Efficient Inter-domain Network Communication in Virtualized Environments Using Shared Memory

PhD candidate: Stefanos Gerangelos
Advisor: Associate Professor Nectarios Koziris
Computing Systems Laboratory,
National Technical University of Athens
{sgerag,nkoziris}@cslab.ece.ntua.gr

Introduction

With the advent of virtualization technology and its propagation to the infrastructure of Cloud systems, there is an emergent request for more effective means of inter-domain communication between virtual machines. By characterizing a communication as *inter-domain*, we are referring to the interaction between two or more virtual machines located in the *same* physical host.

In this work we propose a framework for inter-domain network communication over Ethernet using shared memory techniques eliminating the intervention of the host component in the critical path. Our approach is based on *linux* (as the guest and the host operating system) and the *kvm* hypervisor [8].

Background and Related Work

The *kvm* hypervisor actually consists of a loadable kernel module, which creates a virtual device (namely */dev/kvm*) and exports a number of *ioctl* commands to this device. By issuing a *KVM_CREATE_VM* ioctl system call to the kvm device, a new guest is created, which from the host perspective is nothing more than just a user process with its own virtual address space. From the guest perspective this address space is translated to what is called guest physical address space and is being treated as ordinary physical memory. Kvm takes advantage of the hardware virtualization capabilities (currently Intel VT and AMD-V extensions) in order to virtualize cpu and mmu operations. When it comes to device virtualization though, every I/O request is trapped by QEMU [2], a user space emulator running on the host, and being emulated in order to access the respective device. This emulated approach results in poor performance in terms of network bandwidth. To overcome this problem, Xen's [1] approach was adopted by *virtio* [9]. Virtio is a framework which is based in Xen's *split-driver model* and installs a *frontend* driver in the guest operating system and a respective *backend* one in the host. With this *paravirtualized* solution, the virtio driver of the guest operating system is aware of being in a virtualized world and communicates with the relevant backend in this context. The backend driver communicates into the user space of the hypervisor to facilitate I/O through QEMU. However, using this approach requires at least 2 context switches (one for the TX side and one for the RX side), when a VM sents a network packet to another one located in the same physical machine.

Our approach is inspired by previous research on inter-domain network communication. Most of this research has been focused on implementations concerning the Xen hypervisor [12, 11, 10, 5, 6, 7]. Diakhate et al. [3] have proposed a solution for kvm using shared memory techniques by making changes to user space and specifically to QEMU. Finally, Mohebbi et al. [4] followed a quite similar approach to ours, however, this solution requires changing user applications in order for them to access the proposed virtual device.

Architecture

Our proposed framework includes the allocation of a shared memory space by the host, which it will expose to each guest every time one is created. So, when the kvm module is loaded, it allocates a number of memory pages and creates a ring buffer, which is the structure that every guest is going to use for its inter-domain communication. This model requires the implementation of a virtual network device for the guest. This device is added to the routing table of the guest with a respective rule concerning the inter-domain subnet. The driver of this device is based on the virtio [9] interface.

Eventually, a typical scenario of using our framework follows: upon the installation of the shared ring buffer by the kvm host, a guest (*guest 1*) is created. The host maps the shared ring buffer (which starts from a fixed memory address) to the address space of *guest 1*. After that suppose that another guest

(*guest 2*) is created. The host maps the same memory in the address space of *guest 2*. The host follows the same procedure for every other guest that creates, so at the end of the day every VM which resides in this host and wants to communicate with another VM in the same host must access the same ring buffer. Thus, each guest has to listen for incoming traffic network in this buffer. Now, if *guest 1* decides to send a network packet to *guest 2*, it has to lock for write the ring buffer, place the respective socket buffer (*sk_buff* struct in linux kernel) and make the appropriate changes to the respective ring buffer pointer. *Guest 2*, which is watching the ring buffer, checks the ip header of the *sk_buff*, realizes that this buffer is intended for itself and continues for processing it after it has updated the relevant pointer in the ring buffer. This data path is illustrated in Figure 1.

