enhance efficiency especially for small fields, including single-bit fields.

IX. Conclusion

We have proposed a new multifield classification scheme called $\mathrm{P}^2\mathrm{C}$, which searches the fields independently in a first phase and determines the classification result by a TCAM search on the intermediate search results in a second phase. The key element of $\mathrm{P}^2\mathrm{C}$ is a novel encoding of the intermediate result vectors, which significantly reduces the storage requirements and minimizes the dependencies within the search structures, thus enabling fast incremental updates. $\mathrm{P}^2\mathrm{C}$ involves several encoding styles that can be applied simultaneously and allow the storage efficiency and update dynamics to be tuned at the granularity of individual rules. Furthermore, the $\mathrm{P}^2\mathrm{C}$ encoding has the unique property of being able to adapt to the complexity of classification rule sets. These features make $\mathrm{P}^2\mathrm{C}$ suitable for a broad range of applications.

The $\mathrm{P}^2\mathrm{C}$ scheme has been validated using simulations and by a prototype implemented in an FPGA. Although this prototype runs at a clock speed of only 33 MHz, it is able to achieve wire-speed classification performance for OC-192, clearly demonstrating the enormous performance potential of $\mathrm{P}^2\mathrm{C}$ and its feasibility for achieving OC-768 performance using current off-the-shelf technology. $\mathrm{P}^2\mathrm{C}$ was able to support 1733 rules of a commercial firewall application using only 2 kB of SRAM in combination with 5 kB of TCAM. Performance evaluations have shown that update rates of the order of millions per second are feasible.

Acknowledgment

The authors would like to thank Cisco Systems for its valuable help by providing one of the classification rule sets that was used for evaluating the $\mathrm{P}^2\mathrm{C}$ scheme. Special thanks go to L.-M. Pavka and C. Bolliger for their help preparing this manuscript.

References

[1] M. A. Ruiz-Sánchez, E. W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, vol. 15, no. 2, pp. 8–23, Mar./Apr. 2001.

[2] H. H.-Y. Tzeng and T. Przygienda, "On fast address-lookup algorithms," *IEEE J. Select. Areas Commun.*, vol. 17, pp. 1067–1082, June 1999.

[3] BGP Table. [Online]. Available: http://bgp.potaroo.net

[4] A. Brodnik, S. Carlsson, M. Degermark, and S. Pink, "Small forwarding tables for fast routing lookups," *Comput. Commun. Rev.*, vol. 27, no. 4, pp. 3–14, Oct. 1997.

[5] J. van Lunteren, "Searching very large routing tables in wide embedded memory," in *Proc. IEEE GLOBECOM*, vol. 3, Nov. 2001, pp. 1615–1619.

[6] V. P. Kumar, T. V. Lakshman, and D. Stiliadis, "Beyond best effort: router architectures for the differentiated services of tomorrow's internet," *IEEE Commun. Mag.*, vol. 36, pp. 152–164, May 1998.

[7] C. Macián and R. Finthammer, "An evaluation of the key design criteria to achieve high update rates in packet classifiers," *IEEE Network*, vol. 15, pp. 24–29, Nov./Dec. 2001.

[8] P. Gupta and N. McKeown, "Packet classification on multiple fields," *Comput. Commun. Rev.*, vol. 29, pp. 147–160, Oct. 1999.

[9] Z. Cao, Z. Wang, and E. Zegura, "Performance of hashing-based schemes for internet load balancing," in *Proc. IEEE INFOCOM*, vol. 1, Mar. 2000, pp. 332–341.

[10] SiberCAM Application Note, SiberCore Technologies. [Online]. Available: http://www.sibercore.com/pdf/an_scan001_1.pdf

[11] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," *Comput. Commun. Rev.*, vol. 29, no. 4, pp. 135–146, Oct. 1999.

[12] V. Srinivasan, "A packet classification and filter management system," in *Proc. IEEE INFOCOM*, vol. 3, Apr. 2001, pp. 1464–1473.

