The Performance Evaluation of Netmap Software Switch

Hasan Redžović, *Junior Member, IEEE,* Aleksandra Smiljanić, Member, *IEEE*

***Abstract*—Software switches can provide high level of flexibility and are commonly used in software defined networks. In recent years, development of software routers has shifted towards fast network I/O platforms such as netmap and DPDK that bypass slow kernel network stack. Recently, VALE software switch based on netmap platform has been improved with extension called mSwitch. The VALE architecture is redesigned and divided into two fundamental parts: switching fabric that forwards packets and switching logic module that determines destination port. In this paper, we present the mSwitch (VALE) architecture and guidelines for developing switching logic module. Through extensive tests, we determine performance of packet forwarding through mSwitch.**

***Index Terms*—mSwitch; fast network I/O; network performance evaluation; linux kernel modules.**

I. Introduction

Software switches are becoming increasingly significant for various network environments such as software defined networks (SDN) and virtualized data centers. The virtual machines (VMs) usage in data centers significantly increases the number of virtual ports. Usually, hypervisors of virtual machines use software switches to forward packets between VMs and network through virtual and physical ports. Software switches are well-suited for forwarding packets between virtual ports and they are becoming more important part of the network as the number of virtual ports increases.

Also, SDN rely their development on highly flexible software switches which allow research on commodity servers and easy modification of switching fabrics while retaining high packet rates. Software switches are frequently modified to add new protocols or to improve measurement and debugging features, and to add middlebox-like functions. These various network environments impose important requirements that software switches need to fulfill such as programable data plane, high packet rates, data security, efficient CPU usage and high port density. These features conflict each other and it is difficult to harmonize them. For example, flexibility and simple programable data plane improve speed of development cycles but other requirements such as data security should not be affected. Typically, software switches are based on a large programing code which includes kernel code, and changing the switch is a hard process that required expertise in many different fields of network design and development. The examples of this type of software switches are Linux bridge module [1], FreeBSD switch [2] and Open vSwitch [3], and they all use kernel network stack as switching fabric to forward packets between ports [4].

The kernel network stack is composed of complex programing code which is designed to support wide variety of implementations and it is not optimized for high throughput NICs. Consequently, software switches based on kernel network stack neither provide high packet rate nor adequate flexibility. For this reason, different specialized frameworks emerged such as netmap [5] and data plane development kit (DPDK) [6]. These frameworks bypass kernel network stack and provide new, notably faster, I/O packet platform for network applications. These fast I/O platforms deliver unprocessed packets directly to network applications in user space, where just necessary logic can be implemented avoiding any redundant programing code. The software switches developed using netmap or DPDK platform can outperform software switches based on kernel network stack in various aspects, and, therefore, have higher potential for wide implementation [7]. There are two software switches based on DPDK platform: CuckooSwitch [8] and DPDK vSwitch [9]. The software switch developed on netmap platform is called VALE [10]. Recently, a VALE extension called mSwitch was added which redesigned architecture and implemented new features in order to achieve requirements of virtualized data center and SDN [11].

In this paper, we examine and analyze fast I/O platforms and describe what are benefits and challenges of these platforms. Then, we present detailed architecture of mSwitch (VALE) which contains two main parts: 1. switching fabric that is optimized for fast packet forwarding and port scaling; 2. switching logic that performs lookup and port configuration. Development guidelines for the switching logic module are described. Then, we, performed large number of tests using different network configurations to analyze various mSwitch features and to evaluate packet forwarding.

The paper is organized as follows. The netmap and DPDK platforms are presented in Section 2. Sections 3 describes mSwitch architecture and how to create switching logic. In Section 4, we present testing environment and described conducted tests. Section 5 analyzes and evaluates performance of mSwitch. Finally, Section 7 concludes the paper.

II. Fast I/O network platforms

Unfortunately, the packet throughput of Linux kernel network stack is not sufficient for more specialized workloads. The low performance is due to the complex kernel code which contains large number of systems calls

Hasan Redžović is with the Innovation Center of School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: hasanetf@live.com).

