Packet Classification Algorithms

Viktor Puš[*]
Departement of Computer Systems
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
ipus@fit.vutbr.cz

Abstract

This paper deals with packet classification in computer networks. Classification is the key task in many networking devices, most notably packet filters – firewalls. This paper therefore concerns the area of computer security. The paper is focused on high-speed networks with the bandwidth of 100 Gb/s and beyond. General-purpose processors cannot be used in such cases, because their performance is not sufficient. Therefore, specialized hardware is used, mainly ASICs and FPGAs. Many packet classification algorithms designed for hardware implementation were presented, yet these approaches are not ready for very high-speed networks. This paper addresses the design of new high-speed packet classification algorithms, targeted for the implementation in dedicated hardware. The algorithm that decomposes the problem into several easier sub-problems is proposed. The first subproblem is the longest prefix match (LPM) operation, which is used also in IP packet routing. As the LPM algorithms with sufficient speed have already been published, they can be used in out context. The following subproblem is mapping the prefixes to the rule numbers. This is where the paper brings innovation by using a specifically constructed hash function. This hash function allows the mapping to be done in constant time and requires only one memory with narrow data bus. The algorithm throughput can be determined analytically and is independent on the number of rules or the network traffic characteristics. With the use of available parts the throughput of 266 million packets per second can be achieved. Additional three algorithms (PFCA, PCCA, MSPCCA) that follow in this paper are designed to lower the memory requirements of the first one without compromising the speed. The second algorithm lowers the memory size by 11 % to 96 %, depending on the rule set. The disadvantage of low stability is removed by the third algorithm, which reduces the memory requirements by 31 % to 84 %, compared to the first one. The fourth algorithm combines the third one with the older approach and thanks to the use of several techniques lowers the memory requirements by 73 % to 99 %.

[*]Recommended by thesis supervisor: Prof. Václav Dvořák. Defended at Faculty of Information Technology, Brno University of Technology on September 17, 2012.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays, Algorithms implemented in hardware*;
C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g., firewalls)*

Keywords

Packet Classification, Algorithms, FPGA

1. Introduction

With the rapid development of computer networks, security threats such as viruses and other attacks by hackers are also on the rise. Network security is studied and applied at various layers: direct filtering at the packet level, intrusion detection at the application level, traffic monitoring and detection of anomalous behavior of the whole network. Network traffic filtering has become one of the first steps in securing any network or computer. While the packet filter cannot detect and block all dangerous network traffic, it is still one of the most effective building blocks for any computer security system. It also often cooperates with the higher levels of network security. For example, anomaly detection system updates filtering rules based on the assessment of current threats.

The packet filtering system must perform packet reception, packet header parsing, packet classification and the action based on the instruction from the matching classification rule. Packet classification is the most complex part of the system, and it therefore determines the speed, and also mostly the cost of the system. This task is not only important in practical applications, but also interesting from the theoretical point of view, having implications in geometry and other fields [12].

As network speeds are increasing, the demand for high speed packet processing is also growing. While 10 Gb/s ports are commonly present in various networking devices, the 100 Gb/s technology is expected to be increasingly available in the near future. Standard for 100 Gb/s Ethernet was proposed as IEEE standard 802.3ba in 2008 and ratified in June 2010. This new standard is capable of transmitting one packet each 6.7 ns in each direction.

Packet classification must be able to achieve this throughput, otherwise it would throttle the bandwidth.

Many algorithms for packet classification were proposed, but the goal of 100 Gb/s throughput is either beyond the limits of the current technology, or requires excessive amount of high-speed (and thus expensive) memory.

The algorithms oriented on high speed use various methods of hardware acceleration. Application-specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs) are commonly used. Ternary Content-Associative Memories (TCAMs) can also be found in commercial devices in conjunction with ASICs of FPGAs. Due to high development cost and long time-to-market of ASICs and also the high cost and power consumption of TCAMs, FPGAs gain increasing popularity. Programmability of FPGAs make them suitable for the research in the field of packet classification.

Properties of each packet classification approach are defined by two main factors: The technology used, and the algorithm running on the selected device. While the technology is gradually improved by silicon vendors, and can be approximately predicted by the application of Moore's law, new algorithms may bring significant improvements immediately. That's why the research of packet classification algorithms is important for practical applications.

This paper proposes new packet classification algorithm tailored for high-speed applications, targeting 100 Gb/s networks. Its unique property is the throughput higher than 100 Gb/s, even in the worst case. This speed cannot be achieved by the current algorithms. The lack of guaranteed throughput is generally an issue of all current algorithms which is completely eliminated by the algorithm presented in this paper.

The algorithm is intended for implementation in a hardware accelerator (ASIC or FPGA) and therefore it is designed with consideration of capabilities of these devices. All steps of the algorithm are either very simple arithmetic operations and memory accesses, or were shown to be able to run at the speeds required for 100 Gb/s network in ASIC or FPGA. The idea of problem decomposition introduced in recent literature [6] is also employed in this paper. The algorithm is implemented as a processing pipeline, which brings higher speed by exploiting the parallelism inherent to ASIC and FPGA devices. The algorithm is built around the idea of finding fast direct mapping from the results of independent (and thus potentially parallel) prefix search engines to the correct rule number. Perfect (collision-free) hash function construction algorithm [4] is used to create such direct mapping.

In addition to exploiting inherent parallelism of hardware implementation, the algorithm achieves high speed by employing rather complex software precomputation phase for finding the mapping function which then leads to very simple evaluation of the mapping function in the hardware. Due to the fact that there is no loop in the process of classification, the throughput of the algorithm is perfectly deterministic and constant. Speed of the algorithm is comparable or better than the fastest known algorithms.

While the speed of the algorithm is excellent, it requires significant amount of memory for its data structures. This paper therefore discusses the possibility of lowering memory requirements of the algorithm. Three following algorithms make use of empirically obtained facts about common properties and the structure of rule sets. They lower the memory consumption of the original algorithm by application of several new optimization techniques. These methods add very simple logic to the processing pipeline of the algorithm in order to avoid the states which generate excessive requirements on the amount of memory in the first algorithm.

2. Related Work

This section introduces current algorithms for packet classification. Separate subsections describe some of the important topics of the packet classification and how they are reflected in recent literature.

