mKPAC:Kernel Packet Processing for Manycore Systems

Ramneek[1], Mohan Kumar[2], Taesoo Kim[2], Sungin Jung[1]
[1]Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea
[2]Georgia Institute of Technology, Atlanta, USA
ramneek@etri.re.kr, mohankumar@gatech.edu, taesoo@gatech.edu, sijung@etri.re.kr

ABSTRACT

Network Function Virtualization (NFV) has recently gained popularity due to its ability of offering high scalability and programmability using commodity servers and general-purpose operating system (OS). However, current OSes have failed to deliver the data-plane performance required by the software-based network functions, mainly due to the inherent overheads associated with network stack in the kernel. We present mKPAC, aimed at improving data plane performance of OS kernel in manycore environment. We analyze the high-impact overheads residing in network stack of Linux kernel, and show that the data plane performance for NFV can be accelerated by mitigating major performance penalties and by leveraging the availability of manycores. With 64 bytes packet size, mKPAC can successfully accelerate the Linux Kernel packet forwarding performance up to 40% in packets per second.

CCS CONCEPTS

• **Networks** → **Network protocol design**; • **Software and its engineering** → **Operating systems**;

KEYWORDS

Network Function Virtualization, Linux, Manycore Systems

1 INTRODUCTION

Traditional hardware-centric network functions provide highly optimized packet processing performance. However, they fail to provide the scalability and flexibility required in virtualized data centers that host multiple tenants who dynamically implement their own network topology regardless of the underlying physical network. Envisioned to overcome these challenges, the use of NFV has been proposed [3]. Virtualization of networking equipment allows the network operators to implement a variety of sophisticated network functions by leveraging the programmability, scalability and ease of use offered by commodity servers running conventional operating systems. Despite these benefits, NFV fails to satisfy the performance requirements of network functions.

Existing solutions to achieve high data plane performance include hardware based acceleration (PacketShader, ClickNP), kernel bypassing (DPDK, Netmap), and NFV platforms such as ClickOS,

Middleware '18 Posters and Demos, December 10–14, 2018, Rennes, France

ACM ISBN 978-1-4503-6109-5/18/12…$15.00
https://doi.org/10.1145/3284014.3284022

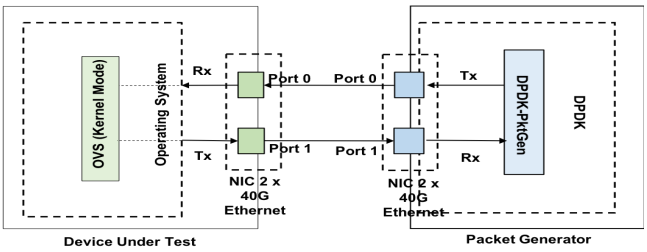

Figure 1: Illustration of the Experimental Testbed

NetVM, [2] etc. However, these approaches still suffer from a number of issues, such as need for specialized hardware, excessive memory usage, scalability issues, relatively low security, need for reimplementation of software functions from scratch in user space, or development of new systems, instead of relying on well-implemented kernel stacks. Other proposals like Megapipe, fastsocket, IX, mTCP etc. focus on user space applications involving system call and packet copying overheads, that do not exist in kernel-space applications considered for the present study. With recent advances in processor technologies supporting large number of cores, and in network equipment supporting high data rates, significant speed up can be supported for network functions that are responsive to parallelization [1]. Moreover, maturity of implementation and robustness of existing kernels, and their inherent features such as iptables, netfilter, ipsec, cgroups, etc. can help to define virtualized network functions with greater ease and flexibility. Hence, there is a need to overcome the drawbacks in network stacks of existing kernels.

In this work, we focus on kernel-level optimizations to enhance the packet processing performance on manycore systems and overcome the limitations in existing techniques. We consider data-plane forwarding applications such as software switching, IP forwarding, daisy chaining based forwarding, and other NFVs e.g software-based firewall, intrusion detection, etc. These applications are implemented in kernel-space and do not incur the system call overheads associated with user-kernel context switch or packet copying overheads. By mitigating the overheads associated with the transmit path and exploiting the scalability of manycore systems, mKPAC can achieve up to 40% improvement in packets per second(PPS) for 64 bytes packets as compared to baseline Linux.

2 EXPERIMENTAL SETUP AND ANALYSIS

The analysis and implementation for the current work is based on Linux Kernel 4.4.1 running on a Intel(R) Xeon(R) CPU E7-8870 v2 @ 2.30GHz, with 8 NUMA sockets, each having 15 cores. Intel Ethernet Controller XL710 with two 40GbE ports is used for data RX and TX. We use OVS kernel module for data plane forwarding [5]. The device under test is connected to DPDK-Pktgen, generating traffic at the rate of 40Gbps, as shown in Figure 1. We used the CPU Flame

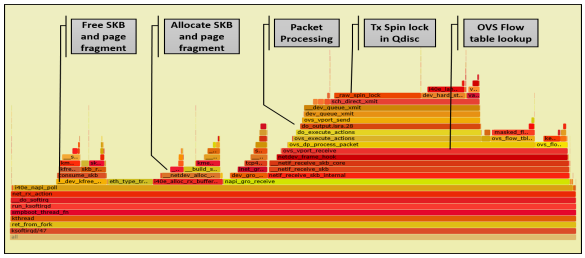

Figure 2: OVS Kernel mode Flamegraph, indicating main packet processing components for data-plane forwarding.

