BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

ADDRESSING ISSUES IN RESEARCH ON PACKET CLASSIFICATION IN CORE NETWORKS

ŘEŠENÍ PROBLÉMŮ VE VÝZKUMU KLASIFIKACE PAKETŮ V PÁTEŘNÍCH SÍTÍCH

EXTENDED ABSTRACT OF PHD THESIS
ROZŠÍŘENÝ ABSTRAKT DISERTAČNÍ PRÁCE

AUTHOR — **Ing. JIŘÍ MATOUŠEK**
AUTOR PRÁCE

SUPERVISOR — **Ing. JAN KOŘENEK, Ph.D.**
ŠKOLITEL

BRNO 2018

Abstract

Although the Internet has changed significantly since the beginning of the 21st century, packet classification is still one of the most common operations implemented in networking devices. Nevertheless, the requirements on its performance are continuously increasing, especially in core networks. Currently, packet classification algorithms have to support 100 Gbps throughput. In addition, classification rule sets are becoming larger and the number of bits involved in the classification decision is growing due to 128-bit IPv6 addresses and classification according to more than 5 header fields in the OpenFlow protocol. Therefore, the majority of contemporary research on packet classification in core networks address the performace of packet classification algorithms, which has to keep pace with continuously increasing requirements. However, the researchers also focus on benchmarking newly developed algorithms because they have to be benchmarked using real rule sets, but such data are not available for most of the packet classification use cases. This thesis deals with both of these issues because it is important not only to design packet classification algorithms having high performance but also to assess their parameters by benchmarking based on proper data sets.

Regarding the performace of packet classification algorithms, this thesis focuses on improving prefix matching, which is used in the majority of 1-dimensional and also multi-dimensional algorithms. Since a software implementation of prefix matching cannot fulfill the requirements imposed on packet classification in core networks, the thesis proposes a novel pipelined prefix matching architecture that targets Xilinx FPGA chips and utilizes their distributed on-chip memory. To fit the whole prefix matching data structure into FPGA's on-chip memory, this thesis also proposes a memory-efficient trie-based representation of a prefix set. The proposed representation is more memory efficient than well-known multibit tries Tree Bitmap and Shape Shifting Trie and for IPv4 prefix sets it also significantly overcomes the Prefix Partitioning lookup algorithm. The architecture then comprises two independent processing pipelines (to utilize both ports of on-chip memory blocks) that are together able to perform almost 255 million lookups per second, which translates into throughput of 170 Gbps for the shortest Ethernet frames.

To allow realistic packet classification algorithms benchmarking, the thesis introduces a new open source synthetic rule set generator called ClassBench-ng, which integrates the generation of IPv4, IPv6, and OpenFlow 1.0.0 classification rule sets following the statistical properties specified in an input seed. Apart from the rule set generation, ClassBench-ng also supports an analysis of a real rule set in the `ovs-ofctl` format producing a corresponding seed that may be used for the generation of a similar synthetic rule set later on. Therefore, researchers having access to real classification rule sets can share their benchmarking data with other members of the community via statistical-based (thus anonymous) seeds produced by ClassBench-ng. With respect to the precision of the rule set generation process, ClassBench-ng is better than original ClassBench and FRuG in case of IPv4 prefixes and than Non-random Generator in case of IPv6 prefixes, when considering an average score for all IP prefix-related parameters. Moreover, it also clearly outperforms FRuG in the precision of OpenFlow rule sets generation.

Contents

1 Introduction **2**
1.1 Thesis Goals 3

2 Packet Classification in Core Networks **4**
2.1 Approaches to Packet Classification 4
2.2 Research Issues 7
2.2.1 Performance of Algorithms 7
2.2.2 Algorithms Benchmarking 8

3 Related Work **9**
3.1 Packet Classification Algorithms 9
3.2 Rule Set Generators 11

4 Addressing Performance of Algorithms **14**
4.1 Proposed Prefix Set Representation 15
4.2 Proposed Hardware Architecture 16
4.3 Experimental Evaluation 18

5 Addressing Algorithms Benchmarking **20**
5.1 ClassBench-ng: Next Generation ClassBench 20
5.2 Experimental Evaluation 23

6 Conclusions **27**

Bibliography **29**

Curriculum Vitae **32**

Chapter 1

Introduction

Computer networks are an indispensable part of our everyday life. We use them as a source of information and a means for communication at work as well as in our free time. The most prominent computer network today is a global network called the Internet. It evolved from research networks during 1970s and 1980s and established as a publicly available global network in 1990s. However, since the beginning of 21st century we have experienced rapid technology development that has significantly changed the Internet. The current Internet is no more the one we knew at the beginning of this century.

While the highest standardized transfer rate for Ethernet was 1 Gbps in 2000 [3], the current operational maximum is 100 Gbps [6]. In addition, since the standard for 400 Gbps Ethernet [9] has been approved in December 2017, the upgrade of the maximum transfer rate in core networks can be expected in the near future. On the other hand, the 802.11 family of standards [8] and the IMT-2000 standard [18] together started the advent of mobile-broadband subscriptions. According to ITU (*International Telecommunication Union*) data [17], in 2016 there were 52.2 active mobile-broadband subscriptions per 100 inhabitants worldwide.

Technology development made access to the Internet more affordable, even in developing countries. The number of individuals using the Internet grew from 495 million in 2001 to 3385 million in 2016 [17]. Moreover, users often own several devices that are able to access the Internet (e.g., personal computer, tablet, smartphone, smart TV, or intelligent sensor). This resulted in 17.1 billion of connected devices (i.e., 2.3 devices per capita) in 2016 and forecasted growth to 27.1 billion (3.5 per capita) in 2021, as reported by Cisco [10]. Because the maximum number of distinct IPv4 (*Internet Protocol version 4*) addresses is less than 4.3 billion, all RIRs (*Regional Internet Registries*) except AFRINIC (*AFRIcan Network Information Center*) exhausted their IPv4 allotments between 2011 and 2015 [10]. Therefore, since 2011 we have experienced accelerated adoption of IPv6 [15], the successor of IPv4. However, not only the principal protocol of the Internet is changing. Architecture of computer networks is being redefined as well, especially due to the concept of network virtualization. Although there are numerous, often competing, technologies enabling this concept at various levels of network architecture [19], one of the most promising approaches is SDN (*Software-Defined Networking*), which regained interest after the introduction of OpenFlow [25], currently the most common protocol for communication between control and data planes of a switch.

Despite all the changes of the Internet were brought to life by upgrading its infrastructure, packet classification at physical link speed is still one of the most common operations implemented in networking devices. Upon arrival, a networking device classifies every

packet according to one or more of its header fields and uses the result of classification for further processing of the packet. Depending on the function of a device, the result of classification may be used for basic forwarding operation, to apply security policies, application-specific processing, or QoS (*Quality of Service*) guarantees.

Even though packet classification has not been replaced by another functionality, requirements on its performance are continuously increased. For instance, because of growth of transfer rates to 100 Gbps and extremely high utilization of the IPv4 address space, a core router has to be able to make a forwarding decision according to a forwarding table containing more than 680 thousand IPv4 prefixes [1] every 6.72 ns. With respect to the IPv6 protocol, due to 4-times longer IP address involved in the forwarding process, the situation is even worse. Currently, the number of IPv6 prefixes in a forwarding table of core routers is almost 43 thousand [1]; however, this number is expected to grow together with IPv6 penetration. Another example of growing demands on packet classification is extending the set of packet header fields involved in the classification process. While the most common set of header fields involved in packet classification consists of 5 header fields, the OpenFlow protocol initially extended this set to 12 fields [4] and the latest version of the OpenFlow protocol defines packet classification based on 45 fields [7]. Both these examples demonstrate that packet classification requires continuous attention of researchers.

From a high level perspective, there can be identified two issues that are addressed in the research on packet classification in core networks.

The first issue is related to the performance of packet classification algorithms, which has to keep pace with growing transfer rates. Parameters that have the greatest influence on the performance are the number of bits involved in packet classification (i.e., the number of utilized header fields and their length) together with the number of utilized classification rules. Therefore, new algorithms have to deal with growing popularity of SDN-based network virtualization utilizing the OpenFlow protocol (more fields) [19] and/or increasing IPv6 penetration (longer fields) [15]. Moreover, since current CPUs (*Central Processing Units*) do not provide enough performance for even 1-dimensional packet classification according to destination IP address (i.e., IP lookup) on a 100 Gbps link, packet classification algorithms targeting core networks have to be accelerated in hardware.

The second issue is related to benchmarking packet classification algorithms, which are continuously improved to meet ever-increasing requirements on their performance. Because real classification rule sets are not usually available for benchmarking, researchers designed and implemented several tools capable of generating synthetic rule sets [36, 37, 34, 32, 14]. Nevertheless, even together these tools are not able to generate all data sets necessary for benchmarking current packet classification algorithms. In addition, it can be shown that the process of rule set generation in the currently available tools is not always accurate.

1.1 Thesis Goals

This thesis aims to address identified issues in the research on packet classification in core networks via achieving the following two goals.

The **first goal** is to address the issue related to the performance of packet classification algorithms by designing a hardware-accelerated prefix matching algorithm that will be able to achieve 100 Gbps throughput for both IPv4 and IPv6 protocols.

The **second goal** is to address the issue related to benchmarking new packet classification algorithms using a tool capable of generating synthetic IPv4, IPv6, and OpenFlow 1.0.0 rule sets with parameters similar to real rule sets.

Chapter 2

Packet Classification in Core Networks

Packet classification is a process determining a class (often referred to as a network flow) that a packet belongs to. The input of packet classification consists of selected header fields extracted from the packet and a set of classification rules with defined priorities, in which each rule represents one class. A classification rule defines a condition for every header field extracted from input packets. The condition is usually specified in one of the following four ways: (1) value—exactly one allowed value, (2) prefix—a range of allowed values having a common binary prefix, (3) range—an arbitrary range of allowed values, and (4) wildcard—any value is allowed.

Regardless the utilized specification, a condition is satisfied when the corresponding header field of a packet contains one of the allowed values. If all conditions of a classification rule are satisfied, then the packet belongs to the corresponding class. Note that classes may overlap, thus multiple classification rules can match the packet. In such a case the matching rule with the highest priority is selected as the output of packet classification. Since packet classes usually define specific processing for their packets, the output of packet classification can also be an action that is going to be applied on the classified packet.

2.1 Approaches to Packet Classification

The problem of packet classification has been approached in many different ways that are described in this section. Apart from publications cited in particular subsections, this description is based on information gathered from [26, 33, 5].

Naive Approaches

The simplest approach to packet classification is a linear search of a rule set with rules sorted from the highest to the lowest priority. An input packet is sequentially matched against classification rules and the first matching rule is selected as the output of packet classification. Both search time and memory requirements of this approach are linear with respect to the number of rules. Because of its search performance, this approach is feasible only for small rule sets.

An orthogonal approach trades-off memory requirements for better search time by pre-computing the best matching rule for every possible packet and storing this information into a table. Classification of an incoming packet then consists of addressing the table by

concatenated header field values and reading the best matching rule information. Search time of this second naive approach is constant (classification is done in a single step) but its memory requirements are exponentially dependent on the number of bits involved in the classification process. Therefore, despite its excellent search time, this approach to packet classification is important mainly from a theoretical point of view.

