IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 6, June 2017

Implementation of Virtual Router in User Space by Bypassing the Kernel using Snabb

Pranav Gokhale[1], Hrishikesh Kulkarni[1], Surajkumar Gunjal[1], Akshay Abhang[1]

Department of Computer Engineering, Pune Institute of Computer Technology, Pune, India[1]

Abstract: As network adapters get faster, the time between packets i.e. the time the kernel has to process each packet gets smaller. That does not leave a lot of time for kernel to figure out what to do with each packet. The kernel has traditionally not done a great job with this kind of network-intensive workload. This is due to the latency of multiple system calls involved while an application reads packets from the network interface. This has led to existence of a number of networking implementations that bypass the kernel's network stack entirely. We use Snabb which is one of many such implementations. Snabb allows us to build our own network design on top of the Snabb core, bypassing the kernel. Our virtual router is built on top of Snabb. The virtual router runs in user space and also bypasses the kernel entirely. This capability allows us to make fast and intelligent forwarding decisions in the virtual router. This is suitable for dedicated servers which have high network traffic and require fast packet forwarding.

Keywords: Software Defined Networking, Kernel bypass, Fast packet processing, Snabb.

I. INTRODUCTION

The current networking 100 Gbps adapters drop the packets at per-packet rate of about 120ns; the interface, at this point, is processing 8.15 million packets per second. That does not leave a lot of time for kernel to figure out what to do with each packet. In a normal kernel network stack, the application has to make multiple system calls to read the packets coming in at the interface. The involvement of system calls leads to latency in the system. Such latency reduces the efficiency by large margin in the case of specialized high packet rate workloads. This has led to existence of a number of networking implementations that bypass the kernel's network stack entirely. The demand for such systems indicates that the kernel is not using the hardware optimally; the out-of-tree implementations can drive adapters at full wire speed from a single CPU. In the Linux ecosystem, there are a few available technologies like PF_RING, Snabb, DPDK, and Netmap. All of these technologies require handing over the full network card to a single process. In other words: it's totally possible to write your own network stack, focusing on super features, and optimize for performance. But there is a big cost incurred, you will be restricted to running at most one process for each network card. This is suitable for dedicated servers which have high network traffic. This paper aims to provide such an alternative approach for packet forwarding through use of open source network virtualization toolkit Snabb by simulating a virtual router's functionality.

II. NETWORK LAYER PROTOCOL OVERHEADS

Let us consider the case of Transmission Control Protocol (TCP). TCP is a transport protocol from the Internet protocol suite. In this protocol, the functions of detecting and recovering lost packets, flow control, and multiplexing are performed at transport level. TCP implements these functions using checksums, sequence numbers and acknowledgement.

According to Analysis of TCP processing overhead [1], most of the overhead is caused not by the actual protocol processing but is caused mainly by the operating system. Packet processing requires considerable support from the system. It is necessary to first take an interrupt, allocate a packet buffer, free a packet buffer, restart the I/O device, wake up the process and finally reset the timer. All these require system calls and thus require service from the kernel. So in a typical operating system, these functions prove to be expensive. [1] states that there is a 240 µs per packet operating system overhead involved during packet processing.

III. ALTERNATIVE APPROACHES

In recent years, there has been emergence of high speed packet I/O frameworks, bringing network performance to user space. Fast user space packet processing by Barbette et al. [2] illustrates that the kernel stack is too slow for packet processing in case of multiple 10-Gbps interfaces. The kernel stack is slow due to the involvement of system calls while processing of packets. Several user space I/O frameworks have been proposed to address this issue. These frameworks enjoy the power of the kernel although now the user space application has added responsibility. These

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 6, June 2017

frameworks allow bypassing the kernel and obtaining efficiently a batch of raw packets with a single system call, with added capabilities of modern Network Interface Card (NIC) such as multi-queueing.
First we review various features exhibited by high performance user space packet I/O frameworks along with a few examples of such frameworks.

