

HPSRouter: A High Performance Software Router Based on DPDK

Zongyao Li *

*Shanghai Key Lab of Scalable Computing and System, Shanghai Jiao Tong University, China

lizongyao12@sjtu.edu.cn

Abstract—all aspects of human life have been changed by Internet at an unprecedented rate. Moreover, the network appliances has exploded. They bring a series of management problems. In 2012, many large carriers initiated an effort, called Network Function Virtualization (NFV), to replace hardware network appliances with software implementations running in VMs. This approach enabled Network Functions (NFs) to run on commodity servers. The road to convert hardware middleboxes to software network functions has never been smooth. The performance gap between software and hardware seems insurmountable when packets are processed through Linux stack. The emergence of data plane development kit (DPDK) brings the chance of bridging the gap by bypassing the Linux kernel stack. We establish a high performance software router based on DPDK framework to offer a solution to the problem and tune the performance to approach the line rate of our 40 Gb network interface card (NIC) using batching, prefetching and affinity technique. We build our system in our four sockets commodity server equipped with 40 Gbps Ethernet interfaces and show that HPSRouter can reach the line rate.

Keywords—DPDK, High Performance, NFV, Software router, Tuning

I. INTRODUCTION

All aspects of human life have been changed by Internet at an unprecedented rate. At the same time, to support the demands of leveraging network, the presence of network applications has exploded and become more and more complex and diverse. The workload in network system has been overwhelmed. The network appliances based on dedicated hardware becomes essential to network system because of high performance and high reliability. Nowadays, they are not only responsible for packet forwarding and filtering but also supporting various other network functions (NFs), such as quality of service (QoS), deep packet inspection (DPI), data encryption service and so on.

Despite the advantages, hardware-based network appliances bring some drawbacks. First, hardware-based network appliances are expensive. Purchasing the devices and the power cost are unaffordable for huge number of these appliances. Second, the management of hardware-based network appliances is difficult. The network operator need to learn different network operating system for the devices offered by different vendor. With the increase of middlebox, it is more and more difficult for operator to master various network operating

systems; meanwhile, looking for the appropriate space to place these appliances is rather burdensome. Third, the hardware-based network appliances have poor flexibility. They have to reserve enough resource to cope with tide traffic, which is wasteful. Finally, it takes years to develop a new hardware. It is too long to catch up the development of new network technologies.

Faced with this dilemma, in 2012, many large carriers propose the concept of Network Function Virtualization (NFV) [1] [9], which aimed at decoupling the network function (NF) from the dedicated hardware devices. This approach enables carriers to shift the NFs to software running on off-the-shelf servers.

However, the move from hardware to software is never smooth sailing. The performance gap between software and hardware becomes a huge obstacle. Though Unix-based software routers provide high flexibility and lowered cost, the slow packets processing and unbearable delay are unacceptable for network system.

In order to narrow the gap, many groups take great endeavour. The RouteBricks [2] project is a meaningful attempt to realize a high performance router. They implemented a prototype of IPv4 router on four servers using Intel quad-core processors (each server was equipped with an external 10Gbps Ethernet card), which could handle packets at the rate of 35Gbps. In 2010, South Korean researchers implemented PacketShader [3], a four-port IPv4 router with a forwarding capability of 39Gbps, based on a single eight-core (including two quad-core processors) server and four 10Gbps Ethernet cards. For the first time, this work applies the GPU to network systems for quick lookup of routing tables. Though the forwarding performance caught up with hardware-based network appliances, the latency caused by offloading network workloads to GPU device still perplexed the researchers.

Various research groups analysed the bottleneck of network performance by building models and experimenting and came to the conclusion that standard software is not suitable for high-speed packet forwarding scenario [4][7][8]. Therefore, Intel Corporation proposed a high-speed packet processing framework named Data plane Development Kit (DPDK) [5] [6], which utilizes user-mode drivers to bypass the Linux kernel stack and uses a range of kernel optimization techniques to enable DPDK to process packets at the same level of performance as dedicated hardware. The DPDK platform

provides a set of underlying software libraries for managing memory, buffers, and queues [10]. By using the DPDK framework, developers can write their own network protocol stacks and implement high-performance datagram forwarding routines in user space. We implemented our high-performance software router with DPDK and performed a series of optimizations to make it reach the line rate of 40Gbit/s network adapters on a general-purpose server.