Aleksandra Smiljanić is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: aleksandra@etf.rs).

per packet, unnecessary packet processing procedure and data copying. The Linux kernel can process only about 2-3 million packets per second (Mpps) on server with moderate hardware components. This is not enough for software switches and other network applications, especially since the network cards are capable of handling much higher throughputs. The performance limitations of the Linux kernel network are nothing new, and, over the years, the specialized platforms were developed to overcome this problem. The most commonly used fast I/O platforms are DPDK and netmap. Mentioned platforms essentially rely on similar approach in accelerating packet throughput – bypassing kernel network stack and providing unprocessed packets directly to applications in user space. However, there are differences between DPDK and netmap.

DPDK uses a number of optimizations such as huge pages, vector instructions, direct cache access, and, provides direct hardware access to/from user space (UIO). DPDK uses a run-to-completion scheduling model. This scheduling model means that NIC devices are accessed in polled mode without any interrupts on the fast path. The downside of DPDK design is untrusted interconnection between clients in DPDK-based switches. DPDK relies on multiqueue NICs with single root I/O virtualization (SR-IOV) to implement a protected data path. SR-IOV allows a single physical port to be shared between multiple virtual machines. However, SR-IOV has some disadvantages such as necessary device support, VLAN filtering limitation to only 64 entries for certain Intel drivers, etc. Also, DPDK platform is using active polling which not only keeps processor cores always busy, but also makes it difficult to scale to large number of ports. DPDK performs unnecessary polling of idle ports. Two switches based on DPDK are CuckooSwitch and DPDK vSwitch. CuckooSwitch achieves high throughput when handling very large numbers of L2 rules, but it has high CPU utilization, does not support virtual ports, and does not have flexible switching logic because it targets replacing hardware Ethernet switches. Also, CuckooSwitch source code is not available. DPDK vSwitch takes the Open vSwitch code base and accelerates it by using DPDK, inheriting the disadvantages of DPDK already mentioned.

The netmap platform is implemented with several kernel modules, and it uses shared unpaged memory to exchange packet data with applications in user space. The netmap creates Rx/Tx rings (netmap rings) in shared memory that replicate NIC rings. Through these netmap rings, application access the packets in user space, while NICs access the packets through original rings. The rings comprise slots that contain memory locations of packets. The access permission to the packet memory locations is handled by dynamically labeling two slots as head and tail. NICs access slots ranging from tail to head slot clockwise, and network applications access slots from head to tail slot clockwise. The relabeling of head and tail slots is triggered by calling system function from user space, and this procedure is called ring synchronization. Time-consuming system functions are only used during ring synchronization, and their impact on performance is mitigated by large number of processed packets. Also, in most cases of packet processing and forwarding, zero-copy optimization can be achieved by exchanging memory location between Tx and Rx rings. Unlike DPDK which relies on SR-IOV, netmap platform uses shared memory and kernel modules to provide secure access between netmap application and NICs ports. Also, there are no unnecessary polls of idle resources, enabling netmap applications to better utilize CPU cores and, in the case of software switches, to scale larger number of ports, when compared to DPDK. Although DPDK provides slightly better performance, netmap platform fulfill better other requirements of software switches such as efficient CPU usage and data security.

The switch based on netmap platform is called mSwitch (VALE), and, recently, it was improved and redesigned to have the following features:

- The mSwitch architecture is divided into two fundamental parts: data plane that switches packets between the ports and switching logic that decides on the packet’s destination port. The clear separation between data plane and switching logic increases flexibility and provides easer programmable interface for implementing new network features;
- Improved scalability of virtual ports, so mSwitch can support up to 120 ports. The scalability is important as the number of virtual ports is constantly increasing;
- Optimization of parallel access to a single port which enables high throughput when multiple senders collide onto single port.

The mSwitch redesigns and improves software switch that could possibly meet all major requirements, and, it can be notably competitive to other software switches. In the following section, we described mSwitch architecture and how to create a simple switching logic.