A. Features
Kernel bypass: An operating system kernel provides wide range of networking functionalities but this comes at the expense of a performance cost in high-speed networking scenarios. So, to boost the performance the frameworks bypass the kernel altogether. These frameworks deliver the raw packets directly into the user space. This involves bypassing the entire existing kernel networking stack. This gives us the capability to create our own user space networking stack to handle specialized workloads. [3]
Zero-copy: While transmitting data to and from a NIC, the data is staged in kernel space buffers. These buffers are part of a Direct Memory Access (DMA), and the application issues read/write system calls to copy the data to user space buffers. This is called memory-to-memory copy. Most of the frameworks try to avoid this memory-to-memory copy by using a dynamic buffer pool in a shared region of memory which is visible to both the NICs and user space application. Thus there is no copying involved in between buffers as it is dynamic, so this is called as zero-copy.
Hardware multi-queue support: Modern NICs allow packets to be received in multiple hardware queues. This helps in load balancing (e.g.: each core in a multi-core machine can be assigned different packets).

B. Frameworks
Packet_mmap [4] provides a size configurable circular buffer mapped in user space that can be used to either send or receive packets. Multiple packets can be sent through one system call to get the highest bandwidth. By using a shared buffer between the kernel and the user also has the benefit of minimizing packet copies.
Netmap: Applications (routers, traffic monitors, firewalls, etc.) need to send and receive packets at line rate even on very fast links. Netmap [5] is a novel framework that enables commodity operating systems to handle the millions of packets per seconds. Netmap has reduced or removed three main packet processing costs: per-packet dynamic memory allocations, removed by pre-allocating resources; system call overheads, amortized over large batches; and memory copies, eliminated by sharing buffers and metadata between kernel and user space, while still protecting access to device registers and other kernel memory areas. Netmap has been implemented in FreeBSD and Linux for several 1 and 10 Gbit/s network adapters.
PF_Ring ZC: PF_RING ZC (Zero Copy) [6] is a flexible packet processing framework that allows you to achieve 1/10 Gbit line rate packet processing (both RX and TX) at any packet size. It implements zero copy operations including patterns for inter-process and inter-VM (KVM) communications.
DPDK: The Intel Data Plane Development Kit [7] is somehow comparable to Netmap and PF RING ZC, but provides more user level functionalities such as a multi core framework with enhanced NUMA-awareness, and libraries for packet manipulation across cores. It also provides two execution models: a pipeline model where typically one core takes the packets from a device and gives them to another core for processing, and a run-to-completion model where packets are distributed among cores using RSS, and processed on the core which also transmits them.
Snabb is the framework that we use to implement the virtual router, which is described in detail in the next section.

IV. SNABB

Snabb [8, 9] is an open source, high-performance networking framework developed by Luke Gorrie since 2012. The project's goal is to offer network administrators, Internet service providers and in general operators that have to process large traffic volumes beyond 1GbE software solutions for existing hardware networking appliances.
The primary idea is to write all low level functionality, the interaction with the hardware, from scratch. Snabb does so for modern Intel NICs. Instead of communicating with the device via the kernel and its drivers, the PCI device is bound directly to Snabb and configured by writing respective values according to the NICs specification using Memory-mapped I/O (MMIO), allowing direct I/O operations. Via Direct Memory Access (DMA) the NIC is allowed to directly access reserved memory regions to read and write packets.
While it is a common approach for packet processing frameworks to focus on the hardware-near C or C++ as primary programming languages and respective established compilers, Snabb uses the lightweight scripting language Lua and the just-in-time LuaJIT [10] compiler instead. Lua is a high-level programming language that has been used in various fields including game development, image processing and robotics for many years.
Firstly, it allows to easy integration with C libraries. Therefore, Snabb can conveniently make use of low-level functions and data structures.
Secondly, LuaJIT which produces target specific machine code for instance for x86 and ARM architectures, is a trace based compiler that gathers profiling information during runtime to compile parts of the application in an optimized

way. The control-flow is analysed in detail by inspecting whole program paths "to capture the smallest set of execution traces that are representative of the dynamic behaviour of the application.

