Performance Modeling and Optimizations for Decomposition-based Large-scale Packet Classification on Multi-core Processors*

Yun R. Qu, Shijie Zhou, Viktor K. Prasanna
Ming Hsieh Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
Email: yunqu@usc.edu, shijiezh@usc.edu, prasanna@usc.edu

***Abstract*—Large-scale packet classification such as OpenFlow table lookup in Software Defined Networking (SDN) is a key task performed at the Internet routers. However, the increasing size of the rule set and the increasing width of each individual rule make large-scale packet classification a challenging problem. In this paper, we present a decomposition-based approach for large-scale packet classification on multi-core processors. We develop a model to predict the performance of the classification engine with respect to throughput and latency. This model involves the architectural parameters of the multi-core processors and the design requirements of packet classification. Based on this model, we employ optimization techniques such as grouping short fields in the search phase and early termination of the merge phase. The performance model can be applied to other generic multi-field classification problems as well. To evaluate the accuracy of the performance model, we implement a 15-field classification engine on state-of-the-art multi-core processors. Experimental results show that, the proposed model predicts the performance with less than $\pm 10\%$ error. For a 32 K 15-field rule set, the optimized decomposition-based approach achieves 2000 ns per packet latency and 33 Million Packets Per Second (MPPS) throughput (49% of the peak throughput). The peak performance assumes an ideal execution model that uses an optimized execution sequence and ignores memory access latency, data dependencies, and context switch overhead.**

***Keywords*-packet classification; multi-core; performance**

I. Introduction

The development of the Internet demands next-generation routers to support a variety of network applications, such as firewall processing, policy routing, traffic billing, and other value added services. In order to provide these services, *multi-field packet classification* [1] is performed on the packet or flow header; this mechanism classifies the packets into different categories based on a set of predefined rules.

Due to the rapid growth of the Internet, packet classification faces three major challenges: the growing size of the classification rule sets, the expanding width of the packet header, and the increasing demand for high performance. For example, the number of rules in a typical rule set for the classic packet classification has increased to over thousands [2]. In SDN [3], the classic 5-field packet classification has been extended to the 15-field OpenFlow table lookup. Meanwhile, higher throughput and lower latency for packet classification are needed. These factors make packet classification a critical task in high-performance routers in the Internet as well as in data center networking.

Many existing hardware-based solutions for packet classification employ Ternary Content Addressable Memories (TCAMs) [5]. TCAMs are notorious for their high cost and power consumption [1]. Recent research has proposed to use Field-Programmable Gate Array (FPGA) technology for real-time network processing engines [6]–[8]. FPGA-based packet classification engines can achieve high throughput for rule sets of moderate size. However, as the rule set increases, it is difficult to scale up the packet classification architecture considering limited on-chip resources, such as the size of the RAM and the number of I/O pins.

The use of software accelerators and virtual machines for network applications is a new trend [9], [10]. State-of-the-art multi-core processors [11], [12] employ caches and instruction level parallelism (ILP) techniques to bridge the increasing gap between processor speed and memory speed. A cache hit typically introduces a latency of a few clock cycles, while the cache misses are overlapped with useful computation using ILP. These features make multi-core processors an attractive platform for low-latency network applications. However, efficient algorithms with predictable performance are still needed on multi-core processors.

Decomposition-based approach provides a scalable solution with respect to both the rule set size and the packet header width for packet classification. In this paper, we present a performance model for decomposition-based approach on state-of-the-art multi-core processors. This model predicts the performance for large-scale packet classification with high accuracy, and suggests possible optimization techniques to improve the performance. Specifically, our contributions include the following:

- We develop a performance model for decomposition-based large-scale packet classification on multi-core processors; this model can be extended to predict the latency and throughput for other generic multi-field classification problems.

*Supported by U.S. NSF under grant CCF-1320211.

Table I: Example: OpenFlow 15-field packet classification rule set [3], [4]

Field	Ingr	Meta-data	Eth _src	Eth _dst	Eth _type	VID	Vprty	MPLS _lbl	MPLS _tfc	SA	DA	Prtl	ToS	SP	DP
No. of bits	32	64	48	48	16	12	3	20	3	32	32	8	6	16	16
Field type†	E	E	E	E	E	E	E	E	E	P	P	E	E	R	R
R_0	5	*	00:13	00:06	0x0800	*	5	0	*	001*	*	TCP	0	*	*
R_1	*	*	00:07	00:FF	0x0800	100	7	16000	0	00*	1011*	UDP	*	*	*
R_2	*	*	*	00:00	0x8100	4095	7	*	*	1*	1011*	*	*	2-2	5-5
R_3	1	*	00:FF	*	*	4095	*	*	*	1*	1*	*	0	0-124	5-5

†: "E" as exact match, "P" as prefix match, and "R" as range match; a "*" in a field of a rule means the rule matches any input in this field.

