Towards Efficient Implementation of Packet Classifiers in SDN/OpenFlow

Kirill Kogan
Purdue University
kirill.kogan@gmail.com

Sergey Nikolenko
National Research University
Higher School of Economics,
Steklov Mathematical Institute
sergey@logic.pdmi.ras.ru

William Culhane
Purdue University
wculhane@purdue.edu

Patrick Eugster
Purdue University
peugster@cs.purdue.edu

Eddie Ruan
Cisco Systems
eruan@cisco.com

ABSTRACT

Traffic classification is a core problem underlying efficient implementation of network services. In this work we draw from our experience in classifier design for commercial systems to address this problem in SDN and OpenFlow. We identify methods from other fields of computer science and show research directions that can be applied for efficient design of packet classifiers. Proposed abstractions and design patterns can significantly reduce requirements on network elements and enable deployment of functionality that would be infeasible in a traditional way.

1. INTRODUCTION

Software-defined networking (SDN) abstracts network infrastructures and services on top of them. OpenFlow [6] enables this through hierarchical tuple matching with set actions. Protocols that can be expressed in this way can be programmed on network elements running OpenFlow. Ternary-content-addressable memory (TCAM) is a class of fast memory for matching packet headers against rules in tuple form. TCAM access has performance guarantees, so TCAM has become the de-facto standard for packet classification in industry [5]. We consider ways to optimize classifier matching for SDN and OpenFlow to implement desired functionality on limited hardware. To further simplify the network abstraction, we explicitly separate *data* from *program*s to be applied to them. The data, which we call a *packet flow*, can be identified by a traffic aggregation level and path through the network topology. The program is the Service-Level-Agreement (SLA) policy to be applied to packet flows. Both of these abstractions can be programmed in OpenFlow, but they have distinct behavior and may require different invariants. Separating them at the model level provides more flexibility to manage the infrastructure: classifiers of packet flows may change frequently and may benefit from "order-independent" matching, while SLA policies can be reused and may match the same traffic with "order-dependent" representations. Second, this separation promotes better reuse of underlying infrastructure. Since there are usually much fewer distinct SLAs than packet flows, one can share resources among flows to further reduce resource requirements. In addition, "virtual pipelines" can now distribute the processing to apply SLAs over the path of a packet flow to better utilize network processing capabilities.

HotSDN'13, August 16, 2013, Hong Kong, China.

2. CLASSIFIERS: ORDER-DEPENDENT VS. ORDER-INDEPENDENT

Optimization techniques that minimize space required for classifiers are usually presented as algorithms developed specifically for this problem [4]. We do not develop new algorithms from scratch but rather suggest to reduce classifier optimization to well-known problems in minimizing Boolean expressions. We consider the problem of optimizing a single classifier: reduce a list or — in the order-independent case — a set of rules $\mathcal{R}$ so that the number of rules is minimized while preserving semantics.

2.1 Optimizing Order-Independent Classifiers

We represent rules and sets of rules as discrete-valued functions. For the purpose of this exposition, we concatenate all fields and treat rules as TCAM strings in the Value–Mask–Action format. We begin with the case of Order-Independent Minimization and decide whether a certain action should be applied to this packet, i.e., construct a representation for a Boolean function. A filter $s = s_1 s_2 \dots s_M$, $s_j \in \{0, 1, *\}$, corresponds to a conjunction $f_s(x_1, \dots, x_M) = \bigwedge_{s_i=1} x_i \wedge \bigwedge_{s_i=0} \bar{x}_i$; an order-independent set of rules $\mathcal{R} = \{R_1, \dots, R_N\}$ corresponds to a formula of depth 2 in DNF of size equal to the sum of filter lengths. Optimization is thus reduced to the *MinDNF* problem: find a minimal size DNF representation for a given Boolean function f. *MinDNF* has been extensively studied; it is Σ_2^P-complete with some inapproximability results [3,7]; efficient heuristics for *MinDNF* include classical Karnaugh maps and an approach based on the Set Cover problem [1].

2.2 Optimizing Order-Dependent Classifiers

We use a heuristic approach to TCAM Minimization: reduce each set of rules that produce the same action separately; in what follows, we denote this port by 1 and all other ports by 0. The difference from the previous case is that we need to encode the fact that a rule applies and none of the previous rules that produce some other action apply. Since we reuse the same zero-producing rules many times, the most natural representation is a Boolean circuit; every ordered set of rules corresponds to a circuit of depth 3 with

OR–AND–OR alternation, i.e., an AC_3^0 circuit. Moreover, TCAM rules reduce to a special case of AC_3^0: all $\wedge$-gates corresponding to rules below a certain rule with action 0 have to be connected to the corresponding level-1 $\vee$-gate. Thus, we arrive at the *Ordered Min-AC_3^0* problem: given a Boolean function f, find its minimal size *ordered* AC_3^0 circuit, i.e., a circuit such that there exists a linear ordering $\prec$ on level-1 $\vee$-gates such that if a level-2 $\wedge$-gate is connected to a $\vee$-gate g, it is also connected to all $\vee$-gates g' such that $g' \prec g$. The Min-AC_d^0 problem has been studied in [1]. It is obviously even harder than *MinDNF*. We are not aware of any results about Ordered Min-AC_3^0 specifically, and for practical algorithms we recommend a generalization of heuristic rules for *MinDNF* discussed in the previous section. In the order-dependent case there is no easy reduction from multiple actions to two actions: rules with different actions may interleave. We can only reduce the general TCAM Minimization problem to optimizing an ordered AC_3^0 circuit with multiple actions. In relation to the optimization of order-dependent classifiers, we pose two important open problems: (1) find new heuristics for the Ordered Min-AC_3^0 problem (it may be easier than the general Min-AC_3^0); (2) find new heuristics for the Ordered Min-AC_3^0 problem with multiple actions.