V. SDN AND VIRTUAL ROUTING

Computer networks are typically built from a large number of network devices such as routers, switches and numerous types of middleboxes (i.e., devices that manipulate traffic for purposes other than packet forwarding, such as a firewall) with many complex protocols implemented on them. Network operators are responsible for configuring policies to respond to a wide range of network events and applications. They have to manually transform these high level-policies into low-level configuration commands while adapting to changing network conditions. And often they need to accomplish these very complex tasks with access to very limited tools. As a result, network management and performance tuning is quite challenging and thus error-prone. The fact that network devices are usually vertically-integrated black boxes exacerbates the challenge network operators and administrators face. Another almost unsurmountable challenge network practitioners and researchers face has been referred to as "Internet ossification". Because of its huge deployment base and the fact it is considered part of our society's critical infrastructure (just like transportation and power grids), the Internet has become extremely difficult to evolve both in terms of its physical infrastructure as well as its protocols and performance. However, as current and emerging Internet applications and services become increasingly more complex and demanding, it is imperative that the Internet be able to evolve to address these new challenges. The idea of "programmable networks" has been proposed as a way to facilitate network evolution. In particular, Software Defined Networking (SDN)[11] is a new networking paradigm in which the forwarding hardware is decoupled from control decisions. It promises to dramatically simplify network management and enable innovation and evolution. The main idea is to allow software developers to rely on network resources in the same easy manner as they do on storage and computing resources. In SDN, the network intelligence is logically centralized in software-based controllers (the control plane), and network devices become simple packet forwarding devices (the data plane) that can be programmed via an open interface.
The underlying network infrastructure may involve a number of different physical network equipment, or forwarding devices such as routers, switches, virtual switches, wireless access points, to name a few. In a software-defined network, such devices are often represented as basic forwarding hardware accessible via an open interface at an abstraction layer, as the control logic and algorithms are off-loaded to a controller. Such forwarding devices are commonly referred to, in SDN terminology, simply as "switches".

VI. OUR APPROACH

Our approach is implementing packet forwarding in user space by bypassing the kernel. The kernel of advanced Operating systems is generic and introduces an overhead for networking functionalities due to system calls. A Virtual Router or vRouter is a software function that replicates in software the functionality of a hardware-based Layer 3 Internet Protocol (IP) routing, which has traditionally used a dedicated hardware device. The idea is to implement the forwarding functionality of vRouter as a user space process by bypassing the kernel in order to improve performance and gain more control over the packet forwarding process. The main source of motivation behind our implementation is poor performance of packet forwarding in virtual network when the traffic of packets is of the order of millions per second. Packet processing is slow due to the latency introduced by the kernel, as the forwarding plane is implemented in kernel space. Our aim is to find an innovative and more efficient user space approach for packet forwarding in virtual network by extending the vRouter functionality with a user space forwarding plane.

We have used the Longest Prefix Matching algorithm for forwarding. Longest prefix match [12] (also called Maximum prefix length match) refers to an algorithm used by routers in Internet Protocol (IP) networking to select an entry from a forwarding table. Because each entry in a forwarding table may specify a sub-network, one destination address may match more than one forwarding table entry. The most specific of the matching table entries the one with the longest subnet mask is called the longest prefix match. It is called this because it is also the entry where the largest number of leading address bits of the destination address match those in the table entry. The application of our project lies in a high traffic handling data centres where SDNs are widely used. In cases where millions of packets are coming in every second at the vRouter for forwarding decisions, the vRouter has only a few microseconds to decide the route of each packet. In such specialized workloads where fast packet processing is required, our project provides the solution in such cases. Our vRouter is capable of fast control-plane packet processing.

