www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 4 April 2015, Page No. 11687-11690

Implementation of VRouter in KVM virtualized host

Kajal Gaikwad, Shweta Nazarkar, Praneta Painthankar, Srushti Khadke
Department of compute Engineering
Pune Institute of Computer Technology
Pune, India
Department of compute Engineering
Pune Institute of Computer Technology
Pune, India
Department of compute Engineering
Pune Institute of Computer Technology
Pune, India
Department of Computer Engineering
Pune Institute of Computer Technology
Pune, India

***Abstract*—A hypervisor is a piece of computer software, firmware or hardware that creates and runs multiple virtual machines. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Virtual switch which is a part of the hypervisor also called as hypervisor switch does the switching between VM traffic. Currently if two VMs those are interested in communication and if they are in same subnet then VM-VM traffic will traverse from source VM to destination VM through hypervisor switch. And if two VMs are in different subnet then traversing of packets will need external devices such as external switch and router which increases the traffic in the network. We can improve the quality of the system by implementing the functionality of router within the hypervisor.**

Keywords—hypervisor, Router, KVM.

I. INTRODUCTION

System virtualization adds a hardware abstraction layer, called the Virtual Machine Monitor (VMM), on top of the bare hardware. This layer provides an interface that is functionally equivalent to the actual hardware to a number of virtual machines. These virtual machines may then run regular operating systems, which would normally run directly on top of the actual hardware. This characterization is a bit oversimplified. There are various virtualization techniques as well as requirements for architectures to be virtualizable[2]. The main motivation for virtualization in the early 70's was to increase the level of sharing and utilization of expensive computing resources such as the mainframes. The 80's saw a decrease in hardware costs that caused a significant portion of the computing needs of an organization to be moved away from large centralized mainframes to a collection of departmental minicomputers. The main motivation for virtualization disappeared and with it their commercial embodiments.

Ubiquitous networking brought the distribution of computing to new grounds. Large number of client machines connected to numerous servers of various types gave rise to new computational paradigms such as client-server and peer-to-peer systems. These new environments brought with them several challenges and problems including reliability, security, increased administration cost and complexity, increased floor space, power consumption, and thermal dissipation requirements. The recent rebirth of the use of virtualization techniques in commodity, inexpensive servers and client machines is poised to address these problems [3,4] in a very elegant way.

Virtualization helps provide software isolation to applications in such shared environments as well as ensures better utilization of server resources. On cloud systems, I/O intensive applications are good candidates for virtualization because they have sufficient spare CPU cycles which can potentially be used by some other co-hosted application. Many virtual machine monitors have emerged in such a scenario, varying from VMware ESX to Xen paravirtualized hypervisor. In recent years, to make development of virtual machine monitors easy, hardware vendors like AMD and Intel have added virtualization extensions to x86 processors which were initially difficult to virtualize and were not in tune with Popek and Goldberg virtualization requirements [1].

But in current virtualized server if we want to communicate between two virtual machines which are in different subnet then routing hardware is necessary. And which may cause high traffic. So this paper describes the basic concepts in virtualization and explains how to manage communication between two virtual machines which are in different subnet without involving hardware.

The rest of the paper is organized as follows. Section II describes the KVM architecture. Section III describes the existing system and how the same can be modified to achieve the desired objective. Section IV describes the design model and section V describes proposed packet flow and section VI concludes by discussing the potential advantages and future work.

II. KVM ARCHITECTURE

Virtualization capabilities in Linux kernel using x86 hardware virtualization extensions [5][6]. It is a full virtualization solution, where guests are run unmodified in

VMs. It consists of two modules, namely, kvm.ko and an architecture dependent kvm-amd.ko or kvm-intel.ko module. Under KVM, each VM is spawned as a regular linux process named KVM and scheduled by the default linux scheduler. For KVM the hardware has to support three processor modes, namely user, kernel and guest mode. The guest mode is added to support hardware assisted virtualization. The virtual machine executes in this guest mode which in turn has user and kernel mode in itself [7][8].

For using shared I/O hardware, these VMs interact with qemu emulator in host user space which provides emulated I/O devices for virtual machines. For instance, in the case of network related applications, Qemu provides emulated Network Interface Card (NIC) to VMs and interacts with tun-tap device on the other side. The tap device is connected to physical NIC through a software bridge.