2.1 The Longest Prefix Match Operation

Packet routing in IP networks can be considered as one-dimensional classification – only destination IP address is important for routing. The routing tables contain pairs of network prefix and the output interface where should the packet belonging to that prefix be sent to. Because a single packet may belong to several prefixes (of different lengths), a router must find the longest prefix that matches the packet. This search on prefixes is the Longest Prefix Match operation, sometimes also called the *IP Lookup*. This operation is also important for classification in more than one dimension.

Because the LPM operation is performed in IP packet routing, many approaches were published [7, 15, 9]. The basic algorithm and the data structure for the LPM is the unibit trie. Trie is often modified to process more input bits in each step and to reduce memory requirements. Popular examples of such algorithms are the Tree Bitmap [7] and the Shape Shifting Trie [15]. The LPM operation can be performed very fast: recently published approaches are able to achieve billions of lookups per second [9].

LPM is used in all following packet classification algorithms. It is performed independently in each dimension, as shown in Figure 1. Each LPM engine contains all prefixes from one dimension found in the rule set (the prefix set). Because the LPM engines are independent, there is good potential for parallel processing. This paper does not provide any new LPM algorithm. Some existing approach should be employed instead.

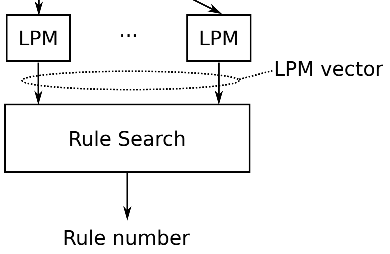

Figure 1: Common decomposition of packet classification algorithms.

2.2 MSCA

Multi Subset Crossproduct Algorithm [6] brought major improvements to packet classification methods in 2006. In this work, Dharmapurikar et al. introduce the concept of *pseudorules* (described in detail in Section 2.3). Because pseudorules expansion is similar to Cartesian product, authors provide heuristics how to break the rule set into several subsets and eliminate the majority of pseudorules. The LPM operation is slightly modified to return a result for each subset, because subsets may contain different prefix sets. A Bloom filter is associated with each subset to perform set membership query. If the Bloom filter output is true, one rule table memory access is performed to retrieve the resulting rule or pseudorule.

MSCA also identifies rules that generate excessive amount of pseudorules. These rules are called *spoilers* and are treated in a separate branch of the algorithm to further reduce number of pseudorules. In hardware implementation, spoilers are moved to a small on-chip TCAM (Ternary Content–Addressable Memory).

MSCA suffers from placement of the rule table in the external memory. Wide data word (over 100 bits if IPv6 is not used) is needed to fetch the rule in one access.

2.3 Pseudorules

The concept of pseudorules was introduced in MSCA and is used also in all four algorithms that follow in this work. First we describe in detail how pseudorules are created: Pseudorules must be added to the rule set to cover all valid combinations of LPM results. In fact, a pseudorule is always a special case of some rule. We explain the emergence of pseudorules on the example in Figure 2 and Table 1. We can see a simplified classification in two three-bit dimensions with three rules. In each dimension, unibit trie is shown to illustrate the LPM operation. Colored arcs are the rules.

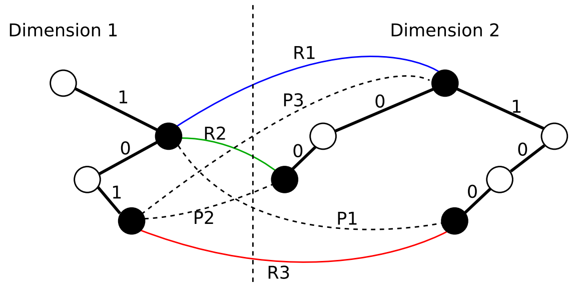

Figure 2: Three rules $R1, R2, R3$ and three added pseudorules.

Rule	Dim. 1	Dim. 2	Priority	Target rule
R1	1*	*	1	
R2	1*	00*	2	
R3	101	100	3	
P1	1*	100	1	R1
P2	101	00*	2	R2
P3	101	*	1	R1

Table 1: Rules and pseudorules.

For example, LPM vector for packet with header fields $(111, 100)$ is $(1*, 100)$. This combination is not in the original rule set, but it is clear that the correct result is rule $R1(1*, *)$. Therefore, pseudorule $P1(1*, 100)$ must be added to handle this situation. Table 1 contains all rules and pseudorules together. The *target rule* in this table points to the correct classification result.

The generation of pseudorules is similar to Cartesian product, and may potentially expand the rule set significantly, but not all possible combinations of prefixes need to be added. Prefix combinations matching no rule are not pseudorules. If the universal rule is in the rule set, then all possible combinations must be added, because all of them match some rule (at least the universal rule). However, this rule can be removed from the rule set and returned as a result only if no other rule matches the packet. Therefore, pseudorules form a subset of Cartesian product of all prefix sets. The algorithm generating all pseudorules from the rule set is shown in Algorithm 1. The algorithm creates pseudorules by finding prefix combinations that match some rule.

Algorithm 1 Generating pseudorules from the rule set.

```
Input: Rule set $R$ without the universal rule and with
    range conditions converted to prefixes.
 1: for all dimension $d$ do
 2:   Create empty prefix set $S_d$.
 3:   for all rules $r \in R$ do
 4:     Add $r.d$ to $S_d$.
 5:   end for
 6:   {$S_d$ is now complete prefix set for dimension $d$.}
 7: end for
 8: Create empty set of pseudorules $P$.
 9: for all rules $r \in R$ sorted from the highest to the
    lowest priority do
10:   for all dimensions $d$ do
11:     Create reduced prefix set $SR_d$ by selecting pre-
        fixes from $S_d$ which are covered by $r.d$.
12:   end for
13:   Create set $CP$ as the Cartesian product of all re-
      duced prefix sets $SR_x$.
14:   for all candidate pseudorules $cp \in CP$ do
15:     if $cp \notin P$ then
16:       Add $cp$ to $P$ with the same priority as $r$ and
          the target rule pointer set to $r$.
17:     end if
18:   end for
19: end for
Output: $P$
```

Packet classification algorithms deal with pseudorules in the Rule Search Stage (see Figure 1). The problem of finding the correct rule has the interesting property of mapping huge number of inputs (LPM vectors) to only moderate number of outputs (rule numbers).