- We propose two optimization techniques: (1) *grouping* of short fields in the search phase, and (2) *early termination* of the merge phase, based on the proposed performance model.
- We implement a decomposition-based packet classification engine on state-of-the-art multi-core processors. Experimental results show our performance model provides an accurate prediction (less than $\pm 10\%$ prediction error) of the performance.
- We sustain over 33 MPPS throughput (49% of the peak throughput) and 2000 ns per packet latency for 32K 15-field rule sets. We achieve $1.5\times$ speed-up compared with designs without any optimization techniques. The peak performance is based on a model that uses an optimized execution sequence and only considers the instructions executed for each packet.

The rest of the paper is organized as follows: Section II introduces the packet classification problem and related work. We present the data structures and algorithms of the decomposition-based approach in Section III. We develop the performance model and propose optimization techniques in Section IV. Section V evaluates the performance and Section VI concludes the paper.

II. Background and Related Work

A. Classic Packet Classification

The classic 5-field packet classification involves classifying packets based on 5 fields of the packet header [1]. The individual entries used for classifying a packet are called *rules*, which are stored in a *rule set*. Each rule has a rule ID (RID), 5 fields and their associated values, a priority, and an action to be taken if matched. Different fields in a rule require various types of match criteria, such as prefix match, range match, and exact match. A packet is considered matching a rule only if it matches all the fields in that rule. A packet may match multiple rules, but usually only the rule with the highest priority is used to take action.

B. Large-scale Packet Classification

A newer version of packet classification in *OpenFlow* [3] requires 15 fields in the packet header to be examined. In Table I, we show an OpenFlow 15-field rule set containing 6 rules (omitting priorities and actions) as an example. We define the packet classification involving no less than 15 fields and no less than 1 K rules as *large-scale packet classification*. Note this definition can be extended to other generic classification problems involving multiple fields.

C. Related Work

Packet classification approaches have been extensively studied [1]. According to the targeted platforms, algorithmic solutions for packet classification can be categorized into hardware-based approaches and software-based approaches. The hardware-based approaches usually exploit FPGA [6] or GPU [13] platforms. For large rule sets, external memory is often used; the low access rate and long access latency of external memory limit the performance for these approaches.

The software-based packet classification approaches explore efficient algorithms on multi/many-core processors. Most of the software-based solutions use decision-tree-based approaches. For example, HiCuts [14] and its enhanced version HyperCuts [15] employ several heuristics to cut the space recursively into smaller subspaces. A software-based router is implemented in [10] using HyperCuts on a 8-core Xeon processor. In [16], a decision-tree-based solution using hashing is proposed on a 4-core 2.8 GHz AMD Opteron system. However, the performance of a decision-tree-based approach is dependent on the statistical feature of the rule set as well as the shape of the tree; it is generally hard to model the performance mathematically. We are not aware of any comprehensive study made to understand its performance.

Another class of the software-based approaches is the decomposition-based approach [1]. The basic idea is to apply lookup operations in multiple fields in parallel, and merge the partial results to produce the final results. To the best of our knowledge, we are not aware of any prior work exploiting decomposition-based approaches on multi-core processors for large-scale packet classification.

Bit Vector (BV) [4] is a data structure used for merging the partial matching results. For a rule set consisting of N rules, a partial matching result from a specific field is a BV of N bits, each bit corresponding to a particular rule in the

Table II: Approach overview

Field type	Prefix/range match	Exact match
Preprocess	Building range-tree	Constructing hash table
Search	Range-tree search	Cuckoo hashing
Merge	Bit-wise AND operation on BV's	

rule set. A bit in a BV is set to "1" only if the input matches the corresponding rule in this field.

III. Decomposition-based Approach

We denote the field requiring prefix match as *prefix match field*, while we define the corresponding part of the rule in this field as *prefix match rule*. For example, "001*" is a prefix match rule in the SA field of the rule set in Table I. Similarly, *range match field*, *range match rule*, *exact match field*, and *exact match rule* can be defined. We present the decomposition-based approach in three phases: *preprocess* phase, *search* phase, and *merge* phase, as shown in Table II. Many decomposition-based approaches employ a straightforward method, *range-tree* [17], [18] in the search phase. Since hash-based algorithms [19] in exact match fields require less memory accesses than the range-tree search, we use hashing in exact match fields to reduce the processing latency and improve the throughput.

We define the rules in the rule set before any preprocessing as *original rules*. We denote the number of the original rules as N, and we denote the index for W fields as k, $k = 0, 1, \ldots, W-1$.

A. Preprocess

A common preprocess operation for any field k of the rule set is to "project" all the original rules onto this field. This operation produces a set of unique values in this field; we denote these values as *unique rules* in the field k. For example, in Table I, projecting all the 4 original rules (R_0 to R_3) onto the VID field results in 3 unique values: "*","100", and "4095". Since the number of the unique rules in a field can be much less than the total number of original rules, we use the unique rules to construct data structures in each field. We denote the number of unique rules in the field k as $q^{(k)}$ ($1 \leq q^{(k)} \leq N$).

For a prefix/range match field, a complete range-tree is constructed using the unique rules. In an exact match field, we employ hashing to reduce the search latency and increase the throughput performance. The basic idea is: (1) for a unique rule, use a set of M hash functions to generate M hash values i, $i = 0, 1, \ldots, M-1$; (2) use a hash value i as the index to access the hash table $T_B^{(k)}$; (3) if this is successful, set the partial result BV_i in the corresponding table location; otherwise try another hash value i; (4) if no vacancy, enlarge $T_B^{(k)}$ and choose another set of M hash functions. We do not introduce the range-tree and the hashing mechanism in detail, since they have been extensively studied [4], [20].

