Modeling Filtering Predicates Composition with Finite State Automata

Marco Leogrande, Luigi Ciminiera, Fulvio Risso
Dipartimento di Automatica e Informatica
Politecnico di Torino
Corso Duca degli Abruzzi, 24 – 10129, Torino, Italy
{marco.leogrande,luigi.ciminiera,fulvio.risso}@polito.it

ABSTRACT

Network virtualization has gained a lot of attention recently, because of some new interesting proposals in the field (i.e. OpenFlow). This trend has had the effect of pushing some filtering operations up at the software level: i.e. extract a potentially large number of protocol fields from a packet, or dynamically combine different filters. The time constraints of working at line rate force the creation of a packet filter model that can guarantee the minimum number of packet checks. This poster proposes mpFSA, a packet filter model based on the Finite State Automata formalism, that aims at achieving optimality w.r.t. the number of packet accesses, without sacrificing efficiency and scalability.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations

General Terms

Algorithms, Design, Performance

1. INTRODUCTION

In the last years, academic researchers and industries spent a significant portion of their development effort in proposals related to the network virtualization topic. OpenFlow [1], the architectural proposal of a network of switches driven by configurable controllers, is probably the most famous example.

To perform virtualization at acceptable performance levels many issues must be solved: how to extract a lot of data from packets to match them against some signatures, or how it is possible to quickly update packet filters on the fly to recognize new flows. Even if it is possible to partially implement such systems at the hardware level, software has to face stricter computational requirements. The constant increase of both data traffic and network-enabled devices forces the filtering systems to always work at line rate.

One possible approach to reach the performance goal is to focus on the reduction of the number of checks that a packet filter must perform to recognize a protocol or to extract a certain field. On the other hand, system flexibility should be preserved as well, i.e. by keeping filter update costs low.

To the best of our knowledge, the most common filtering architectures (such as for **BPF+** [2] or **Swift** [3]) aim at speeding up the execution by applying compiler-oriented optimizations to the generated code, or deploying smart memory strategies to coalesce packet accesses, or using hardware-efficient assembly instructions. These approaches have the downside of not guaranteeing the execution of the minimum number of checks. Filter update flexibility, on the other hand, is usually overlooked.

2. PROBLEM OVERVIEW

This poster proposes a new model, called **mpFSA** (standing for **Multilevel Finite State Automata with Predicates**), that uses the FSA formalism [4] to achieve the minimum number of packet accesses. Optimizations are applied directly at the *filtering graph* level, the data structure that stores information about how the filtered protocols might be encapsulated[1].

Our FSA approach, described later, does not focus only on the efficient analysis of the filtering predicate, but concentrates also on quickly scanning packets to determine the protocols included within them and, therefore, a timely extraction of useful protocol fields. Our model checks separately for: *(i)* protocol encapsulation and *(ii)* predicates included in the provided filter. A preliminary analysis suggests that this separation might benefit both the goal of reaching the minimum number of checks and the achievement of keeping the system flexible w.r.t. filter updates.

3. MULTILEVEL FINITE STATE AUTOMATA WITH PREDICATES

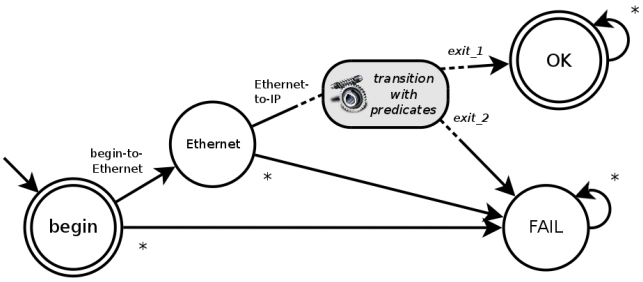

Figure 1: mpFSA for the filter `ip.dst == 1 or ip.src == 2 or ip.src == 3`. Refer to Figure 2 for the transition with predicates.

The proposed mpFSA architecture adapts the Finite State

[1]If the filtering implementation uses a native code generator, the filtering graph is most likely the data structure used by the compiler to emit the desired filtering code.

Automata model to packet filtering. In particular, the *transition function* is extended, by tuning its behavior according to a set of **Boolean predicates**.

The mpFSA optimizes the protocol scan by mapping the protocol encapsulation on FSA *input symbols*. Each symbol consumed by the mpFSA represents a "jump" between two consecutive protocols in the packet under investigation: so the symbol `Ethernet-to-IP` carries the meaning that the `IP` protocol was found inside the `Ethernet` encapsulation. As a well-defined algebra exists already for FSA, the composition among different mpFSA is performed with solid guarantees of optimality.