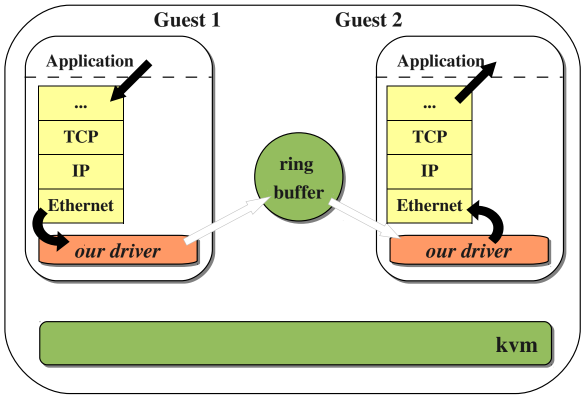

Figure 1: Proposed Framework's Data Path

The communication of the VM with the "outside world" is being done as usual using the default network interface (e.g. *eth0*). Our optimized approach eliminates the intervention of the host every time each guest sends/receives a network packet to/from the virtual device. The only time the host interferes, is when it installs a mapping to the ring buffer for a newly created guest, but this is outside of the critical path of the network communication. However, this scheme has some limitations concerning the locking mechanisms and the available concurrency of the system. Consider for example the case which several VMs decide to send packets through our device. In this case, the access to the ring buffer is expected to become a non-negligible bottleneck.

Summary and Future Plans

Our framework allows VMs located in the same physical host to communicate between them using TCP/IP semantics in an efficient manner. It includes minor modifications to the kvm hypervisor as well as the addition of a driver in the guest side in order to support our virtual network device. Our current agenda consists of evaluating this prototype in order to estimate our framework's efficiency. In the future, we plan to experiment further and add support for low latency protocols (like Myrinet, InfiniBand), which are highly used in the HPC world.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. A. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[2] F. Bellard. Qemu, a fast and portable dynamic translator. In *ATEC '05 Proceedings of the annual conference on USENIX Annual Technical Conference*, 2005.

[3] F. Diakhate, M. Perache, R. Namyst, and H. Jourdren. Efficient shared memory message passing for inter-vm communications. In *4th Workshop on Virtualization in High-Performance Cloud Computing (VHPC '08), Euro-par 2008*, 2008.

[4] O. K. Hamid Reza Mohebbi and M. Sharifi. Zivm: A zero-copy inter-vm communication mechanism for cloud computing. In *Computer and Information Science*, pages 18–27, 2011.

[5] Z. Hongyong, G. Kuiyan, L. Yaqiong, S. Yuzhong, and M. Dan. A highly efficient inter-domain communication channel. In *IEEE Ninth International Conference on Computer and Information Technology*, 2009.

[6] Y. S. Hongyong Zang and K. Gu. Optimizing interdomain communication. In *15th International Conference on Parallel and Distributed Systems*, 2009.

[7] K. Kangho, K. Cheiyol, J. Sung-In, S. Hyun-Sup, and K. Jin-Soo. Inter-domain socket communications supporting high performance and full binary compatibility on xen. In *ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2008.

[8] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liquori. kvm: the linux virtual machine monitor. In *Linux Symposium*, pages 225–230, Ottawa, Ontario, Canada, 2007.

[9] R. Russel. virtio: Towards a de-facto standard for virtual i/o devices. In *ACM SIGOPS Operating Systems*, 2008.

[10] M. J. K. Wei Huang and D. K. Panda. Efficient one-copy mpi shared memory communication in virtual machines. In *IEEE International Conference on Cluster Computing*, 2008.

[11] Q. G. Wei Huang, Matthew J. Koop and D. K. Panda. Virtual machine aware communication libraries for high performance computing. In *IEEE conference on Supercomputing*, 2007.

[12] P. R. Xiaolan Zhang, Suzanne McIntosh and J. L. Griffin. Xensocket: A high-throughput interdomain transport for virtual machines. In *International Conference on Middleware*, 2007.