Data compression algorithms for flow tables in Network Processor RuNPU

N.I. Nikiforov, ORCID: 0000-0000-3394-1766 <nickiforov.nik@gmail.com>
D.Yu. Volkanov, ORCID: 0000-0001-9900-5822 <volkanov@asvk.cs.msu.ru>
Lomonosov Moscow State University, GSP-1, Leningr. Gory, Moscow, 119991, Russia

Abstract. This paper addresses the problem of packet classification within a network processor (NP) architecture without the separate associative device. By the classification, we mean the process of identifying a packet by its header. The classification stage requires the implementation of data structures to store the flow tables. In our work, we consider the NP without the associative memory. Flow tables are represented by an assembly language program in the NP. For translating flow tables into assembly language programs, a tables translator was used. The main reason for implementing data compression algorithms in a flow tables translator is that modern flow tables can take up to tens of megabytes. In this paper, we describe the following data compression algorithms: Optimal rule caching, recursive end-point cutting and common data compression algorithms. An evaluation of the implemented data compression algorithms was performed on a simulation model of the NP.

Keywords: network processor; software-defined networks; packet classification; data compression

For citation: Nikiforov N.I., Volkanov D.Yu. Data compression algorithms for flow tables in Network Processor RuNPU. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 77-86. DOI: 10.15514/ISPRAS-2021-33(4)-6

Acknowledgments. This work is partially supported by the Russian Foundation for Basic Research under grant 19-07-01076.

1. Introduction

At present, software-defined networks (SDN) are in active development and require high-performance switches [1]. The main functional element of the high performance SDN switch is a programmable network processor (NP). The network processor is a system-on-chip specialized for network packet processing. In this work, we consider a programmable NP. A programmable NP is one that supports on-the-fly modification of the packet processing program and the set of header fields to be processed.

In this article, we discuss data compression algorithms used for flow tables. Flow tables are needed for packet classification process. A flow table is the set of rules defined by OpenFlow protocol. OpenFlow is one of the most common protocols for controlling a network SDN switch. This paper considers OpenFlow version 1.3 [2]. Each rule contains a match field, a bit string by which a packet can be identified and a set of actions, that the NP performs on this packet. Classification is the process of the identification of a network packet by its header.

This article has the following structure: in second section we introduce problem, in third section we introduce the NP architecture and flow tables translator, in fourth section we describe related work, in fifth section we describe data compression algorithm implementation and in sixth section we introduce our evaluation methodology.

2. The problem

Let us consider OpenFlow tables formalisation. An ordered set of all considered attributes is denoted as \( I = \{m_1, m_2, \ldots, m_k\} \). Every attribute \( m_i \) from the set \( I \) is described by a bit string \( m_i \in \{0, 1\}^k \). In this article symbol \( \exists \) denotes any bit. But, if \( \exists m_i \in m_1 \) and \( m_i = \ast \), then for \( \forall m_i^k \), where \( k > j \), \( m_i^k = \ast \). The length of the attribute is denoted \( len(m_i) = W_i \). Flow tables are represented by a set of rules \( R = \{r_1, r_2, \ldots, r_n\} \). With every rule \( r_j \) binding the features:

- An index \( j \);
- A priority \( p_j \in Z^+ \);
- A vector of values of attributes \( f_j = \{f_{j1}, f_{j2}, \ldots, f_{jn}\} \), where \( f_{jl} \) is an attribute value \( m_{l} \in I \);
- A set of actions \( A_l = \{a_1, a_2, \ldots, a_j\} \).

A network packet header \( x \) and its metadata with vector values of attributes \( g = \{g_1, g_2, \ldots, g_n\} \) matches the matching rule \( r_j \in R \) if a vector of values of the attributes \( f_j = \{f_{j1}, f_{j2}, \ldots, f_{jn}\} \) and a priority \( p_j \) (a rule \( r_j \in R \) identifies a network packet with a vector of values of attributes \( g \)), if:

1) a vector values of attributes \( g \) match a vector of values of the attributes \( f, \forall g_i \in g, len(g_i) = \) \( len(f_i), \forall f_{ij} \in f_i, f_{ij} \in \{0,1\}, l = \frac{k}{2}; \)  
2) a priority \( p_j \) is the highest among all rules \( r_j \in R \), if a vector \( g \) match a vector \( f_j \).