The *Boolean predicates* that are used inside transitions with predicates, on the other hand, are modeled as hypotheses on the fields of the protocols included in the packet itself: i.e. `ip.dst == 1`. Their optimized analysis is achieved by grouping the checks on the fields of the same protocol inside the same transition and ordering them in consecutive steps. According to the kind of checks needed, a hybrid structure is created, that mixes a tree-like comparison hierarchy (for range operators) with a jump table structure when testing for equality. It is possible to optimize the field accesses by keeping the structure balanced and using FSA-like algorithms to advance through the different steps of the packet scan.

As an example, the mpFSA in Figure 1 and 2 models the filter `ip.dst == 1 or ip.src == 2 or ip.src == 3`.

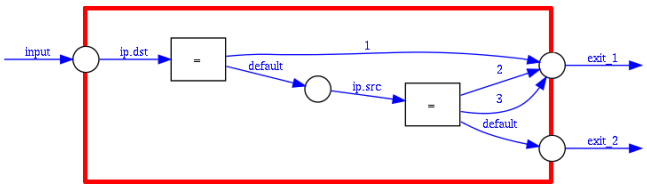

Figure 2: Transition with predicates internal structure.

Our model focuses on reducing the execution time, therefore pushes towards the latter in the space-time tradeoff; furthermore, some details of the model have a high degree of generality. It is still unclear whether these choices are correct; more testing on real-world scenarios is needed.

4. PRELIMINARY RESULTS

The proposed mpFSA model has been validated by implementing it inside the NetBee framework, which includes an experimental compiler that creates run-time code for the NetVM [5] virtual machine. In particular, the mpFSA abstraction has been implemented inside the compiler front-end.

The prototype has been tested against a BPF implementation, representative of the state of the art. In order to avoid overheads related to code interpretation, both the mpFSA and the BPF code were compiled Just-In-Time [6].

A packet trace was prepared, that included 300 packets generated during an ordinary web browsing session. Five filtering predicates were chosen and, for each of them, the number of clock cycles needed to complete the packet analysis was measured, by using the `RDTSC` assembly instruction available on the x86 architecture.

The results are shown in Figure 3. Because of the fewer packet checks needed, caused by the elimination of some redundancies that BPF did not take into account, mpFSA outperforms BPF in all chosen filters. The reduction of the number of ticks needed to complete the filter, in our tests, varies between 10 and 35 %.

Even if these results are satisfactory and show already a slight performance improvement, this evaluation must be considered a preliminary test: many more comparisons must be performed to assess the complete validity of our claims. Our next step is to perform a thorough scalability analysis.

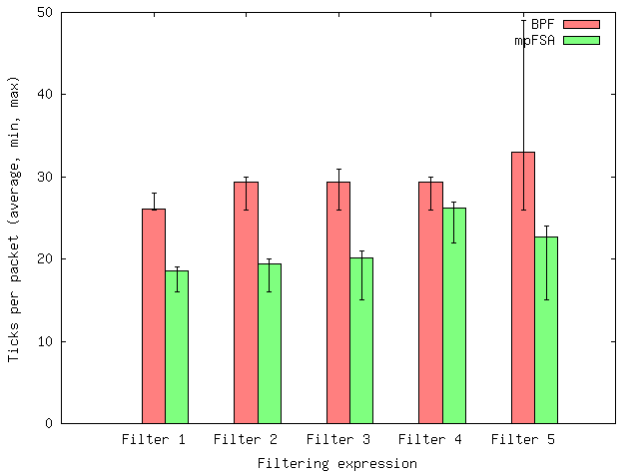

Figure 3: Comparison between BPF and mpFSA performance.

5. REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74, March 2008.

[2] A. Begel, S. McCanne, and S. L. Graham. Bpf+: exploiting global data-flow optimization in a generalized packet filter architecture. *SIGCOMM Comput. Commun. Rev.*, 29:123–134, August 1999.

[3] Z. Wu, M. Xie, and H. Wang. Swift: a fast dynamic packet filter. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 279–292, Berkeley, CA, USA, 2008. USENIX Association.

[4] J. Hopcroft, R. Motwani, and J. Ullman. *Automata theory, languages, and computation*. Addison-Wesley, 2006.

[5] O. Morandi, F. Risso, P. Rolando, S. Valenti, and P. Veglia. Creating portable and efficient packet processing applications. *Design Automation for Embedded Systems*, 15:51–85, 2011. 10.1007/s10617-011-9072-8.

[6] L. Degioanni, M. Baldi, F. Risso, and G. Varenni. Profiling and optimization of software-based network-analysis applications. *Symposium on Computer Architecture and High Performance Computing*, 0:226, 2003.