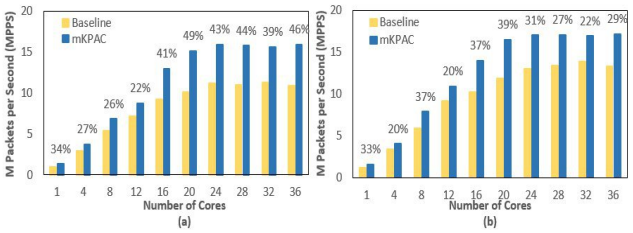

Figure 3: Throughput vs. number of cores for baseline kernel and mKPAC for (a) 64 bytes and (b) 128 bytes packets.

Graphs to identify the hot or busy-on-CPU code paths by visualization of sampled stack traces as shown in Figure 2. We identify the packet processing components as: Kernel receive packets (RX), OVS processing (application handling), Kernel send packets (TX), and meta data/data buffer allocation and freeing. Profiling with perf top shows that raw spin locks in Qdisc and TX queue contribute to 14.2% of overhead. Hence, the main overheads can be traced down to high TX interrupt handling, cleanup and transmit component overheads, meta-data/data buffer allocation and freeing.

3 DESIGN AND IMPLEMENTATION

In the current work, we aim at improving the packet processing performance by proposing design solutions to overcome TX path overheads, as described below:

Adaptive TX Clean-up: With the Linux NAPI mechanism, besides polling for receiving packets, kernel cleans the TX ring on the NIC port. Cleanup includes unmapping the DMA region, freeing the data and meta-data buffer after transmission is complete. When no packets are available in RX ring, the port is removed from the polling list and interrupt is enabled. Also, the TX polling loop executes only for a small fixed budget, defined by the driver. The frequent interrupts and inefficient cleanup can disrupt the CPU cycles, resulting in low performance. To address this issue, we propose the use of adaptive TX cleanup. To avoid the CPU resource exhaustion because of non-preemptive cleanup, we modify TX cleanup budget calculation to dynamically optimize the number of TX elements that can be cleaned up in one polling loop.

*TX spin locks and TX queue selection:*We focus on two main overheads: TX spin locks and TX queue selection. In Linux, there are two spin locks: one for Qdisc and the other for TX queue. Even when there is no contention, such that each core has its exclusive TX queue and a dedicated Qdisc for each TX queue, the atomic instructions for lock and unlock create a high overhead (14%). Hence, we can avoid both the locks in the TX path, resulting in an increased performance without contention. In Linux kernel, TX queue selection logic is implemented by the driver code and is based on hashing or transmit path steering (XPS) in case of multi-queue systems [4]. However, it does not ensure that a unique queue is assigned to each CPU. Hence, we modify the queue selection function such that a unique queue dedicated to each core can be assigned for TX.

4 EVALUATION AND FUTURE DIRECTIONS

Packet processing performance gains achieved by mKPAC are shown in Figure 3. As compared to the baseline Linux kernel, we could achieve up to 40% improvement in PPS for 64 Bytes packet size. The per-core decrease in latency using mKPAC is around 23% and 22%, for 64 bytes and 128 bytes packets, respectively. The current implementation spans across the NAPI module, driver module, and the dev_queue_xmit path of Linux kernel. Our preliminary results show that by focusing on the design issues in the kernel, a considerable performance improvement can be achieved on many-core systems, without any kind of kernel-bypassing or specialized hardware support. The packet processing performance can be further improved by addressing other overheads and scalability issues in manycore systems. For instance, overheads related to meta data and data buffer allocation and de-allocation can be addressed by pre-allocation of buffers by the NIC driver during initialization. This can help to reduce the allocation cycles and prevent mapping and un-mapping of data buffers for every packet transmission, and will be implemented in the future work.

Tuning various system parameters, such as irq affinity, buffer sizes, etc., can help to achieve some performance gain; however, these are not significant for high pps workloads. Hence, mKPAC rather focuses on design issues related to the network stack in OSes. Future work will include (i) design solutions for overcoming additional overheads in the network stack and scalability issues in manycore systems; (ii) evaluation with other use cases (e.g. contrail vRouters, eBPF based InKev,etc.), more complex network functions (e.g.intrusion detection, daisy-chaining based forwarding, load balancing, etc), and smart NICs and drivers capable of providing low latency and other advanced features.

ACKNOWLEDGMENTS

The work is supported by the Institute for Information and Communications Technology Promotion under Grant No.: 2014-3-00035.

REFERENCES

[1] Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, and Nickolai Zeldovich. 2010. An Analysis of Linux Scalability to Many Cores. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. 1–16.

[2] Sebastian Gallenmüller, Paul Emmerich, Florian Wohlfart, Daniel Raumer, and Georg Carle. 2015. Comparison of Frameworks for High-Performance Packet IO. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '15)*. IEEE Computer Society, Washington, DC, USA, 29–38.

[3] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. 2015. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine* 53, 2 (Feb 2015), 90–97.

[4] Tom Herbert and Willem de Bruijn. 2018. Scaling in the Linux Networking Stack. https://www.kernel.org/doc/Documentation/networking/scaling.txt

[5] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado. 2015. The Design and Implementation of Open vSwitch. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. 117–130.

2