Software Switch Performance Factors in Network Virtualization Environment

Yimeng Zhao, Luigi Iannone and Michel Riguidel
INFRES ,Telecom Paristech, Paris
Email: {yimeng.zhao, luigi.iannone, michel.riguidel}@telecom-paristech.fr

***Abstract*—The maturity of Software Defined Network (SDN) and network virtualization has inspired the development of software switch. This paper studies the behavior of softswitch in network virtualization environment. We investigate main performance factors and evaluate their impacts. Based on these measurements, we extend resource allocation among multiple softswitch instances with consideration of topology design. Simulation results show that proper coordination between resource allocation and topology greatly improves performance.**

***Index Terms*—Software switch, Network virtualization, Software Defined Network, Resource allocation.**

I. Introduction

Recently there is a growing interest in programmable software forwarding devices along with the maturity of Software Defined Network (SDN) and emergence of Network-as-a-Service (NaaS)[1]. Software switch (softswitch) provides agile upgrades and rapid evolution, and even can be extended to fulfill part of application logic in cloud. Moreover, in terms of lowering OPEX, software implementation is more competitive than dedicated hardware. In summary, softswitch brings flexible implementation and deployment model.

In this paper, the term "softswitch" refers to the software switch built on commodity x86 server. Compared with dedicated hardware, softswitch has performance concern. Many efforts attempt to boost softswitch performance from various aspects. For instance, OpenvSwitch[2] applies kernel-built-in fast forwarding path and supports multi-thread. Data plane tools like Netmap[3] and Data Plane Development Kit (DPDK)[4] aim to improve the efficiency of interaction with OS kernel and NIC. Different from previous works that only focus on improving individual performance, we aim at the orchestration of multiple softswitch instances within limited resources from a global view. This can be treated as a resource allocation problem that has been widely studied in cloud, such as VM placement or virtual network embedding[5]. This paper demonstrates the specific resource allocation work based on the features of softswitch in network virtualization.

In order to optimize resource allocation, it is essential to understand performance factors and their impacts. Since former benchmark tests only rely on partial parameters, we carry out a systematic measurement that highlights various performance factors. Numerical results are shown to evaluate their impacts. We discuss the cause of performance issues on softswitch and show improvement directions. A new deployment model for softswitch is necessary to combine its features instead of traditional hardware models. For example, in multiple softswitch instances scenarios, resource allocation should take topology design into consideration.

The rest of the paper is organized as follows, in section II, we show the main performance factors of softswitch highlighting concerns and proposing new deployment models. In section III, we combine resource allocation algorithm with topology customization among multiple softswitch instances in service chaining scenario. Finally the conclusion and future work are summarized in sections IV.

II. Softswitch Performance Factors

In this paper, we only cover SDN softswitches that follow OpenFlow specifications. Because their behaviors are well defined and regulated, which is beneficial to lead to general conclusion. We choose two typical OpenFlow compatible softswitches, namely OpenvSwitch and CPqD OpenFlow softswitch (OFsoftswitch)[6]. OpenvSwitch is the kernel-level implementation which supports multi-thread processing. OFsoftswitch is the user-space implementation that supports OpenFlow specification 1.3 but runs only in single thread.

Fig.1 summarizes a standard flow chart for soft packets processing according to OpenFlow. The theoretical time for processing a packet should be: $T = t_{polling} + t_{I/O} + t_{rules} + t_{actions} + t_{overhead}$. Softswitch uses polling mode to check available data on ports. $t_{polling}$ represents this time cost for polling check. If new packet arrives on the port, I/O module first reads the packet from that port into memory. Then this packet is matched with rules in flowtable in order of priority until a rule is matched, and t_{rules} refers to total matching time. Next, related actions listed in matched rule are applied to packet, which time is represented by $t_{actions}$. Finally, I/O module sends the packet out from memory. After processing on one port, softswitch loops back to polling step and repeats above process on next port. $t_{polling}$ is treated as an overhead, as it always exists no matter whether there is available packet. In each polling round, there exists another fixed overhead $t_{overhead}$. It is due to software design like recycling resources or registering event handler. All these overheads are marked as gray in flow chart. According to flow chart, we divide total time cost into independent parts and measure their performance factors respectively. We use emulation-based evaluation for our softswitches in Mininet[7]. Simulation platform is built on Intel E5-1620 with 4 cores each running at 3.6 GHz.

978-1-4799-6204-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ICNP.2014.69

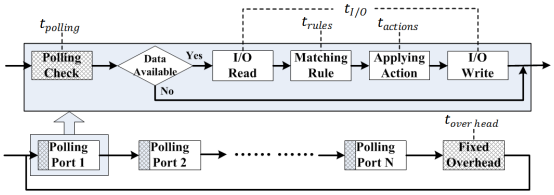


Fig. 1. Packet processing flow chart.

