An OpenFlow Compliant Smart Switch for monitoring applications

Andrea Di Pietro
Dept. of Information Engineering, University of Pisa, ITALY
Email: {andrea.dipietro@for.unipi.it

***Abstract*—In this paper we propose a novel hardware-software co-design vision that aims at enhancing flexibility and reusability of hardware based packet forwarding engines. In particular, we move on the path of the well-known OpenFlow architecture. However, although such approach is certainly powerful, it is biased towards routing-related applications: its main goal is to allow the software control plane to arbitrarily route a packet flow. We believe that a similar paradigm, encompassing high performance packet forwarding hardware driven by a flexible software control plane, may be beneficial even to other kinds of applications, like monitoring and measurements. Unfortunately, the primitives that the OpenFlow protocol provides are in many cases not flexible enough for such purposes. For this reason, we propose a flexible packet forwarding architecture based on regular expression that, besides enabling standard-compliant OpenFlow switching, can be easily reconfigured through its control plane to support other kinds of applications. We implemented a working prototype of out architecture by using the well-known NetFPGA board.**

I. Introduction

The big advantage of the OpeFlow paradigm is the separation between the data plane (whose functionality is fixed and which can be optimized for high performance) and the routing intelligence, which is left for the user to implement through a well defined standard interface. Several works, such as [1], showed that network monitoring applications can also benefit from an OpenFlow switch, which can be used in order to dispatch packets and flows to an array of software based sensors. However, the standard OpenFlow protocol, which limits the definition of a flow to a 10-tuple of fields extracted from layer 2-4 headers, is a bit limiting for such use case. For example, it does not allow to demultiplex packets based on a the presence of a certain pattern in their payload (in turn, revealing a particular application), which would be very useful for application level monitoring. For this reason, we propose a novel switching architecture which, unlike OpenFlow, is based on regular expressions. A regular expression provides a concise and flexible means for matching particular patterns over a flow of characters. A common use case for regular expressions is deep packet inspection for intrusion detection. In our architecture, instead of defining a flow in terms of a tuple (with possibly undefined values), we define it in terms of a pattern described as a regular expression. Such a different approach allows to define a flow in a very flexible way: the fields of interest (which can be specified according to the application and may include part of the payload) in the packets are extracted to make up a string, which is then walked through by our regular expression processor. The rest of the paper is organized as follows: in section II we describe the high level architecture, while in section III we present the implementation details of the prototype we implemented on a NetFPGA board and in section IV we explain how the proposed architecture (beside supporting the OpenFlow standard) can support further applications.

II. Smart Switch Architecture

The process of IP packet forwarding depending on arbitrary metadata (i.e., one or more OSI layers) contained in the packets themselves is logically (and practically) equivalent to perform *pattern matching*. The search in the forwarding table can therefore be obtained by simply applying pattern matching algorithms upon arbitrary metadata extracted from a specific IP packet. As pattern matching is a widely addressed topic in literature, the above observation opens a wide horizon of theoretical and practical solutions to address the problem of lookup and classification. Typically, finite automata (FAs) are employed to implement regular expression search, but for realistic rule sets they need a memory amount which turns out to be too large for practical implementation. We have chosen for our scheme the δFA [3] which presents interesting performance characteristics. In particular, in addition to maintaining a data structure which is much more compact that the standard automaton, it needs a lower number of memory accesses than most compressed automata. The smart switch architecture is organized into a software control layer and a hardware–based packet switching component. While the latter is in charge of actually performing regular expression matching over the incoming packets and to forward them on the proper port based on its output, the former is in charge of updating the data structure representing the automaton, handling packets which do not match any rule and implementing the interface with the application(e.g. the OpenFlow protocol). The core module of the switch is shown in figure 1. The first operation performed on the incoming packet is parsing the fields of the packet in order to compose the string which will be fed into the δFA state machine. This operation is performed by "Metadata Exctractor", a configurable block that extracts the right fields depending on the application (e.g. the OpenFlow 10-tuple). The obtained string is passed to the "Pattern Matching Engine", which starts walking through the δFA data structure one character at a time. The result (i.e., the output port associated with the flow) is then used in order to decide the specific operation to be performed over the packet. When a packet

Resource	Logic Utilization (%)		Logic Distribution (%)		
	Slice Flip Flop	4 input LUTs	Occupied Slices	Total of 4 inputs LUTs	RAMB16s
Smart Switch	32	42	66	49	65
OpenFlow on NetFPGA	47	75	91	80	65

TABLE I
COMPARISON IN RESOURCE UTILIZATION.

does not match any rules is sent to the control plane software which, depending on the application logic, will take the proper decision and, if necessary, update the automaton (in the case the OpenFlow standard is being implemented, this will involve interaction with the controller).