The set of rules \( R \) must satisfy the following constraint. For any two rules \( r_j, r_l \in R, r_j \neq r_l \), if their vectors of values intersect, there is a set of attribute values. This set corresponds to vectors of values of attributes of both rules \( r_j \neq r_l \).
Let us introduce the function for network packet identification \( x \rightarrow g_i \) flow table \( R \), (denotes as \( R(x) \)). It returns a set of actions, that corresponded to the rule \( x \rightarrow g_i \).

\[
R(x) = A_x \quad \text{where} \quad A_x \quad \text{is the set of actions} \quad r_j \in R.
\]

We need to introduce a similar concept of the sets of rules \( R_1 \) and \( R_2 \). The set \( R_1 \) is similar to the set \( R_2 \) when for any network packet header, that can be identified by some rule from the set \( r_j \in R_1 \), and there exists another rule that identifies it as \( r_j \in R_2 \), and \( A_1 = A_2 \).

We need to develop an algorithm for compressing flow tables. This algorithm must translate an input flow table (a set of rules \( R_1 \)) into a new compressed set of rules \( R_2 \):
1. The set of rules \( R_1 \) is similar to the set of rules \( R_2 \).
2. The cardinality of the set \( R_2 \) must be lower than the cardinality of the set \( R_1 \).

3. Network processor architecture

In the considered NP the pipeline architecture is used, with each pipeline consisting of 10 computing blocks. To avoid complex memory organization, there is no associative memory in the considered NP. The NP uses the same memory both for commands and data. Let us consider the pipeline NP architecture. Each computing block has an access to the memory area where the program with data is located. There is a limit of 25 clock cycles per packet on each processing block. There is up to 512 kilobytes to store assembly language program representing flow tables. Due to the instruction set architecture, there is no separate memory area where data is stored. Therefore, the microcode contains all the data, required to classify packets.

3.1 Flow tables translator

Flow tables translator is a tool that is executed on CPU. It is used for flow tables translating into assembly language programs, that can be interpreted by NP. Flow tables translator uses tree structures for flow table representation. Every node of the tree structure can be associated with a table rule. After building a tree every node is translated into a part of an assembly language program. Here is a flow tables translator workflow:
1) Load a flow table from file.
2) Check every rule in the table.
3) Build a tree structure from a set of rules.
4) Translate every node into a part of an assembly language program.
5) Combine all translated parts into the one assembly language program.
6) Add a header that corresponds to used protocol.
7) Write the assembly language program into file.

This tool was implemented in work [3]

4 Related work

In this section, we introduce a review on data compression algorithms, that already used for other network processors [4]. To choose algorithms for implementation in NP we used the following criteria:
1) Compression rate, is needed for algorithm performance evaluation.
2) Evaluation of compression algorithm complexity.
3) Usability of compressed flow tables without decompression.
4) The necessity to use external memory by the algorithm.

4.1 Most common data compression algorithms

Data compression algorithms have evolved over the years. Nowadays compression algorithms can be used in many different ways. In this section we describe the algorithms that compress data in binary format. There are most known of them: Huffman codding, JPEG, LWZ, zip. These algorithms require decompression for data usage. And this is why we will not use them in our flow table translator.

4.2 Optimal rule caching

Optimal rule caching algorithm is more specific data compression algorithm. It is used for table compressing in SND switches [5]. It is based on search tree structure, that is built based on rules usage frequencies. There are two trees: the first tree consists of the most used rules. This tree is translated into assembly language program. The second tree consist from other rules; it is stored in CPU memory.

4.3 Recursive end-point cutting

Recursive end-point cutting algorithm is based on HyperSplit tree usage. Compressing is performed by destroying duplication rules [6]. This algorithm permits operations with flow tables without full rebuilding tree. By rules duplication we understand the following rules:
- A rule storing in a node duplicates the rules in leaf nodes (particle duplication).
- A rule storing in a node duplicates the rules in all leafs nodes (full duplicating rule).

This algorithm recursively uses NewHypersplit tree to remove duplicate rules from the currently being built tree. The deleted duplicate rules are then collected into a second rule table, called a recursive table, to build a second tree. It is possible that duplicate rules still exist in the second tree, and some of them are also removed and used to build the third tree.