Dharmapurikar et al. in [6] observes that the number of pseudorules is up to 200 times higher than the number of original rules. Experiments with Algorithm 1 however show even higher numbers in some cases. Figure 3 shows the histogram of pseudorules for one rule set giving the number of associated pseudorules for each rule. It can be seen that the majority of pseudorules is generated by minority of rules (note the logarithmic scale of the vertical axis). In this particular case, the top 10 rules (10 % from a total of 103 rules) generate 42.34 % pseudorules. Removing those rules should decrease the number of pseudorules significantly. Histograms for other rule sets show the similar feature.

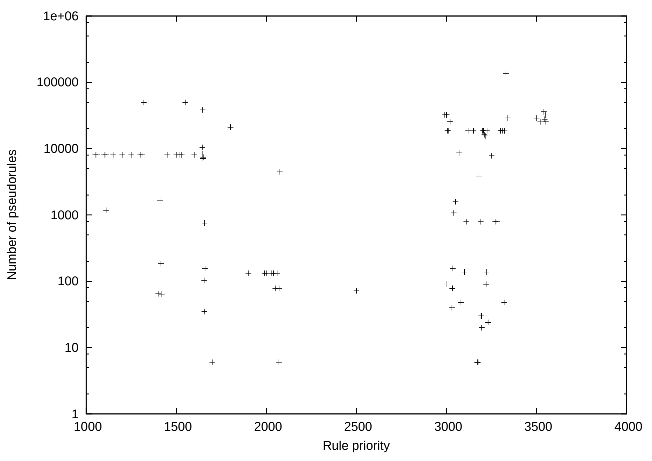

Figure 3: Pseudorules histogram.

3. Perfect Hashing Crossproduct Algorithm

The Perfect Hashing Crossproduct Algorithm [14] introduced in this chapter addresses the most critical part of the decomposition algorithms, which is the Rule Search stage. The problem in the Rule Search stage is that it has to handle large amount of possible inputs (all possible LPM vectors), while some inputs match no rule, some match one rule and some match several rules, from which the one with the highest priority has to be selected. The 100 Gb/s target requires that this processing is done very fast.

We start the algorithm description with the simplified situation when there are no pseudorules. We add pseudorules handling later in the text.

In the domain of packet classification, the general concept of hash table specializes to the rule table and the set of hash keys specializes to the set of all rules expressed as LPM vectors. This set is known in advance, therefore the static perfect hash function suffices in this case. The result of the hash function is the pointer to the rule table.

The perfect hash table (rule table) stores all rules, except for the universal rule, which covers all packets. The reason for removing the universal rule is explained later. The perfect hash function maps each LPM vector to the correct rule if the packet matches some rule. Therefore, the classification algorithm performs the LPM and then computes the perfect hash function. The result is a pointer into the rule table, where the rule is read from. This rule is then compared to the original packet. If it matches, then the correct rule was found. If not, then the packet matches no rule or the universal rule (if present). Figure 4 shows the basic structure of PHCA.

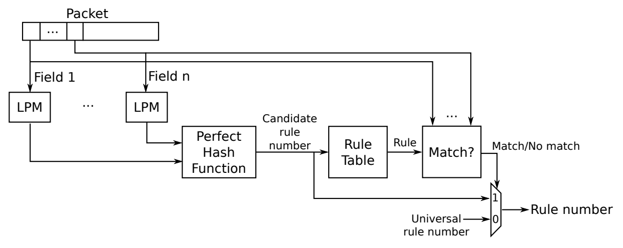

Figure 4: Basic structure of the PHCA.

This arrangement allows the perfect hash function to return any (even incorrect) result if the packet matches no rule. This situation may occur only if the LPM vector does not correspond to any rule and is always rectified by matching the packet to the rule in the rule table. In the case of no match, the universal rule is returned. Now it is also visible why the universal rule is not included in the perfect hash function construction: If it was included, then all LPM vectors not corresponding to any rule would have to be correctly hashed to the universal rule. Allowing the perfect hash function to return in some cases an incorrect rule number removes the necessity to handle the complete Cartesian product of all prefix sets and therefore saves significant amount of memory in the perfect hash function implementation.

We continue by describing the perfect hash function construction algorithm as presented by Czech et al. in [4]. The perfect hash construction algorithm creates an acyclic graph where edges are the hash keys and vertices are results of two different ordinary hash functions. Vertices are then assigned values so that they sum up to the desired hash value. The hash construction is shown in Algorithm 2. The function associations (a, b) are associations between rules in the LPM vector representation and the rule numbers, which are also pointers to the rule table. Prefixes in LPM vectors are represented by some symbols, for example 16-bit integers. The created perfect hash function has the same number of inputs and outputs and is bijective.

Algorithm 2 Construction of the hash function.

```
Input: Set of function associations $A$, each association is
    a tuple $(a, b)$ of the LPM vector and the correct rule
    number.
 1: Create new graph $G$ with $g = c|A|$ vertices and no
    edges, where $c > 1$. The real number $c$ is used to
    increase the graph size when needed.
 2: Pick two different ordinary hash functions $f_1, f_2$ that
    output integers from interval $[0, g-1]$.
 3: for all $(a, b) \in A$ do
 4:   $h_1 \leftarrow f_1(a)$
 5:   $h_2 \leftarrow f_2(a)$
 6:   Add an edge between vertices $h_1$ and $h_2$ into the
      graph $G$ and label that edge $b$.
 7: end for
 8: if $G$ contains cycle then
 9:   Increase $c$ and repeat the algorithm. The increment
      is typically a small number, for example 0.2.
10: end if
11: Associate values to each vertex such that for each
    edge the sum of the values of both its vertices is the
    value of that edge. This can be done by depth-first
    search algorithm, because the graph is acyclic.
Output: $f_1, f_2$ and vertex values of $G$.
```

After the graph is created, the hash value computation is simple. At first, two ordinary hash functions f_1 and f_2 of the input LPM vector are computed. Then two vertex values are read from the Vertex Table and added. For each vertex, only one integer is stored. Functions f_1 and f_2 may be virtually any convenient hash functions, the only requirement is that the same functions are used in the perfect hash construction and the following computation. CRC32 is used for experiments in this paper.

Complete table of rules is stored in PHCA, because if packet matches no rule, the perfect hash function still returns some value (the function is not built for LPM vectors which are not rules). Therefore, packet must be compared to the selected rule: If it matches, the correct rule was found. Otherwise, the packet matches no rule or universal rule (if present). Structure of the Perfect Hashing Crossproduct Algorithm is shown in Figure 5.