B. Search and Merge

The search process in an exact match field requires at most M hash accesses. We use the same hash-based search techniques as in [4]. Note only one BV is extracted for each field in the search phase. For W packet header fields, a total number of W BV's are extracted as partial results. We use bitwise AND operations to merge all the W BV's into a single N-bit BV; this BV represents the final packet classification result.

C. Motivation

Developing a performance model for decomposition-based approaches on multi-core processors is one of the main contributions of this paper. The performance model can be used for: (1) This model can lead to multiple optimization techniques (Section IV-D) to improve the performance with respect to throughput and latency. (2) The decomposition-based approach and its performance model can also be applied to other generic multi-field classification problems, such as traffic classification. Given the required parameters, this model can give a rough estimate on the classification performance on state-of-the-art multi-core processors. Our methodology can also be extended to other platforms such as GPU co-processors.

IV. Performance Model

In this section, we first give an overview of the state-of-the-art multi-core processors in Section IV-A. In Section IV-B, we cover all the architectural parameters and design requirements related to the performance on multi-core processors. We formulate the performance model in Section IV-C. Based on this performance model, we propose two optimization techniques in Section IV-D.

A. Multi-core Processor

A multi-core processor consists of multiple processor cores, each having access to its designated caches as well as a shared cache. Each core has access to large but much slower main memory. Modern multi-core processors are usually organized in multiple sockets, where the cores are separated into groups (sockets); communication between different sockets is realized by technologies such as Quick Path Interconnect (QPI) [12]. Figure 1 shows an example of a state-of-the-art multi-core platform. It has two 8-core sockets; each core runs at 2.4 GHz. Each core is integrated with a designated 16 KB L1 data cache and a designated 2 MB L2 cache. All the 8 cores in the same socket share a 6 MB L3 cache, and communicate with main memory through a DDR3 memory controller.

B. Parameters and Metrics

1) Architectural Parameters:

f Clock rate of each independent core;
C Total number of physical cores;

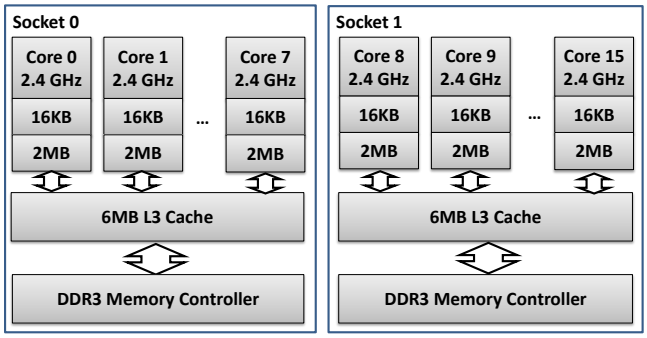

Figure 1: A state-of-the-art multi-core platform [11]

I	Instructions executed per core per clock cycle;
Γ	Cache line width;
S_1	Size of the L1 data cache on each core;
S_2	Size of the L2 cache on each core;
S_3	Size of the L3 cache;
S_4	Size of the main memory;

2) Design Requirements: The following parameters are related to specific designs; they have significant impact on the overall performance:

N	No. of the original rules in the rule set;
W	No. of fields in each original rule;
W_r	No. of prefix/range superfields;
W_e	No. of exact match superfields;
$\rho^{(k)}$	Ratio of number of unique rules over number of original rules, for $k = 0, 1, \ldots, W-1$;
$w^{(k)}$	Width of the field k, for $k = 0, 1, \ldots, W-1$;
P	No. of packets processed concurrently;
M	No. of hash functions used in each exact match field.

Clearly, $q^{(k)} = \rho^{(k)} \cdot N$. As can be seen in Section IV-D, $W_e + W_r \leq W$. For P, since packets can be processed in batches to improve cache performance and hide thread-creation overhead, we usually have $P > 1$ (but $P = 1$ is also possible). We also have another group of parameters:

V_{inst}	Average no. of instructions executed per packet;
V_1	Average no. of accesses[1] to L1 cache per packet;
V_2	Average no. of accesses to L2 cache per packet;
V_3	Average no. of accesses to L3 cache per packet;
V_4	Average no. of accesses to main memory per packet;
λ_n	Multiplicative factor for no. of accesses to various memories ($n = 1, 2, 3, 4$ for L1, L2, L3, and main memory, respectively);
α	Compensation for no. of L1 data cache accesses;
η_m	Compensation for latency per packet ($m = 0, 1$);
μ	Early termination factor.

[1]We refer "memory access" to memory read access only unless otherwise specified, since we focus on the search and merge phases.

This group of parameters are implicitly related to the performance of a multi-core platform. λ_n's and η are used to calibrate our performance model to achieve better accuracy (Section IV-C); μ is related to the early termination technique (Section IV-D).