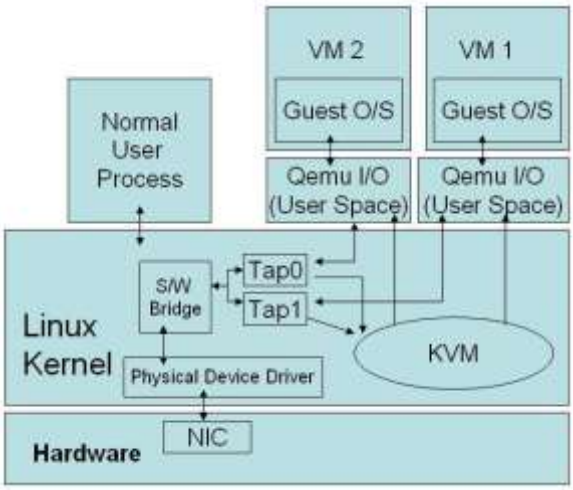

Fig.1 KVM Architecture

Fig 1 shows the typical KVM architecture, with reference to a network related application. A typical network packet flows through the KVM virtualized host in the following way. As depicted in picture, when a packet arrives at physical NIC, interrupts generated by NIC are handled by the physical device driver. The device driver forwards the packet to software bridge. The bridge, then pushes the packet to the tap device of the corresponding VM. The tap device is a virtual network device that sends a signal to KVM module. KVM module in turn, generates a virtual interrupt to the user space qemu of the target VM. Qemu then copies the packet from tap device and generates the interrupt for the guest OS emulating the virtual NIC. Again, the physical device driver in the guest OS handles the packet transfer to the VM's address space.

Consequently, for a VM process in KVM virtualized server, guest mode execution (both kernel or user mode) corresponds to execution within a VM while other modules in user mode (qemu) and kernel mode (KVM module, tun-tap module, bridge module, etc.) correspond to hypervisor execution. Among these, Qemu I/O module runs separately for each VM but is co-ordinated by a single KVM module which manages VMs by signal and virtual interrupts. Hence, it is easy to understand that the KVM module can potentially become a bottleneck when it tries to execute on behalf of many VMs.

III. EXISTING SYSTEM

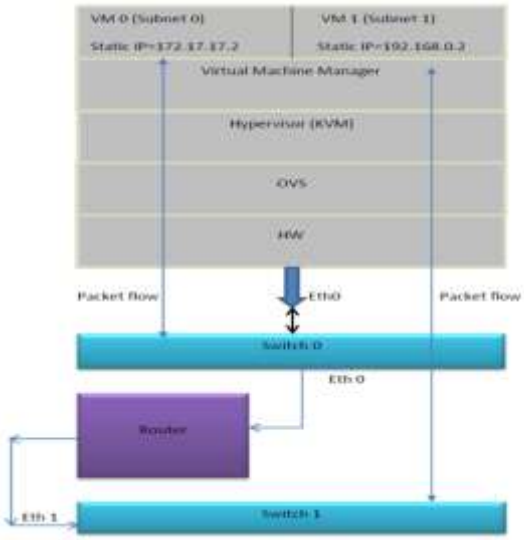

Fig.2 Existing System

Fig.2 represents the flow of packets in current scenario .Here both VM0 and VM1 are in different subnets deployed on same physical host. There are multiple components in fig.2 which are as given below.

- VM0:

  It is source virtual machine which is having an IP address 172.17.17.2.This virtual machine will send some message to another virtual machine through hypervisor control.

- VM1:

  It is destination virtual machine which is having an IP address 192.168.0.2.

- Hypervisor:

  A hypervisor or virtual machine monitor (VMM) is a piece of computer software, firmware or hardware that creates and runs virtual machine. KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module, kvm.ko, that provides the core virtualization infrastructure and a processor specific module, kvm-intel.ko or kvm-amd.ko. KVM also requires a modified QEMU although work is underway to get the required changes upstream.

- OVS:

  Open vSwitch, sometimes abbreviated to OVS, is a production-quality open source implementation of a distributed virtual multilayer switch. The main purpose of Open vSwitch is to provide switching stack for hardware virtualization environments, while supporting multiple protocols and standards used in computer networks.

  In this scenario the packet flow from VM0 to VM1 is as given below:

- The packet from VM0 i.e. from guest moves towards VNIC (virtual NIC).
- From VNIC it gets forwarded to host through hypervisor (e.g. KVM).

- Here in host kernel checking is done for destination address. And since the destination address is in another subnet the packet moves towards host hardware i.e. NIC.
- From NIC it gets forwarded to switch1 and then to physical router. The physical router finds out the path for packet and forwards it to switch0.
- Switch0 then forwards the packet to destined virtual machine i.e.VM1.

But, in this case the extra network traffic gets created because of packet forwarding in physical devices where the source and destination are on same physical machine. This overhead in the network can be minimized by implementing routing functionality at hypervisor level.