III. MSWITCH ARCHITECTURE AND SWITCHING LOGIC

As we mentioned before, the main mSwitch modification is dividing architecture to switching fabric that forwards packets between ports and switching logic that implements lookup and port configuration. This separation supports mSwitch with fixed and stable high throughput, while providing users with customable and easy to use switching logic. Fig. 1 shows the mSwitch architecture with virtual and physical ports. The switching logic views abstract representation of each connected port as a unique index number in the switch. When a packet arrives to a port, switching logic uses arbitrary packet processing function to modify the packet if necessary and to determine its destination port. The virtual ports connected to the switch can provide access to different types of applications in user spaces:

- The netmap applications – applications designed to use netmap API can also be connected to virtual ports, allowing multiple netmap applications to access single physical port. Without mSwitch, only one netmap application can use physical port at any single point in time;
- QEMU virtual machines – QEMU hypervisor is modified to use netmap API and allows QEMU instances to be connected to the virtual ports;
- All regular user space application – kernel network stack can be attached to virtual ports allowing all

applications to communicate with the network through mSwitch. In this case, packets are traversing not only through mSwitch but also through kernel network stack which leads to low performance.

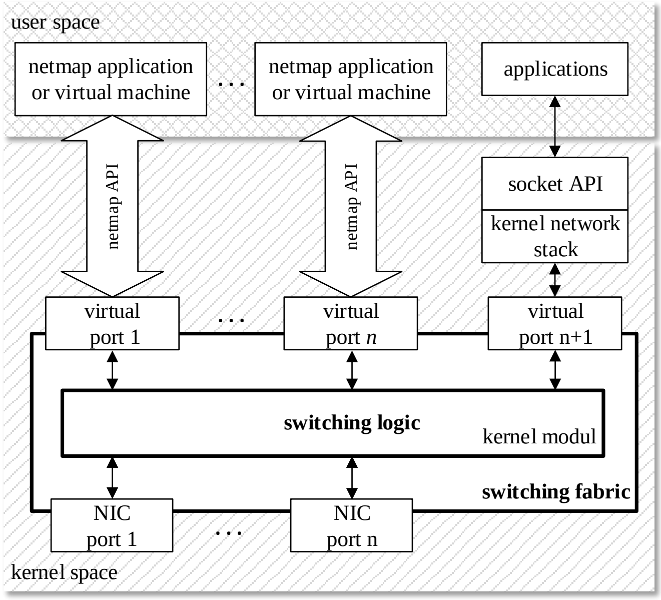

Fig. 1 mSwitch architecture is divided into two basic parts: switching fabric and switching logic.

Physical ports are connected to the mSwitch using modified version of netmap API, providing much higher performance than the default drivers. Also, mSwitch utilizes receive side scaling technology implemented as a part of modern NICs that provides multiple packet queues. Each queue is mapped onto one ring that can be assigned to separate CPU core in order to scale the performance. Depending on a packet receiving port, packet forwarding is performed with the corresponding thread. User application threads execute packet forwarding for virtual ports and kernel threads execute forwarding of packets arriving to physical ports or kernel network stack. Thus, the multiple threads may contend for access to destination ports. The forwarding algorithm differs significantly from previous version of VALE in order to scale to large number of ports.

The switching logic runs as a kernel module that can register to the switching fabric three custom function: lookup, port configuration and callback function in the case if some process dies, i.e., stops working. Fig. 2 shows structure of the kernel module implementing switching logic. Each kernel module contains two basic function: (i) Initialization function to register kernel module into kernel space and to assign that module to some device or process; (ii) Termination function to unregister kernel model and release all allocated resources.

In the case of switching logic, the kernel module is assigned to mSwitch. The custom functions are registered using netmap API and structures provided by mSwitch. Thus, in order to compile switching logic as a kernel model it is necessary to provide not only kernel source code but also netmap source code. Registered custom functions are used according to the event driven principle – if packet arrives, the lookup function is used, if application sends configuration request, this request is processed by the configuration function, and if some port stops working the callback function reconfigures the resources. It is not mandatory for the switching logic to contain all three types of functions – if some of these custom functions are not provided, mSwitch will use the corresponding default functions. The mSwitch default lookup function is the learning bridge called netmap_bdg_learning. The termination function, shown in Fig. 2, unregisters custom functions by passing function pointers set to NULL to the switching fabric. Also, the termination function must release all allocated memory in kernel space or otherwise the memory will be lost until the system is rebooted.

