Experimental Study on Packet Forwarding Performance of Virtual Software Routers (Postprint)

Authors: He Peng, Yang Jianhua, Zhang Jianhua, Xie Gaogang

Date: 2017-03-09T00:00:00+00:00

Abstract

The architecture of future networks must evolve adaptively according to network services, making flexible and high-performance routers fundamental to constructing future networks. Virtual software routers support network virtualization and programmability, enabling them to host multiple logical networks with different architectures on a single physical network, thereby facilitating service innovation and offering excellent flexibility. However, the overhead introduced by virtualization significantly impacts the packet forwarding performance of virtual routers. Consequently, how to improve forwarding performance while preserving flexibility has become an urgent issue to address. This paper introduces different virtualization technologies and scalable routing software, constructs an experimental platform, and measures the forwarding performance of KVM (full virtualization), Xen (paravirtualization), OpenVZ (operating system-level virtualization), among others. It also evaluates the performance improvements achieved by various I/O acceleration technologies. The experimental data provides important guidance for identifying forwarding performance bottlenecks in virtualized software routers and for enhancing forwarding performance.

Full Text

Preamble

Vol.8 No.6
Information Technology Letter

Experimental Study on Packet Forwarding Performance of Virtual Software Routers

He Peng, Yang Jianhua, Zhang Jianhua, Xie Gaogang

Abstract: Future network architectures require adaptive evolution based on network services, making flexible yet high-performance routers fundamental for

building next-generation networks. Virtual software routers support network virtualization and programmability, enabling multiple logical networks with different architectures to coexist on a single physical infrastructure for service innovation. However, virtualization overhead significantly impacts packet forwarding performance. How to improve forwarding performance while maintaining flexibility has become an urgent challenge. This paper introduces different virtualization technologies and scalable routing software, constructs an experimental platform, and measures the forwarding performance of KVM (full virtualization), Xen (paravirtualization), and OpenVZ (operating system-level virtualization). We also evaluate the performance improvements achieved by various I/O acceleration techniques. These experimental results provide important guidance for identifying performance bottlenecks and optimizing forwarding performance in virtual software routers.

Keywords: virtualization, software router, forwarding performance, I/O acceleration

1 Introduction

Network virtualization has attracted widespread attention in recent years. As core network devices in virtual networks, virtual routers offer two key advantages over existing network equipment. First, virtual routers can partition a single physical routing device into multiple virtual routers that process different traffic flows in parallel and provide corresponding functional extensions based on service characteristics. This meets the diverse packet processing requirements of emerging applications such as IoT and cloud computing services. Additionally, virtual routers can allocate specific resources to particular virtual devices according to the scale of the virtual network, maximizing resource utilization. For example, a telecom operator can use one virtual network to carry a large enterprise' s traffic while using another for multiple small companies, with complete isolation between virtual networks enhancing network robustness. Second, in a virtual network built with virtual routers, different virtual devices can support different protocol stacks—one might use TCP/IP while another employs an entirely new protocol. This enables parallel experiments on the same physical infrastructure without interference, facilitating future Internet research while providing faster deployment and lower costs than physical testbeds. Many nations and organizations are actively researching future Internet development, such as the U.S. GENI project [3] and the EU FIRE project [4], which have proposed numerous new architectures. Virtual networks can support experimental needs for these studies without affecting normal network operations, enabling faster deployment of advanced architectures into production networks.

