vNS: a modular programmable virtual network switch

Massimo Gallo and Fabio Pianese
Nokia Bell Labs
first.last@nokia-bell-labs.com

CCS CONCEPTS

• **Networks** → **Network design principles**; **Transport protocols**; • **Software and its engineering**;

KEYWORDS

Network stack, Virtualization, Virtual Switch

ACM Reference Format:
Massimo Gallo and Fabio Pianese. 2018. vNS: a modular programmable virtual network switch. In *SIGCOMM Posters and Demos '18: ACM SIGCOMM 2018 Conference Posters and Demos, August 20–25, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3234200.3234242

1 INTRODUCTION

Virtual network switches are a staple component of Cloud environments involving virtual machines (VMs). They implement an efficient inter-VM communication facility, which the guest operating systems see as a standard network-based interface. In Cloud deployments, where a plurality of tenants each manage groups of several VMs on shared hardware, a virtual network switch is a great vantage point to supervise and optimize network resources. Given the popularity of Open vSwitch [7], research on architectures that expand the virtual switch potential has been gaining increasing attention. Recent literature highlights several cases in which a virtual switch might profitably exceed its role as a transparent communication interface, leading to systems that integrate various kinds of protocol processing.

For instance, Oko [1] can execute stateful filtering and monitoring functions over inter-VM packet flows. Two examples of virtual switches that transparently modify the congestion control behavior of traffic flows they carry and make optimization of end-to-end application performance possible, are Virtualized Congestion Control (VCC) and AC/DC TCP [2][5]. The most radical approach, adopted in NSaaS [8], aims to offload the entire transport stack to cloud provider controlled VMs requiring small modifications to the guest VM's to rely upon an external TCP socket implementation. These systems are beneficial for datacenter workloads, where congestion can severely degrade flow throughput, and help improve the behavior of end-to-end applications sensitive to latency, jitter, and loss.

We believe that existing virtual switches would be greatly enhanced by the integration of a full-fledged programmable packet processor between VMs and the physical network. By delegating complex network functionalities (e.g., L4 termination) to the virtual switch, the data plane can be efficiently tapped, leveraging zero copy and batching to boost performance. On one hand, similar to the systems cited above, packet processing can be handled in user space and lives outside the tenants' computing resources, ensuring safety and ease of upgrade of sensitive functionality by the service provider. On the other hand, the use of a modular and open-ended approach to packet manipulation [6] would retain the flexibility of general-purpose programming languages to implement complex and stateful virtualized network functions.

Accordingly, we are developing vNS , a system that turns the transparent switch of a virtualized server into a general-purpose flow processing platform based on ClickNF, a recent high-performance derivative of the Click software router [4]. This poster presents the design principles and architecture of vNS , along with some early results of the current prototype. We then discuss how vNS can benefit a number of use-cases that are relevant for virtualized Cloud environments.

2 DESIGN

Besides performing packet switching, to support the implementation of advanced networking functionalities (e.g., L4 termination and dedicated application-layer processing), a programmable virtual switch should *(i)* provide a flexible and extensible network stack (L2-L4) *(ii)* offer efficient communication primitives among the VMs and with the outside network, and *(iii)* conceal its presence to the applications deployed in the VM. The ClickNF framework [4] we developed is a great starting point for such an architecture, as it already provides a complete L2-L4 modular and efficient network stack and fast packet I/O via the DPDK library [3].

SIGCOMM Posters and Demos '18, August 20–25, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5915-3/18/08…$15.00
https://doi.org/10.1145/3234200.3234242

Fig. 1 shows vNS 's architecture as a graph of ClickNF modules. Applications run unmodified in VMs using a standard BSD Socket API. They are linked at runtime with *libns*, a dynamic library with two goals: emulating the standard socket API and driving the virtual network stack inside the virtual switch. More precisely, libns intercepts the BSD socket API calls and multiplexes/demultiplexes control packets sent to/received from the virtual network stack. Control packets simply consist in socket API commands encapsulated in ClickNF packet data structures, e.g., `|socket|domain|type|protocol|` corresponds to `socket(int,int,int)`. Each control packet triggers a response by the network stack deployed in vNS , a packet reports back the results, e.g., `|socket|ret|err` is used to communicate the result of a `socket()` call. In this way, VM applications can operate exactly as if the network stack were deployed locally.

The main component of vNS is a ClickNF instance running in the host. For efficiency, we develop *vHost* a Click element that uses the DPDK vhost library [3], allowing the switch to efficiently move packets from/to the VMs without incurring in copies and taking advantage of batching. A critical part of vNS is the *vApp* element in charge of executing and responding to the control commands sent by guest VMs' applications, and notifying them of network-initiated changes in the local virtual stack, e.g., when a connection is closed by the remote peer. In particular, vApp uses an event queue to register relevant events triggered by control packets, e.g., a `recv()` command issued by a VM's application or a `RST` packet coming from a remote peer that communicates with some process in a VM. A ClickNF blocking task waits for incoming events and `yield()`s back the CPU when done.