VII. RESULTS AND CONCLUSION

The timestamps in the figures below are represented in Unix Epoch time. We calculate the difference between the timestamp of the first sent packet and the last received packet. We compare the transmission time of Snabb i.e. kernel

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 6, June 2017

bypass and the traditional kernel networking stack. The libcrafter [13] tool is used in the case of the traditional networking stack. The timestamps in the figures below are represented in seconds.

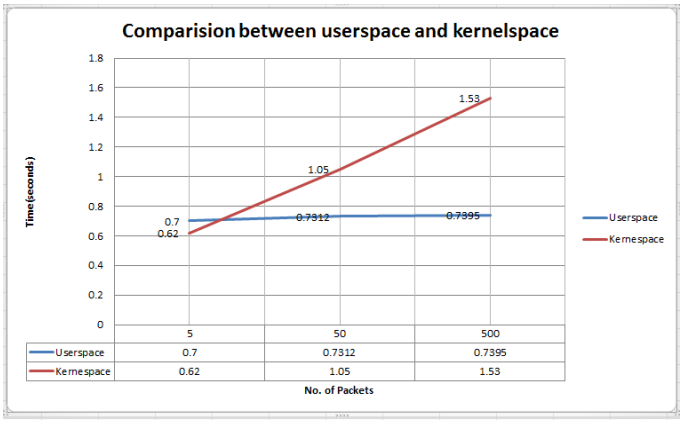

Fig. 1 A comparison between performance of user space packet forwarding and kernel space packet forwarding

Thus through various literature surveys we have carried out an extensive study of alternative approaches for packet forwarding in virtual networks. We build a virtual router in user space by bypassing the kernel. In our approach, we bypass the kernel and forward the packets through user space process. The advantages of this approach are increased efficiency and better performance in case of high number of packets coming in at the NIC, as there is no redundancy i.e. no need of copying packets from user space to kernel space.

VIII. FUTURE SCOPE

OpenContrail [14] is a network virtualization platform for the cloud. Open Contrail is an open source project that implements the networking constructs and elements entirely in software. The component of Open Contrail responsible for packet forwarding is vRouter. The vRouter implements the forwarding plane as a kernel module. Our future scope mainly lies in extending the OpenContrail architecture to add user space forwarding functionality in the vRouter. We also plan to replace the Longest Prefix Matching algorithm with MPLS (Multi-Protocol Label Switching) over GRE(Generic Routing Encapsulation). Multiprotocol Label Switching (MPLS) is a type of data carrying technique for high-performance telecommunications networks that directs data from one network node to the next based on short path labels rather than long network addresses, avoiding complex lookups in a routing table. Generic Routing Encapsulation (GRE) is a tunnelling protocol developed by Cisco Systems that can encapsulate a wide variety of network layer protocols inside virtual point-to-point links over an Internet Protocol network.

REFERENCES

[1] Clark, David D., et al. "An analysis of TCP processing overhead." IEEE Communications magazine 27.6 (1989): 23-29.
[2] Barbette, Tom, Cyril Soldani, and Laurent Mathy. "Fast userspace packet processing." Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems. IEEE Computer Society, 2015.
[3] I. Marinos, R. N. Watson, and M. Handley “Network stack specialization for performance” Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, page 9. ACM, 2013.
[4] https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt
[5] http://info.iet.unipi.it/~luigi/netmap/
[6] http://www.ntop.org/products/packet-capture/pf_ring/
[7] http://dpdk.org/
[8] Scholz, Dominik. "Diving into Snabb." Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM) 31 (2016).
[9] https://github.com/snabbco/snabb
[10] http://luajit.org/
[11] Nunes, Bruno Astuto A., et al. "A survey of software-defined networking: Past, present, and future of programmable networks." IEEE Communications Surveys & Tutorials 16.3 (2014): 1617-1634.
[12] Comer, Douglas E. Computer networks and internets. Prentice Hall Press, 2008.
[13] https://github.com/pellegre/libcrafter
[14] http://www.opencontrail.org/