Open vSwitch: performance improvement and porting to FreeBSD

Gaetano Catalli
University of Pisa
gaetano.catalli@gmail.com

ABSTRACT

OpenFlow switching enables flexible management of enterprise network switches and experiments on regular network traffic. OpenFlow switches are in charge of packet forwarding, whereas a controller sets up switch forwarding tables on a per-flow basis, to enable flow isolation and resource slicing.

The present work deals with Open vSwitch, a software implementation for Linux platforms of an OpenFlow switch. The current implementation includes both a specific kernel-module and a totally userspace version of the program. The bulk of the code is written in platform-independent C and is easily portable to other environments.

In the first part of the work we performed the porting of the userspace version of the software to FreeBSD. Then, being the obtained performance unsatisfying, we proceeded reorganizing portion of the code and eliminating some inefficiencies. This led to performance about ten times greater than before.

Finally, combining OVS with *netmap*, a recently developed framework for very fast access to network packets, we had been able to further increase the program performance of about four times, with peaks of *3.0 Mpps*.

1. INTRODUCTION

Networks have become part of the critical infrastructure of our businesses, homes and schools. On the other hand, the enormous installed base equipment, and the reluctance to experiment with production traffic, have created a considerable obstacle for new ideas.

Virtualized programmable networks, such as *GENI*, could lower this barrier to entry for new ideas, increasing the rate of innovation in the network infrastructure. Programmable networks call for programmable switches and routers that can process packets for multiple isolated experimental networks simultaneously.

The OpenFlow [2] project aims to the deployment of such a device, based on an Ethernet switch, with an internal flow table and a standardized interface to add or remove flow entries. The basic idea is to exploit the fact that most modern Ethernet switches and routers contain flow-tables that run at line-rate (e.g. to implement firewalls, NAT, QoS, etc.). While each vendor's flow-table is different, an interesting common set of functions that run in many switches and routers has been identified. OpenFlow exploits this common set of functions. Furthermore, OpenFlow provides an open protocol to program the flow-table in different switches and routers.

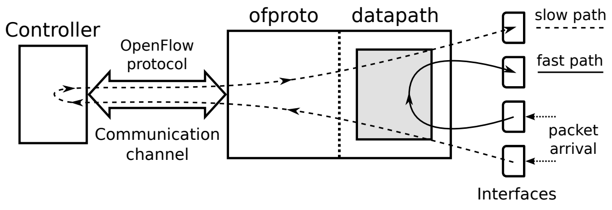

Figure 1: Open vSwitch structure, representing the flow of traffic in the system.

2. OPEN VSWITCH

Open vSwitch [3] (OVS) is an open-source software implementation of an OpenFlow switch, principally designed to work as a virtual switch in virtualized environments. OVS core is essentially formed by two entities that implements different functionalities (Fig. 1).

The *datapath* performs the main switching activity, that is taking network packets from an input port and managing it in some way, depending on what is specified in the *flow table*. For example, a packet could be simply forwarded into another port, or dropped, or be modified before being forwarded.

The *ofproto* module, instead, implements the switching logic. In other words, it is *ofproto* that tells the *datapath* what actions to associate to different flows by adding, removing or modifying entries in the *flow table*. This module also implements the remote configuration interface of the switch, or rather the OpenFlow protocol.

Currently two different implementation of the *datapath* are available, one that run completely in userspace and the second that exploits a special Linux kernel module.

2.1 The porting process

The porting process involved the userspace portion of the program. To make OVS run on FreeBSD, one of the lower layers of the program, the one in charge of passing packets from the NIC up to userspace and viceversa, had been reimplemented basing on *bpf* and *libpcap*.

During the porting process, some inefficiencies in the original implementation, that led to poor performance (about *65 Kpps*), became apparent. So, in the second part of the work, we analysed the program bottlenecks and designed and implemented possible solutions.

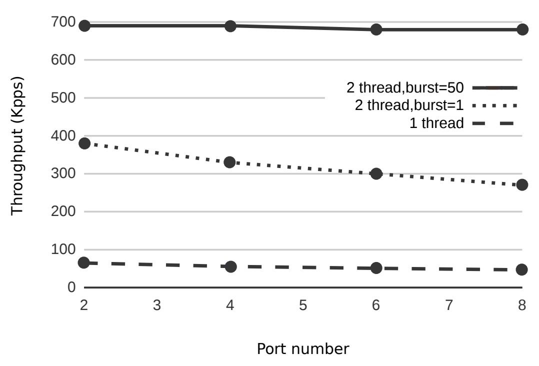

Figure 2: Forwarding performance of the optimized Open vSwitch code depending on the number of ports attached to the switch.