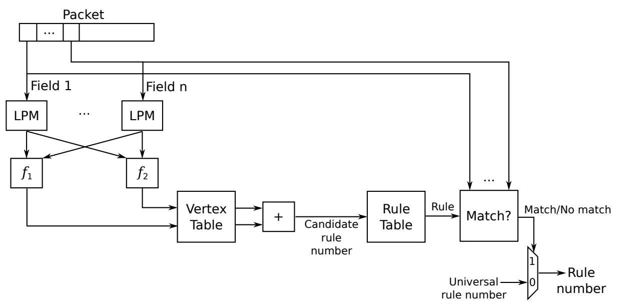

Figure 5: Detailed structure of the PHCA.

The arrangement described so far does not support pseudorules. To add the pseudorules support to the algorithm, surprisingly little must be done. Refer to the Algorithm 1: each pseudorule has the number of correct rule associated to it. It is therefore known which rule is the desired algorithm output for each pseudorule. This information is used to create additional input to the perfect hash function construction Algorithm 2. In addition to the function associations (a, b) between the rules in the LPM vector representation and the rule numbers, the input A now contains also the associations between the *pseudorules* in the LPM vector representation and the *rule* numbers.

The perfect hash function is not bijective anymore. It maps large number of pseudorules into the smaller number of rules. In fact, the hash function is not perfect anymore. It instead contains many collisions among each rule and all of its associated pseudorules. The use of intended hash function collisions is an innovative, non-traditional application of perfect hash functions. The important point is that none of pseudorules is stored in PHCA – the rule table still contains only rules after adding the pseudorules support into the algorithm. Therefore, a significant amount of memory is saved.

4. Prefix Filtering Classification Algorithm

The method of lowering the number of pseudorules is based on the observation that many classification rules often contain universal conditions. For example, if the user of a firewall wants to block a specific source IP address, the filtering rule does not specify any destination IP address nor the port number. This means that packet with any destination IP and any port number matches this rule. However, the rule can create many pseudorules because all more specific destination IPs and ports have to be covered.

Figure 6 is an example for two fields, where the universal condition in the rule set produces many pseudorules. The situation is even worse for multiple fields, because pseudorules create Cartesian product.

PFCA [8] inserts a *Generalization Stage* (GS) into the classification algorithm after LPM engines (Figure 7). If it is applicable, GS is able to replace LPM results (parts

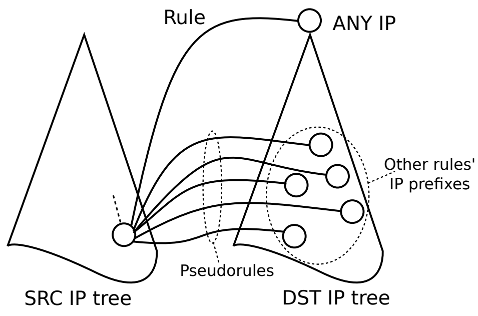

Figure 6: One of the most severe causes of pseudorules: ANY values in the rule set.

of LPM vector) by the ANY value without loosing any information needed for correct packet classification. As a result, number of output combinations is reduced after GS. This will reduce the data structures of the following stages of all crossproduct algorithms. In other words: If GS replaces pseudorule by a rule, then the pseudorule does not need to be treated. Other parts of the algorithm remain the same as in PHCA.

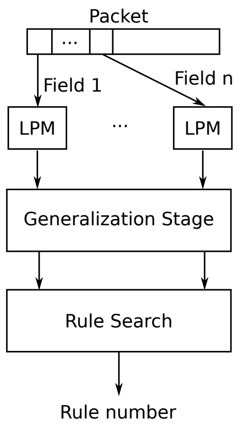

Figure 7: Structure of the Prefix Filtering Classification Algorithm.

Definition 1 (Generalization Rule). *The generalization Rule (GR) is a 3-tuple $g = (b, v, G)$ where b is an index to the LPM vector, v is a value of a particular field of LPM vector, and G is a set of indices to the LPM vector.*

Definition 2 (Generalization Rule Effect). *The effect of one GR is: if $LPM[b] = v$, then for each index $i \in G$ set $LPM[i] := ANY$. All GRs may be applied together, their ordering is unimportant.*

This scheme corresponds to the following situation: we know that if a field $LPM[b]$ has a particular value v, then some other fields $LPM[i], i \in G$ are unimportant, because the result of classification is already determined.

It remains to find an algorithm to create GRs. First, all pseudorules must be found by Algorithm 1. The algorithm creates a list of pseudorules where all pseudorules corresponding to one rule are stored continually. We call these continual sections *P-blocks*. We can say that the list of pseudorules consists of k P-blocks, where k is the number of rules and the P-blocks are sorted from the highest to the lowest priority. Note that each rule is also added as a pseudorule before all of its pseudorules are added to the list. This leads to the fact that in each P-block, the first pseudorule is the most general of all pseudorules in that P-block, and it is the original rule itself. This ordering is helpful in the next part of the algorithm where GRs are created and some pseudorules are removed.

Algorithm 3 Creating generalization rules and removing pseudorules.

```
 1: Input: List of pseudorules $P$.
 2: Create empty set of generalization rules $GR$.
 3: for all pseudorules $p_{this} \in P$ do
 4:   for all dimensions $d$ do
 5:     if $p_{this}[d] \neq ANY$ and no previous $p_{prev} \in P$
        exists such that $(p_{prev}[d] = p_{this}[d]$ or $p_{prev}[d] = ANY)$ then
 6:       Create new generalization rule $g_{new} = (d, p_{this}[d], G)$, where $G = \{i | p_{this}[i] = ANY\}$.
 7:       Add $g_{new}$ to $GR$.
 8:       Remove all pseudorules $p_{after}$ that follow in $P$
          after $p_{this}$ where $p_{after}[d] = p_{this}[d]$ and exists
          $i \in G$ such that $p_{after}[i] \neq ANY$.
 9:     end if
10:   end for
11: end for
12: Output: Set of generalization rules $GR$, reduced list
    of pseudorules $P$.
```

Algorithm 3 identifies situations when the rule defines some field with index d and allows the ANY value in fields with indices from G. Then in certain cases, for all LPM vectors with the same value at index d, values at indices from G may be replaced by ANY value, and the result of the classification is still uniquely determined.