TCAM

Another straightforward approach to packet classification is to use TCAM (*Ternary Content-Addressable Memory*). Since TCAM supports addressing by content and ternary matching, it can be viewed as the second naive approach with the ability to use the prefix specification. Indeed, each TCAM record (i.e., a rule after the conversion of all conditions to prefixes augmented by the X value for each unspecified bit) represents all entries of the second naive approach's table that correspond to the packets matching the record.

Constant search time and reasonable memory requirements are arguments behind the extensive use of TCAM in commercial devices. Nevertheless, this approach suffers from several non-negligible issues. Parallel matching, which allows constant search time, leads to high power consumption of this kind of memory. Because of supporting ternary matching, its cost per bit is also higher than of other memories. Moreover, if TCAM is used for packet classification, its capacity is not utilized efficiently due to rules replication during the range-to-prefix conversion. Last but not least, the need for storing records of width equal to the number of bits involved in packet classification limits scalability of this approach to classification according to more and/or longer header fields. All these disadvantages motivate research in algorithmic solutions to packet classification.

Representation Using Tuples

The representation of the packet classification problem using tuples has been introduced by Srinivasan, Suri, and Varghese in [31]. It is the first out of three seminal approaches to multi-dimensional packet classification described in the thesis. In this approach, each classification rule is represented by a tuple, whose elements define the number of bits used for specification of the corresponding rule's conditions. Such a representation is motivated by the observation that real rule sets contain only a few combinations of specification lengths. Therefore, the number of distinct tuples representing a rule set is expected to be much lower than the actual number of rules.

Classification of an incoming packet using a rule set represented by tuples is done by a (possibly parallel) linear search of a tuple set. Each tuple represents a subset of the original rules that is searched for a matching rule using only a limited amount of information from packet's header fields. The search for a matching rule within the tuple is thus the exact matching problem, which can be for example solved using a hashing table.

Geometric Representation

Another representation of the general packet classification problem is based on multi-dimensional space where each dimension corresponds to one header field utilized in packet classification. In this space, each condition of a classification rule can be represented as an interval in the corresponding dimension, thus the rule is equal to a multi-dimensional rectangle defined by intervals corresponding to its conditions.

Since a packet contains exact values in its header fields, it is represented as a point in the multi-dimensional space. A rule matching an input packet is then represented by a rectangle that contains a point of the multi-dimensional space that corresponds to the packet.

Combinatorial Representation

The last seminal approach to multi-dimensional packet classification described in this thesis is based on viewing a classification rule as a combination of the given number of conditions. In order to speed up a classification process, this approach builds a Cartesian product of sets of conditions utilized in particular dimensions. The entries of the Cartesian product correspond to all possible combinations of matching results for individual dimensions and each entry has the best matching rule associated to it. This is similar to the second naive approach to packet classification. However, the condition sets of particular dimensions of a real rule set are usually significantly smaller than the number of rules in the rule set [16]. Therefore, the number of Cartesian product entries is much smaller than the number of all possible packets.

Packet classification based on combinatorial representation of the rule set fully utilizes properties of the constructed Cartesian product. First of all, matching conditions for individual dimensions are determined, possibly using different 1-dimensional packet classification approaches in each dimension. The matching results from individual dimensions are then combined together and the best matching rule associated to a corresponding Cartesian product's entry is selected as the result of the classification process.

Range Matching

Apart from approaches to multi-dimensional packet classification, there are also specific approaches to 1-dimensional packet classification, which reflect different condition specifications utilized in classification rules. The seminal range matching approach to 1-dimensional packet classification has been introduced by Lakshman and Stiliadis in [20]. It is based on non-overlapping intervals (often referred to as elementary intervals), which are created by dividing the full range by start and end points of each range utilized in a rule set.

In order to allow packet classification, each elementary interval has to store a list of classification rules, whose range overlaps with the interval. Packet classification then consists of the search for an elementary interval covering the header field value extracted from a packet, followed by returning the list of overlapping (i.e., matching) rules.

Prefix Matching

Similarly to range matching, prefix matching is beneficial especially when classification rules contain the prefix specification. However, the prefix specification is also often used as a uniform way of specifying a condition in a classification rule.

The most utilized prefix matching approach to 1-dimensional packet classification is called LPM (*Longest Prefix Matching*). This approach looks for all prefixes matching the input value and returns the longest one (i.e., the most specific one) as the best matching prefix. Some packet classification algorithms based on combinatorial representation might require prefix matching that returns all matching prefixes instead of just the longest one. In such a case, LPM can be easily modified to satisfy this requirement by omitting its last step.

2.2 Research Issues

Current research on packet classification techniques for core networks faces two issues. The first one can be characterized as *performance of packet classification algorithms*. Clearly, the performance of classification algorithms has to be increased to meet ever-increasing requirements on packet classification in core networks. The second issue rises from the first one and it revolves around *packet classification algorithms benchmarking*. As new classification algorithms are designed, it is necessary to benchmark their performance and compare their properties to each other.

2.2.1 Performance of Algorithms

Even though the Internet is still growing and accelerating, packet classification stays one of the most common operations implemented in networking devices. Nevertheless, the changes of the Internet increase requirements on the performance of packet classification algorithms, which has to keep pace with growing transfer rates. The standard for 400 Gigabit Ethernet [9] has been approved in December 2017, but current core networks widely support 100 Gigabit Ethernet defined in [6]. Therefore, packet classification algorithms targeted at core networks are required to support throughput of 100 Gbps.

Considering the shortest Ethernet frame, the maximum packet rate of 100 Gigabit Ethernet is approximately 148.81 MPPS (*Million Packets Per Second*). Thus, packet classification algorithms have to be able to provide a classification result every 6.72 ns. To achieve such matching performance, their designers have to deal with growing requirements on selected parameters, which include the number of bits involved in packet classification (i.e., the number of utilized header fields and their length) and the number of specified classification rules.

The number of bits involved in packet classification depends on a specific use case. However, in general it is increasing because nowadays more as well as longer header fields are being utilized in packet classification. Extensions of the set of utilized header fields are closely related to growing interest towards network virtualization, which may be realized, for instance, by OpenFlow-based SDN [19]. Each version of the OpenFlow protocol incrementally extended the set and its latest version has defined packet classification according to 45 header fields [7]. On the other hand, longer header fields are mainly due to increasing IPv6 penetration [15].

The number of specified classification rules varies even for different instances of the same use case. In case of IP routing, forwarding tables of core routers currently contain 680 thousand IPv4 and 43 thousand IPv6 prefixes [1] but these numbers continuously grow as the allocation of prefixes from IPv4 and IPv6 address spaces progresses. The situation is different with firewalls, in which the actual number of installed classification rules depends on security policies of a particular network. Since firewall rule sets are not usually publicly available (because of security reasons), the researchers evaluating 5-tuple-based classification algorithms often use synthetic data sets consisting of thousands or tens of thousands of classification rules [35, 30]. Focusing on SDN-enabled switches in a datacenter, each OpenFlow rule may correspond to an active virtual machine, thus their number may be in the order of tens of thousands [22].

In summary, packet classification algorithms targeting core networks have to be able to classify an incoming packet according to tens or hundreds of bits into tens or hundreds of thousands of classes and provide a new classification result every 6.72 ns. Such

requirements on performance prohibit software implementation of packet classification algorithms [11]. Although hardware implementation can be realized using both an ASIC (*Application-Specific Integrated Circuit*) or an FPGA (*Field-Programmable Gate Array*), because of its availability, flexibility, and configurability, FPGA-based implementation will be considered further in this thesis.

2.2.2 Algorithms Benchmarking

The requirements on the performance of packet classification algorithms are continuously increasing, as demonstrated in the previous section. Therefore, the development of classification algorithms that meet these requirements is a never-ending story. To benchmark a new classification algorithm, benchmarking tools (e.g., Netbench [27]) usually assess the following parameters of the algorithm: (1) classification speed defined as the number of memory accesses required for the classification of a single packet and (2) the memory requirements of a data structure representing a set of classification rules. In addition, the speed of updates of a classification rule set representation might also be assessed in the course of algorithm's benchmarking.

The practical implementation of a packet classification algorithm has to consider the worst case performance of the algorithm in terms of the above mentioned parameters, but the worst case performance solely depends on a utilized classification rule set. This means that in the ideal case a real rule set is utilized while benchmarking the classification algorithm. Nevertheless, real rule sets are not publicly available for the majority of packet classification use cases (often because of security reasons). One of the few exceptions to this are IPv4/IPv6 prefix sets from forwarding tables of core routers [28], which can be used for benchmarking IP lookup (i.e., the key part of IP routing). However, since IPv6 penetration is expected to grow exponentially in the future, current IPv6 prefix sets cannot be directly used for benchmarking IP lookup algorithms in the future environment.

The researchers addressed the issue of missing real benchmarking data in various ways. While a limited number of research groups obtained access to real rule sets via NDAs (*Non-Disclosure Agreements*), others developed several tools capable of generating synthetic data sets for common packet classification use cases. Because the size of IPv4 forwarding tables is not expected to grow significantly in the future, the tools for generating IP prefix sets focus on IPv6 prefix sets only. To get this kind of benchmarking data, it is possible to use for instance Non-random Generator [36] or V6Gene [37]. On the other hand, in the area of 5-tuple rules it makes sense to generate both IPv4 and IPv6 5-tuples. The former is almost exclusively generated using ClassBench [34], while the latter can be generated using ClassBenchv6 [32]. The most problematic is the situation with OpenFlow rules generators. Although FRuG (*Flexible Rule Generator*) [14] is able to generate a set of rules that specify a condition for an arbitrary number of header fields, it does not explicitly consider any specifics of OpenFlow rule sets.

Even though the existing generators are capable of producing IPv4/IPv6 prefixes and 5-tuples as well as rules specifying a condition for more than 5 header fields, none of them explicitly provides support for OpenFlow rules generation. In addition, none of the tools is able to generate all types of benchmarking data, which complicates the situation in case of benchmarking packet classification algorithms targeted at various use cases. Moreover, it can be shown that the process of rule set generation in available generators is not always accurate. These issues show that packet classification benchmarking is still an open problem.

Chapter 3

Related Work

While the previous chapter introduced the packet classification operation along with various approaches to its implementation, this chapter contains the selection of already implemented packet classification algorithms and synthetic rule set generators. These algorithms and generators represent current solutions to the performance of packet classification algorithms issue and the packet classification algorithms benchmarking issue, respectively, which were also discussed in the previous chapter. Since the goal of the thesis is to address these issues as well, the algorithms and generators presented in this chapter serve as a starting point for work described in this thesis and they will be used as a baseline for the evaluation of proposed solutions to the identified issues in research on packet classification in core networks.

3.1 Packet Classification Algorithms

This section describes existing packet classification algorithms that implement the prefix matching approach to packet classification. The reasons for including prefix matching algorithms only are that the prefix specification is utilized in IP lookup (i.e., probably the most common type of packet classification in core networks) and it is also commonly used as a uniform way of specifying all conditions of a classification rule. Therefore, prefix matching is utilized not only in 1-dimensional packet classification, but also in the majority of multi-dimensional classification algorithms. As prefix matching algorithms usually encode a prefix set using a binary prefix tree called trie, this section presents selected trie-based prefix matching algorithms.