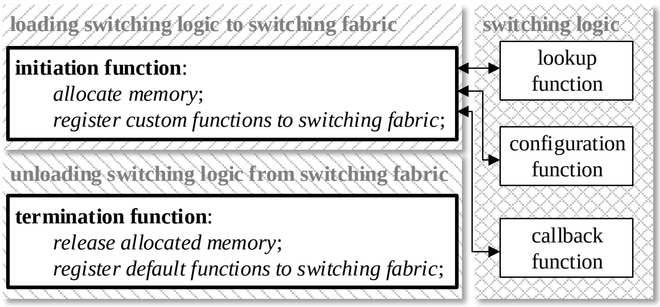
loading switching logic to switching fabric
initiation function:
allocate memory;
register custom functions to switching fabric;
unloading switching logic from switching fabric
termination function:
release allocated memory;
register default functions to switching fabric;
switching logic
lookup function
configuration function
callback function

Fig. 2 Structure of switching logic kernel module.

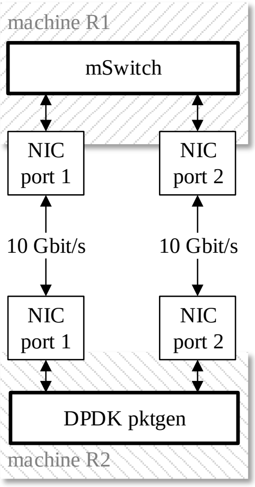

Fig. 3 Testing environment comprised two physical machines connected with 10 Gbit/s ports.

We created simple switching logic module called swlog_bridge in order to test mSwitch packet throughput. The swlog_bridge comprises a basic lookup function that forwards packets from some port to another predetermined port. Our goal is to test performance of the switching fabric in order to determine how redesigned and improved features of mSwitch affect its overall performance. In the following sections, we describe conducted tests and analyze results.

IV. Testing environment

Our testing environment shown in Fig. 3 consists of two physical machines connected with two 10 Gbit/s links. Tests were performed on machine R1 where netmap platform and mSwitch were implemented. Application DPDK pktgen was implemented on machine R2 and used as a packet generator and measurement tool. The DPDK pktgen can generate different types of packets at the maximal packet rate of 10 Gbit/s links (equivalent to 14.88 Mpps) and it also supports ARP protocol, which was useful for some of our tests. On both links, traffic was generated with a total maximal packet throughput of 29.76 Mpps. Essentially, in all tests, we measure the overall packet forwarding rate for different configurations. All test configurations are depicted in Fig. 4:

- Test A: NIC ports are directly connected to kernel network stack;
- Test B: NIC ports are connected to mSwitch with default lookup function netmap_bdg_learning;
- Test C: NIC ports are connected to mSwitch and to kernel network stack through corresponding virtual ports. mSwitch uses default lookup function netmap_bdg_learning;
- Test D: Same configuration as for test C, but swlog_bridge module was used as switching logic instead of default mSwitch functions. The swlog_bridge is configured to forward packets between NIC ports;
- Test E: Same as previous configuration, but the swlog_bridge is configured to forward packets exclusively to virtual ports;
- Test F: Netmap bridge application is directly connected to NIC ports using netmap API.

In following section analyze results of described tests.

V. Performance evaluation

The results of all tests described in previous section are shown in Fig. 5. Test A determines kernel network stack packet throughput without netmap API and mSwitch. The results of test A represent a benchmark for comparing all of the following measurements and determining how different configurations affect performance. The kernel network stack was able to forward 1.5 Mpps.

Test B measures packet throughput of mSwitch with default lookup function netmap_bdg_learning. In this test mSwitch was able to achieve 28.1 Mpps which was 18 times faster processing than of the kernel network stack.