There are several conditions that must be met when creating a GR:

- The rule must contain at least one ANY value. This condition is not explicitly written in the algorithm, because it is implicit: for rules with no ANY value, G would be empty and the GR would make no sense.
- The same value of the field at index d has not appeared earlier in the list of pseudorules. If this condition is not true, we cannot be sure that the value in this field unambiguously determines the correct classification result.
- The ANY value has not appeared at the index d earlier in the list of pseudorules. The reason for this condition is the same as for the preceding one.

5. Prefix Coloring Classification Algorithm

Let us focus on the fact that the LPM operation is performed independently for each field in decomposition–based packet classification algorithms such as PHCA and PFCA. (see Figure 1). The advantage of this scheme is the strong potential for parallel computation. On the other hand, LPM results are logically related – only certain combinations of LPM results form a rule, the rest of them are unwished pseudorules. Thus, the knowledge of LPM result from one dimension *should* affect LPM result in other dimensions. Figure 8 shows an example situation when the knowledge of LPM result from one dimension has impact on the LPM of the other dimension: When the Dimension 2 LPM result is $00*$, then there is no need to continue searching below prefix $1*$ in the Dimension 1 LPM. The result 101 would not bring any new information significant for the packet classification, because no other (pseudo) rule is reachable. The aim of this section is to find an effective way of exchanging information among dimensions before the LPM results reach the Rule Search stage.

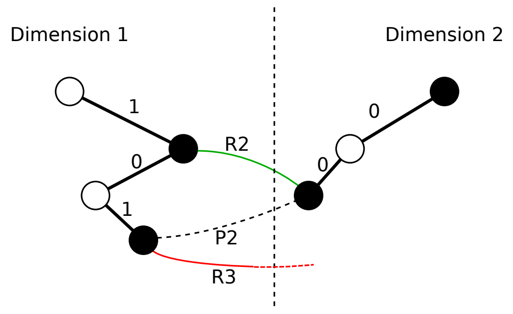

Figure 8: Motivation for communication between LPMs. There is no need to traverse the Dimension 1 trie below $1*$ if the Dimension 2 LPM result is $00*$.

The example of algorithm which takes one LPM result into account when performing the other LPM is the Grid-of-Tries [16]. In the basic version of that algorithm, there is one second-level trie for each valid result of the first-level trie. This scheme does not scale well. The implied sequential processing of dimensions is not an issue, because it is easily pipelined. The worse fact is that the number of tries at lower levels can grow exponentially. There is one second-level trie for each result of the first level trie, and there is one third-level trie for each result of each of the second-level tries etc. Also, Grid-of-Tries selects some particular ordering of dimensions, making the communication between LPMs only unidirectional. However, the idea of one LPM result affecting (and being affected by) other dimensions' LPM results is the key to the PCCA presented in this section.

In the Grid-of-Tries algorithm the tries at lower levels always store some subset of the full prefix set of the corresponding dimension. PCCA [13] delays the communication among dimensions *after* the LPM operation, so the LPM engines must return maximum amount of data. Let's suppose that the LPM operation is modified to return *all* matching prefixes, not only the longest one. The newly added Color Processing Stage aims to select from these prefixes only combinations which are in the rule table. This selection would remove all pseudorules, making the Rule Search step easier.

Let each prefix p_1 contain a precomputed bitmap for all other dimensions. There is one bit for each prefix of each dimension in the bitmap. The bit stored in prefix p_1 corresponding to prefix p_2 is set to 1 if prefixes p_1 and p_2 appear together in some rule. Otherwise the bit is set to 0. We call this bitmap the *full bitmap.* Each LPM result (prefix) now contains an information about "allowed" and "suppressed" prefixes from other dimensions. From all LPM results, only allowed prefixes are then used to create the LPM vector which is then passed to the following stages of the algorithm.

It is possible to remove almost all pseudorules this way, because most unwanted LPM vectors are filtered away. Not *all* pseudorules are removed, because the information about rules priority is missing in prefixes. (see Figure 9).

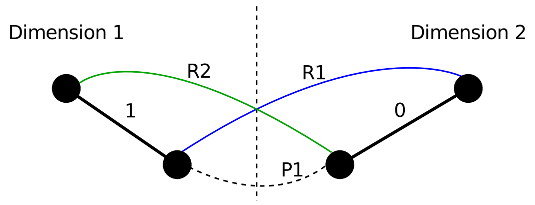

Figure 9: Pseudorule $P1$ cannot be removed without the information about rules priority.

The full bitmap has a disadvantage in adding large memory overhead to the LPM results. Number of prefixes may be large, therefore LPM results table would have to use very wide data words. Table 2 shows that numbers of prefixes are too large for full bitmaps in larger rule sets. Moreover, sizes of full bitmaps vary with each rule set.

Instead of using full bitmaps, fixed amount of *groups of prefixes* are created in PCCA to limit the size of bitmaps. Only small bitmaps for groups are stored. Grouping of prefixes may be based on different criteria. There is always some *implicit* information in the prefixes: Each prefix has its length, which can be used directly as a group number. The length may also be divided by a constant to create more coarse-grained grouping and thus smaller bitmaps. Number of parents in the LPM tree is called prefix *nesting* and can also be used for prefix grouping.

Explicit grouping information can be added to the prefixes: we assign an abstract *color* property to each prefix. Number of colors is set to be much smaller than the number of prefixes. Instead of carrying large full bitmaps of prefixes, each prefix contains its own color and only a small bitmap of allowed colors for each other dimension.

Explicit prefix coloring is the most general option, because all other grouping criteria can be simulated by proper assignment of colors. Therefore, only explicit grouping by colors is used in the following text.

The bitmap stored in prefixes is called the *Allowed Colors Bitmap* (ACB). Instead of returning all matching prefixes, the LPM operation returns the longest matching prefix for each color. Also, it returns the *Aggregate Allowed Colors Bitmap* (AACB) which is a bitwise logical disjunction (OR) of all ACBs observed during the LPM tree descent.

The modified processing pipeline of the classification algorithm is in Figure 10. The Color Processing Stage works like a filter, which checks each input prefix against AACBs from other dimensions. Only one prefix for each dimension may pass through it. The Color Processing Stage selects the longest prefix from prefixes that are allowed by all AACBs. Operations of the Color Processing Stage are shown in detail in Algorithm 4.

One possible method of assigning colors to prefixes is presented here. This method is designed to achieve the balanced distribution of prefixes among colors: Prefixes are sorted by length and then assigned colors sequentially with repetition (i.e. 0, 1, 2, 3, 0, 1, ...).