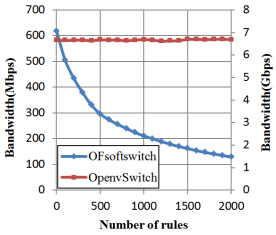


Fig. 2. Impact of packet size

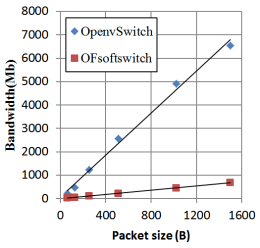


Fig. 3. Impact of rules

A. I/O

We measure the influence of packet size on I/O. "Iperf" is used for TCP traffic bandwidth test. In spite of the performance difference due to implementation, we find that maximum bandwidth is proportional to packet size in both cases of OpenvSwitch and OFsoftswitch. The degree of linear fitting is over 99.5% and 99.95% respectively. So packets per second (pps) value stays relatively stable regardless of packet size.

B. Number of rules

The number of rules has a great effect on performance due to lookup operation. OFsoftswitch uses linear lookup, hence t_{rules} is proportional to the number of rules it matches. We force each incoming packet to be compared with given number of rules before matching one. Fig.3 shows the relationship between the number of rules and bandwidth. We further deduce that the time for matching single rule: $t_{rule} \approx 20ns$.

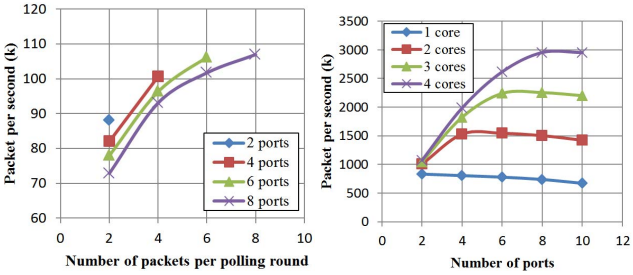
OpenvSwitch uses two-layer rule matching: a kernel-built-in hashtable and a user-space daemon. The kernel hashtable works as a cache to store recently matched rules for fast lookup, while user-space daemon further handles unmatched packets in kernel. In this measurement, we only focus on the impact of rules in kernel hashtable and avoid the interaction with user-space. The hashtable should guarantee that lookup time is a constant value regardless of number of rules as long as the packet hits hashtable. This can be proved by stable maximum bandwidth of OpenvSwitch in Fig.3, since the number of rules has no impact on total throughput.

C. Action on packet processing

OpenFlow allows switch to modify packet before sending it out. The applied actions are various that cover from basic operations like rewriting headers to complex ones like encapsulation. In softswitch, the time cost of action depends on its complexity. But in measurement, we find that the complexity of actions shows no impact on performance. Because compared to all of the other work that needs to be done during packet processing, this overhead is negligible. Gross et al.[8] obtain similar results, where OpenvSwitch can achieve equivalent performance with the existence of STT tunneling.

TABLE I
IMPACT OF CPU FREQUENCY.

Traffic load	CPU Frequency	RTT Delay	
		OpenvSwitch	OFsoftswitch
Light	1.2 GHz	0.035ms	0.142ms
Heavy	3.6 GHz	0.007ms	0.032ms



(a) OFsoftswitch (b) OpenvSwitch

Fig. 4. Impact of traffic pattern.

D. CPU frequency

Intel CPUs have features to change its running frequency according to workload. This behavior brings benefits in energy saving, but results in unstable performance. As shown in Table I, both OpenvSwitch and OFsoftswitch reach nearly 5 times performance increase on RTT delay in heavy traffic load than light load, which is contrary to common intuition. Because light traffic load motivates CPU to lower its running frequency at only 1.2GHz, while the frequency is 3.6GHz in heavy traffic load. There is a tradeoff between energy saving and high performance. Thus the deployment of softswitch should clarify its consideration on both QoS and energy aspects.

E. Polling & Overhead

Different from hardware implementation where multiple ports can run in parallel, softswitch uses polling mode on checking ports. Polling check introduces additional overhead. In order to maximize the total packets softswitch can process, it should try to send out packets as many as possible to lower the share of overhead.

In OFsoftswitch, when the number of ports is fixed as n, the theoretical effective packet processing load ratio is:

$$r = \frac{ppr \times t_{processing}}{n \times t_{polling} + ppr \times t_{processing} + t_{overhead}}$$

where $t_{processing} = t_{I/O} + t_{rules} + t_{actions}$ and $ppr \leq n$. The total throughput is proportional to r. The results in Fig.4(a) show that when the number of ports is fixed, the larger the ppr, the more throughput. When ppr is equal to n, the increase of n make r tend to reach upper limit as $t_{processing}/(t_{processing} + t_{polling})$. Accordingly in Fig.4(a), when n is over 6, the total throughput tends to converge to a constant value.

OpenvSwitch improves polling mechanism and eliminates the performance degradation about ppr on single core. However, its performance is still affected by the number of ports

on multi-core system. In Fig.4(b), more cores lead to better performance. When the number of cores is fixed, the maximum throughput is achieved when the number of ports is two times the number of cores, and excessive ports result in slight performance degradation. This is the result of the mapping system between multiple cores and multiple ports.