Existing virtual router platforms fall into two categories: general-purpose hardware platforms and specialized hardware platforms. General-purpose platforms use scalable router software (e.g., Click [1], XORP [2]) combined with virtualization software (e.g., Xen [16], OpenVZ [17]). Recent years have seen significant breakthroughs in software router forwarding performance on general-purpose

hardware. Dobrescu et al. [5] designed and implemented the RouteBricks software router based on a cluster approach, fully exploiting the throughput potential of individual physical devices. Experimental results show that a RouteBricks cluster built from four general-purpose servers can provide up to 35 Gbps throughput. Sangjin Han et al. [6] leveraged the powerful parallel computing capabilities of GPUs to develop PacketShader, a general-purpose hardware router achieving 39 Gbps throughput on a single machine. Specialized hardware platforms redesign the router data plane using dedicated hardware. For instance, Anwer et al. [9] first used NetFPGA [21] to implement data planes for multiple virtual routers, with each plane adopting the same structure to support functions like table lookup and forwarding. Guohan Lu et al. [10] proposed CAFÉ, a configurable packet forwarding engine that uses different APIs to configure underlying hardware for different data planes, avoiding hardware redesign. Deepak Unnikrishnan et al. [11] introduced hardware reconfigurability and software-hardware migration concepts—when traffic to a virtual router increases, its routing table can be copied to hardware for migration to a hardware data plane, while the original hardware data plane migrates to software. When deploying new data planes, all can be moved to software while writing new logic to the FPGA. Anwer et al. [12] proposed SwitchBlade, a modular pipeline architecture supporting hardware programmability that enables rapid prototyping and deployment of new protocols while maintaining high-speed hardware forwarding performance. Virtual routers using specialized hardware must ensure both high performance and scalability of the underlying data plane.

The surge in network bandwidth demands extremely high router throughput. Traditional routers, with simple packet processing and extensive optimization of data and control planes, provide very high throughput. Virtual software routers, due to virtualization and extensibility features, require even higher forwarding performance to support deep packet processing for emerging services. Regardless of design approach, virtualization technology is essential, and its introduction inevitably impacts router I/O efficiency. Therefore, evaluating the performance overhead of different virtualization technologies is necessary. This paper measures the forwarding performance of several representative software routers implemented on virtualization platforms and evaluates common I/O acceleration techniques to identify performance bottlenecks and optimize forwarding performance.

The remainder of this paper is organized as follows: Section 2 introduces virtual router technologies, including principles of different virtualization approaches and architectures of several scalable routers. Section 3 tests and presents results for packet forwarding performance of common virtual software routers. Section 4 introduces common I/O acceleration techniques, constructing and evaluating these technologies. Section 5 concludes and outlines future research directions for I/O acceleration.

2.1 Virtual Software Router Architecture

The typical framework of a virtual software router consists of three major components: virtualization software platform, data forwarding module, and control module, as shown in [Figure 1: see original paper]. The virtualization software acts as a hypervisor layer, providing virtual I/O channels to different virtual machines. The data forwarding module can be implemented with either specialized hardware or general-purpose NICs. Inside the virtual machine, the control module computes routes or supports experimental networks for implementing new protocols (e.g., OpenFlow [7]).

2.2 Virtualization Implementation

Virtualization technologies used in virtual software routers can be broadly classified into three categories based on virtualization layer:

Full Virtualization

Full virtualization, also known as native virtualization, employs a hypervisor layer between the operating system and underlying hardware, enabling hardware sharing among upper-layer operating systems. When a guest OS executes a privileged instruction, the hypervisor intercepts the code and performs the corresponding operation (e.g., writing a value to a hardware register). Since guest operating systems cannot directly control underlying hardware and must go through the hypervisor proxy, I/O performance is impacted. Compared to hardware virtualization, full virtualization supports different operating systems but cannot simulate different hardware platforms. Representative software includes VMware and KVM [14]. Recent x86 CPUs provide instruction-level support for full virtualization, enabling near-native performance for guest operating systems.

Paravirtualization

Paravirtualization, also called quasi-virtualization, similarly uses a hypervisor layer to isolate hardware and operating systems. However, it modifies the OS code so the guest OS knows it runs on a virtualization platform, significantly improving I/O performance. For example, when a NIC receives a packet, full virtualization requires the hypervisor to simulate a hardware interrupt before transferring the packet to an OS. In paravirtualization, the OS knows it runs on a virtualized platform, allowing the hypervisor to batch-transfer packets without simulating hardware interrupts. Since paravirtualization requires OS code modification, it is difficult to apply to non-open-source commercial operating systems like Windows. Representative software includes Xen [16] and UML [15].