The method of filling ACBs in prefixes is straightforward: At the beginning, all bitmaps contain zeros. Then for each rule r and each prefix $r.d$ of rule r, set bits in the ACB to one, such that bitmaps allow colors of all other prefixes of the rule r.

It remains to find an algorithm that generates pseudorules. Due to colors and color bitmaps, majority of pseudorules are suppressed. However, some pseudorules are generated for most rule sets. The experience with previous algorithms is that generating all pseudorules during the software precomputation phase may be highly time–consuming operation. Therefore it is undesirable to generate all pseudorules and then remove some of them. Instead, an algorithm that directly generates only pseudorules which have to be considered in the Rule Search Stage is presented (Algorithm 5).

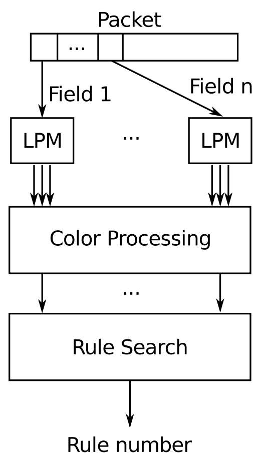

Figure 10: Improved scheme of decomposition algorithm.

Algorithm 4 Operations of the Color Processing Stage.

```
Input: AACB for each dimension, prefix for each dimension and color.
  for all dimensions $d$ do
    create the Present Colors Bitmap (PCB$_d$) where bits correspond to colors in dimension $d$. Each bit of PCB$_d$ is set to 1 if some prefix with that color was returned by the LPM, and to 0 otherwise.
  end for
  for all dimensions $d$ do
    Final Allowed Colors Bitmap FACB$_d$ $\leftarrow$ bitwise_and(PCB$_d$, corresponding AACBs from all other dimensions).
  end for
  if some FACB contained all zeros then
    Packet matches no rule.
  else
    Create empty output LPM vector $V$.
    for all dimensions $d$ do
      Add the longest matching prefix allowed by FACB$_d$ to $V$.
    end for
  end if
Output: LPM vector $V$
```

Algorithm 5 Pseudorules generating with respect to colors.

```
Input: Rules with prefixes containing ACBs and their own color.
  Create empty list of pseudorules $P$.
  The rule set is traversed from the highest to the lowest priority:
  for all rules $r$ do
    for all dimensions $d$ do
      $L_d \leftarrow$ list of all prefixes from dimension $d$ matching rule $r$. In this list, there is prefix $r.d$ from rule $r$, and all more specific (longer) prefixes from other rules.
    end for
    A decision tree is traversed. Each tree level corresponds to one dimension $d$, dimension ordering is unimportant. Tree edges are prefixes from $L_d$. The tree is traversed by a depth-first traversal algorithm. Descent is performed only if colors and ACBs in prefixes allow the combination of prefixes from the root of the tree to the current leaf.
    if the lowest tree level is reached then
      if (the combination of prefixes from root to leaf) $\notin P$ then
        Add the prefix combination into $P$. ($r$ is also added by this operation)
      end if
    end if
  end for
Output: List of pseudorules $P$
```

6. Multi Subset Prefix Coloring Classification Algorithm

The Multi Subset Prefix Coloring Algorithm (MSPCCA) aims to combine PCCA with MSCA [6]. MSCA is the first algorithm to introduce the concept of pseudorules. It also proposes the technique of spoilers removal. But the most important optimization of MSCA is the division of rule set into subsets. The algorithm exploits the fact that the sum of Cartesian products of small sets is much smaller than the single Cartesian product of large sets.

MSPCCA employs four optimization techniques:

1. Spoilers removal in a separate algorithm branch.
2. Division of rule set into subsets.
3. Prefix coloring and subsequent filtering.
4. Perfect hash function construction.

To combine all four techniques into a single algorithm, their ordering must be specified. While PCCA uses the techniques in the order 1, 3, 4, MSPCCA inserts the division of rule sets as the second step.

Because MSCA contains several rule sets, matching single packet in all of them would slow down the processing. MSCA therefore uses additional set membership query implemented by the Bloom filters [2] to filter out unnecessary rule table accesses. This filtering is also included in MSPCCA. It is advantageous to use Bloom filters *after* the color filtering, because less items must be stored in Bloom filters in that phase of the algorithm.

The classification engine has the final structure shown in Figure 11. (Spoilers branch is not shown.) After the LPM is computed over all fields, Color Processing blocks filter out impossible prefix combinations. A Bloom filter for each subset is then queried for the presence of the prefix combination in the respective subset. Only in the case of positive result the Perfect Hash Function is computed to obtain the pointer to the rule table. The packet is then matched to the selected rule. In case of match, the selected rule is the output, otherwise the default rule is applied. In parallel to the main algorithm there is a separate branch for classification of spoilers. The end of the algorithm performs simple priority resolution between the two branches.

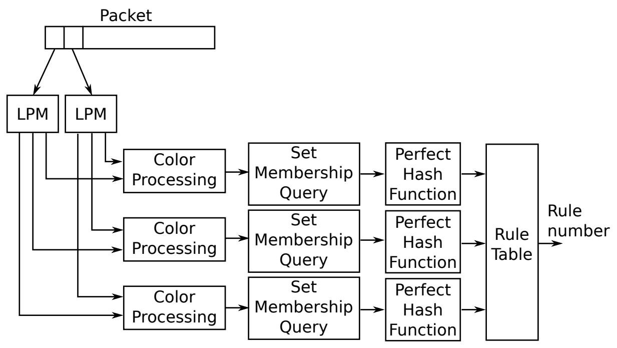

Figure 11: Structure of algorithm combination.

In parallel pipelined hardware implementation, several further optimizations are possible, compared to Figure 11. Supposing that MSCA is able to avoid match of single packet in multiple subsets, only one instance of the Perfect Hash Function logic is needed. The Vertex Table used by Perfect Hash Function is separate for each subset. The Vertex Table is also the only part of the algorithm that is stored in the external memory. The Perfect Hash Function reads two 16-bit integers from the external memory for each packet.

Similar to previous algorithms, MSPCCA works in two phases – precomputation and the classification itself. The precomputation phase is composed of separate methods as follows: After removing spoilers, MSCA splitting algorithm is used to split the rule set into subsets. Then for each subset, PCCA coloring algorithm is used to assign colors to prefixes and to create the reduced set of pseudorules. The Bloom filters are then filled with the rules and pseudorules of each subset. The last precomputation step is building the perfect hash function for each subset.