3) Performance Metrics:

Throughput T	No. of packets processed per second;
Latency L	Average classification time per packet;
Prediction error ϵ	Relative error between the predicted latency and the actual latency.

ϵ is a metric to evaluate the accuracy of the performance model. We compute prediction error for latency. Let L_0 and T_0 denote the latency and throughput predicted by our performance model, respectively; we define $\epsilon = \frac{L_0 - L}{L}$.

C. Performance Model

To implement our packet classification engine, we can assign each core a single packet header field; however, the communication overhead between different cores during the merge phase limits the performance of this implementation [4]. Therefore, in our implementation, each packet header of W-fields stays in a single core.

1) L1 Data Cache Read Accesses: First, let us consider an exact match field. At most M hash keys are compared with the input, therefore the number of bits loaded for comparing keys is at most $M \cdot w^k$. The number of memory accesses for comparing hash keys is at most $M \cdot \lceil \frac{w^{(k)}}{\Gamma} \rceil$, where "$\lceil \cdot \rceil$" denotes the ceiling function. The memory accesses required for extracting an N-bit BV is $\lceil \frac{N}{\Gamma} \rceil$. Hence we estimate the total number of memory accesses for searching an exact match field as: $M \cdot \lceil \frac{w^{(k)}}{\Gamma} \rceil + \lceil \frac{N}{\Gamma} \rceil$.

Second, let us consider a prefix/range match field. For $q^{(k)}$ unique rules, the resulting range-tree has $2 \cdot q^{(k)}$ distinct range boundaries in the worst case. Therefore the number of levels for a complete binary tree is around $\log(2 \cdot q^{(k)})$. In each of the first $\log(q^{(k)})$ levels, a range boundary of length $w^{(k)}$ is retrieved and compared with the input; therefore the number of memory accesses required for these tree levels is: $\lceil \frac{w^{(k)}}{\Gamma} \rceil \cdot \log(q^{(k)})$. For the last level of the tree, an N-bit BV has to be loaded, which requires $\lceil \frac{N}{\Gamma} \rceil$ memory accesses.

Third, let us consider the number of memory accesses for the merge phase. If all the $(W_e + W_r)$ BV's, each of N bits, are loaded, we need $(W_e + W_r)\lceil \frac{N}{\Gamma} \rceil$ memory accesses. We introduce a factor μ ($0 < \mu \leq 1$) to consider the effect of the early termination (Section IV-D).

Note all the memory accesses are issued to L1 cache first. Hence we model the average number of L1 data cache read accesses to be:

$$V_1 = \alpha + \lambda_1 \cdot \{W_r \cdot (\lceil \frac{N}{\Gamma} \rceil + \lceil \frac{w^{(k)}}{\Gamma} \rceil \cdot \log(q^{(k)})) + W_e \cdot (M \cdot \lceil \frac{w^{(k)}}{\Gamma} \rceil + \lceil \frac{N}{\Gamma} \rceil) + \mu(W_e + W_r)\lceil \frac{N}{\Gamma} \rceil\} \quad (1)$$

Assuming the cache hit rate varies little, the number of additional L1 accesses is proportional to the number of L1

accesses initially issued. Hence a multiplicative factor λ_1 can be used to cover the additional memory accesses after a cache miss. α represents a fixed number of L1 accesses that are independent of the rule set size.

2) No. of accesses to L2, L3, and main memory: Let $S(key)$ and $S(BV)$ denote the memory sizes required to store the hash keys and the BV's, respectively. It is clear that $S(key) \propto q^{(k)}w^{(k)} = N\rho^{(k)}w^{(k)}$, $S(BV) \propto q^{(k)}N = N^2\rho^{(k)}$. Note $S(key) \ll S(BV)$, therefore we ignore $S(key)$ in the performance model. Similarly, a range/prefix match field requires a memory space of $N^2\rho^{(k)} + N\rho^{(k)}w^{(k)} \approx N^2\rho^{(k)}$.

Equation 1 can be verified using software tools. However, since data and instructions share L2 and L3 caches, it is difficult to directly model and benchmark the number of accesses to L2 cache, L3 cache, and the main memory. Here we propose a simple linear model for the number of accesses to L2 and L3 caches, as well as the main memory. This model is based on the assumption that the cache hit rate remains almost constant even for large data sets. Let $\lambda_n\langle\cdot\rangle$ denote a function of the parameters inside the angle brackets:

$$V_2 = \lambda_2\langle C \cdot S_1, (W_e + W_r)N^2\rho^{(k)}\rangle \cdot V_1 \quad (2)$$

$$V_3 = \lambda_3\langle C \cdot S_1, C \cdot S_2, (W_e + W_r)N^2\rho^{(k)}\rangle \cdot V_1 \quad (3)$$

$$V_4 = \lambda_4\langle C \cdot S_1, C \cdot S_2, S_3, (W_e + W_r)N^2\rho^{(k)}\rangle \cdot V_1 \quad (4)$$

Here all the λ_n's are multiplicative factors implicitly related to the size of the memory and the size of the data structures. Modeling these factors precisely is very difficult. As can be seen later, a simple linear model for V_2, V_3, and V_4 based on experiments is sufficient to accurately predict the performance of the decomposition-based approach.