II. DESIGN AND IMPLEMENTATION

In this section, we present the design and implementation of HPSRouter on a general proposed server.

A. Overall architecture

Figure 1 shows the overall architecture of HPSRouter. We follow the principle of decoupling the control plane and the data plane, and design it as a multi-threaded program and implement it using the DPDK framework. The HPSRouter consists of two threads: router thread and controller thread. The router thread is responsible for the data plane processing of packets. Moreover, the controller thread exposes interface to network operators to control the state of HPSRouter.

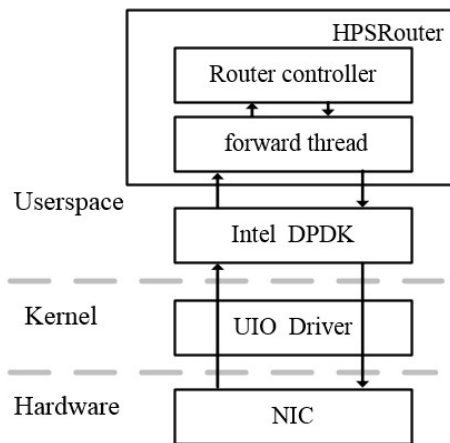


Figure 1. The overall architecture of HPSRouter

B. Router processing

The Router processing mainly involves the analysis of IPv4 basic headers. The protocol stack parses basic information such as the source address and destination address. After comparing the key information of the native machine, the packets processing is completed according to the established strategy. If the destination address is a local address, the current protocol stack processes the data packet and strips the IPv4 header. Then, the data is submitted to the upper level protocol for the next step processing. If the parsed IPv4 destination address is not a local address and the local machine can act as a router for data forwarding, the protocol stack searches for a static routing table. Find the right IP address by routing. Finally, the protocol stack modifies CRC check information and MAC address, sends the data to the sending queue. Then, the downstream packet-forwarding engine uses the next-hop IP/destination IP address, and VLAN ID to check the ARP entry to obtain the destination

MAC address. Then, the port MAC address is obtained based on the outbound interface information and the router forwarding process is completed. The router processing flow chart shows in Figure 2.