This tree building process is performed recursively while there are duplicate rules in the last tree.

4.4 Algorithms comparison

Let us describe data compression algorithms comparison in Table 1. Each algorithm has its own pros and cons.

1) Optimal rule caching – has the highest compression ratio, and is quickly implemented in the considered NP. The need to use external memory imposes additional overhead on some packet processing.
2) Recursive end-point cutting – has the lowest compression ratio, it is more difficult to implement than the optimal rule caching algorithm. Moreover, this algorithm does not require the use of external memory.
3) Common data compression algorithms – have good compression ratios on average, but require data decompression.

Table 1: Data compression algorithms comparison

| Name                               | Complexity of construction | Compression rate | CPU memory usage | Decompression needed |
|------------------------------------|-----------------------------|------------------|------------------|---------------------|
| Optimal rule caching               | \( O(N^2) \)                | 0.1...0.9        | yes              | no                  |
| Recursive end-point cutting        | \( O(N + log(N)) \)         | 0.1              | no               | no                  |
| Common data compression algorithms | \( O(K + log_2 N) \)        | 0.1...0.8        | no               | yes                 |

Related work

In this section, we introduce a review on data compression algorithms, that already used for other network processors [4]. To choose algorithms for implementation in NP we used the following criteria:

1) Compression rate, is needed for algorithm performance evaluation.
2) Evaluation of compression algorithm complexity.
3) Usability of compressed flow tables without decompression.
4) The necessity to use external memory by the algorithm.
5. Our solution

In this section, we introduce our solution of flow tables compressing.

5.1 Flow table optimization

First of all, we need to introduce operation getting last important bit \( last(m_i) = j, m_j^i \in \{0, 1\} \) and \( m_{j+1}^i = \ast \). We claim that the rules \( r_i \in R \) and \( r_j \in R \) are the same if \( \forall u \in len(f_j^i) \ last(f_j^i) = last(f_j^i) \) but \( f_j^i \neq f_j^u \) and \( A_i = A_j \). For flow table optimization we need to remove all same rules.

5.2 Main flow table compression algorithm

Let us introduce a packet header distribution \( P \) where \( p_x \) mean packet network probability \( x \rightarrow g = \{g_0, g_1, \ldots, g_{2^n}\} \). We need a correction ratio \( T_p(R_1, R_2) \) where \( R_1 \) and \( R_2 \) are two different flow tables. Thus correction ratio means probability of incoming network packet header by distribution \( P \). As well as probability of identifying this network packet by rules \( r_1 \in R_1 \) and \( r_2 \in R_2 \). Moreover, the sets of actions of this rules are similar \( A_1 = A_2 \), \( A_1 \in r_1, A_2 \in r_2 \). The optimal correction ratio for flow table \( R \) and a number of rules \( n \) and a network packet header distribution \( P \) is:

\[
\zeta(n, R, P) = \max_{R_p, R_q \in \mathbb{R}^n} T_p(R_1, R_2).
\]

Let \( p^i \) be probability of choosing rule \( r_i \in R \), in distribution \( P \). Let rules in flow table \( R \) be in not increasing order of their probabilities. Then:

\[
\zeta(n, R, P) = \sum_{i \in [1, n]} p^i + 1 - \sum_{i \in [1, n]} p^i \geq n/n0.
\]

This algorithm needs exploration and building a flow table \( R_0 \) based on input flow table \( R \). There is a minimal set of rules \( (n_0) \) and a maximum optimal correction ratio \( \zeta(n, R, P) \).

5.3 Software solution

In this section, we introduce software workflow of our algorithm. First of all we need to add a new fields in tree structure nodes for our algorithm.

- A probability into tree node. It must be filled if node contains rule.
- A sum of probabilities of leaf nodes.

Let us introduce program operation for split tree.

- Generate a set of tree nodes.
- Sort this set in non-increasing order.
- Create a counter that stores a sum of node probabilities.
- Get the first node with maximum self-probability.
- Increase the counter.
- Add this node into another set and remove from first.
- Repeat last three operations while counter less than 0.95.
- Build tree from second set of rules.