3) Latency: An estimation of the classification latency is:

$$L_0 = \eta_1 \cdot \frac{V_{inst}}{f \cdot I \cdot C} + \sum_{n=1}^{4} \beta_n V_n + \eta_0 NP \quad (5)$$

$$= \eta_1 \cdot \frac{V_{inst}}{f \cdot I \cdot C} + V_1 \cdot (\beta_1 + \sum_{n=2}^{4} \beta_n \lambda_n) + \eta_0 NP \quad (6)$$

The value of a specific β_n is correlated to the access delay of the corresponding memory. We noticed in our experiments the frequency of context switch is relatively low, meaning the context switch has very limited impact on the overall performance. Hence we add the context switch overhead implicitly into the compensation factors η_m's. The intuition behind η_m's is the contention over shared resources among multiple threads. Larger rule sets or more parallel threads both require more shared resources to be utilized, leading to higher chances of contention and longer latency.

4) Peak Throughput: Let T_{peak} denote the peak throughput; T_{peak} is calculated by only considering the total number of instructions executed, the number of cores, and the clock rate (excluding all the memory access delay). Hence T_{peak} is an upperbound[2] on the throughput. Let V_{peak} denote the number of instructions executed for each packet in the best case (Section V-E):

$$T_0 = \frac{P}{L_0} \leq T_{peak} = \frac{f \cdot I \cdot C}{V_{peak}} \quad (7)$$

D. Optimization Techniques

1) Grouping: We notice smaller values of W_e and W_r lead to a smaller value of V_1, which in turn leads to smaller L_0. Therefore we can group multiple short fields consisting of only few number of bits (usually exact match fields) into one *superfield* by concatenating them together. For example, in Table I, the Vprty and MPLS_tfc, both 3-bit wide, have 3 and 2 unique rules, respectively; grouping them results in a 6-bit wide range match field consisting of 6 unique rules. As can be seen, grouping more than one field leads to: (1) The superfield requires range match. (2) The unique rules in a superfield are the crossproduct of the unique rules in the original fields. (3) The width of the superfield is the sum of the original field widths.

2) Early Termination: Accessing a long N-bit BV can be expensive: large $\lceil \frac{N}{\Gamma} \rceil$ results in large L_0. Now suppose the rules in the rule set are sorted by their priorities, with the higher-order bits in a BV corresponding to rules with higher priorities. We merge $(W_e + W_r)$ BV's into the final result starting from the MSB position. If a bit "1" appears in a higher-order position of the final BV, the lower-order part of the final BV can be ignored, since only the highest-priority match has to be reported. Hence, To accelerate the merge process, we split each N-bit BV into sub-BV's of length $\lceil \frac{N}{\Gamma} \rceil$; we terminate the merge process when we find a match for the first time. We define this technique as *early termination* of the merge process. This technique does not improve the worst-case performance; however, it reduces the average number of memory accesses in the merge phase from $(W_e + W_r)\lceil \frac{N}{\Gamma} \rceil$ to $\mu(W_e + W_r)\lceil \frac{N}{\Gamma} \rceil$.[3]

V. Evaluation

A. Experimental Setup

We conducted the experiments on a 2× AMD Opteron 6278 platform [11] and a 2× Intel Xeon E5-2470 platform [12]. The dual-socket AMD platform has 16 physical cores, each running at 2.4 GHz. Each core is integrated with a 16 KB L1 data cache and a 2 MB L2 cache. A 6 MB L3 cache is shared among all 16 cores. The processor has access to 64 GB DDR3-1600 main memory. The dual-socket Intel platform also has a total number of 16 cores, each running at 2.3 GHz. Each core has a 32 KB L1 data cache and a 256 KB L2 cache. All 16 cores share a 20 MB L3 cache. The processor has access to 48 GB DDR3-1600 main memory.

[2]Better theoretical performance bounds include memory bandwidth, PCIe bus speed, *etc*. However, these bounds are too loose.

[3]We assume $\mu \approx 0.5$, because the early termination technique reduces the memory accesses in the merge phase by half on the average.

Table III: Synthetic rule set

Field	$\rho^{(k)}$	Field	$\rho^{(k)}$	Field	$\rho^{(k)}$
Ingr	0.02	Metadata	0.02	Eth_type	0.01
Eth_src	0.25	Eth_dst	0.25	VID	0.2
MPLS_lbl	0.02	MPLS_tfc	1	Vprty	1
SA	0.25	DA	0.25	Prtl	0.02
SP	0.1	DP	0.1	ToS	1

Table IV: Summary of parameters

Parameter	2× AMD Opteron 6278	2× Intel Xeon E5-2470
f, C, Γ	2.4 GHz, 16, 64	2.3 GHz, 16, 64
S_1, S_2	16 KB, 2 MB	32 KB, 256 KB
S_3, S_4	6 MB, 64 GB	20 MB, 48 GB
N	1 K to 32 K	
W, W_r, W_e	15, 6, 3	
$\rho^{(k)}$, $w^{(k)}$	shown in Table III and Table I	
P	1 to 64	
M	2	
I, μ	1, 0.5	
λ_n, α, β_n, η_m	To be determined during experiments	

We implemented the decomposition-based approach using Pthreads on openSUSE 12.2 (gcc version 4.7.1). We used *perf*, a performance analysis tool in Linux, to monitor the hardware and software events such as the number of cache misses. The value of I can be determined for each experiment; however, we assume $I \approx 1$, since η_1 can compensate the prediction error for latency.