The contribution of MSPCCA is the fact that all four optimization techniques are successfully integrated into single algorithm. Spoilers removal, division into subsets and Color Processing contribute to lower the number of pseudorules, while the Perfect Hash Function avoids storing the pseudorules. Therefore it is expected that the new algorithm will require less memory than its predecessors.

7. Results

This section provides overall results of all four presented algorithms. The algorithms are compared to the MSCA, which takes similar approach and therefore can be used for a fair comparison. MSCA is also a candidate for 100 Gb/s solution.

Six rule sets are used for experiments in this paper. Four of them are real-life rule sets from university network firewalls (rules1-4) and two are synthetic rule sets generated by ClassBench [17] tool (synth1-2). Numbers of rules and numbers of unique prefixes in each dimension are shown in Table. 2.

Rule set	Rules	Src IPs	Dst IPs	Proto	Src Ports	Dst Ports
synth1	219	55	53	1	14	1
synth2	394	65	57	1	14	1
rules1	103	28	48	4	6	40
rules2	173	84	84	3	1	16
rules3	275	46	64	3	1	22
rules4	1 107	158	80	4	1	56

Table 2: Basic properties of rule sets.

7.1 Throughput

The processing time for each packet is split into several steps: The first step is the LPM operation in all dimensions. The time of tree-based LPM algorithms is linear with the size of the dimension. However, the size of each dimension is always known in advance and almost never changes. For example the LPM processing IPv6 address must match at most 128 bits. Therefore, the LPM may be considered as operation with constant time in the scope of packet classification. Moreover, approaches running in truly constant time were already published [5].

The FPGA implementation of spoilers TCAM memory gives one result per clock cycle [3]. The remaining steps of the PHCA are the perfect hash function, one rule table access and one match of the original packet with the rule from the rule table. All of these operations run in constant time and therefore, the PHCA runs in constant time.

PFCA adds the prefix filtering step, which can be implemented as a simple pipelined hardware. Implementation of the PCCA Color Processing Stage in Virtex-6 FPGA logic consumes 1364 LUT-FlipFlop pairs, and can run at 262 MHz (after synthesis for 5 dimensions and 8 colors) [8]. It only adds four cycles of latency. The Bloom filters in MSPCCA were shown to have very good throughput in hardware implementation [6, 5, 10]. The throughput of all three improved algorithms is the same as PHCA.

All algorithms are designed for implementation in FPGA or ASIC. The LPM computation can run in parallel in each dimension, because LPMs are independent. The algorithms are designed as a pipeline of simple steps. There are no complex mathematical operations such as division, which is particularly hard to implement in FPGA.

Supposed that all logic can be implemented on-chip to achieve any required throughput (possibly even replicated to achieve it), the overall throughput of the algorithms is determined by the off-chip communication. Only the Vertex Table is too large to fit on the FPGA chip or into ASIC, and is proposed to be stored in the external SRAM. The perfect hash function evaluation requires to read and add two integers from the Vertex Table for each packet. The width of integers must be enough to store the rule number. For example, memory width of 16 bits will support up to 65536 rules. The throughput with commodity FPGA and SRAM is 266 million packets per second (provided that RLDRAM2 running at 533 MHz [1] is used as external SRAM to store the Vertex Table and there is no other obstacle in the algorithm performance). This can be compared to 150 million packets per second throughput of 100 Gbps Ethernet for the shortest 64 B frames.

7.2 Memory

Table 3 shows the overall amount of memory for different algorithms and rule sets. The memory for LPM operation is not included, because all algorithms share this step. However, MSCA, PCCA and MSPCCA algorithms add some extra memory to LPM, compared to the other algorithms. This additional memory is included in table 3. Eight spoilers are removed in each algorithm. For MSCA and MSPCCA three subsets are used and the probability of Bloom filters' false positive is set to 0.005. Eight prefix colors in each dimension are used for the PCCA and MSPCCA.

Rule set	MSCA	PHCA	PFCA	PCCA	MSPCCA
synth1	1 622	553	365	227	122
synth2	2 928	874	682	601	232
rules1	2 303	21 362	4 403	3 347	82
rules2	162	5 983	5 310	1 075	81
rules3	211	865	26	260	122
rules4	898	1 306	169	814	568

Table 3: Memory size of the Rule Search Stage (kbits) for algorithms.

MSCA performs well on real-life rule sets rules2 and rules3, but has the worst results from all compared algorithms for synthetic rule sets. PHCA has poor results for rules1 and rules2 and these rule sets are hard also for PFCA and PCCA. PFCA perform very well on rules3 and rules4, while all other rule sets consume the least memory in the MSPCCA. Rule set rules1 is hard for both MSCA and PCCA, while the combination of both – MSPCCA handles this rule set very well.

Table 3 however does not reflect the throughput of the algorithms. While the MSCA must fetch whole rule from the external memory to classify a packet, other algorithms use external memory only to compute the perfect hash function and are therefore better suited for high speed networks. The original paper describing MSCA [6] claims the throughput of 38 million packets per second with the rule table stored in one 300 MHz SRAM. If we consider the same RLDRAM running at 533 MHz as used in other calculations in this paper, we can suppose that MSCA achieves the throughput of 67 million packets per second. This is well under 266 million packets per second for PHCA, PFCA, PCCA and MSPCCA.

We introduce the *memory to speed ratio* that takes the speed into account. The size of required memory is divided by the number of rules and the algorithm throughput to get the *bits per rule and million packets per second.* Table 4 compares the algorithms using this metric. PCCA has the lowest average memory to speed index and also its worst result (rules1) is the lowest of all measured algorithms.

PHCA is the first algorithm that achieves the 100 Gb/s throughput, but at the cost of inefficient memory utilization, compared to older MSCA. PFCA maintains the throughput of PHCA and reduces the memory signifi-

Rule set	MSCA	PHCA	PFCA	PCCA	MSPCCA
synth1	110.60	9.49	6.27	4.76	2.11
synth2	110.93	8.34	6.50	5.73	2.21
rules1	333.83	779.70	160.73	122.18	3.02
rules2	13.99	130.03	115.40	23.38	1.77
rules3	11.48	10.57	0.36	3.55	1.67
rules4	10.78	3.94	0.51	2.46	1.71
average	98.60	157.01	48.30	27.01	2.08

Table 4: Memory to speed index.

cantly. PCCA reduces memory consumption with better results than PFCA in most cases. MSPCCA combines MSCA and PCCA to achieve high throughput and to improve the memory to speed ratio by the order of magnitude in average.