III. Prototype Implementation

We implemented a prototype of our switch as a module of the well-known Netfpga platform. Our implementation can hold more than 100000 flow entries and it is capable of running at line-rate across the four NetFPGA ports. Differently from the design of [4] (which places output queues in the fast yet small BRAM memory) we decided to store both the automaton meta-data and the output queues in the external SRAM. This minimizes the risk of queue overflow, but on the other hand, requires to limit the number of lookups in order to hit line rate. For this reason, we implemented a flow–cache (in this case a flow is defined as the standard five-tuple), where a new entry is added when the first packet of a new flow enters the system. Upon cache miss, the switch performs a lookup in the δFA and stores the result in the flow cache. Since the number of flows can be very high, a hash table is an efficient way to implement such a cache. In order to avoid collisions, we implemented a Perfect Hash Function (PHF) through double hashing. The basic idea to create a PHF is using a two-level hashing scheme with universal hashing at each level. In the first level, the n keys are hashed into m slots by using a hash function h carefully selected from a family of universal hash functions. To handle collisions in a slot j, a small secondary hash table S_j with an associated hash function h_j is used. By carefully choosing the hash functions h_j, we can guarantee that there are no collisions at the secondary level. We compared the complexity of our Smart Switch to the original OpenFlow running on NetFPGA as implemented in [4]. The build results for the two designs are shown in table I. These results were obtained by using Xilinx's implementation tools from ISE10.1.03. We report the values of logic utilization and logic distribution considered in percentage. These results suggest that in terms of logic utilization, our implementation is more efficient than the original, although being able to handle a larger number of rules (100000 against the 32000 claimed in [4]). The BRAM occupation of our prototype, instead, is equal to that of the other system, due to the amount of memory required for implementing perfect hashing.

IV. Use cases

Besides supporting the OpenFlow standard, our switch can support different applications by using a proper set of regular expressions and by configuring the meta–data extraction block.

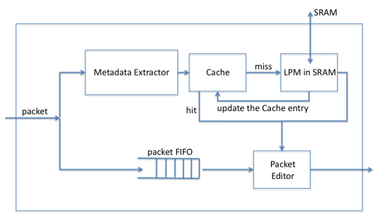

Fig. 1. Core module of the Smart Switch.

In [5] we showed that the high entropy of the least significant bits of IP addresses allows to use them as keys for load balancing. In particular, by writing a set of regular expressions that assigns different combinations of suffixes to different interfaces it is possible to split the traffic coming from a port into roughly equivalent portions. [5] shows that such partitioning, although not perfect, is comparable to that of a hash function computed of the flow keys (which is the typical solution adopted by special purpose load–balancing devices). Another possible use case of the smart switch (which we have not implemented yet) is using it as hardware-based pre-filter for Voip traffic monitoring. As RTP port numbers are notoriously dynamically assigned, that cannot be achieved by just observing the OpenFlow 10-tuple and may be a cumbersome task to be implemented in software at high speed. However, some patterns in the packet payload (codec identifiers etc.) can be expressed in terms of regular expressions and used to filter out non–interesting traffic. The suspicious RTP traffic (hopefully a much–reduced stream) can be then forwarded to a software based sensor for further inspection.

References

[1] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy, "Flow processing and the rise of commodity network hardware," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 20–26, March 2009. [Online]. Available: http://doi.acm.org/10.1145/1517480.1517484

[2] G. Antichi, A. D. Pietro, D. Ficara, S. Giordano, G. Procissi, and F. Vitucci, "On the use of compressed dfas for packet classification on netfpga," in *IEEE CAMAD*, 2010.

[3] D. Ficara, S. Giordano, G. Procissi, F. Vitucci, G. Antichi, and A. DiPietro, "An improved dfa for fast regular expression matching," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 5, 2008.

[4] J. Naous, D. Erickson, A. Covington, G. Appenzeller, and N. McKeown, "Implementing an openflow switch on the netfpga platform," in *ACM ANCS*, 2008.

[5] G. Antichi, A. D. Pietro, D. Ficara, S. Giordano, G. Procissi, and F. Vitucci, "Design and development of an openflow compliant smart gigabit switch," in *IEEE GLOBECOM, to appear*, 2011.