Since the typical number of unique values ($q^{(k)}$) in each field is not very large, we generated synthetic rule sets according to Table III. As shown in Table III, Vprty (3-bit), MPLS_tfc (3-bit) and ToS fields (6-bit) are short fields, the maximum number of unique rules in these fields is very small; therefore the numbers of unique rules in these fields are saturated ($\rho^{(k)} = 1$). Increasing the number of unique rules is similar to enlarging the original rule set [4], which has a negative effect on the performance (Section V-C). To reduce the number of unique rules after grouping, we only group those (short fields) containing a small number of bits: Eth_type, MPLS_lbl, and ToS are grouped as one superfield, while Ingr, VID, Vprty, Prtl, and MPLS_tfc are grouped as another superfield. Each of the remaining 7 fields is treated as one superfield without the grouping technique.

Since we target large-scale OpenFlow packet classification, we have a fixed $W = 15$; our model can be extended to other types of large-scale packet classification as well. For each exact match field (superfield), we choose $M = 2$ hash functions since a small value of M leads to a small number of memory accesses. The number of entries in a hash table is around $2 \cdot q^{(k)}$ in our implementations.

We generated random packet headers[4] to test our packet classification engine. We summarize the parameters in Table IV. The rest of this section is organized as follows: In Section V-B, we evaluate the parameters to be determined in Table IV. In Section V-C, we vary the value of N from 1 K to 32 K to show the scalability of the decomposition-based approach. In Section V-D, we vary the value of P, and study its impact on the performance. In Section V-E, we study the effect of the optimization techniques. We also compare the sustained throughput T with the peak throughput T_{peak}. In Section V-F, we compare the performance of the decomposition-based approach with prior work.

[4]Real packets usually have dependencies among them, this leads to better cache performance; the performance for classifying random packet traces is pessimistic compared to real-world scenarios.

Table V: Calculating λ_1 and α

	N	Theoretical value of $(V_1 - \alpha)\lambda_1^{-1}$	Experimental value of V_1	λ_1	α
AMD	1 K	2.59×10^2	8.47×10^2	1.30	5.08×10^2
	2 K	4.94×10^2	1.15×10^3		
Intel	1 K	2.58×10^2	4.99×10^2	1.21	1.83×10^2
	2 K	4.93×10^2	7.85×10^2		

Table VI: Estimation results of λ_n, β_n, and η

	η_0, η_1	λ_2, λ_3, λ_4
AMD	9.87×10^1, 2.54×10^1	5.43×10^{-2}, 5.81×10^{-2}, 1.11×10^{-4}
Intel	9.57×10^1, 1.79×10^1	9.75×10^{-3}, 1.59×10^{-3}, 8.86×10^{-4}

B. Estimating Parameters

We first determine the value of λ_1 and α in Equation 1. We start with $P = 1$ and choose the rule set size to be either $N = 1$ K or $N = 2$ K[5]. We measure the value of V_1 in experiments. For each N, We conduct 20 experiments on each platform; in each experiment, 1 million packets are classified. These extensive experiments provide an accurate estimation on the values of λ_1 and α. In Table IV, we show the average values of λ_1 and α on both the AMD and Intel platforms. Note both λ_1 and α are platform-dependent and correlated to S_1 and $C \cdot S_1$: (1) More L1 data cache accesses on the AMD platform results in larger values of λ_1 and α. This is because the AMD platform has a smaller S_1 than the Intel platform; it is harder to fit large amounts of data inside the L1 data cache. (2) Smaller values of λ_1 and α on the Intel platform show that we have better estimation of V_1, since we have a larger S_1 and less L1 cache misses.

With the λ_1 and α values in Table V, we can estimate the value of V_1 under various scenarios using Equation 1. If the number of L2 cache accesses (V_2) for a large N has to be estimated, we need to compute λ_2 in Equation 2 first. This can be done by dividing the experimental value of V_2 by V_1 for $N = 1$K. Similarly, λ_3, λ_4, β_n's and η can be determined. We summarize the results in Table VI.

[5]We need at least two sets of experiments to decide λ_1 and α. An alternative way to evaluate λ_1 and α is to vary N and use curve fitting techniques; however, as can be seen later, the methodology used in our model provides an estimation of V_1 that is accurate enough.