8. Conclusion

This paper deals with packet classification, which is integral part of many networking applications, most notably firewalls. Aim of the paper is to design new algorithm that is applicable for 100 Gb/s networks and above. FPGAs and ASICs are considered as the target technology for the algorithm implementation.

All previous algorithms fail to achieve high throughput without the use of specialized and expensive TCAMs. Furthermore, most known algorithms suffer with the issue of non-deterministic throughput. Their worst case throughput may be significantly worse than in the average case. The family of decomposition methods appears to be the best candidate for the 100 Gb/s solution.

Analysis of several real life and synthetic rule sets gives the starting point for designing the new algorithm. The LPM stage, created by the problem decomposition, does not seem to be the biggest issue, because rule sets often contain only limited amount of distinct prefixes. More important is the efficient handling of LPM results - the Rule Mapping stage. After considering the ordinary hash function to map the LPM results into the rule table, a smarter approach is chosen. The perfect hash function construction algorithm is used to create direct mapping. The resulting function employs collision whenever it is suitable.

The new Perfect Hashing Crossproduct Algorithm [14] achieves the throughput of 266 million packets per second, which is well above the throughput of 100 Gb/s Ethernet for the shortest packets in one direction. This throughput is maintained under all circumstances, regardless the rule set size and the properties of the incoming packets. The high throughput is achieved also by effective mapping of the algorithm to hardware. Almost all parts of the algorithm are designed to fit into the FPGA or ASIC, where extremely high throughput can be achieved. The only access to the external memory is the perfect hash function evaluation and it takes exactly two narrow integer reads to classify each packet. This way the scarce external memory bandwidth is saved.

While the PHCA achieves very high throughput, its memory requirements are rather high. The Prefix Filtering Classification Algorithm [8] improves the memory efficiency with the throughput unchanged. The algorithm is based on the empirical observation that many rules define conditions only for several packet header fields, leaving other fields with the ANY value. These rules however significantly contribute to the final memory size of PHCA. By finding and applying generalization rules to the LPM results, the perfect hash table size is reduced by up to 94.9 %.

The disadvantage of PFCA is its low stability. The memory reduction is only 11 % for one of the used rule sets. The Prefix Coloring Classification Algorithm [13] is more general approach, which is not limited to one specific property of the rule set. After assigning colors to prefixes, combinations of nonsense colors are found and avoided. This optimization lowers the size of the perfect hash table by an order of magnitude for most rule sets, while the throughput is still not affected – it is the same as in the original PHCA.

PCCA also brings several open questions. The most important question is whether there exists an optimal coloring strategy which always finds the best possible coloring in better than exponential time. The performed experiments show that multiple runs of the algorithm with random coloring give results similar to the normal distribution. Given the fact that all inputs to the algorithm are discrete and there is no concept of erroneous or random behavior (except for the coloring) which is often source of the normal distribution, it appears that some very complex behavior emerges in the algorithm. This behavior is yet to be fully understood.

Final algorithm is the MSPCCA, which combines PCCA with older MSCA. It takes the idea of dividing the rule set into several independent subsets from the MSCA. This technique brings significant reduction of memory. The average memory size of MSPCCA is by an order of magnitude smaller than in other algorithms. What is probably even more important is that MSPCCA shows high stability, keeping total memory requirements between 81 and 568 kbit for all available rule sets.

Throughput of all presented algorithms is 266 million packets per second, which corresponds to 178 Gb/s for the shortest 64 B packets and 548 Gb/s for the 440 B packets (reported as average in [11]).

References

[1] Rldram part catalog. Micron Technology, Inc.

[2] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970.

[3] J.-L. Brelet. An overview of multiple cam designs in virtex family devices. Xilinx Application Note 201, Xilinx Inc., 1999.

[4] Z. J. Czech, G. Havas, and B. S. Majewski. An optimal algorithm for generating minimal perfect hash functions. *Information Processing Letters*, 43(5):257–264, 1992.

[5] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor. Longest prefix matching using Bloom filters. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 201–212, New York, NY, USA, 2003. ACM.

[6] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood. Fast packet classification using Bloom filters. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 61–70, New York, NY, USA, 2006. ACM.

[7] W. Eatherton, G. Varghese, and Z. Dittia. Tree bitmap: hardware/software IP lookups with incremental updates. *SIGCOMM Computer Communication Review*, 34(2):97–122, 2004.

[8] J. Koženek, V. Puš, and J. Blaho. Memory optimization for packet classification algorithms. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, Association for Computing Machinery, pages 165–166. Association for Computing Machinery, 2009.

[9] H. Lee, W. Jiang, and V. K. Prasanna. Scalable High-Throughput SRAM-Based Architecture for IP Lookup Using FPGA. In *FPL '08*. IEEE, 2008.

[10] Y. Lu, B. Prabhakar, and F. Bonomi. Perfect hashing for network applications. *Information Theory, 2006 IEEE International Symposium on*, pages 2774–2778, July 2006.

[11] S. McCreary and K. Claffy. Trends in wide area IP traffic patterns - A view from Ames Internet Exchange. In *ITC Specialist Seminar*, Monterey, CA, Sep 2000.

[12] M. H. Overmars and A. F. van der Stappen. Range searching and point location among fat objects. *J. Algorithms*, 21(3):629–656, 1996.

[13] V. Puš, M. Kajan, and J. Kořenek. Hardware architecture for packet classification with prefix coloring. In *IEEE Design and Diagnostics of Electronic Circuits and Systems DDECS'2011*, pages 231–236. IEEE Computer Society, 2011.

[14] V. Puš and J. Kořenek. Fast and scalable packet classification using perfect hash functions. In *FPGA '09: Proceedings of the 17th international ACM/SIGDA symposium on Field programmable gate arrays*, New York, NY, USA, 2009. ACM.

[15] H. Song, J. Turner, and J. Lockwood. Shape shifting tries for faster ip route lookup. In *ICNP '05: Proceedings of the 13TH IEEE International Conference on Network Protocols*, pages 358–367, Washington, DC, USA, 2005. IEEE Computer Society.

[16] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. *SIGCOMM Comput. Commun. Rev.*, 28(4):191–202, 1998.

[17] D. E. Taylor and J. S. Turner. Classbench: a packet classification benchmark. *IEEE/ACM Trans. Netw.*, 15(3):499–511, 2007.