Operating System-Level Virtualization

Operating system-level virtualization differs from the previous two approaches, resembling UNIX's chroot system call. It uses OS mechanisms (e.g., namespaces in Linux) to place a process group into a "container" isolated from the host OS processes, limiting CPU and memory usage to achieve resource isolation and allocation. This technology only runs on specific operating systems (almost exclusively Linux). All "virtual machines" (called servers in this context) share a single kernel. Representative software includes OpenVZ [17], Linux-VServer [18], and LXC [19]. The overhead of OS-level virtualization is minimal, but its main disadvantage is the inability to run multiple heterogeneous operating systems.

2.3 Data Forwarding Module

Click

Click is a modular software router that modularizes router design components (called "elements" in Click), providing a flexible and extensible router software suite. Click abstracts two operations for connecting elements: Push, where a packet is passed from source to destination after processing at the source; and Pull, where the destination initiates a request to the source for packet processing. [Figure 2: see original paper] shows an Ethernet switch framework built using Click. However, while Click can forward packets to designated ports, it does not provide routing computation capabilities and cannot calculate routes dynamically. The Click software suite modifies the Linux kernel to provide a polling driver, significantly improving small-packet forwarding performance on ordinary NICs. Many software router studies [5,8] use Click as their data forwarding module. Some router software like XORP can also use Click as a forwarding engine. RouteBricks achieved remarkable throughput using Click' s polling driver.

Hardware Data Plane

Using specialized hardware to build router forwarding modules offers incomparable performance advantages over software, with hardware designed to integrate routing tables. Using TCAM for fast route lookup significantly reduces CPU burden. The drawback is that hardware resources are extremely limited—currently the largest TCAM can only store thousands of routing entries, far from meeting the 200,000-300,000 entry requirements of real networks. Hardware-based forwarding modules require algorithm design to build "fast tables," handling frequently looked-up IP entries in hardware and less frequent ones in software.

2.4 Control Module

XORP

Recognizing that the closed nature of commercial router software hindered new protocol research and implementation, Handley et al. designed and implemented XORP to provide an open, extensible router software platform for researchers. XORP has been widely adopted in industry and academia. XORP uses a multi-process mechanism that isolates packet switching, routing computation, and forwarding engines through a Forward Engine Abstraction layer, ensuring robustness, openness, and performance. [Figure 3: see original paper] shows the XORP architecture, which uses Click as its forwarding engine.

OpenFlow

OpenFlow, proposed by Stanford University, represents a new network architecture using a master-slave structure for network management. In an OpenFlow network, the Controller stores all forwarding policies while OpenFlow Switches handle data forwarding. Each switch contains a flow table storing forwarding rules. When a packet matches a rule, the switch forwards it according to the rule. When no match occurs, the switch consults the controller for new rules. OpenFlow uses a "ten-tuple" (MAC pairs, IP pairs, port pairs, etc.) as the basis for forwarding policies, enabling complex forwarding strategies. The controller provides APIs for operating the flow table, meeting the needs of programmable networks.

3 Forwarding Performance Testing on Different Virtual Platforms

We selected three mainstream virtualization platforms based on virtualization layer differences—KVM (full virtualization), Xen (paravirtualization), and OpenVZ (operating system-level virtualization)—for packet forwarding tests. The experimental platform parameters are shown in .

Table 1. Experimental Platform Parameters - CPU: Intel Xeon L5420 (2 sockets, 4 cores each, total 8 cores) - L1 Cache: 32KB per core (16KB instruction + 16KB data) - L2 Cache: 6MB shared between pairs of cores - Memory: 16GB DDR2 667MHz - Motherboard: SuperMicro X7DWU - NIC: Intel 82571EB Gigabit, 4 ports

3.1 Experimental Configuration

We used Spirent Testcenter as the test instrument, adopting the RFC 2544 standard test method to measure throughput across different virtualization software. Selected packet sizes were 64 bytes (minimum), 512 bytes (average network packet size), and 1518 bytes (maximum). Based on these results, we designed additional experiments for multi-VM scenarios, running different VMs on different network segments to test overall forwarding performance. Since Spirent does

not support this testing method, we used an alternative approach to characterize throughput, as detailed in Section 3.2.2.

