A Transparent Highway for inter-Virtual Network Function Communication with Open vSwitch

Mauricio Vásquez Bernal[1], Ivano Cerrato[1], Fulvio Risso[1], David Verbeiren[2]
[1]Dept. of Computer and Control Engineering, Politecnico di Torino, Turin, Italy
[2]Tessares SA, Louvain-la-Neuve, Belgium, formerly at Intel Corporation NV/SA, Belgium
[1]{mauricio.vasquez, ivano.cerrato, fulvio.risso}@polito.it; [2]david.verbeiren@gmail.com

ABSTRACT

This paper presents a software architecture that can dynamically and transparently establish direct communication paths between DPDK-based virtual network functions executed in virtual machines, by recognizing new point-to-point connections in traffic steering rules. We demonstrate the huge advantages of this architecture in terms of performance and the possibility to implement it with localized modifications in Open vSwitch and DPDK, without touching the VNFs.

CCS Concepts

•**Networks** → *Middle boxes / network appliances;*

Keywords

NFV; Open vSwitch; DPDK; performance

1. INTRODUCTION

In Network Functions Virtualization (NFV), complex services can be delivered by rearranging multiple Virtual Network Functions (VNFs) in arbitrary *graphs* (Figure 1(a)), with multiple VNFs often executed on a single physical server as distinct machines (VMs). This paper presents a set of interacting software components that optimize the inter-VNF communications by creating a direct connection between two VMs, hence bypassing the vSwitch when two VNFs are logically connected through a point-to-point (`p-2-p`) link.

Differently from other proposals [1], we can accelerate *transparently* and *dynamically* the packets exchange between VMs on a *widespread* vSwitch. *Transparency* refers to the possibility for an application to exploit the advantages of our technology without even knowing it is there, and for an OpenFlow controller to attach to a vSwitch without noticing it has been modified. *Dynamicity* refers to the capability to either create a direct VM-to-VM channel or return to a traditional VM-to-vSwitch-to-VM path on the fly, based on the run-time analysis of OpenFlow rules. Finally, our idea has been integrated in a *widespread* vSwitch, particularly, it extends the version of OvS based on the Data Plane Development Kit (DPDK), and then it is oriented to optimize connections between VMs *executing DPDK-based network applications*, bringing its benefits to the entire class of the above VNFs.

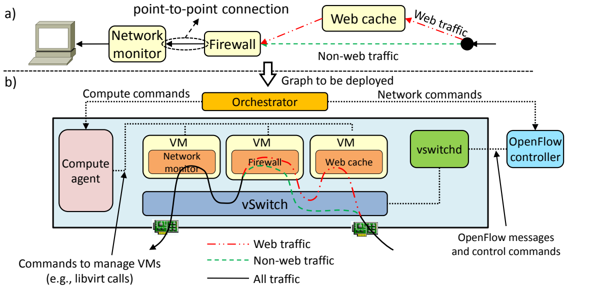


Figure 1: Traffic crossing VNFs: (a) the service graph; (b) its implementation on a server.

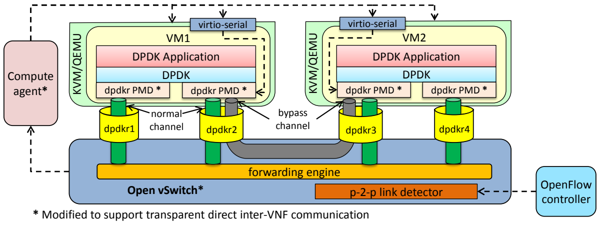
2. PROTOTYPE ARCHITECTURE

Our DPDK-based applications run inside VMs connected to the forwarding engine of OvS through `dpdkr` ports; this module handles packets according to the content of its forwarding table, which can be configured with OpenFlow `flowmods`. `dpdkr` ports are implemented using shared memory, and are exposed to the VM through `ivshmem` devices; moreover, applications access `dpdkr` ports using a poll mode driver (PMD).

As shown in Figure 2, our architecture modifies the `dpdkr` port to include a *normal* channel connected to the OvS forwarding engine and the optional *bypass* channel that is directly connected to another VM. Also the PMD has been modified, so that the same instance can handle both channels and expose them as a single `dpdkr`

SIGCOMM '16, August 22-26, 2016, Florianopolis , Brazil

DOI: http://dx.doi.org/10.1145/2934872.2959068



* Modified to support transparent direct inter-VNF communication

Figure 2: Overall software architecture.

port to applications, which are not aware of the actual implementation of that port. We also extended OvS with a new *p-2-p link detector* module, which analyses each `flowmod` received by the vSwitch in order to dynamically detect when a new `p-2-p` link between two `dpdkr` ports is either requested or removed.