Each VM is connected through a pair of vHost / vApp elements to a dedicated *cTCP* instance, the modular network stack implementation in ClickNF.vNS implements packet switching and fast I/O by means of the *DPDK* and *Etherswitch* elements provided by ClickNF and legacy Click respectively.

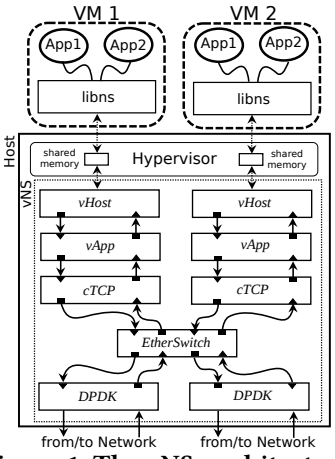

Figure 1: The vNS architecture.

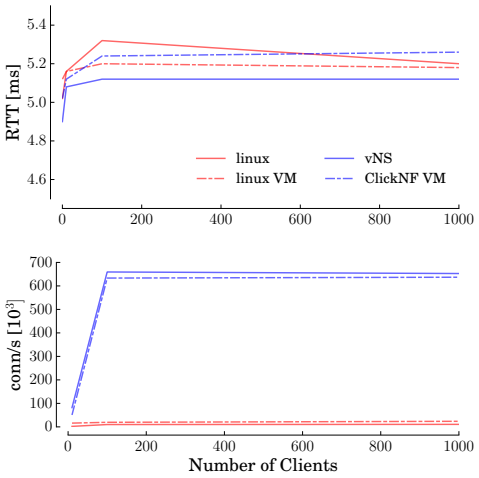

Figure 2: Comparison between vNS and Linux stack.

3 PRELIMINARY RESULTS

We prototyped vNS by developing two Click elements, vApp and vHost, and a preliminary version of libns. To test the architecture's feasibility and potential, we run a simple experiment in which an increasing number of clients connects to a server running in the VM with network stack delegated to a single core vNS instance. After the connection is established, the remote peer closes it without exchanging any data. We compare our solution against more traditional architectures where the network stack, the linux kernel's or the optimized ClickNF one, and the server are co-located and run in the host (linux) or in the VM (linux VM, ClickNF VM). Our testbed consists of 2 machines with 40-core Intel Xeon® 2.60GHz processors, 64 GB RAM, and 10 GbE Intel® 82599ES cards, running Ubuntu 16.10, ClickNF, and DPDK 17.11. The VM features a single core, 1GB of RAM, Ubuntu 16.10 and runs on top of the QEMU KVM 2.6.1 hypervisor.

Fig. 2 (top) presents connection establishment RTT (the delay between SYN and SYN-ACK) which generally increases with the number of clients in the scenario. When the stack is executed inside the virtual switch, a small performance benefit is observed. This is due to the fact that connections are terminated inside the virtual switch, which deals with TCP connection setup packets without involving the VM. Fig. 2 (bottom) shows the number of connection setups per second handled with increasing numbers of parallel clients. As expected, a ClickNF stack optimized for performance sustains a much higher conn/s load compared to linux, until the clients saturate the CPU processing power (>100 clients).

We are currently progressing toward the complete implementation of vNS including further testing and comparisons with alternative approaches. Moreover, we plan to investigate the management and control of several co-located vNS instances and to introduce dynamic reconfiguration.

REFERENCES

[1] Paul et al. Chaignon. 2018. Oko: Extending Open vSwitch with Stateful Filters. In *Proc. ACM SOSR'18.*

[2] Bryce Cronkite-Ratcliff et al. 2016. Virtualized Congestion Control. In *Proc. ACM SIGCOMM'16.*

[3] Linux Foundation. 2018. DPDK framework. (2018). http://dpdk.org.

[4] Massimo Gallo et al. 2018. ClickNF: a Modular Stack for Custom Network Functions. In *Proc. of USENIX ATC'18.*

[5] Keqiang He et al. 2016. AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks. In *Proc. ACM SIGCOMM'16.*

[6] Eddie Kohler et al. 2000. The Click Modular Router. *ACM Trans. Comput. Syst.* (2000).

[7] Linux Foundation. 2009. Open vSwitch. (2009). http://openvswitch.org.

[8] Zhixiong Niu et al. 2017. Network Stack As a Service in the Cloud. In *Proc. of ACM HotNets'17.*