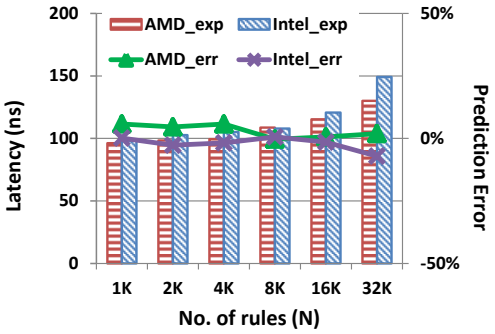

Figure 2: Latency with respect to N ($P = 1$)

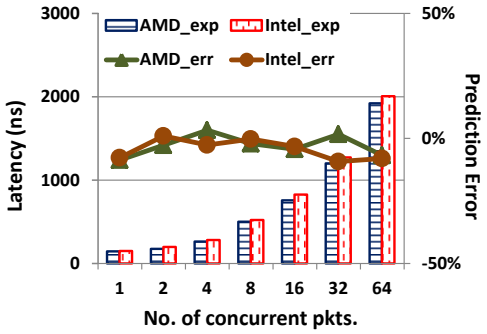

Figure 3: Latency with respect to P ($N = 32\,\text{K}$)

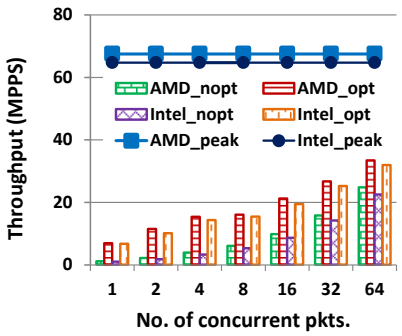

Figure 4: Throughput with respect to P ($N = 32\,\text{K}$)

To model the latency performance, given N and P, we see from Equation 6 that L_0 is only a function of V_i, V_c, and V_1; all the other parameters are either from experimental results, or known constants[6]. Hence we use Equation 6 to model the latency performance directly. The experimental results for the factor $(\beta_1 + \sum_{n=2}^{4} \beta_n \lambda_n)$ are 1.46×10^{-3} for the AMD platform, and 1.78×10^{-3} for the Intel platform.

C. Varying No. of Rules N

In Figure 2, we fix the number of concurrently processed packets $P = 1$, and show the latency performance for various rule set sizes on both the AMD (*AMD_exp*) and Intel (*Intel_exp*) platforms; we also show the prediction errors ϵ (*AMD_err* and *Intel_err*). As can be seen:

- As the number of rules N increases, the latency per packet increases sublinearly. A similar trend can also be seen when we keep N unchanged and increase $\rho^{(k)}$. This means a large $q^{(k)}$ in each superfield degrades the latency performance.
- Our implementations can achieve less than 150 ns per packet latency for 32 K rule sets. The latency predicted by Equation 6 has less than $\pm 10\%$ prediction error ($|\epsilon| < 10\%$).

D. Varying No. of Concurrent Processed Packets P

In Figure 3, we show the latency performance for $N = 32\,\text{K}$ and various values of P on both the AMD (*AMD_exp*) and Intel (*Intel_exp*) platforms, with the prediction errors ϵ (*AMD_err* and *Intel_err*), respectively. Note:

- The processing latency for each packet increases rapidly as we increase P. According to Equation 7, the throughput increases sublinearly with respect to P.
- The performance model accurately estimates the latency with less than $\pm 10\%$ prediction error.

In our experiments, we also observe that, for $P > 64$, there are too many threads competing for the same resources; the latency degrades significantly, leading to a diminishing return on the throughput performance. Also, the performance model needs to be adjusted to capture higher order effects for $P > 64$. Hence, we used $P \leq 64$ in our experiments.

E. Optimization and Sustained Throughput

In Figure 4, we show the throughput performance[7] with respect to P. We show the performance for the designs without any optimization techniques (*AMD_nopt* and *Intel_nopt*) and the designs with both the grouping and early termination techniques (*AMD_opt* and *Intel_opt*). In our experiments, we observe that the grouping technique itself shortens the search time of 15 individual fields by nearly 50%. Further, the early-termination technique reduces the average number of memory accesses, especially for large rule sets. As shown in Figure 4, with the proposed two optimization techniques, we achieve 33 MPPS throughput (up to $1.5\times$ speed-up compared to the implementation without any optimization).

We analyzed the assembly code and counted the number of instructions executed for each operation in Table VII. For the best-case scenario in the search phase, we assume the outermost "*if*" statement in nested conditional statements is always satisfied, and the first hash access to $T_B^{(k)}$ always finds a match. In the merge phase, we assume a match

[6]Note $(\beta_1 + \sum_{n=2}^{4} \beta_n \lambda_n)$ can be viewed as a single parameter.

[7]We also expect higher throughput by using more cores; *i.e.*, the performance is scalable with respect to C.

Table VII: No. of instructions executed for each operation

Search a tree level	Search a hash table	Merge 9 ($W_e + W_r$) BV's each of $\lceil \frac{N}{\Gamma} \rceil$ bits
23	76	115

is found in the first $\lceil \frac{N}{\Gamma} \rceil$ bits of the final matching result for the best-case early termination. Therefore, $V_{peak} = 23W_r \cdot \min_k[\log(q^{(k)})] + 76W_e + 115 = 2275$ in our implementation. We calculate T_{peak} using Equation 7; in this equation we assume $I = 4$ since both of the two processors can execute at most 4 instructions per clock cycle. We show in Figure 4 the peak throughput T_{peak} (*AMD_peak* and *Intel_peak*). The measured throughput ($P = 64$) $T \approx 49\% \times T_{peak}$; T considers all overheads in execution including memory access delay, data dependency, and context switch overhead. We expect higher peak as well as sustained throughput as the compilation framework improves.