IV. PROPOSED SYSTEM

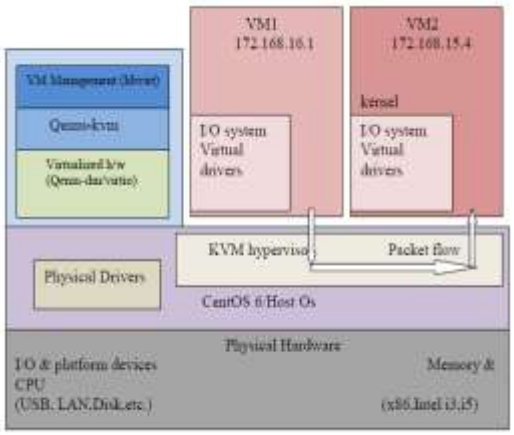

Fig.3 Proposed system

Fig. 3 describes the proposed system. It consists of physical machine with KVM installed on it. And the virtual machines are deployed on it. KVM is supported with VM management tool.

The components of the system are

- LibVirt
- Qemu KVM
- Virtio
- Virtual machines
- OVS
- Hypervisor

1. LibVirt: A toolkit to interact with the virtualization capabilities of recent versions of Linux. It has many C libraries. Libvirt is used by various virtualization programs and platforms. Graphical Interfaces are provided by Virtual Machine Manager and others. The most popular command line interface is 'virsh'
2. Qemu-KVM: The KVM project is maintaining a fork of QEMU called qemu-kvm. It is used as hypervisor management.
3. Virtio: Virtio is a virtualization standard for network and disk device drivers where just the guest's device driver "knows" it is running in a virtual environment, and cooperates with the hypervisor. This enables guests to get high performance network and disk operations, and gives most of the performance benefits of paravirtualization.

V. PRPOSED PACKET FLOW

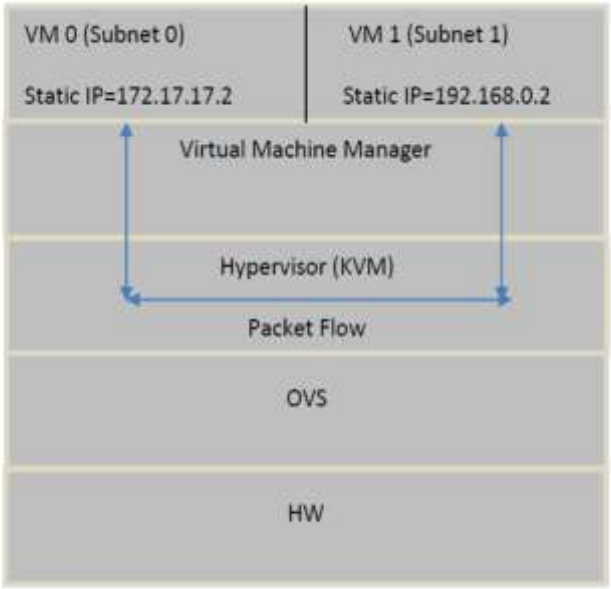

FIG.4 PROPOSED PACKET FLOW

The proposed packet flow is as shown in the fig.4. Our main aim with proposed packet flow is

- VM-to-VM communication on the same system without intervention of external router or switch.
- To maintain QoS (bandwidth control for the individual VM).
- Openstack, OpenNebula or any other private and public cloud depends on kvm Can control VM from central orchestrator.

VI. CONCLUSION

- Improvement in routing will reduce trafficking overhead of traversing packets (Data-Plane) through router though sender and receiver are in same physical host.
- This project will improve functionality of routing and thus will reduce traffic overhead.

But this modification will apply for communication between two different virtual machines which are in different subnets within same physical server and not for multiple physical servers.

References

[1] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," Commun. ACM, vol. 17, no. 7, pp. 412–421, Jul. 1974. [Online]. Available: http://doi.acm.org/10.1145/361011.361073

[2] G.J. Popek and R.P. Goldberg, "Formal Requirements for Virtualizable Third-Generation Architectures," Comm. ACM , July 1974, pp. 412–421.I.S. Jacobs and C.P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

[3] R. Figueiredo, P.A. Dinda, and J. Fortes, "Resource Virtualization Renaissance," IEEE Internet Computing, May 2005, Vol. 38, No. 5.

[4] M. Rosenblum and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends," IEEE Internet Computing, May 2005, Vol. 38, No. 5.

[5] (2012) Main page-kvm. [Online]. Available: http://www.linux-kvm. org/page/MainPage

[6] A. Kivity, "kvm: the Linux virtual machine monitor," in OLS '07: The 2007 Ottawa Linux Symposium, Jul. 2007, pp. 225–230.

[7] S. Zeng and Q. Hao, "Network i/o path analysis in the kernel-based virtual machine environment through tracing," in Information Science and Engineering (ICISE), 2009 1st International Conference on, dec. 2009, pp. 2658 –2661.

[8] J. Zhang, K. Chen, B. Zuo, R. Ma, Y. Dong, and H. Guan, "Performance analysis towards a kvm-based embedded real-time virtualization architecture," in Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on, 30 2010-dec. 2 2010, pp. 421 –426.