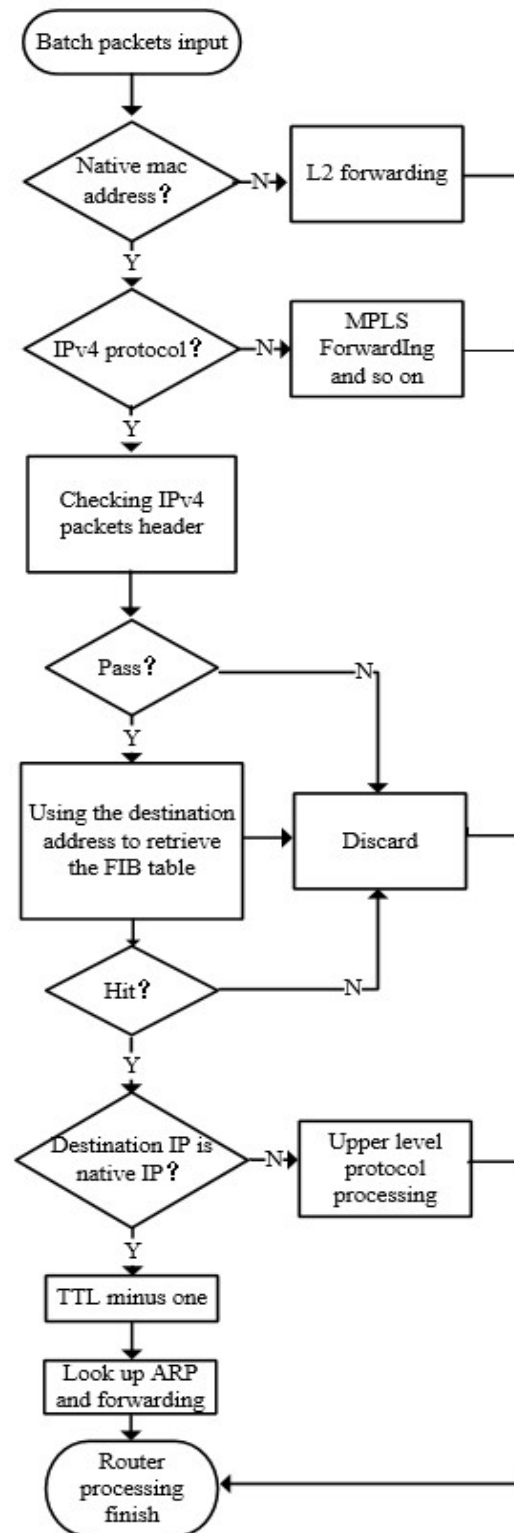


Figure 2. The router processing flow chart

C. Router controller

The router controller decouples the control plane from the data plane and provides the interface to network operators to control the state of HPSRouter. When running the HPSRouter, we start by creating a router controller thread. Then, the controller will initialize the routing thread including allocating memory, setting affinity and so on. After that, network operator can use the controller thread to configure the router and make it in an inactive state. Afterwards, the operator can start or stop the router thread to process the packets and update the router table, ARP table, FIB table and so on. When the processing task is finished, the operator can terminate the router thread and exit the program. The state transition diagram shows in figure 3.

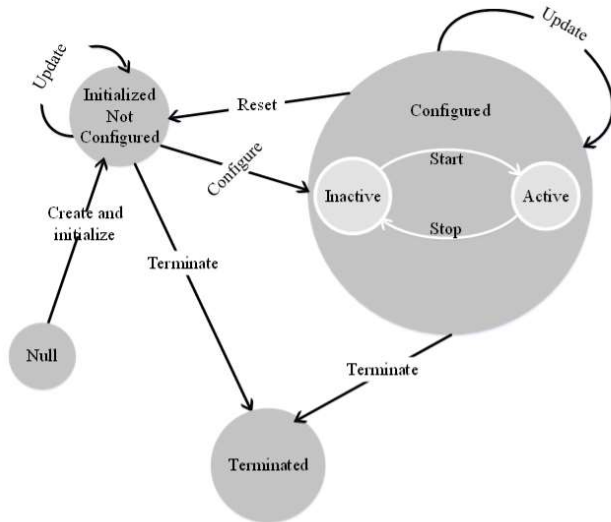


Figure 3. State transition diagram

D. Network packets processing model

Traditional packets forwarding models can be divided into run to completion (RTC) model and pipeline model.

1) Pipeline model: From the name, we can see that the pipeline model draws on the industrial pipeline model; a function is decomposed into multiple independent stages, different stages of product delivery through the queue.

2) Run to completion model: a program generally divided into several different logic functions, but these logic functions will be in a CPU core operation, we can scale horizontally to enable the same logic codes to run on multiple cores in a symmetric multiple processor (SMP) system to increase the amount of transactions per unit of time.

Because of the DPDK user space poll-mode driver, we used to bind the threads corresponding to the receiving / sending queue to the corresponding processor. The entire life cycle of packets can only appear in one of the threads. In this case, compared with pipeline model, the run-to-completion model performs better on scalability and takes full advantage of local cache to avoided performance degradation causing by cache inconsistence. Therefore, we build our HPSRouter in run to completion model so that the router processing can conveniently scale over multiple CPU cores.

E. Performance tuning

1) Batching: Batching is an important technique for achieving high performance in SMP systems. Instead of updating the receive queue tail index register every time a packet is received, we do not really update the receive queue tail index register until the number of releasable receive descriptors reaches a threshold. Hence, we set this threshold to a size appropriate for the cache line size, which is usually 32. when we assemble 32 packets in a batch and deliver them to the CPU for the following processing, the time consumption in accessing to memory, writing the descriptor and so on can be bisected.

2) Affinity binding: DPDK work in user mode. The thread's scheduling is still dependent on the kernel. Using thread affinity bindings, specific tasks can be assigned to work on only one core. The advantage is to avoid the frequent switching of threads between different cores, thread switching between cores easily lead to a large number of performance losses due to cache miss and cache write back. We further restrict certain cores from participating in Linux system scheduling and make it possible for the thread to monopolize the core, guarantee more cache hit, and avoid the multitasking overhead of the same core.