Different virtualization software uses different network configuration methods; we only selected bridge mode for our tests. Below is a brief introduction to each software and its network configuration.

KVM KVM (Kernel Virtual Machine) differs from mainstream virtualization technology Xen in that it does not implement a separate hypervisor layer but instead adds modules to the kernel, using the kernel itself as the hypervisor. KVM is a full virtualization technology requiring CPU virtualization support. [Figure 4: see original paper] shows the KVM network configuration, where eth_x represents Ethernet interfaces, tap_x represents mirrored network ports of guest OS interfaces in the hypervisor layer, and Br_x represents bridge modules.

Xen Xen, developed by Cambridge University, is a paravirtualization software implementing a minimal hypervisor layer. Xen uses the term "Domain" to describe virtualization environments, with two types: Dom0 (Driver Domain) and DomU (Guest Domain). Dom0 can directly access hardware, while DomU must go through Dom0 proxy. Tests used bridge mode for both Dom0 and DomU forwarding performance. [Figure 5: see original paper] shows Xen DomU network configuration, where peth_x represents physical NIC ports and vif_x represents mirrored network ports of guest OS interfaces in the hypervisor layer.

OpenVZ OpenVZ is the open-source version of Swsoft's proprietary Virtuozzo software, using virtual network interface pairs to implement bridging. [Figure 6: see original paper] shows OpenVZ network configuration, where veth_x represents mirrored network ports of guest OS interfaces in the hypervisor layer.

3.2.1 Native Operating System Testing

We first tested a system platform without any virtualization software (Native Linux). The results are shown in . The results show that ordinary NICs perform poorly with minimum-sized packets, primarily due to excessive system overhead from interrupt-driven approaches. Studies [5,6] indicate that using polling instead of interrupts can significantly improve small-packet forwarding performance.

Table 2. Native Linux Forwarding Performance Test | Packet Size | Theoretical Max Rate (fps) | Actual Rate (fps) | Rate (%) | |———-|————————|————————|———-| | 64 bytes | 2,976,190 | 778,898 | 26.17% | | 512 bytes | 469,924 | 469,924 | 100% | | 1518 bytes | 162,548 | 162,548 | 100% |

3.2.2 Virtualization Technology Performance Test Results and Analysis

Table 3. Forwarding Performance Test Results for Various Virtualization Technologies | Packet Size | Xen Dom0 (fps) | Xen DomU (fps) | OpenVZ (fps) | |———–|————-|————-|———–| | 64 bytes | 611,488 | 113,452 | 590,565 | | 512 bytes | 469,924 | 139,506 | 469,924 | | 1518 bytes | 162,548 | 124,831 | 162,548 |

Table 3 shows packet forwarding performance results for different virtualization software. Full virtualization introduces substantial I/O performance overhead due to long kernel paths and excessive context switches. Paravirtualization, optimized for virtualized environments, shows some improvement over full virtualization but still lags significantly behind native OS performance. OS-level virtualization, with minimal overhead, performs very close to native OS in all metrics except small-packet forwarding.

Based on these results, we conducted multi-VM performance tests on Xen and OpenVZ, shown in using percentage of theoretical maximum forwarding rate.