After performing these operations, we get the set of nodes. We could build first tree from second set of rules and second tree from first set of nodes. After this, we need to translate the first tree into an assembly language program.

5.3.1 Notation used

Let \( node_1, node_2 \) – tree vertices, value – some feature value. Let’s introduce the following notations:

- Tree.root – the root node of the tree Tree.
- node(value) – the descendant of the node \( node \), connected to it by an arc with the mark value.
- node.rules – set of rules corresponding to node \( node \).
- node.edges – set of marks of arcs coming from node \( node \).
- node.prob – an amount of probabilities of rules.
- copy(node_1, val, node_2) – a procedure that adds to the node \( node_1 \) a descendant with an arc marked value, copying the tree that forms the node \( node \).
- equals(node_1, val, node_2) – function that returns true if the trees formed by nodes \( node_1 \) and \( node_2 \) are the same, otherwise it returns false. The comparison takes into account the rule sets and arc labels associated with the nodes.
- same(rules_1, rules_2) – function that returns true if rules are same.
- isleaf(node_1) – function that returns true if \( node_1 \) – tree leaf, otherwise it returns false.

5.3.2 Flow table optimization algorithm

Let us introduce procedure Same (Listing 1), it returns a set of rules that are a union of same rules in the sets of two nodes.

Let us introduce flow table optimization algorithm. It can be described by the procedure Optimize (Listing 2), where \( node \) – tree node. For optimizing flow table, we need to perform this procedure to Tree.root.

6. Evaluation

6.1 Evaluation methodology

In this section, we describe evaluation methodology. Flow table compression algorithms can be assessed by an assembly language program evaluation. This is so because flow tables translator with
Our simulation model of the NP allowed us to execute a number of tests on different sized flow tables. We have worked with flow tables that contained up to 6000 rules. The compressed flow table stored the same information as the uncompressed one but required less memory. The uncompressed flow table contained about 1500 rules.

6.3 Evaluation results

We performed an evaluation on the simulation model of the NP. This evaluation shows that our solution allowed us to use several times less memory of the NP.

6.4 Conclusion

We have proposed a new flow table translator tool that allows to implement a variety of compression algorithms, such as optimal caching, lossy compression, and bucket compression. This tool can be used in network processors to improve their performance and reduce memory usage.

References

[1] Smeliantsky R.L. System Defined networks. Open Systems. DBMS, issue 9, 2012, pp. 15-26 (in Russian).
[2] Open Networking Foundation. OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04). 2012.
[3] Markoborodov A., Skobtsova Y., and Volkov D. Representation of the OpenFlow Switch Flow Table. In Proc. of the International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC). 2020, pp. 1–7.
[4] Braun Wolfgang and Menth Michael. Wildcard compression of inter-domain routing tables for OpenFlow-based software-defined networking. In Proc. of the Third European Workshop on Software Defined Networks, 2014, pp. 25–30.
[5] Rottenstreich Ori and Tapolcai János. Optimal rule caching and lossy compression for longest prefix matching. IEEE/ACM Transactions on Networking, vol. 25, issue 2, 2016, pp. 864–878.
[6] Chang Yen-Kuan and Chen Han-Chen. Fast packet classification using recursive endpoint-cutting and bucket compression on FPGA. Computer Journal, vol. 62, no. 2, 2019, pp. 198–214.

Information about authors

Nikita Igorevitch NIKIFOROV is a Master's student at the department of Computational Mathematics and Cybernetics at the University of Computer Systems and Automation of the Faculty of Computational Mathematics and Cybernetics. Research interests include software-defined networks, data compression algorithms.

Nikita Igorevitch NIKIFOROV – студент магистратуры кафедры АСВК ф-та ВМК. Научные интересы включают программно-определяемые сети, алгоритмы сжатия данных.
Dmitry Yuryevitch VOLKANOV – candidate of physical and mathematical sciences, associate professor at the department of Computer Systems and Automation of the Faculty of Computational Mathematics and Cybernetics. Areas of research: analysis and design of network processing unit architecture.

Дмитрий Юриевич ВОЛКАНОВ – кандидат физико-математических наук, доцент кафедры АСВК ф-та ВМК. Направления исследований: анализ и разработка архитектуры сетевого процессора.