In test C, kernel network stack was connected to mSwitch through the corresponding virtual ports. In this configuration, NIC port 1 and virtual port 1 have the same MAC addresses and this also applies to NIC port 2 and virtual port 2. When packet arrives on either of NIC ports, mSwitch default lookup function forward them to both virtual port and other NIC port. For example, if packet arrives on NIC port 1, it will be forward to virtual port 1 and NIC port 2. In this configuration mSwitch was able to forward 5.3 Mpps which was 3.5 times higher throughput than in the case of the kernel network stack, but 5.3 time lower than in the test B configuration. The configuration in test C is important because it allows all applications in user space to communicate with the network while providing 3.5 times higher packet forwarding throughput for rest of the traffic.

In test D, the configuration is the same as in test C but we used our switching logic swlog_bridge instead of netmap_bdg_learning. The swlog_bridge was adjusted to exclusively forward packet between NIC ports and ignore virtual ports. In this case, mSwitch was able to forward 28.05 Mpps, which was almost the same as in test B. This result shows that idle virtual ports do not noticeably affect the performance.

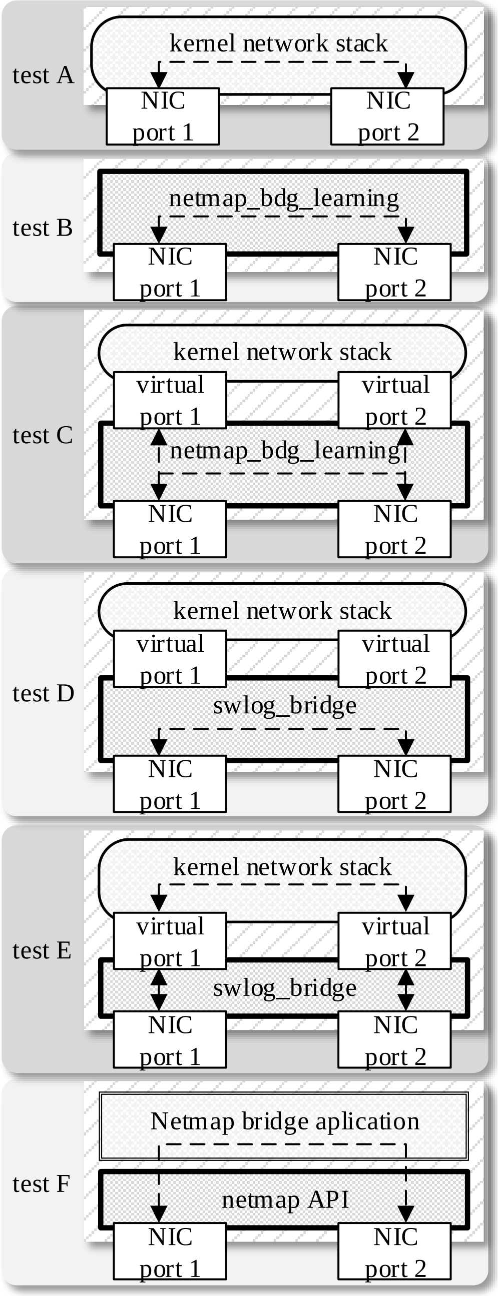

Fig. 4. Configuration of all conducted tests.

Test E measures packet throughput when swlog_bridge is adjusted to exclusively forward packet between NIC port and the corresponding virtual ports. In this case, packets are transferred between NIC and kernel network stack through mSwitch. The packet forwarding was done in kernel network stack and maximal packet throughput was 0.7 Mpps. This was 2.1 times lower that in the test A. The test E configuration force packets through the switching fabric and kernel network stack which lowers the performance.

Finally, test F measures netmap application performance without mSwitch, where applications are attached directly to

NIC port using netmap API. The netmap bridge application was used that are very similar to swlog_bridge and netmap_bdg_learning. The maximal measured packet throughput in test F was 28.5 Mpps which is almost identical as results in tests B and D. The test F shows that network applications such as bridge based on the netmap API can have equal performance as the switching logic module of mSwitch. However, mSwitch has advantages as it can connect regular network stack to the ports.