Trie

The trie data structure, which has been proposed by Fredkin [13], represents the basis of prefix set encoding in the majority of prefix matching algorithms. It is a binary prefix tree in which each node represents a prefix. The root node represents the empty prefix and left and right child nodes of any trie node represent prefixes created from their parent's prefix by appending 0 and 1, respectively. Trie nodes representing prefixes from a prefix set are called prefix nodes, while other trie nodes are referred to as place holder nodes.

Matching prefixes of a prefix set represented by a trie is done by traversing the trie from the root to the leaves according to the bits of an input value (e.g., a destination address in case of IP lookup) taken from the MSB (*Most Significant Bit*) to LSB (*Least Significant Bit*). All prefix nodes visited during such a traversal represent prefixes matching the input

value and the last visited prefix node represents the longest matching prefix, which is the output of the LPM operation.

In order to diminish the main disadvantage of a trie (i.e., matching only one input bit in each step), modern trie-based prefix matching algorithms employ the concept of so called multibit trie. These algorithms propose new types of node that represent subtrees of a trie and allow to match multiple bits per step. Thus, although a trie is not directly applicable for high-performance prefix matching, it is a seminal data structure for modern prefix matching algorithms.

Tree Bitmap

Tree Bitmap (TBM), which has been developed by Eatherton, Varghese, and Dittia [12], is one of the best known multibit trie algorithms. It represents a set of prefixes using a 2^{SL}-tree, where parameter SL (stride length) determines the number of input bits matched in each step. This tree is built on the top of a trie by mapping TBM nodes, each of which corresponds to trie's subtree of the maximum depth equal to SL, onto the trie in a non-overlapping fashion such that each trie node is covered by a TBM node.

Each TBM node is encoded using two bitmaps and two pointers. The external bitmap contains 2^{SL} bits determining whether a corresponding child node is present (value 1) or missing (value 0). Note that MSB of the external bitmap corresponds to the leftmost child, while its LSB corresponds to the rightmost child. On the other hand, $2^{SL} - 1$ bits of the internal bitmap correspond to the nodes of the underlying trie in breadth-first order and each bit encodes information whether a corresponding internal node is a prefix node (value 1) or non-prefix node (value 0). The child and prefix pointers refer to information about child nodes and prefix-related data, respectively.

The compact representation of a TBM node makes possible to retrieve it from a memory in just one clock cycle. Additionally, due to the use of bitmaps for node's encoding, this prefix matching algorithm is easily implementable in hardware. The TBM algorithm is thus able to achieve high matching performance. In addition, the fixed structure of a node allows easy updates of the represented prefix set. However, it may also cause high memory overhead when the underlying trie is sparse. Trade-off between matching performance and memory requirements of the TBM algorithm for a given prefix set can be tuned via parameter SL. Clearly, a higher value of parameter SL results in a lower number of matching steps but higher memory overhead and vice versa.

Shape Shifting Trie

Another important multibit trie algorithm called *Shape Shifting Trie* (SST) has been introduced by Song, Turner, and Lockwood [29]. This algorithm is based on TBM, but it tries to overcome its main drawback by introducing an adaptive shape of a node, which reduces memory overhead when the underlying trie is sparse. Instead of parameter SL, the shape of SST nodes is influenced by parameter K, which determines the maximum number of underlying trie nodes that can be represented by an SST node.

The encoding of SST nodes utilizes external an internal bitmaps as well as child and prefix pointers that have the same function as in TBM nodes. Apart from these bitmaps and pointers, the data structure of an SST node also contains the shape bitmap consisting of $2K$ bits, which encodes the shape of the node. Similarly to the internal bitmap, pairs of shape bitmap's bits correspond to the nodes of the underlying trie in breadth-first order. The first bit of each pair determines whether the SST node contains the left child of the

corresponding node (value 1) or its left child is missing (value 0), while the second bit encodes the same information for the right child of the corresponding node.

The adaptive shape of SST nodes allows to represent a prefix set using a very small amount of memory. However, the variability of nodes makes the construction of the prefix set representation a computationally complex task and it also significantly limits the possibility of prefix set updates. Moreover, the need to decode the shape of a node from its shape bitmap negatively influences matching performance. Because of all these limitations, SST is not a viable option for prefix matching in core networks, which is also illustrated by the fact that there is no hardware architecture implementing this algorithm. Nevertheless, it may be useful as a reference algorithm for an assessment of memory requirements.

Prefix Partitioning Lookup Algorithm

Prefix Partitioning Lookup Algorithm (PPLA) developed by Le and Prasanna [21] is the newest prefix matching algorithm described in the thesis. This algorithm focuses on minimizing the memory footprint of a prefix set representation, while keeping sufficient throughput for 100 Gbps networks. As all other prefix matching algorithms described so far in this section, PPLA represents a prefix set by a trie. However, it utilizes the trie representation just for partitioning the set of prefixes into k disjoint subsets that are used for prefix matching in k separate processing pipelines. Once the prefix set is partitioned, each subset is represented by a separate binary search tree (for minimum memory requirements) or 2-3 tree (for easy prefix set updates) and mapped onto stages of a corresponding pipeline.

Although an FPGA implementation of PPLA is able to perform 410/390 MLPS (*Million Lookups Per Second*) for an IPv4/IPv6 prefix set consisting of over 330 000 unique prefixes, the main advantage of the PPLA algorithm is its high memory efficiency. When prefix subsets are represented by a binary search tree, an average memory efficiency ratio (i.e., the number of bytes of memory needed for storing one byte of a prefix) achieved by the algorithm is 1.00 for IPv4 prefix sets and 0.91 for IPv6 prefix sets. The main drawback of this algorithm is connected with the initial partitioning of a prefix set, which introduces very high preprocessing overhead. Therefore, PPLA may experience the degradation of matching performance and memory efficiency when the prefix set is frequently updated.

3.2 Rule Set Generators

This section focuses on synthetic rule set generators for benchmarking both 1-dimensional and multi-dimensional packet classification, which were already briefly introduced in Section 2.2.2. All 1-dimensional generators implement the generation of IP prefixes (i.e., classification rules for benchmarking IP lookup). More precisely, they are specialized on generating IPv6 prefixes because currently available real IPv6 prefix sets are not suitable for benchmarking as they are expected to significantly grow in size in the future. The situation is different in case of multi-dimensional generators. They primarily support the generation of IPv4 5-tuples, but some of them also allow to generate IPv6 5-tuples or classification rules specifying a condition for more than 5 header fields.

Non-random Generator

Non-random Generator, which has been developed by Wang et al. [36], is the first of two IPv6 prefix set generators described in this thesis. It converts an input IPv4 prefix set to an

output IPv6 prefix set in a way that preserves selected features of the input and generates the output with respect to IPv6 allocation policies.

As each input IPv4 prefix is converted to a single output IPv6 prefix, the size of the output prefix set is directly inherited from the input set. To determine the exact length of an output IPv6 prefix, Non-random Generator doubles the length of an input IPv4 prefix and sometimes adjusts the resulting value (e.g., to get also odd prefix lengths). Finally, the value of the output IPv6 prefix is composed of leading three bits `001` followed by the number of AS (*Autonomous System*) corresponding to the input prefix, the input IPv4 prefix itself, and possibly some randomly generated bits (to get a prefix of the given length).

Since Non-random Generator was the first generator of synthetic IPv6 prefix sets, it was extremely useful at the time of its origin. Nevertheless, prefix sets generated by this tool do not correspond to current real IPv6 prefix sets because their generation is based on sets of IPv4 prefixes. Therefore, newer generators are based on currently available real sets of IPv6 prefixes.

V6Gene

A well-known example of a generator based on real sets of IPv6 prefixes is called V6Gene [37]. This tool has been designed by Zheng and Liu, who aimed to overcome the reasons that prevent real IPv6 prefix sets from being used for benchmarking IPv6 lookup algorithms. V6Gene thus starts from a real prefix set of a small size and systematically enlarges this set by new prefixes generated in a way that simulates the allocation of address prefixes in the real IPv6 world.

The generation phase of the V6Gene's run is divided into two parallel branches, which implement different algorithms introducing new IPv6 prefixes into the input prefix set. While the first branch simulates the allocation of IPv6 prefixes from LIR (*Local Internet Registry*) prefixes that already exist in the prefix set to their subscribers, the second branch allocates IPv6 prefixes from a limited number of newly introduced LIR prefixes.

As the generation process is based on real IPv6 prefix sets in V6Gene, this tool can generate synthetic prefix sets that are closer to real IPv6 prefix sets than the output of Non-random Generator. However, the generator does not explicitly account for prefix aggregations and splits known from real prefix sets. Moreover, although the authors highlighted the availability of their tool, currently there is no publicly available V6Gene's implementation.

ClassBench

ClassBench [34], which has been introduced by Taylor and Turner, is probably the most often used multi-dimensional synthetic rule set generator. The most important component of this tool is the Filter Set Generator that implements the generation of synthetic IPv4 5-tuples according to an input seed (referred to as a parameter file in [34]) and a small set of high-level parameters. The remaining tools then implement the construction of the seed from a real rule set (the Filter Set Analyzer) and the generation of a synthetic header trace, which is able to comprehensively exercise a packet classification algorithm utilizing a given rule set (the Trace Generator).

The Filter Set Generator's input seed contains several statistical distributions that jointly make possible to generate a condition for each header field belonging to an IPv4 5-tuple. In general, the Filter Set Generator implements the generation of individual classification rule's conditions as sampling corresponding distributions stored in the input seed.

However, because of interdependence of some distributions (e.g., the PPC distribution depends on the protocol value and the prefix pair length distribution depends on the selected PPC), sampling has to be done in a specific order. Moreover, generating some conditions of a classification rule is not as straightforward as sampling a distribution, which is true especially for source/destination address prefixes.

A great popularity of ClassBench has been caused not only by the Filter Set Generator's ability to produce realistic IPv4 5-tuples for benchmarking packet classification algorithms, but also by providing a set of twelve input seeds extracted from real classification rule sets of various origin (access control lists, firewalls, IP chains). Since the time of ClassBench's publication, these seeds have become de facto standard inputs for the generation of synthetic data for benchmarking various packet classification algorithms. However, ClassBench is no longer sufficient for current needs of the research community as it focuses on IPv4 5-tuples only and does not support the generation of IPv6 5-tuples or classification rules comprising a condition for more than 5 header fields. These drawbacks have been addressed by more recent multi-dimensional synthetic rule set generators ClassBenchv6 and FRuG.

ClassBenchv6

In order to allow the generation of synthetic classification rules with IPv6 address prefixes, Sun et al. have proposed ClassBenchv6 [32], a reshaped version of ClassBench for the IPv6 world. Similarly to original ClassBench, this tool generates IPv6 5-tuples according to statistical distributions stored in an input seed. However, rather than using a seed extracted from a real set of IPv6 5-tuples, ClassBenchv6 builds on similarities between IPv4 and IPv6 environments (similar allocation policies, retained Internet topology and classification use cases, continuing evolution of the Internet) and predicts an IPv6 seed from an IPv4 seed corresponding to a real rule set.

Since ClassBenchv6 generates IPv6 5-tuples in a similar way as the Non-random Generator produces IPv6 prefixes, it also has similar benefits and drawbacks. The prediction of IPv6 trie-based distributions based on corresponding IPv4 distributions was useful during the initiation phase of IPv6 deployment. Nevertheless, it is no longer valid.