3) Software Prefetching: It is important to use software prefetching in performance-related areas. If we load upcoming data from memory into Cache, when the current data processing is completed, the upcoming data is already in the Cache, greatly reducing not only the overhead of direct read from memory but also the processor latency. In consequence, performance has been greatly improved. The prefetching operation is only suitable for data rather than instructions.

III. EXPERIMENTAL RESULTS

A. Experimental Setup

We evaluate HPSRouter on a test bed of two quad-socket servers equipped with Intel Xeon E7 CPUs, each of which has 64 logical cores. The servers run Linux kernel 4.8 and HPSRouter uses DPDK version 16.11.

To evaluate HPSRouter, we use a DPDK-based packet generator. It directly connects to the server, which our HPSRouter runs on via two dual-port 40-GbE NICs. Its role is both the source and the sink of packets. It produces up to 150 Gbps network traffic that contains packets with various types of lengths.

B. Compared to Linux IP stack

We run our HPSRouter and Linux IP stack on one of the server respectively. In addition, let the packet generator produce 64B, 128B, 256B, 512B and 1024B packets. Moreover, we measured the performance of the Linux kernel protocol stack and DPDK respectively. The figure 4 shows the result. In the case of Linux IP protocol stack, we found that the throughput of Linux IP protocol stack could only reach 813Mb / s when using 64B packets, even with 1024B packets, the throughput of Linux IP protocol stack can only reach 6.3 GB / s. In our HPSRouter, in the case of the 64B packet, the throughput can reach 6.8 Gb / s, in the case of 512B data packets; HPSRouter throughput has been able to achieve the

line rate of the network card. Moreover, in the case of 1024B data packets, the throughput of our HPSRouter can reach the maximum rate of the network card. Our HPSRouter bypassed Linux protocol stack and eliminated system calls and redundant packet copying. Compared to the Linux kernel stack, our HPSRouter throughput increased by 8-10 times.

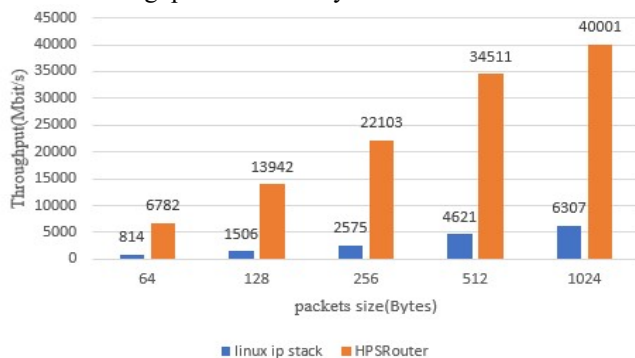


Figure 4. The throughput of HPSRouter and Linux IP stack

C. HPSRouter performance

Then, we test our HPSRouter performance and scalability. We run our HPSRouter on one of the server. Moreover, let the packet generator produce 64B, 128B, 256B, 512B and 1024B packets. At the first time, we just use one port in 40-GbE NIC. Then we leverage all of the ports in 40-GbE NICs to check its scalability. The result is shown in figure 4 and figure 5.

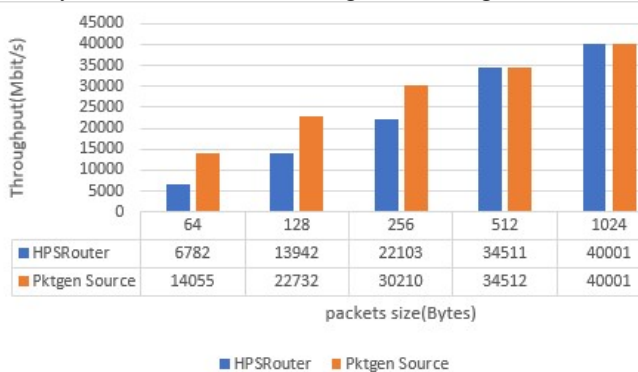


Figure 5. The performance of HPSRouter

In Figure 5, we see that the 512B data packets sourced by Pktgen can be completely processed by our HPSRouter. Line rate of NIC can be achieved when processing packets larger than 512B. A recent study [11] shows. The average size of data packets in the network is 744B. If we take our target to be packets of around this size, the HPSRouter can sustain line rate.