[13] T. Y. C. Woo, "A modular approach to packet classification: algorithms and result," in *Proc. IEEE INFOCOM*, vol. 3, Mar. 2000, pp. 1213–1222.

[14] J. Xu, M. Singhal, and J. Degroat, "A novel cache architecture to support layer four packet classification at memory access speeds," in *Proc. IEEE INFOCOM*, vol. 3, Mar. 2000, pp. 1445–1454.

[15] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, Mar./Apr. 2001.

[16] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," *Comput. Commun. Rev.*, vol. 28, no. 4, pp. 191–202, Oct. 1998.

[17] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *Proc. Hot Interconnects 7*. Stanford University, Stanford, CA, Aug. 1999, . .

[18] A. Feldmann and S. Muthukrishnan, "Tradeoffs for packet classification," in *Proc. IEEE INFOCOM*, vol. 3, Mar. 2000, pp. 1193–1202.

[19] P. Gupta and N. McKeown, "Dynamic algorithms with worst-case performance for packet classification," in *Proc. IFIP Networking*, Paris, France, May 2000, pp. 528–539.

[20] C.-F. Su, "High speed packet classification using segment tree," in *Proc. IEEE GLOBECOM*, vol. 1, Nov. 2000, pp. 582–586.

[21] T. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *Comput. Commun. Rev.*, vol. 28, no. 4, pp. 203–214, Oct. 1998.

[22] F. Baboescu and G. Varghese, "Scalable packet classification," *Comput. Commun. Rev.*, vol. 31, no. 4, pp. 199–210, Oct. 2001.

[23] J. van Lunteren and A. P. J. Engbersen, "Prefix-based parallel packet classification," IBM Res. Rep., RZ 3210 (#93256), 2000.

[24] D. Shah and P. Gupta, "Fast incremental updates on ternary-CAMs for routing lookups and packet classification," in *Proc. Hot Interconnects 8*, Aug. 2000, pp. 145–153.

[25] FPGA-Based Rapid-Prototyping Platform Spyder-Virtex-X2/XCV800 [Online]. Available: http://www.x2e.de

[26] P. Gupta, "Routing lookups and packet classification: theory and practice," Tutorial at Hot Interconnects VIII, Stanford, CA, [Online]. Available: http://klamath.stanford.edu/~ pankaj/talks/, Aug. 2000.

[27] Network Processing Forum (NPF) [Online]. Available: http://www.npforum.org

[28] Embedded Microprocessor Benchmark Consortium (EEMBC). [Online]. Available: http://www.eembc.org

Jan van Lunteren (M'02) received the M.Sc. degree in electrical engineering, the M.Sc. degree in technological design, and the Ph.D. degree in electrical engineering in 1992, 1994, and 1998, respectively, all from the Technical University of Eindhoven, The Netherlands.

He has been with the IBM Zurich Research Laboratory, Rüschlikon, Switzerland, since 1994, doing research on high-speed networking. His current interests are focussed on network processors in general and wire-speed routing lookup and packet classification algorithms specifically.

Ton Engbersen (M'92–SM'02) has been employed at the IBM Zurich Research Laboratory, Rüschlikon, Switzerland, since 1984. He was instrumental in bringing VLSI design skills to the laboratory in the mid 1980s, and in the early 1990s developed the PRIZMA switch architecture. PRIZMA has become a family of communication switch offerings that IBM is marketing through its IBM Microelectronics Division. From 1996 to 1997, he spent two years at the IBM T. J. Watson Research Center, Yorktown Heights, NY, where he led the initial development of MPLS. Since 1997, he has been managing the Network Technology Research Group at the Zurich Research Laboratory. His current research interests are in networking technology, scalable network processors, scalable switching technology, and SDH/Sonet. He is a Member of the IBM Academy of Technology and is currently serving on the Technology Council of this Academy.