Table 4. Forwarding Performance with Different Numbers of VMs | # VMs | Xen DomU (64B) | Xen DomU (512B) | Xen DomU (1518B) | OpenVZ (64B) | OpenVZ (512B) | OpenVZ (1518B) | |—-|————-|————-|————|————-|————| | 1 | 7.54% | 29.70% | 76.77% | 26.17% | 100% | 100% | | 2 | 6.12% | 22.73% | 60.20% | 26.17% | 100% | 100% | | 4 | 10.39% | 53.84% | 69.62% | 26.17% | 100% | 100% | | 8 | 9.53% | 52.50% | 61.16% | 26.17% | 100% | 100% |

The results show that VM count has minimal impact on throughput. Therefore, improving throughput requires shortening packet kernel paths and enabling VMs to directly access devices. Existing methods can significantly improve virtual I/O performance and are discussed in the next chapter.

4.1 I/O Performance Acceleration Technologies

The previous test results demonstrate that full virtualization overhead significantly impacts I/O performance. However, full virtualization allows unmodified operating systems to run, offering high applicability. Researchers have developed acceleration techniques that substantially improve virtual I/O performance.

Virtio (Paravirtualized I/O) [20] Virtio, designed and implemented by Australian programmer Rusty Russell, provides a standardized paravirtualized I/O interface for the growing number of virtualization solutions in Linux. Paravirtualized I/O enables guest operating systems to recognize they run in a virtualized environment and adopt optimizations to improve I/O performance. [Figure 13: see original paper] illustrates the principle. Currently, KVM supports this virtual I/O acceleration technology.

Intel VT-d and SR-IOV Technologies While paravirtualized I/O improves guest OS I/O performance, it still suffers from high context-switch overhead. The solution lies in eliminating hypervisor proxy and enabling direct device access. This involves two problems: (1) Direct Memory Access (DMA) remapping for guest OS device buffer memory, and (2) interrupt remapping. Intel VT-d (Intel Virtualization Technology for Directed I/O) solves both. As shown in [Figure 14: see original paper], the left half shows full virtualization I/O requiring hypervisor proxy, while the right half shows DMA remapping allowing hardware to write data directly to guest OS buffers, achieving near-native I/O performance.

VT-d provides a direct path from hardware to guest OS, while SR-IOV (Single-Root I/O Virtualization) provides a method to virtualize a single device into multiple devices. SR-IOV provides independent memory space and interrupt resources for each VM, introducing two function types: Physical Functions (PFs) and Virtual Functions (VFs). PFs support SR-IOV extension features for configuration and management, while VFs are "lightweight" PCIe functions containing essential data migration resources. The SR-IOV architecture allows an I/O device to support multiple virtual functions, enabling I/O device and port sharing without software emulation. [Figure 15: see original paper] shows the SR-IOV architecture.

PCI Passthrough Technology PCI Passthrough, developed by Xen developers, enables guest OS direct device access by using software to emulate IOMMU, allowing guest OS to bypass the hypervisor. Compared to VT-d, which is hardware-based IOMMU support, PCI Passthrough provides software-level IOMMU support for systems without VT-d, enabling I/O performance optimization.

Software Router Design Experiences RouteBricks Design Experience: RouteBricks optimizes single-router forwarding performance through three methods: (1) Using polling instead of traditional hardware interrupts, dramatically improving 64-byte small-packet forwarding on general-purpose hardware; (2) Employing batch processing to transfer multiple packet addresses per bus transaction, fully utilizing PCIe bandwidth; (3) Leveraging NIC multi-queue features by binding different queues to different cores, greatly improving data processing parallelism. These methods enable a single server to achieve 9.77 Gbps 64-byte packet forwarding rates.

PacketShader Design Experience: PacketShader optimized the Linux kernel network stack, reducing the kernel packet structure sk_buff from 208 bytes to 8 bytes and using batch processing to significantly improve packet processing performance. Experiments show this optimization alone increases packet throughput to 10.5 Gbps, a 13.5x improvement. PacketShader also leverages multi-core multi-queue features and, considering its NUMA architecture, carefully optimizes packet distribution balance and data locality to achieve 40 Gbps

overall forwarding performance.