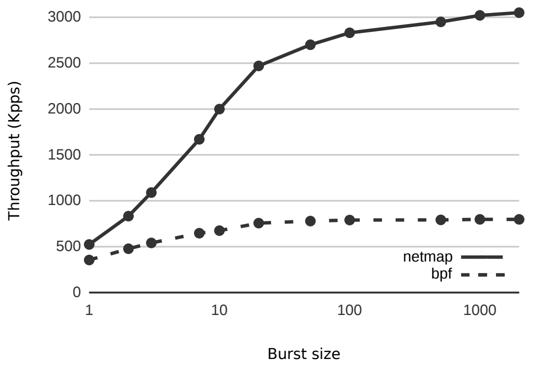

Figure 3: Forwarding performance of Open vSwitch on *netmap*.

2.2 Performance improvement

Analysing the program we discovered that its inefficiency could be attributed to an ineffective management of the system call `poll()`, and to the structure of the program main loop.

After some attempts of optimizing the library that provides support for poll operations, without any significant result, we decided to drastically revise the program structure, dividing it into two separate parts.

In the original implementation, all the *ofproto* and *datapath* activities were performed within a single process, even if they have evidently different requirements in terms of performance.

Our idea had been to split them into two separate threads so that the *datapath* should be able to run at a very fast rate interacting with *ofproto* only in rare cases. One possibility is that the *datapath* is not able to manage a packet, so it has to pass it to *ofproto* by means of a shared queue. Another possibility is that *ofproto* wants to modify the *datapath* flow-table in response to a message from the controller. In all cases the communication between the threads had been implemented with shared structures.

The *datapath* thread body, had been entirely reimplemented in order to optimize some operations. In particular, we eliminated the inefficiencies related to the management of the `poll()` system call, we improved the network packets reception exploiting the *burst* technique and finally we modified the program in order to avoid the memory copy for incoming packets.

As a result, the program performance increased more than ten times (Fig. 2), with peaks of about *700 Kpps*. At this point, performance was no longer limited by the program itself, but by the inefficiency of the kernel in providing high speed access to raw network packets.

2.3 *netmap* integration

Netmap [4] represents an excellent solution to the above mentioned problem. It is a recently developed framework that provides very fast access to network packets, essentially exploiting memory mapping, so that userspace programs can directly access the kernel packet buffers without the need of any memory copy.

Combining OVS with *netmap* had been the last step of the work. Thanks to the simplicity of its API, we had been able to implement a small library that maps *pcap* functions onto *netmap*. This way replacing *bpf* with *netmap* had not required any change to OVS code and, moreover, any existing application based on *pcap* should take the same advantage. The final results are very satisfying, since peaks of more than *3.0 Mpps* have been reached (Fig. 3). A more detailed performance analysis is presented in [5].

3. CONCLUSIONS

In this work we dealt with Open vSwitch. We started making the program executable on FreeBSD, reimplementing the module that abstracts network devices. In the first test session we obtained very poor forwarding performance (about *65 Kpps*). Consequently, we started inspecting the code, discovering some inefficiencies related to the management of the `poll()` system call by the program main loop.

In the second part of the work, we proceeded with the implementation of possible solutions for such issues, substantially separating the fast portion of the program from the slow one into two different threads. At the end of this process, we were able to reach throughput values as much as ten times greater than before (about *700 Kpps*), making OVS performance comparable with the kernel bridge one [1].

Finally, we combined Open vSwitch with a recently developed system, called *netmap*, that provides very fast access to network packets. Thanks to this last enhancement we touched very high packet rates, up to *3.0 Mpps*.

4. REFERENCES

[1] A. Bianco, R. Birke, L. Giraudo, and M. Palacin. *OpenFlow Switching: Data Plane Performance.*

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. *OpenFlow: Enabling Innovation in Campus Networks.*

[3] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. *Extending Networking into the Virtualization Layer.* HotNets-VIII, 2009.

[4] L. Rizzo. *netmap: fast and safe access to network adapters for user programs.* http://info.iet.unipi.it/∼luigi/netmap/.

[5] L. Rizzo, M. Carbone, and G. Catalli. *Transparent acceleration of software packet forwarding using netmap.* Tech. Report, University of Pisa.