III. Resource allocation

In this section, we extend our vision from individual softswitch to global optimization among multiple instances. The deployment of softswitch is still challenging. First, softswitch requires large amount of computation resources. In order to realize 10 Gbps line rate forwarding speed on single port, one CPU core running at 3 GHz is needed[9]. Second, in NaaS which provides network programmability for tenants, there should be isolated softswitch instances deployed for each tenant. It is common for multiple softswitches sharing limited physical resources, which shifts performance issue to a more critical level. Third, NaaS considers global performance optimization between network and applications as a whole. Hence softswitch needs specific resource allocation mechanism to contribute to global optimization. Instead of over provisioning of physical computation resources, we aim to orchestrate multiple softswitch instances within limited resources to achieve optimized global throughput.

On multi-core system, the process is distributed among multiple cores dynamically. This introduces additional overhead for switching among cores and prevents flexible and accurate control on CPU resource. Thus customized and fine control mechanism on CPU is necessary for performance improvement. Moreover, former section shows that the network topology also influences performance, as it defines the connection graph and ports configuration. This inspires us to combine topology design with resource allocation coordinately for optimization in scenarios like NaaS where we can customize the network topology. To formulate this optimization problem: 1) according to given task, we pick out representative topologies among all possible ones; 2) in each selected topology, we apply resource allocation mechanism; 3) we choose the best combination of topology and resource allocation mechanism.

We demonstrate the effectiveness of coordination between topology design and resource allocation in service chaining scenario. The given task is described as follows: the network service region is constituted of 3 intermediate switches (S1, S2, S3), and S1 is ingress switch while S3 is egress. 2 service functions (F1, F2) are needed following in order as F1–F2, and both functions can be attached to any switch. For simplicity, F1 and F2 are set up as softswitch, and no real service function executes inside. Fig.5 enumerates 2 representative topologies that lead to 2 different sets of resource models. The service paths are listed under each topology. We further assume that these 5 softswitch (S1, S2, S3, F1, F2) share 3 CPU cores (C1, C2, C3). We use OFsoftswitch in simulation for easy implementation without involving kernel-level programming.

The table in Fig.6 indicates the binding relationships between softswitches and CPUs. 3 customized CPU allocation

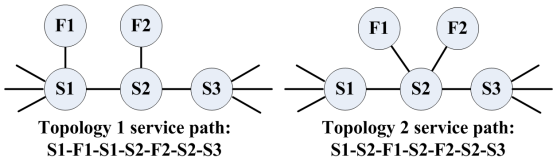
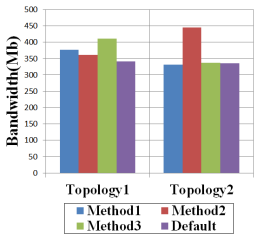


Fig. 5. Service chaining topology.

methods are listed in detail to compare with default OS scheduling. These allocation methods are applied to each topology. From the results of throughput in Fig.6, default OS scheduling shows slight performance difference on two topologies, which is 341 and 336 Mbps respectively. In both topologies, customized CPU allocation provides equivalent or better performance than default OS scheduling. In topology 1, method 3 leads to best performance as 411Mbps while method 2 reaches to 445Mbps in topology 2. Thus the best coordination between topology and resource allocation is "topology 2 + method 2", which achieves over 30% performance improvement than default control by OS.



CPU Allocation					
Allocation	S1	S2	S3	F1	F2
Method 1	C1	C2	C3	C1	C2
Method 2	C1	C2	C3	C1	C3
Method 3	C1	C2	C3	C2	C3
Default	Deployed randomly by OS				

Fig. 6. Simulation on resource allocation.

IV. Conclusion

This paper focuses on the performance analysis of softswitch in virtualization environment. We summarize the main performance factors and measure their impacts. Based on these facts, we extend resource allocation among multiple softswitches to take topology design into consideration and achieve notable improvement. Some common conclusions can be extended to other types of softswitch. Our work raises a couple of open problems like how to build softswitch resource model, and how to combine softswitch features in deployment. These problems are the subjects for future work.

References

[1] P. Costa, M. Migliavacca, *et al.*, "Naas: Network-as-a-service in the cloud," in USENIX Hot-ICE'12, Berkeley, CA, USA, 2012, pp. 1–1.

[2] "Open vSwitch," http://openvswitch.org/.

[3] L. Rizzo, "Netmap: A novel framework for fast packet i/o," in USENIX ATC'12. Berkeley, CA, USA, 2012, pp. 9–9.

[4] "Intel dpdk: Data plane development kit," http://dpdk.org/.

[5] A. Rai, R. Bhagwan, and S. Guha, "Generalized resource allocation for the cloud," in SoCC '12. New York, NY, USA, 2012, pp. 15:1–15:12.

[6] "Openflow 1.3 software switch," http://cpqd.github.io/ofsoftswitch13/.

[7] "Mininet," http://mininet.org/.

[8] J. Gross, B. Basler, and et al., "The overhead of software tunneling," http://networkheresy.com/2012/06/08/the-overhead-of-software-tunneling/.

[9] G. Pongracz, L. Molnar, and Z. Kis, "Removing roadblocks from sdn: Openflow software switch performance on intel dpdk," in *2013 Second European Workshop on SDN (EWSDN)*, Oct 2013, pp. 62–67.