FRuG

Even though both ClassBench and ClassBenchv6 theoretically allow to generate conditions for selected header fields beyond IPv4/IPv6 5-tuple (e.g., IPv4 flags in ClassBench and IPv6 flow label in ClassBenchv6), the full support of rules with a condition for more than 5 header fields was not available until Ganegedara, Jiang, and Prasanna had introduced FRuG [14].

This synthetic rule set generator allows the user to fully control the size and structure of generated rule sets and also to define specific distribution for each included header field, which makes it a powerful benchmark to assess various packet classification algorithms. However, only MAC address and IP address fields can be set to follow corresponding distributions from an input set. Distributions for other header fields have to be manually configured by the user, making this generator less attractive if a realistic set of synthetic rules needs to be generated. Moreover, even though FRuG currently supports the generation of condition for 12 header fields utilized in OpenFlow 1.0.0, it neither explicitly considers specifics of OpenFlow rule sets nor allows to easily specify correlation among header fields of generated rules.

Chapter 4

Addressing Performance of Algorithms

As described in details in Section 2.2.1, packet classification algorithms targeting core networks are required to support growing transfer rates and an increasing number of rules as well as bits involved in packet classification. Therefore, this chapter deals with improving the performance of classification algorithms in order to meet the above mentioned requirements.

Since matching a condition specified as an exact value or wildcard is trivial, the greatest improvement of classification algorithms' performance can be achieved by optimizing either prefix matching or range matching. Because prefix matching is utilized in IP lookup (i.e., probably the most common type of packet classification in core networks) and also in the majority of multi-dimensional packet classification algorithms, this thesis addresses the performance of packet classification algorithms via improvements of prefix matching. Focus on matching prefixes is also beneficial because a prefix is a typical representation for conditions on IP addresses, which are the widest dimension in current classification rules. Moreover, supporting IPv6 may be more expensive than supporting additional header fields with respect to the number of bits involved in packet classification.

The performance of current CPUs prohibits software implementation of packet classification algorithms. Thus, packet classification algorithms have to be implemented in hardware. Because of availability, flexibility, and configurability, this thesis targets implementation in FPGAs. The inherent parallelism of this technology and the availability of distributed on-chip memory on current FPGAs also seamlessly supports pipelined processing, which is a necessity for a high-performance implementation of classification algorithms. Nevertheless, because of a limited number of on-chip memory blocks, the rule set has to be represented using a memory-efficient data structure.

This chapter addresses the performance of classification algorithms by introducing a memory-efficient trie-based representation of a prefix set together with a pipelined hardware architecture for prefix matching based on this representation in an FPGA. The prefix set representation was designed according to the results of the analysis of available prefix sets (real IPv4 and IPv6, generated IPv6) and it allows to store the whole data structure representing a prefix set into memory blocks available on current FPGAs. Subsets of these memory blocks are allocated to particular pipeline stages of the architecture, which allows the pipeline stages to access their memory blocks independently of each other. Thus, the architecture achieves matching performance required in 100 Gbps networks.

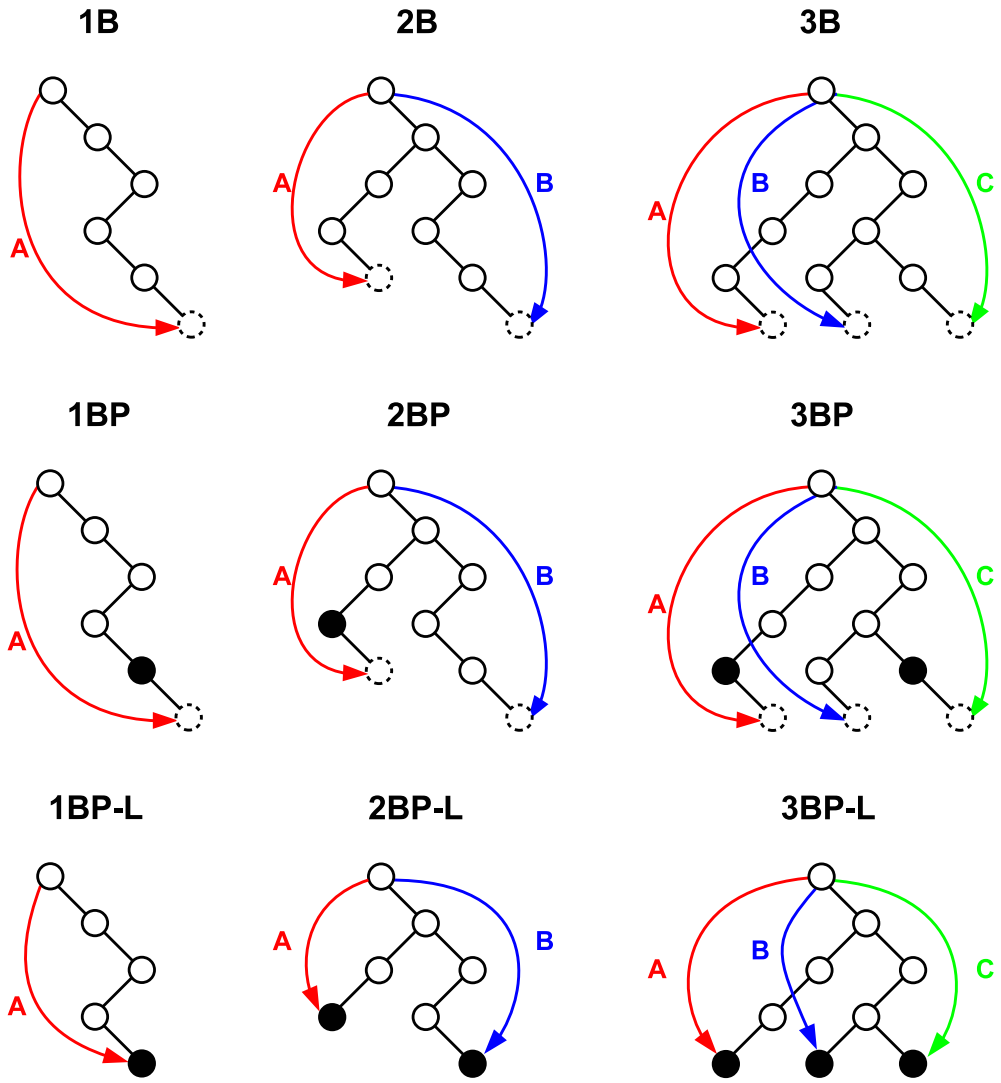


Figure 4.1: Newly proposed nodes.

4.1 Proposed Prefix Set Representation

Since the representation of a prefix set in TBM consists mainly of internal nodes without prefixes or leaf nodes, memory requirements of this algorithm may be substantially reduced by even a bit more efficient encoding of these nodes. To this end, this section proposes a new multibit trie representation of a prefix set that utilizes thirteen different types of node.

The nodes of the proposed prefix set representation can be divided into two groups: (1) nine newly proposed nodes and (2) four variants of a standard TBM node. The newly proposed nodes are illustrated in Figure 4.1, which is organized as a grid of three columns and three rows. The nodes of each column and row share the same property, which is also reflected in their name (#B encodes the number of branches, P stands for prefixes, and L marks leaf nodes). Utilized variants of a TBM node ensure completeness and efficiency of the representation in less common situations. They include a standard node for $SL = 3$ (TBM3) and a set of leaf nodes for $SL = 3, 4, 5$ (TBM3-L, TBM4-L, TBM5-L), which do not contain the external bitmap and child pointer.

The size of each newly proposed node mainly depends on its maximum allowed branch length, which is same for all branches of a node. Apart from the branch length, the size of newly proposed nodes is also influenced by the presence of child and prefix pointers, which is already encoded in the node's name. The child pointer is utilized in every node without the -L suffix in its name and the prefix pointer is present in every node, whose name contains P. The size of TBM node's variants is determined by the number of bits in the external and internal bitmaps as well as by the presence of the child pointer, which is utilized in TBM3 only (note that all TBM-based nodes contain the prefix pointer).

The mapping of proposed nodes onto a trie is done according to an algorithm, which starts from the root node and continues in breadth-first order towards the leaves of the trie. At each position, the algorithm performs a trial mapping of all proposed types of node and determines the best type for the current position using Equation 4.1, where p is the number of covered prefix nodes, n is the number of all covered trie nodes, and $size$ is the size of a selected node type. Although the algorithm does not guarantee a globally optimal mapping of proposed nodes onto the trie, it represents a working solution that is locally optimal and has acceptable time complexity.

$$cost = \begin{cases} \frac{p}{size} & \text{if } p > 0 \\ \frac{n}{size} & \text{otherwise} \end{cases} \tag{4.1}$$

4.2 Proposed Hardware Architecture

In order to achieve performance required in 100 Gbps networks, prefix matching based on the proposed representation has to be implemented in hardware using a processing pipeline, in which each processing element (PE) performs one step of the matching algorithm. Since the prefix set representation proposed in the previous section can be classified as a multibit trie, the result of prefix matching is available after processing at most n nodes, where n is the height of a tree representing the prefix set; therefore, the pipeline has to consist of n PEs. Because each of these PEs accesses a memory storing a prefix set representation independently, the memory has to be able to support n parallel memory accesses per clock cycle. To satisfy this requirement, it is convenient to implement the processing pipeline in an FPGA chip, which allows to allocate one or more independent on-chip memory blocks to each PE.

A proposed hardware architecture for prefix matching based on the prefix set representation introduced in the previous section is shown in Figure 4.2. This architecture consists of two processing pipelines with uniform PEs and dual-port memories shared between PEs from corresponding stages. By utilizing the dual-port nature of memory blocks available in an FPGA, the architecture can achieve double the performance of a single-pipeline architecture without the need to compromise on memory accesses. A memory allocated to each pipeline stage comprises two parallel parts, each of which has data width of 80 bits (i.e., the maximum size of a node). While the first part of the memory stores data words with an even address, the second part is used for storing data words with an odd address. Such internal organization of the memory makes possible to read the whole node in one clock cycle even if its representation is not aligned on the beginning of a data word, which may cause storing the node in two consecutive data words.

Figure 4.2 also shows a high-level architecture of a PE that implements one step of the prefix matching algorithm. The processing of a prefix set representation's node in the PE

follows the *fetch-decode-execute paradigm*, which is known from processing instructions in a CPU. First of all, the PE fetches from the memory the representation of the node, which is subsequently decoded and the obtained values are sent in parallel to the *execute* submodule. This submodule actually executes one step of prefix matching (i.e., it searches for matching prefixes and determines the address of the next node) and sets the PE's outputs accordingly.

Combinatorial logic of the *fetch* and *execute* submodules of the PE is relatively complex, which limits the maximum frequency they can safely operate on. Therefore, in order to achieve desired matching performance, each of them contains two sets of intra-stage registers. In total, each PE contains four sets of intra-stage registers, which means that the latency of processing a node within the PE is five clock cycles.