3. DATA AND PROGRAM OPTIMIZATION

In this part we consider directions/design patterns to optimize data and programs.

3.1 Data Optimization

Packet flows have a different nature than SLA policies; they are added and deleted frequently. For a special case when packet flow is identified by Destination IP, classifiers match with the longest prefix. In a TCAM, order-dependent representation is most likely chosen for simplicity of management, but longest prefix match does not generalize easily to the case when a packet flow is represented by a tuple. Therefore, there is no clear well-defined priority for order-dependent classifiers, although some priority can be used to avoid false positive matches when two filters with different output port actions match the same traffic. Thus, we can significantly simplify management by preserving order-independency during allocation (if feasible); note that it is different from translation of order-dependent to order-independent classifiers.

3.2 Program Optimization

An SLA can be viewed as an ordered set of services and/or protocol descriptions. Multiple rules may match the same traffic, and various services may occur in multiple SLAs in different orders, so order-dependent classifiers are better suited for representing SLAs. The most promising direction to optimize required TCAM space is to share instances of SLAs among different packet flows when those flows have the same SLA. If an SLA policy consists of a single ACL then its classifier can be "sharable" among all packet flows with the same policy. An action is *sharable* when it is not packet flow specific; e.g., "set TOS", "PERMIT", "DENY" are sharable actions, but "POLICE" is unsharable since it has a specific packet flow state. We cannot assume that all actions are sharable; the problem is to reduce TCAM space in case when some or all of the actions are unsharable. This technique can reduce required TCAM space proportionally to the number of flows with the same SLA.

As SLA complexity increases, the number of required processing cycles increases as well, so faster network processors or longer pipelines are necessary to maintain desired line-rate, increasing the costs. An SDN controller can split processing along the path of a packet flow using its knowledge of the network; this split may be either static or dynamic.

As for static splitting, the first major approach is to split a classifier into semantically equivalent policies that can be accessed in parallel; this is done, e.g., in the Palette infrastructure [8]. A more natural way to split a hierarchical policy representation is to distribute its hierarchical levels across several linked network elements. An interesting space optimization problem was introduced in [2]: given a hierarchical policy P with depth $d > 1$, convert P to an equivalent policy P' with depth at most l, $l < d$, that minimizes the required TCAM space. The motivation behind this is to reduce required TCAM space allowing packet classification in line-rate without incurring a possibly huge memory blowup that may happen if P is flattened to a single level. This lends itself well to splitting SLAs across network elements. In SDN environments, the value of l can be related to the path length of a packet flow. Classifiers can be categorized into "well-structured" and "non-well-structured" [2]; flattening a policy loses information about classifier structure. Algorithms for "well-structured" classifiers reuse the size characteristic and do not deal with intersections of rules, so all algorithms that begin with an already flattened classifier and find an optimal depth policy should be more complex.

We can encapsulate a "program counter" in a packet to indicate where a subsequent network element should dynamically start processing. If such a program counter is implemented, we need to find out whether it can be part of OpenFlow or another abstraction. For a single packet flow, static and dynamic methods introduce a clear tradeoff between utilization of underlying infrastracture and management simplicity. Efficient static splitting is even more challenging for multiple flows. More research is required to identify the use-cases for each virtual pipeline implementation.

4. CONCLUSION

In this work, we have presented research directions that can significantly reduce TCAM and control plane requirements via classifier sharing and reuse of existing infrastructure elements. We show how to generalize a virtual pipeline architecture to distribute workload. With these ideas, OpenFlow can be extended to distribute processing. We also introduce new abstractions that simplify network components management and show how to reduce these problems to apply existing research in other fields of computer science.

5. REFERENCES

[1] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M. E. Saks. Minimizing DNF formulas and AC_d^0 circuits given a truth table. In *IEEE Conference on Computational Complexity*, pages 237–251, 2006.

[2] A. Kesselman, K. Kogan, S. Nemzer, and M. Segal. Space and speed tradeoffs in TCAM hierarchical packet classification. *J. Comput. Syst. Sci.*, 79(1):111–121, 2013.

[3] S. Khot and R. Saket. Hardness of minimizing and learning DNF expressions. In *FOCS*, pages 231–240, 2008.

[4] Alex X. Liu, Chad R. Meiners, and Eric Torng. TCAM razor: a systematic approach towards minimizing packet classifiers in TCAMs. *IEEE/ACM Trans. Netw.*, 18(2):490–500, 2010.

[5] Netlogic Microsystems. Content addressable memory. http://www.netlogicmicro.com.

[6] OpenFlow 1.3 specification, 2012. http://www.openflow.org/wk/index.php/OpenFlow_1_3_proposal.

[7] C. Umans. The minimum equivalent DNF problem and shortest implicants. *J. Comput. Syst. Sci.*, 63(4):597–611, 2001.

[8] D. Hay Y. Kanizo and I. Keslassy. Palette: Distributing tables in software-defined networks. In *INFOCOM*, 2013.