When the VM is created (e.g., by the compute agent), it is connected to `dpdkr` ports that have only the *normal* channel. When the vSwitch detects the request to setup a `p-2-p` link between two VMs, it creates a new pair of `dpdkr` *bypass* channels mapped on the same piece of memory, shared by both communicating VMs. This way, the two VMs will be able to exchange packets without the intervention of the OvS forwarding engine.

The two new *bypass* channels are plugged in the proper VMs and assigned to the right PMD instance. Since OvS does not know which VM is attached to a specific port (it just knows ports and the rules used to forward packets among them), for these operations the vSwitch has to rely on an external component. Consequently, we modified the compute agent[1] to receive requests from OvS and: *(i)* plug the *bypass* channel (as an `ivshmem` device) into the VM by interacting with QEMU; *(ii)* configure the PMD instance to send/receive packets through the *bypass* channel, by means of a control channel based on a `virtio-serial` device. Notably, the PMD can still receive packets from the *normal* channel, hence allowing an OpenFlow controller to send `packet-out` messages to that port. Finally, when the *p-2-p link detector* recognizes that a `p-2-p` link no longer exists, the *bypass* channel is removed and the proper PMD instances are configured to use only the *normal* channel.

To maintain compatibility with external entities such as the OpenFlow controller, OvS exposes the two (*normal* and *bypass*) channels as a single (standard) `dpdkr` port, so that such entities can continue to issue commands involving `dpdkr` ports as they usually do (e.g., get statistics, turn them on/off), without noticing any change in their actual implementation.

Finally, in order to export statistics related to ports and flows implementing a `p-2-p` link, the PMD has been extended so that, each time a packet is sent through the *bypass* channel, it increases the counters associated to that OpenFlow rule and port, which are stored in a shared memory. When OvS needs to export statistics, it just reads the proper values from that shared memory. The vSwitch is in fact not able to count statistics related to `p-2-p` links by itself, as it is not involved in moving packets flowing through these connections.

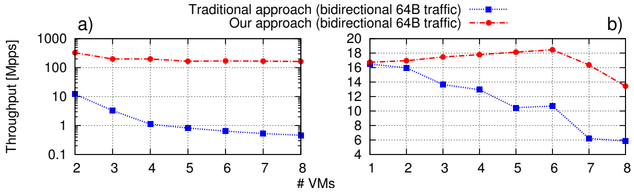


Figure 3: (a) memory-only; (b) NICs involved.

3. EXPERIMENTAL VALIDATION

We characterized our prototype on an Intel Xeon E5-2690 v2 @ 3GHz, equipped with two 10G Intel 82599ES NICs, comparing our approach with the vanilla OvS-DPDK. In all the tests, we consider chains of VMs connected only through `p-2-p` links, where each VM has two `dpdkr` ports and runs a single core DPDK application that moves packets from one port to another. Notably, thanks to the transparency of our technology, exactly the same VMs have been used in all the tests.

Figure 3 reports the throughput obtained with chains of growing length. Particularly, Figure 3(a) refers to the case in which the first and the last VM of the chain act as traffic source/sink; this test validates our approach without the NICs and PCI-e bus bottlenecks. Figure 3(b) refers instead to the case in which traffic is delivered/drained to/from the chain through the 10Gbps NICs. Both the tests show that a chain of VMs exploiting our technology provides better throughput than the same chain based on the vanilla OvS-DPDK.

Our prototype brings also advantages in terms of latency, especially with long chains (in case of 8 VMs, we get an improvement of 80%); however, due to space constraints, detailed results are not reported here.

Finally, the establishment of a direct channel between two VMs, from the moment in which OvS recognizes a `p-2-p` link, to the moment in which the PMD starts to use the *bypass* channel, is on the order of 100 ms.

Acknowledgments

This work was conducted within the framework of the FP7 UNIFY project (http://www.fp7-unify.eu), which is partially funded by the Commission of the European Union.

4. REFERENCES

[1] S. Garzarella, G. Lettieri, and L. Rizzo. Virtual device passthrough for high speed vm networking. In *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on*, pages 99–110, 2015.

[1] This prototype extends a special NFV node available at http://github.com/netgroup-polito/un-orchestrator.