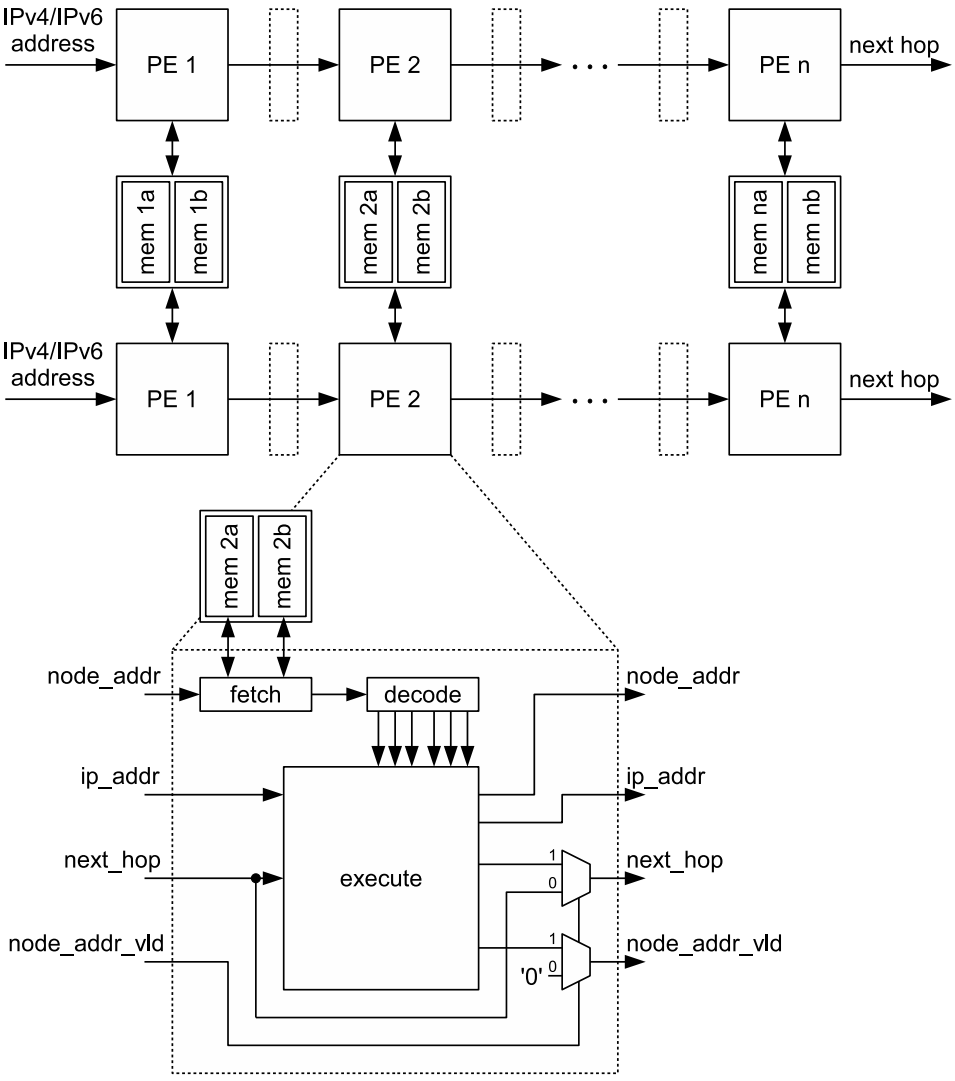


Figure 4.2: The pipelined hardware architecture that implements prefix matching based on the prefix set representation from Section 4.1. The figure also shows a high-level architecture of a PE utilized in employed processing pipelines.

4.3 Experimental Evaluation

This section evaluates the prefix set representation proposed in Section 4.1 and the pipelined hardware architecture implementing prefix matching based on the representation, which was designed in Section 4.2. The mapping of the proposed representation's nodes on a trie has been implemented in the Netbench tool [27]. Thus, it is possible to compare the memory efficiency of the proposed representation with prefix set representations utilized in TBM and SST algorithms. On the other hand, the evaluation of the designed architecture is allowed by its implementation on a Xilinx Virtex-6 XC6VSX475T FPGA, which has been done using Xilinx ISE 14.3.

The proposed prefix set representation was assessed on real IPv4 and IPv6 prefix sets from forwarding tables of core routers and synthetic IPv6 prefix sets generated by Non-random Generator [36]. Its comparison with with the TBM and SST algorithms in terms of memory requirements is provided in Table 4.1 that shows memory required for the proposed representation and the percentage of memory it saves when compared to TBM and SST. The proposed representation overcomes both TBM and SST, but the amount of saved memory is higher for TBM (between 34.67 % and 76.31 %) than for SST (between 8.65 % and 19.98 %), which was designed with the aim of minimizing memory requirements. Moreover, it is shown that sparse prefix trees of IPv6 prefix sets allow to save more memory because they are well suited for the utilization of memory-efficient newly proposed nodes introduced in Figure 4.1. It is also clear that all selected prefix sets can be stored in an on-chip memory available on the target FPGA, when encoded using the proposed representation.

Prefix Set	**Prefixes**	**Memory [kb]**	**Saved Memory**	
IPv4		**Proposed**	**TBM (*SL=5*)**	**SST (*K=32*)**
`rrc00`	332 118	6 330.8	34.67 %	8.65 %
`IPv4-space`	220 779	3 571.4	37.37 %	12.49 %
`route-views`	442 748	7 779.8	34.85 %	11.34 %
IPv6		**Proposed**	**TBM (*SL=3*)**	**SST (*K=32*)**
`AS1221`	10 518	475.8	55.82 %	19.16 %
`AS6447`	10 814	493.8	56.11 %	19.98 %
Generated IPv6		**Proposed**	**TBM (*SL=4*)**	**SST**
`rrc00_ipv6`	319 998	21 264.3	75.63 %	N/A
`IPv4-space_ipv6`	150 157	10 412.2	76.31 %	N/A
`route-views_ipv6`	439 880	29 039.5	75.57 %	N/A

Table 4.1: Memory requirements of the proposed prefix set representation and the percentage of memory it saves when compared to the TBM and SST algorithms on selected prefix sets.

Table 4.1 also allows to compute an average memory efficiency ratio (i.e., an average number of bytes required for storing one byte of a prefix) of the proposed prefix set representation. This parameter of the proposed representation, which equals to 1.08 in case of generated IPv6 prefix sets and 0.85 in case of IPv4 prefix sets, is necessary for comparison with the PPLA algorithm. According to [21], the average memory efficiency ratio of PPLA on generated IPv6 prefix sets is 0.90 when prefixes are encoded using a set of binary search trees. Therefore, the proposed representation is slightly worse than PPLA on in this case. However, it is significantly better on IPv4 prefix sets, for which [21] reports the average

memory efficiency ratio of 1.00. The memory efficiency of the proposed representation on real IPv6 prefix sets cannot be compared with PPLA, because PPLA was not evaluated on this type of data.

Since the proposed prefix set representation is based on a trie, in which multiple prefixes share the same path through a prefix tree up to some level, it should exhibit a better memory efficiency ratio on large prefix sets (e.g., generated IPv6) then on small prefix sets (e.g., real IPv6). Nevertheless, this is not the case on real and generated IPv6 prefix sets used in the performed evaluation. The most probable explanation of this unexpected situation is that the utilized Non-random Generator [36] does not model the process of IPv6 address allocation correctly, as discussed in Section 3.2. Although there are other IPv6 prefix set generators (e.g., V6Gene [37]), the Non-random Generator was used in order to fairly compare the results of the performed evaluation with results presented in [21].

The utilization of FPGA resources (both absolute and percentage) and the maximum operating frequency after place & route of the proposed architecture on the target FPGA chip are shown in Table 4.2. Apart from resource utilization and the maximum operating frequency of a single PE, the table also shows these values for a complete processing pipeline and the whole proposed architecture comprising two pipelines. Even though the length of each processing pipeline (23 PEs) makes possible to perform prefix matching using any of the selected prefix sets, the whole architecture fits into the target FPGA.

	LUTs (% of All)	**Registers (% of All)**	**Frequency [MHz]**
1 PE	3 647 (1.23 %)	1 825 (0.31 %)	127.162
1 pipeline (23 PEs)	83 881 (28.19 %)	41 957 (7.05 %)	127.162
2 pipelines (46 PEs)	167 762 (56.37 %)	83 950 (14.11 %)	127.162

Table 4.2: Resource utilization and the maximum frequency of the proposed pipelined hardware architecture after place & route on Xilinx Virtex-6 XC6VSX475T using Xilinx ISE 14.3.

Each processing pipeline is able to provide one matching result per clock cycle, thus the total matching performance of the whole architecture is almost 255 MLPS, translating into throughput of 170 Gbps for the shortest Ethernet frames. Operating frequency and the number of pipeline stages also together determine the overall latency of the proposed architecture. Because each PE consists of five pipeline stages, the whole pipeline contains $5 \cdot 23 = 115$ stages. Since processing in one stage takes approximately 7.86 ns, the overall latency of processing in the full pipeline is 903.90 ns. Overall latency also dictates the size of a buffer for packets that wait for the result of prefix matching in the proposed architecture. To support throughput of 170 Gbps, the capacity of the packet buffer has to be at least 18.75 kB. Nevertheless, this buffer can be implemented in an external memory in order to save precious memory resources available on an FPGA chip.

Chapter 5

Addressing Algorithms Benchmarking

Because the requirements on the performance of packet classification algorithms are continuously increasing, the development of classification algorithms that meet these requirements is still an active process. To verify that newly developed algorithms fulfill given requirements, they have to be benchmarked using a set of classification rules. However, since many packet classification algorithms leverage inherent properties of real classification rule sets to improve their performance [33], benchmarking using an arbitrary set of classification rules would not provide valid results. Therefore, to correctly benchmark a packet classification algorithm, a real rule set has to be used. The problem of this requirement is that real rule sets are not publicly available for the majority of packet classification use cases, mostly because of security reasons.

This chapter addresses the issues of packet classification benchmarking by introducing ClassBench-ng, a new open source tool for the generation of synthetic IPv4, IPv6, and OpenFlow 1.0.0 rule sets. Its generation process is based on an input seed that specify statistical properties of all header fields, for which the matching conditions are to be generated. Therefore, to make the ClassBench-ng output rule set as close as possible to a real classification rule sets, it is important to ensure that such a seed contains properties that precisely reflect the current trends. The chapter thus presents also the feature of ClassBench-ng that allows to create input seeds from real rule sets. This feature aims to make the proposed tool attractive in the long term and for a wide number of different use cases.

5.1 ClassBench-ng: Next Generation ClassBench

The ClassBench-ng toolkit tightly integrates rule set generation and analysis in order to allow accurate as well as flexible generation of IPv4, IPv6, and OpenFlow rule sets. It also defines the structure of a seed, which stores the results of rule set analysis and serves as an input to the rule set generation process. The ClassBench-ng seed contains several statistical distributions that allow to completely characterize all considered types of generated rule sets in an anonymous and scalable way.

Since original ClassBench already defines seed's structure for the IPv4 case and provides a rule set generator that accepts such seeds, ClassBench-ng utilizes these components and supplements them with a rule set analyzer, which is not publicly available, although it has been presented in [34]. Using parts of ClassBench is valid even though they were

designed more than 10 years ago and the Internet has changed significantly since that time. Indeed, the results of the performed analysis demonstrate that the value of IPv4 prefix set parameters has not almost changed and the expected changes were correctly reflected. This is also the case for parameters related to protocol and ports.