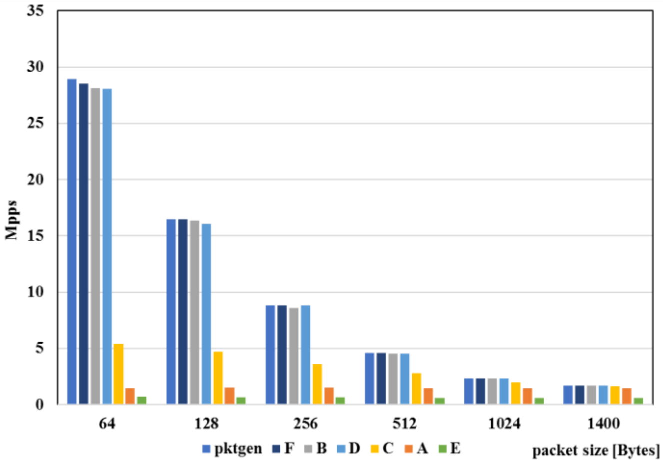

Fig. 5 Packet throughput for different configuration of machine R1.

VI. Conclusion

We have presented significance of software switches as a part network environment such as SDN and virtualized data centers. The kernel network stack has low packet processing performance due to the complex and non-optimized code and, as such, it is not a suitable environment for developing high-end software switches. We described most commonly used fast I/O platforms, and what are the basic challenges in creating software switch using these platforms. We have selected to analyze mSwitch as a software switch that is trying to fulfill all network requirements (flexibility, high throughput, low CPU utilization, high port density, etc.). The mSwitch architecture and its switching logic were described in details. Through performance evaluation of relevant scenarios, we have shown that mSwitch can achieve high packet forwarding throughputs. Compared to pure netmap solutions, mSwitch is able to connect regular network stack to the network as well. In this way, it allows utilization of legacy applications.

Acknowledgment

This work was supported by the Serbian Ministry of Science and Education (project TR-32022), and companies Telekom Srbija, and Informatika.

References

[1] "Bridge," Linux Foundation, 2016. [Online]. Available: https://wiki.linuxfoundation.org/networking/bridge

[2] "Bridging Advanced Networking," FreeBSD, [Online]. Available: https://www.freebsd.org/doc/handbook/network-bridging.html.

[3] "Open vSwitch," Linux Foundation, 2016. [Online]. Available: http://openvswitch.org/.

[4] P. Emmerich, D. Raumer, F. Wohlfar and G. Carle, "Performance Characteristics of Virtual Switching," in *3rd International Conference on Cloud Networking (CloudNet)* , Luxembourg, 2014.

[5] L. Rizzo, "netmap: A Novel Framework for Fast Packet I/O," in *USENIX*, Boston, 2012.

[6] Intel, "Data Plane Development Kit," [Online]. Available: http://dpdk.org/.

[7] G. Pongracz, L. Molnar and Z. L. Kis, "Removing Roadblocks from SDN: OpenFlow Software Switch Performance on Intel DPDK," in *European Workshop on Software Defined Networks (EWSDN)*, Berlin, 2013.

[8] D. Zhou, B. Fan, H. Lim, M. Kaminsky and D. G. Andersen, "Scalable, High Performance Ethernet Forwarding with CuckooSwitch," in *Conference on emerging Networking EXperiments and Technologies*, Santa Barbara, 2013.

[9] "Intel DPDK vSwitch Getting Started Guide," 2013. [Online]. Available: https://01.org/sites/default/files/downloads/packet-processing/intel_dpdk_vswitch_050f.pdf. [Accessed 2017].

[10] L. Rizzo and G. Lettieri, "VALE, a Switched Ethernet for Virtual Machines," in *USENIX Association is the Advanced Computing Systems Association*, Boston, 2012.

[11] M. Honda, F. Huici, G. Lettieri and L. Rizzo, "mSwitch: a highly-scalable, modular software switch," in *ACM SIGCOMM Symposium on Software Defined Networking Research*, Santa Clara, 2015.