From Figure 6, we can conclude that HPSRouter's performance is completely unaffected when CPU extending from single-core to multi-core. When we run HPSRouter with four CPU cores to process traffic from four network ports, the packets processing speed of HPSRouter reaches line speed and maintains line speed as the packet size increasing. Due to the limitation of the micro-switch in our dual-port full-duplex 40GbE network interface card, when using both ports of the network interface card for sending and receiving at the same time, our network interface card can only send data packets at the speed of 50Gb / s. So our HPSRouter achieves 100Gb / s high performance on our platform.

IV. CONCLUSIONS

We have established a high performance software router based on DPDK framework to offer a solution to perform problem existed in software router and tune the performance to approach the line rate of our 40 Gb network interface card (NIC) using batching, pipelining, prefetching and affinity technique. We build our system in our four sockets commodity server equipped with 40 Gbps Ethernet interfaces and show that HPSRouter can reach the line rate. In the future, we may integrate more network capabilities while maintaining the same performance on common servers as dedicated hardware.

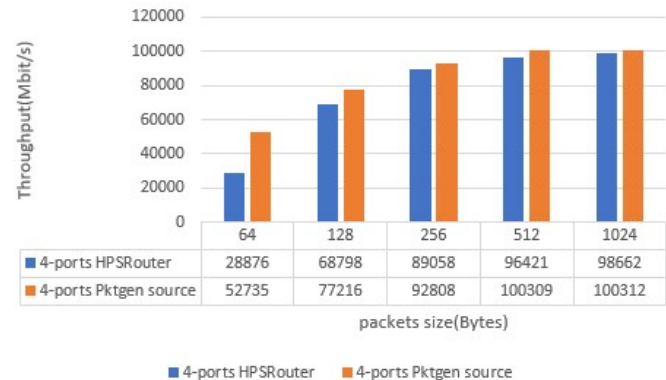
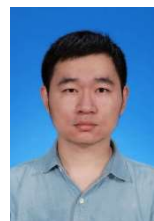


Figure 6. The scalability of HPSRouter

REFERENCES

- [1] ETSI GS NFV.001: Network Functions Virtualisation (NFV) [J]. Architectural Framework, 2013.
- [2] Dobrescu M, Egi N, Argyraki K, et al. RouteBricks: exploiting parallelism to scale software routers[C]. Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009: 15-28.
- [3] Han S, Jang K, Park K S, et al. PacketShader: a GPU-accelerated software router[C]. ACM SIGCOMM Computer Communication Review. ACM, 2010, 40(4): 195-206.
- [4] R.Bolla et al. Linux Software Router: Data Plane Optimization and Performance Evaluation, in: Journal of Networks, 2(3), pages 6-17, 2007.
- [5] Intel DPDK: Data Plane Development Kit Project Page, <http://www.dpdk.org>, last visited June 2017.
- [6] Intel DPDK: Programmers Guide, January 2017.
- [7] Wu W, Crawford M, Bowden M. The performance analysis of Linux networking—packet receiving [J]. Computer Communications, 2007, 30(5): 1044-1057. PDCA12-70 data sheet, Opto Speed SA, Mezzovico, Switzerland.
- [8] Rizzo L. Netmap: a novel framework for fast packet I/O[C]. 21st USENIX Security Symposium (USENIX Security 12). 2012: 101-112.
- [9] Virtualisation NF. Virtual Network Functions Architecture [J]. ETSI GS NFV-SWA, 1: V1, 2014.
- [10] Scholz D. A Look at Intel's Dataplane Development Kit [J]. Network, 2014, 115.
- [11] P. M. Santiago del Rio, D. Rossi, F. Gringoli, L. Nava, L. Salgarelli, and J. Aracil. Wire-speed statistical classification of network traffic on commodity hardware. In Proceedings of the 2012 ACM conference on Internet measurement conference, IMC '12, pages 65-72, New York, NY, USA, 2012. ACM.



Li Zongyao received the B.S. degree in software engineering from Wuhan University. He currently pursues the M.S. degree in the school of software engineering in Shanghai Jiao Tong University, Shanghai, China. His research interests are mainly focus on cloud computing, high performance network, network function virtualization and software defined network.