To support the analysis and generation of IPv6 and OpenFlow rules, ClassBench-ng extends seed's structure and also both IPv4 rule set analyzer and generator. Fortunately, the IPv4 prefix set parameters defined in original ClassBench are also able to characterize sets of IPv6 prefixes, when extended to 129 instead of 33 levels of a trie. On the other hand, OpenFlow support has to be added from scratch using the distribution for OpenFlow *rule types* and separate statistical distributions for OpenFlow-specific header fields, both of which represent important characteristics of OpenFlow rule sets. The analysis and generation tools have to be extended such that they are able to produce and consume, respectively, such a modified seed.

Figure 5.1 shows a high-level architecture of ClassBench-ng comprising four main building blocks, which are presented in details in the following subsections.

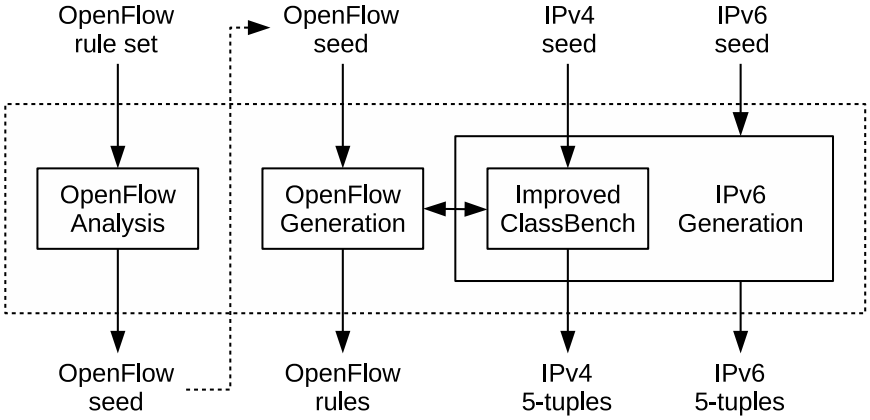


Figure 5.1: The high-level architecture of ClassBench-ng.

Improved ClassBench

For over a decade, researchers have used original ClassBench in order to generate synthetic classification rule sets for the benchmarking of packet classification algorithms. However, a test campaign evaluating the fidelity of ClassBench, which was performed in the course of this thesis' preparations, revealed that the rule set generation process of ClassBench is not always accurate. While the layer four ports and protocol of generated rules accurately follow the input seed, the IPv4 prefixes show lower accuracy. The most probable explanation of such errors is based on parameters interdependence. To prevent a complex resolving of dependencies among the prefix set parameters, ClassBench assigns each parameter a priority and tries to meet the required distributions in an order given by the priority of corresponding parameters.

ClassBench-ng improves the generation process of ClassBench by iteratively building an output rule set with source and destination prefix set characteristics as close as possible to the distributions from the input seed. First of all, the tool uses original ClassBench and generates a 100-times larger initial rule set. Then it prunes the tries representing source and destination IP prefixes of this rule set to converge on a solution which is accurate and

contain the target number of IP prefixes. Once the pruned source and destination tries are available, *Improved ClassBench* selects such rules that contain source and destination IP prefixes available in the pruned tries. To find these rules, the tool constructs a bipartite graph in which each node represents a prefix from either source or destination trie and each edge connecting two nodes represents a rule that contains corresponding source and destination IP prefixes. The rules that ClassBench-ng is looking for are represented by maximum matching in the constructed bipartite graph.

IPv6 Generation

The *IPv6 Generation* block extends the improved version of the original ClassBench's rule set generator with support for the generation of IPv6 rules. Since both IPv4 and IPv6 prefix sets can be represented using a trie and the trie-related parameters utilized in original ClassBench are able to catch current IPv6 dynamics, an IPv6 seed straightforwardly extends the trie-related parameters to allow the specification of corresponding distributions for up to 129 trie levels. In the same way ClassBench-ng also extends the improved process of IPv4 rule sets generation, i.e., it adds support for the generation of IP prefixes according to trie-related distributions specified in the IPv6 seed.

OpenFlow Analysis

The *OpenFlow Analysis* block takes as an input OpenFlow rules and generates the corresponding OpenFlow seed. The OpenFlow seed is a backward-compatible extension of a 5-tuple seed (i.e., the seed utilized in original ClassBench) consisting of three main sections: (1) a rule type distribution, (2) a 5-tuple seed, and (3) an OpenFlow-specific fields seed. The first section represents header fields dependency observed in the analyzed rule set. The structure of the second section is exactly the same as of a standalone 5-tuple seed, thus it is possible to use an OpenFlow seed for the generation of 5-tuple rules using original ClassBench (or its improved version that is employed in ClassBench-ng). Finally, the last section contains separate statistical distributions for OpenFlow-specific header fields. Each of these distributions utilizes one of the following representations:

- *values* — a distribution over a set of original values;
- *parts* — a distribution over a set of the selected part of original values;
- *size* — a total number of unique original values;
- *null* — no representation.

Currently, ClassBench-ng is able to correctly parse rule sets represented in the format utilized by the `ovs-ofctl` command line tool [2] and generate the appropriate OpenFlow 1.0.0 seed. This format is primarily aimed at representing flow table entries of OpenFlow switches (note that each flow table entry corresponds to an OpenFlow rule). However, since the IPv4 prefix and IPv4 5-tuple are also valid OpenFlow rules, the *OpenFlow Analysis* block is able to parse these types of classification rules as well. Although the `ovs-ofctl` tool supports both IPv4 and IPv6 prefixes, ClassBench-ng currently supports parsing of IPv4 prefixes only.

OpenFlow Generation

The *OpenFlow Generation* block generates a set of OpenFlow rules from an input OpenFlow seed. It starts with the generation of a set of IPv4 5-tuples, which follow the parameters specified in the seed, using the *Improved ClassBench* block of ClassBench-ng. Each generated 5-tuple is then transformed to an OpenFlow rule that complies with a *rule type* generated according to the corresponding distributions stored in the seed. In particular, some of the 5-tuple fields might be removed and other OpenFlow-specific fields might be added.

The generation of values for OpenFlow-specific header fields is driven by representation utilized for particular header fields. However, in order to generate consistent OpenFlow rules, further constraints on the generated values have to be sometimes applied (e.g., the value of *eth_type* must be set according to utilized header fields from higher levels or the value of *vlan_id* must not be `0x000` nor `0xFFF`).

5.2 Experimental Evaluation

This section evaluates the fidelity of ClassBench-ng's rule set generation for IPv4 prefixes, IPv6 prefixes, and OpenFlow rules. The evaluation does not focus on layer four ports and protocol because ClassBench-ng directly uses the values of these header fields generated by original ClassBench, which provides accurate results in this case.

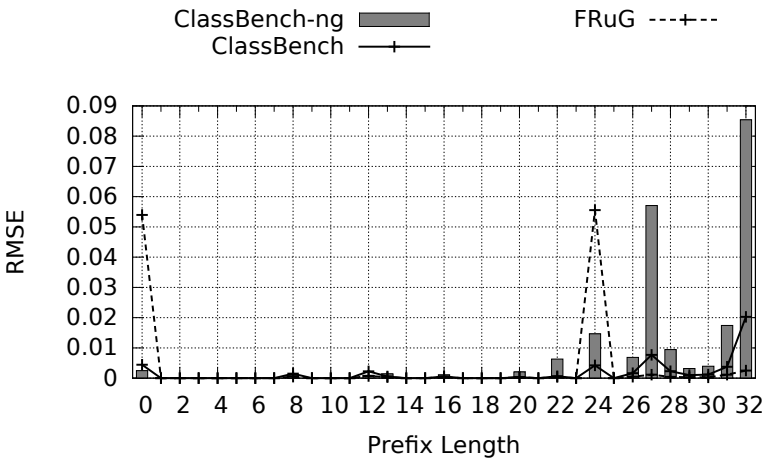
In order to fairly compare ClassBench-ng with other synthetic rule set generators, the evaluation presented in this section is based on the value of *root-mean-square error* (RMSE) that is computed using Equation 5.1. In this equation, n represents the number of generated rule sets, $\bar{y}$ is the target value of an evaluated parameter, and y_i stands for the parameter's value extracted from the generated sets. The performed experiments were carried on by generating $n = 10$ rule sets using tool-specific seeds extracted from an *original rule set*. The characteristics of the *original rule set* thus represent the target values (i.e., $\bar{y}$) against which were compared the same characteristics of the generated sets (i.e., y_i) obtained from various rule set generators.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\bar{y} - y_i)^2} \tag{5.1}$$

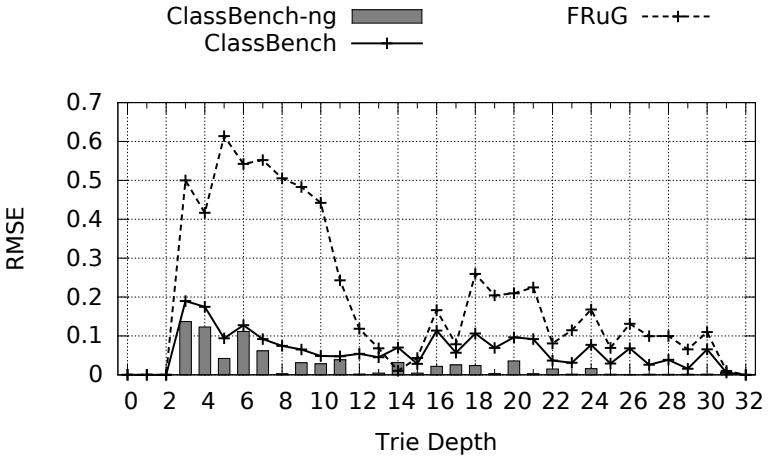
IPv4 Prefixes Generation

The comparison of ClassBench-ng, ClassBench, and FRuG on the generation of IPv4 prefixes utilized an *original rule set* that had been generated by ClassBench using the `acl4` seed provided with this tool. Because both ClassBench-ng and FRuG support the transformation of an input rule set into the corresponding seed, they were used to generate input seeds for the compared rule set generators from the *original rule set* (note that a seed for ClassBench-ng can also be used in original ClassBench). Finally, these seeds were utilized in the compared tools to generate rule sets, whose trie-related characteristics were assessed using RMSE.

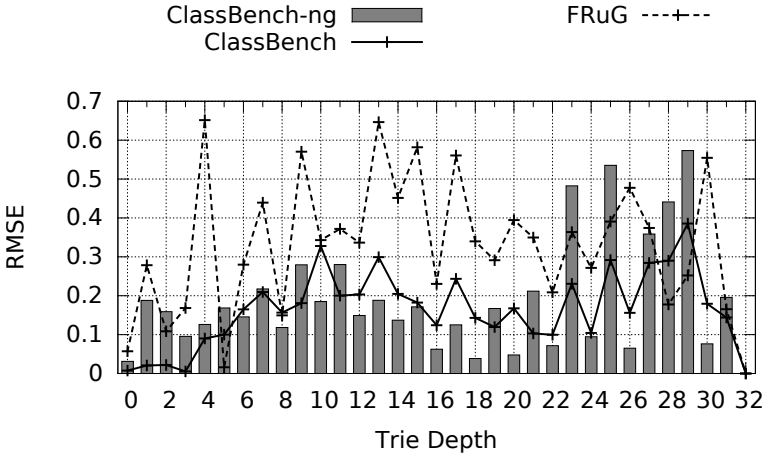
Figures 5.2 show the comparison of RMSE obtained for ClassBench-ng, ClassBench, and FRuG on each trie level. According to these results, ClassBench-ng fully outperforms original ClassBench and except for only one trie level also FRuG in terms of the branching probability distribution (Figure 5.2b). The situation is more balanced with respect to the



(a) Prefix length distribution.