4.2 Virtual I/O Acceleration Technology Performance Evaluation

This section evaluates the I/O performance improvements from VT-d, paravirtualized I/O, and PCI Passthrough using the RFC 2544 standard throughput test. Additionally, IPerf [22] bandwidth tests were conducted for paravirtualized I/O. Since Xen does not support Virtio and our platform could not enable VT-d for Xen, we tested VT-d and Virtio on KVM and PCI Passthrough on Xen.

Table 5. KVM Test Results | Packet Size | KVM Virtio (fps) | KVM VT-d (fps) | |———-|————|————-| | 64 bytes | 113,452 | 841,696 | | 512 bytes | 139,506 | 413,750 | | 1518 bytes | 124,831 | 162,548 |

Table 6. PCI Passthrough Test Results | Packet Size | Xen DomU (fps) | Xen DomU + PCI Passthrough (fps) | |———-|————-|———————-| | 64 bytes | 113,452 | 230,650 | | 512 bytes | 139,506 | 222,110 | | 1518 bytes | 124,831 | 162,548 |

Table 7. KVM Paravirtualized I/O IPerf Bandwidth Test | Test Type | KVM Virtio | |——-|———| | TCP | 749 Mbps | | UDP | 555 Mbps |

Experimental results show that paravirtualized I/O indeed improves TCP transmission bandwidth for guest operating systems, but offers limited performance gains for packet forwarding. VT-d technology provides guest OS with direct hardware access, achieving near-native forwarding performance and minimizing virtualization overhead. PCI Passthrough performance falls between the two due to software emulation overhead.

5 Conclusion

This paper explores virtualization technologies, related I/O acceleration techniques, and widely recognized scalable router software in the context of virtual routers. We measured the forwarding performance of several representative virtualization platforms, providing insights for virtual router research and design. Results demonstrate that OS-level virtualization introduces minimal overhead but has limitations due to its low virtualization level. Hardware-assisted virtualization technologies like VT-d enable guest operating systems in full virtualization environments to achieve native I/O efficiency while maintaining high virtualization levels. Although current virtual router designs primarily use lightweight virtualization like OpenVZ, VT-d and SR-IOV technologies will significantly impact future virtual router design.

Most current research does not use general-purpose platforms for virtual routers, instead relying on specialized hardware to reduce virtualization overhead and lightweight virtual systems like OpenVZ to further minimize overhead. However,

with SR-IOV and related technologies reducing virtualization overhead to a minimum, experiences from software router I/O optimization can be applied to virtual router design. We anticipate that research on software virtual routers using SR-IOV technology will emerge within the next one to two years.

References

[1] Robert Morris, et al. The Click modular router. SOSP, 1999
[2] Mark Handley, et al. XORP: An Open Platform for Network Research. ACM SIGCOMM Computer Communication Review, 2003
[3] GENI. http://www.geni.net/
[4] FIRE. http://cordis.europa.eu/fp7/ict/fire/
[5] MiHai Dobrescu, et al. RouteBricks: Exploiting Parallelism To Scale Software Routers. SOSP 2008. Best Paper Awards.
[6] Sangjin Han, et al. PacketShader: a GPU-Accelerated Software Router. SIGCOMM 2010.
[7] Nick McKeown, et al. OpenFlow: Enabling Innovation in College Networks. ACM SIGCOMM Computer Communication Review, 2008
[8] Nobert Egi, et al. Towards High Performance Virtual Routers on Commodity Hardware. ACM CoNEXT, 2008.
[9] Muhammad Bilal Anwer, et al. Building a Fast, Virtualized Data Plane with Programmable Hardware. VISA 2009.
[10] Guohan Lu, et al. CAFÉ: A Configurable pAcket Forwarding Engine for Data Center Networks. PRESTO' 09.
[11] Deepak Unnikrishnan, et al. Scalable Network Virtualization Using FPGAs. FPGA' 10
[12] Muhammad Bilal Anwer, et al. SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware. SIGCOMM 2010.
[13] QEMU. http://wiki.qemu.org/
[14] Kernel Virtual Machine. KVM. http://www.linux-kvm.org/

Note: Figure translations are in progress. See original paper for figures.