F. Comparison with Prior Work

Many hardware-based packet classification approaches [6], [13] ignore the host-to-device transfer time when processing packets. Considering this I/O overhead, for example, a throughput of 2 MPPS can be achieved using GPU co-processors for 5-field packet classification [13].

Most prior work on multi-core processors target the classic 5-field packet classification ($W = 5$); in general, the decision-tree based approaches can achieve around 10 MPPS [14]–[16] on state-of-the-art multi-core processors. However, these approaches do not scale easily with respect to W: each cut in one field may lead to an exponential increase in the number of subspaces in other fields, therefore the latency and throughput performance deteriorate quickly as W increases. We are not aware of any prior work on large-scale packet classification ($W \geq 15$). According to Equation 1, the latency increases sublinearly with respect to W ($W_e + W_r \leq W$), which means the decomposition-based approach is scalable with respect to W.

VI. Conclusion

We presented a decomposition-based packet classification approach in this paper. A performance model was developed and evaluated on state-of-the-art multi-core platforms. We proposed two optimization techniques to improve the performance based on this model.

In the future, we plan to use this performance model for other generic classification problems, and compare the multi-core implementation with the FPGA-based counterparts. We will investigate the performance for more packet classification algorithms.

References

[1] D. E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238–275, 2005.

[2] "Evaluation of Packet Classification Algorithms," http://www.arl.wustl.edu/~hs1/PClassEval.html.

[3] "OpenFlow Switch Specification V1.3.1," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf.

[4] Y. Qu, S. Zhou, and V. K. Prasanna, "Scalable Many-Field Packet Classification on Multi-core Processors," in *Proc. of SBAC-PAD*, 2013, pp. 33–40.

[5] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for Advanced Packet Classification with Ternary CAMs," in *Proc. ACM SIGCOMM*, 2005, pp. 193–204.

[6] W. Jiang and V. K. Prasanna, "Scalable Packet Classification on FPGA," *IEEE Trans. VLSI Syst.*, vol. 20, no. 9, pp. 1668–1680, 2012.

[7] Z. Chicha, L. Milinkovic, and A. Smiljanic, "FPGA Implementation of Lookup Algorithms," in *Proc. HPSR*, 2011, pp. 270–275.

[8] M. Bando, Y.-L. Lin, and H. J. Chao, "FlashTrie: Beyond 100-Gb/s IP Route Lookup Using Hash-based Prefix-compressed Trie," *IEEE/ACM Trans. Netw.*, vol. 20, no. 4, pp. 1262–1275, 2012.

[9] Y. Luo, P. Cascon, E. Murray, and J. Ortega, "Accelerating OpenFlow Switching with Network Processors," in *Proc. ANCS*, 2009, pp. 70–71.

[10] Y. Ma, S. Banerjee, S. Lu, and C. Estan, "Leveraging Parallelism for Multi-dimensional Packet Classification on Software Routers," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 1, pp. 227–238, 2010.

[11] "AMD Opteron 6200 Series Processor," http://www.amd.com/us/products/server/processors/6000-series-platform/6200/Pages/6200-series-processors.aspx.

[12] "Intel Xeon Processor E5-2470," http://ark.intel.com/products/64623/Intel-Xeon-Processor-E5-2470-20M-Cache-2_30-GHz-8_00-GTs-Intel-QPI?wapkw=e5+2470.

[13] A. Nottingham and B. Irwin, "Parallel Packet Classification Using GPU Co-processors," in *Proc. SAICSIT*, 2010, pp. 231–241.

[14] P. Gupta and N. McKeown, "Classifying Packets with Hierarchical Intelligent Cuttings," *IEEE Micro*, vol. 20, no. 1, pp. 34–41, 2000.

[15] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting," in *Proc. ACM SIGCOMM*, 2003, pp. 213–224.

[16] F. Pong, N.-F. Tzeng, and N.-F. Tzeng, "HaRP: Rapid Packet Classification via Hashing Round-Down Prefixes," *IEEE Trans. on Parallel and Dist. Sys.*, vol. 22, no. 7, pp. 1105 –1119, 2011.

[17] P. Warkhede, S. Suri, and G. Varghese, "Multiway Range Trees: Scalable IP Lookup with Fast Updates," *Computer Networks*, vol. 44, no. 3, pp. 289 – 303, 2004.

[18] P. Zhong, "An IPv6 Address Lookup Algorithm based on Recursive Balanced Multi-way Range Trees with Efficient Search and Update," in *Proc. CSSS*, 2011, pp. 2059–2063.

[19] A. Yoshioka, S. Shaikot, and M. S. Kim, "Rule Hashing for Efficient Packet Classification in Network Intrusion Detection," in *Proc. ICCCN*, 2008, pp. 1–6.

[20] R. Pagh and F. F. Rodler, *Cuckoo Hashing*. Springer, 2001.