(b) Branching probability distribution (two-children nodes).



(c) Average skew distribution.

Figure 5.2: The comparison of trie-related parameters' *root-mean-square error* on each trie level when generating IPv4 prefix sets using ClassBench-ng, ClassBench, and FRuG.

average skew distribution (Figure 5.2c). In this case, ClassBench-ng is more precise on approximately 50 % of trie levels when compared against ClassBench and on more than 80 % of levels when compared against FRuG. However, Figure 5.2a shows a poor fidelity of ClassBench-ng with respect to the prefix length distribution.

It is not possible to improve ClassBench-ng's generation fidelity for the prefix length distribution without impacting negatively on the other parameters. Nevertheless, in case of the prefix length distribution RMSE is 10-times lower compared to other trie-related parameters. Therefore, ClassBench-ng is the most accurate rule set generator among the compared tools on average.

IPv6 Prefixes Generation

When comparing the quality of ClassBench-ng's IPv6 prefix set generation against Non-random Generator, two *original rule sets* were used. An input seed for ClassBench-ng was extracted from an IPv6 prefix set from a core router, while Non-random Generator's input consisted directly of an IPv4 prefix set from the same router obtained on the same day. Although such a setup does not lead to an entirely fair comparison of the tools, it is enforced by their different requirements on input data. ClassBench-ng requires a seed extracted from a rule set of a target type (an IPv6 prefix set in this case) and Non-random Generator expects an IPv4 prefix set on its input. Thus, using IPv4 and IPv6 prefix sets originating from the same core router leads to the fairest comparison of the tools.

Both ClassBench-ng and Non-random Generator achieve a comparable quality of IPv6 prefixes generation in terms of the prefix length distribution. However, ClassBench-ng is more precise with respect to the branching probability distribution, while Non-random Generator wins the comparison on the average skew distribution. Individual figures illustrating these results are not shown due to space limitations.

OpenFlow Rules Generation

The fidelity of ClassBench-ng's OpenFlow rules generation was compared against FRuG on two important characteristics of an OpenFlow rule set: (1) header fields dependency represented by the *rule type* parameter and (2) separate statistical distributions for OpenFlow-

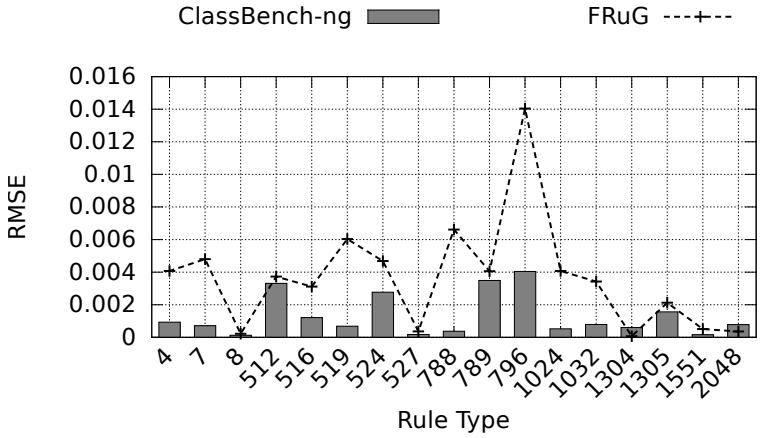


Figure 5.3: The comparison of *root-mean-square error* for *rule types* utilized in OpenFlow rule sets generated by ClassBench-ng and FRuG.

specific header fields. To fairly compare ClassBench-ng with FRuG, input seeds for both generators were extracted from a rule set utilized in an Open vSwitch deployed in a data-center, which was chosen as an *original rule set*.

Figure 5.3 compares ClassBench-ng's RMSE on particular *rule types* utilized in the *original rule set* against RMSE obtained on these *rule types* with FRuG. In this experiment ClassBench-ng clearly outperforms FRuG as it achieves higher RMSE for *rule types* 1304 and 2048 only. Therefore, ClassBench-ng is more accurate in characterizing the relationship between header fields, i.e., which fields are more likely to be specified together in a rule.

ClassBench-ng is also more accurate than FRuG with respect to the generation of OpenFlow-specific header fields, as shown in Figure 5.4. Since header fields *vlan_id*, *vlan_prio*, and *ip_tos* are always wildcarded in the *original rule set*, the figure compares average RMSE of the generators on the *in_port*, *mac_src*, *mac_dst*, and *eth_type* header fields only. While the average RMSE of both generators is almost the same (and very low) for *in_port*, ClassBench-ng is clearly more accurate than FRuG for all other fields.

The whole comparison of ClassBench-ng and FRuG is in favor of the sooner. It is also important to note that ClassBench-ng produces consistent OpenFlow rules, which satisfy all constraints introduced in Section 5.1.

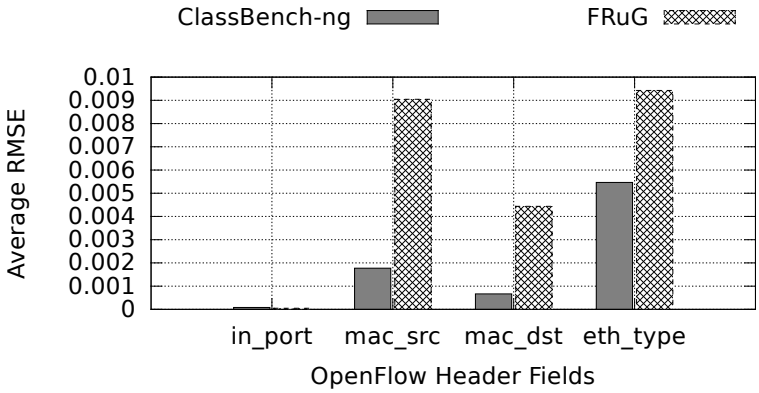


Figure 5.4: An average *root-mean-square error* of ClassBench-ng and FRuG for all evaluated OpenFlow-specific header fields. The average for each header field was computed from all RMSE values of that particular field.

Chapter 6

Conclusions

This thesis deals with packet classification that is one of the most common operations implemented in networking devices. Although the basic principle of packet classification has always been the same, a rapid development of the Internet, which we have been experiencing since the beginning of this century, has significantly increased the requirements that have to be met by current packet classification algorithms deployed in core networks. Namely, growing transfer rate has led to a shorter time available for the classification of a single packet, while the number of bits involved in the classification decision has increased due to accelerated adoption of the IPv6 and OpenFlow protocols. Moreover, the complexity of packet classification has also been increased by a growing number of classification rules. Therefore, the majority of current packet classification research address the performance of packet classification algorithms. However, as the requirements on the algorithms are continuously increasing, improving their performance is still an active process producing novel classification algorithms that have to be benchmarked, ideally using real sets of classification rules. Unfortunately, such rule sets are not publicly available for most of the packet classification use cases. Current research thus further focuses on the generation of synthetic rule sets applicable to benchmarking packet classification algorithms. These two issues in current research on packet classification are also addressed in this thesis, which has been directed by the goals set in Section 1.1.

The performance of packet classification algorithms is addressed by an FPGA-based implementation of prefix matching [24, 23] that is able to perform almost 255 MLPS for both IPv4 and IPv6 prefixes, which translates into throughput of 170 Gbps when considering the shortest Ethernet frames. Such lookup performance is enabled by a newly proposed pipelined hardware architecture utilizing on-chip memory blocks available in current FPGA chips. Although the whole architecture consists of two processing pipelines (to use both ports of on-chip memory blocks), each of which comprises 23 stages (to support matching IPv6 prefixes), it easily fits into the target FPGA chip (Xilinx Virtex-6 XC6VSX475T). In addition, because the amount of the on-chip memory is limited, prefix sets are encoded using a novel memory-efficient representation that allows to completely store any of the available prefix sets in the on-chip memory of the target FPGA. The proposed prefix set representation is more memory efficient than representations utilized in both TBM and SST algorithms, especially for IPv6 prefix sets corresponding to sparse prefix trees. Furthermore, even though the proposed representation is slightly worse than the representation utilized in the PPLA algorithm in case of generated IPv6 prefix sets, it is significantly better on real IPv4 prefix sets. Since the proposed prefix set representation and pipelined hardware architecture together allow to perform prefix matching with throughput required

in 100 Gbps networks (regardless the version of the IP protocol), it can be concluded that the first goal of this thesis has been successfully achieved.

To enable a realistic assessment of classification algorithms' performance parameters, this thesis introduces a synthetic rule set generator called ClassBench-ng [22], which combines features of existing 1-dimensional and multi-dimensional generators in a single tool and explicitly supports also the generation of OpenFlow 1.0.0. rule sets. Though original ClassBench provides all necessary features for the generation of IPv4 5-tuples, generated IPv4 prefix sets do not precisely follow corresponding distributions specified in an input seed. ClassBench-ng thus builds on the original ClassBench tool, but it improves the ClassBench's process of IPv4 prefixes generation and extends this process with the support for the generation of IPv6 prefixes. Nevertheless, the main contribution of ClassBench-ng is a newly added OpenFlow toolchain comprising not only a rule set generator but also a rule set analyzer that is capable of analyzing IPv4 and OpenFlow rule sets specified in the `ovs-ofctl` format. Since ClassBench-ng contains the rule set analyzer producing a seed together with the rule set generator utilizing such seed on its input, it is able to generate synthetic rule sets with properties similar to the analyzed real rule set and also to adjust the seed when the properties of the real set changes. Although the generation of IPv4 prefixes in ClassBench-ng is not more accurate than in original ClassBench or FRuG with respect to all trie-related parameters, on average ClassBench-ng outperforms both of these tools. The situation is similar for IPv6 prefixes generation, in which case the precision of ClassBench-ng is comparable with Non-random Generator, i.e., a specialized 1-dimensional generator of IPv6 prefixes. ClassBench-ng achieves the best results in case of OpenFlow rules generation, where it is clearly more accurate than FRuG with respect to both *rule type* and individual OpenFlow-specific header fields. In summary, the proposed ClassBench-ng toolchain is capable of generating synthetic IPv4, IPv6, and OpenFlow 1.0.0 rule sets, which follow statistical distributions extracted from real rule sets, and the precision of its rule set generation process is comparable or better than the precision of similar rule set generators. Therefore, the second goal of this thesis has also been achieved.

Despite the goals set for this thesis have been successfully achieved, there are numerous options for future work in the addressed areas. Currently, there is an ongoing effort at the development of a trace generator for ClassBench-ng that will allow to generate a packet header trace for a given rule set. Using such trace it will be possible to comprehensively assess not only the expected worst case performance of a packet classification algorithm utilizing the given rule set, but also its actual average performance under a traffic load. Once the trace generator will be finished, the next steps may focus on further extensions of the presented solutions. It would be interesting either to allow incremental updates of a prefix set while keeping a high memory efficiency of its representation or to make the ClassBench-ng's rule set analyzer able to extract a seed from different types of real rule sets specified in various formats. Nevertheless, even more appealing would be to address challenges brought by a continuous and accelerating evolution of the Internet. This category currently includes improving the performance of the prefix matching architecture to support 400 Gbps Ethernet and extending ClassBench-ng to support further versions of the OpenFlow standard.

Bibliography

[1] IPv6: IPv6 / IPv4 Comparative Statistics. Available online [September 2017]: http://bgp.potaroo.net/v6/v6rpt.html.

[2] `ovs-ofctl(8)` (Open vSwitch 2.8.90 Manpages). Available online [August 2018] https://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.html.

[3] Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. IEEE Std 802.3, 2000 Edition. IEEE. 3 Park Avenue, New York, NY 10016-5997, USA. October 2000. ISBN 0-7381-2574-8.

[4] OpenFlow Switch Specification. Version 1.0.0 (Wire Protocol 0x01). Open Networking Foundation. December 2009.

[5] Hardwarová akcelerace klasifikace paketů. Technická zpráva. Výzkumná skupina ANT at FIT. Květen 2010.

[6] IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 3: CSMA/CD Access Method and Physical Layer Specifications Amendment 4: Media Access Control Parameters, Physical Layers, and Management Parameters for 40 Gb/s and 100 Gb/s Operation. IEEE Std 802.3ba-2010. IEEE. 3 Park Avenue, New York, NY 10016-5997, USA. June 2010. ISBN 978-0-7381-6322-2.

[7] OpenFlow Switch Specification. Version 1.5.1 (Protocol version 0x06). Open Networking Foundation. March 2015.

[8] IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Std 802.11-2016. IEEE. 3 Park Avenue, New York, NY 10016-5997, USA. December 2016. ISBN 978-1-5044-3645-8.

[9] IEEE Standard for Ethernet - Amendment 10: Media Access Control Parameters, Physical Layers, and Management Parameters for 200 Gb/s and 400 Gb/s Operation. IEEE Std 802.3bs-2017. IEEE. 3 Park Avenue, New York, NY 10016-5997, USA. December 2017. ISBN 978-1-5044-4450-7.

[10] The Zettabyte Era: Trends and Analysis. White paper. Cisco. June 2017.

[11] Ahmed, O.; Areibi, S.; Grewal, G.: Hardware Accelerators Targeting a Novel Group Based Packet Classification Algorithm. *International Journal of Reconfigurable Computing*. vol. 2013. January 2013. ISSN 1687-7195.

[12] Eatherton, W.; Varghese, G.; Dittia, Z.: Tree Bitmap: Hardware/Software IP Lookups with Incremental Updates. *ACM SIGCOMM Computer Communication Review*. vol. 34, no. 2. April 2004: pp. 97–122. ISSN 0146-4833.

[13] Fredkin, E.: Trie Memory. *Communications of the ACM*. vol. 3, no. 9. September 1960: pp. 490–499. ISSN 0001-0782.

[14] Ganegedara, T.; Jiang, W.; Prasanna, V.: FRuG: A Benchmark for Packet Forwarding in Future Networks. In *Proceeedings of the IEEE 29th International Performance Computing and Communications Conference (IPCCC)*. IEEE. December 2010. pp. 231–238. ISBN 978-1-4244-9330-2.

[15] Google: IPv6 - Statistics. Available online [September 2017]: https://www.google.com/intl/en/ipv6/statistics.html.

[16] Gupta, P.; McKeown, N.: Packet Classification on Multiple Fields. *ACM SIGCOMM Computer Communication Review*. vol. 29, no. 4. August 1999: pp. 147–160. ISSN 0146-4833.

[17] International Telecommunications Union (ITU): Active mobile-broadband subscriptions per 100 inhabitants, 2007-2017. Available online [September 2017]: https://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2017/Stat_page_all_charts_2017.xls.

[18] International Telecommunications Union (ITU): List of ITU-R Recommendations on IMT. Available online [September 2017]: https://www.itu.int/net/ITU-R/index.asp?category=information&rlink=imt-advanced-rec.

[19] Jain, R.; Paul, S.: Network Virtualization and Software Defined Networking for Cloud Computing: A Survey. *IEEE Communications Magazine*. vol. 51, no. 11. November 2013: pp. 24–31. ISSN 0163-6804.

[20] Lakshman, T. V.; Stiliadis, D.: High-speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching. *ACM SIGCOMM Computer Communication Review*. vol. 28, no. 4. October 1998: pp. 203–214. ISSN 0146-4833.

[21] Le, H.; Prasanna, V. K.: Scalable Tree-Based Architectures for IPv4/v6 Lookup Using Prefix Partitioning. *IEEE Transactions on Computers*. vol. 61, no. 7. July 2012: pp. 1026–1039. ISSN 0018-9340.

[22] Matoušek, J.; Antichi, G.; Lučanský, A.; et al.: ClassBench-ng: Recasting ClassBench After a Decade of Network Evolution. In *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE. May 2017. pp. 204–216. ISBN 978-1-5090-6386-4.

[23] Matoušek, J.; Skačan, M.; Kořenek, J.: Memory Efficient IP Lookup in 100 Gbps Networks. In *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE. September 2013. pp. 1–8. ISBN 978-1-4799-0004-6.

[24] Matoušek, J.; Skačan, M.; Kořenek, J.: Towards Hardware Architecture for Memory Efficient IPv4/IPv6 Lookup in 100 Gbps Networks. In *2013 IEEE 16th International*

Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS). IEEE. April 2013. pp. 108–111. ISBN 978-1-4673-6136-1.

[25] McKeown, N.; Anderson, T.; Balakrishnan, H.; et al.: OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*. vol. 38, no. 2. April 2008: pp. 69–74. ISSN 0146-4833.

[26] Puš, V.: *Packet Classification Algorithms*. PhD. Thesis. Brno University of Technology, Faculty of Information Technology. Brno. 2012.

[27] Puš, V.; Tobola, J.; Košař, V.; et al.: Netbench: Framework for Evaluation of Packet Processing Algorithms. In *2011 Seventh ACM/IEEE Symposium on Architecture for Networking and Communications Systems*. IEEE Computer Society. October 2011. pp. 95–96. ISBN 978-0-7695-4521-9.

[28] RIPE Network Coordination Centre: RIS Raw Data. Available online [March 2018]: https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data.

[29] Song, H.; Turner, J.; Lockwood, J.: Shape Shifting Tries for Faster IP Route Lookup. In *13th IEEE International Conference on Network Protocols (ICNP'05)*. IEEE. November 2005. pp. 358–367. ISBN 0-7695-2437-0.

[30] Song, H.; Turner, J. S.: ABC: Adaptive Binary Cuttings for Multidimensional Packet Classification. *IEEE/ACM Transactions on Networking*. vol. 21, no. 1. February 2013: pp. 98–109. ISSN 1063-6692.

[31] Srinivasan, V.; Suri, S.; Varghese, G.: Packet Classification Using Tuple Space Search. *ACM SIGCOMM Computer Communication Review*. vol. 29, no. 4. August 1999: pp. 135–146. ISSN 0146-4833.

[32] Sun, Q.; Huang, X.; Yang, W.; et al.: ClassBenchv6: An IPv6 Packet Classification Benchmark. In *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*. IEEE. November 2009. pp. 1–6. ISBN 978-1-4244-4148-8.

[33] Taylor, D. E.: Survey and Taxonomy of Packet Classification Techniques. *ACM Computing Surveys*. vol. 37, no. 3. September 2005: pp. 238–275. ISSN 0360-0300.

[34] Taylor, D. E.; Turner, J. S.: ClassBench: A Packet Classification Benchmark. *IEEE/ACM Transactions on Networking*. vol. 15, no. 3. June 2007: pp. 499–511. ISSN 1063-6692.

[35] Vamanan, B.; Voskuilen, G.; Vijaykumar, T. N.: EffiCuts: Optimizing Packet Classification for Memory and Throughput. *ACM SIGCOMM Computer Communication Review*. vol. 40, no. 4. August 2010: pp. 207–218. ISSN 0146-4833.

[36] Wang, M.; Deering, S.; Hain, T.; et al.: Non-random Generator for IPv6 Tables. In *Proceedings of the 12th Annual IEEE Symposium on High Performance Interconnects*. IEEE. August 2004. pp. 35–40. ISBN 0-7803-8686-8.

[37] Zheng, K.; Liu, B.: V6Gene: A Scalable IPv6 Prefix Generator for Route Lookup Algorithm Benchmark. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06)*, vol. 1. IEEE. April 2006. ISBN 0-7695-2466-4.

Curriculum Vitae

Personal Data

Name:	Jiří Matoušek
Date of birth:	June 14, 1987
Nationality:	Czech Republic
E-mail:	imatousek@fit.vutbr.cz
Homepage:	http://www.fit.vutbr.cz/~imatousek/

Education

since 2011	Brno University of Technology, Faculty of Information Technology Ph.D. student of Computer Science and Engineering Supervisor: Ing. Jan Kořenek, Ph.D.
2009–2011	Brno University of Technology, Faculty of Information Technology Master's degree in Mathematical Methods in Information Technology Thesis: *Network Traffic Simulation and Generation*
2006–2011	Brno University of Technology, Faculty of Information Technology Bachelor's degree in Information Technology (honors degree) Thesis: *Implementation and Verification of Network Interface Blocks*
2002–2006	Střední průmyslová škola Zlín Technical Lyceum

Study Visits

10–12/2014	University of Cambridge, The Computer Laboratory, United Kingdom visitor
08–12/2009	Lappeenranta University of Technology, Finland Erasmus student

Awards

2013	Jan Hlavička Award for Outstanding Results in PhD Studies
2011	BUT FIT Dean's Award for Outstanding Master's Thesis
2008	GE Foundation Scholar-Leaders Program

Work Experience

since 01/2015	CESNET, z. s. p. o. researcher
since 10/2011	Brno University of Technology, Faculty of Information Technology junior researcher and assistant lecturer
08/2011–09/2014	CESNET, z. s. p. o. researcher

Trainings

06/2017	Quartus Prime Workshop (3 days) El Camino, Mainburg, Germany
12/2013	Vivado Advanced XDC and STA Doulos, Ringwood, United Kingdom
11/2012	Essential Tcl Scripting for the Vivado Design Suite so-logic, Vienna, Austria

Projects

- 4× Ministry of Education, Youth and Sports of the Czech Republic (ED1.1.00/02.0070, LM2010005, LM2015042, LQ1602)
- 1× Ministry of the Interior of the Czech Republic (VI20172020064)
- 3× Technology Agency of the Czech Republic (TA03010561, TH01010229, TH02010214)
- 2× Grant Agency of Brno University of Technology (FIT-S-14-2297, FIT-S-17-3994)

Publications

- 7× international conference paper (ANCS: 2017, 2018; DDECS: 2011, 2013, 2014, 2017; FPL: 2013)
- 1× international conference poster (FCCM: 2018)
- 3× national PhD workshop paper (PAD: 2012, 2013, 2014)
- 1× student competition paper (EEICT: 2011)
- 1× technical report (CESNET: 2013)

Products

- 1× software (2017)
- 1× prototype (2015)
- 1× specimen (2014)

Research Interests

- hardware acceleration of network algorithms using FPGAs
- network algorithms benchmarking