

MMNet: An Efficient Inter-VM Communication Mechanism

Prashanth Radhakrishnan, Kiran Srinivasan
NetApp, Inc.
{shanth,skiran}@netapp.com

Abstract

This paper presents MMNet, an efficient, transparent, non-intrusive and dynamic communication mechanism between Xen Virtual Machines (VMs) that are co-located on the same physical machine. MMNet communication happens in a zero-copy manner, with minimal VMM (Virtual Machine Monitor) switches in the data path. All existing applications and most OS components can operate over MMNet without any modifications. MMNet can be deployed as a kernel module, without requiring patching of the guest OS or the VMM. Finally, MMNet can be enabled or disabled dynamically without restarting any network applications in the VM. Though there are other inter-VM communication mechanisms that satisfy a subset of the above features, MMNet convincingly satisfies all of them, albeit in an environment with a degree of trust among VMs.

1 Introduction

Virtual machine monitor (VMM) technology is becoming increasingly ubiquitous. In the desktop environment, virtual machines (VM) are being used for running multiple different operating systems simultaneously on a physical machine. Similarly, in enterprise IT, VMM technology is a popular tool for server consolidation to help reduce data center costs.

Increasingly, VMs are also finding their way into the architecture of enterprise class software systems and appliances (eg., Oracle VM [2]). Usually, in such systems, the software system is broken into co-operating components and each component is placed in a separate trusted VM. This sort of a design allows for increased fault-isolation between the collaborating components. In addition, it leverages other attractive features in a VM environment - migration, snapshotting, replay etc. In such architectures, the throughput and latency of communication between the component VMs are crit-

ical to the overall system performance.

We have identified four key design requirements for a communication mechanism between collaborating VMs that are co-located on a physical machine:

- **Performance:** This implies high throughput, low latency and acceptable CPU consumption.
- **Transparency:** The mechanism should present a transparent interface that requires no modification to a majority of the guest OS subsystems and all user applications. Also, this enables VM live migration to work seamlessly without disrupting running applications.
- **Dynamism:** The mechanism should be designed to be enabled or disabled without disrupting the execution of any applications within the VM. When the mechanism is disabled, packets should pass through the original data path.
- **Non-intrusiveness:** The mechanism should be designed to be deployed without requiring a patch to the guest OS or the VMM sources.

2 Design

2.1 Key Features

In this subsection, we discuss how MMNet satisfies the aforementioned design requirements.

Performance: In order to achieve high performance, we eliminate data copies and hypervisor calls in the critical path by employing a variant of shared-memory based communication.

During data transfer, typically shared-memory mechanisms either setup a designated, limited shared memory segment which entails copy costs [1] or employ hypervisor calls to enable mapping of pages to avoid copying [3]. Both these approaches add latency in the critical data path. In contrast, we eliminate data

copies by mapping in the entire physical memory of a VM into the address space of its communicating peer VM a priori in a read-only fashion. Since the collaborating VMs are assumed to have some degree of trust between themselves, the security implications are not a concern. Moreover, a bug in one VM cannot corrupt the other VM’s memory because of the read-only mapping, thus ensuring fault-isolation.

Shared-memory communication requires a signalling mechanism between the communicating peers. Existing systems typically utilize hypervisor functionality like virtual interrupts for signalling. Signalling happens in the critical data path and can affect performance. Our design is independent of the signalling mechanism and can employ variants of polling via shared memory counters or interrupts via the hypervisor (using the Xen event channel). We present results using the interrupt mechanism.

Transparency: In order to be transparent, the communication mechanism should present a standard well-known interface. Co-locating the VMs on a physical machine impacts the link-level properties of the communication layer between the VMs. Thus, we interject at the link-layer and export a standard ethernet interface (Figure 1). Since this is the lowest layer in the stack, most of the OS subsystems and all the user applications can work transparently.

Dynamism: MMNet tracks VM membership and dynamically establishes (or dismantles) efficient connections between co-located VMs. Within a VM, MMNet enables running applications to seamlessly switch to (or from) the efficient connection path by updating the IP routing tables appropriately.

Non-intrusiveness: Most OS kernels enable extensibility by exporting a standard generic link layer device interface. This allows us to make our link-layer driver a non-intrusive loadable kernel module.

2.2 Architecture

Control Path: MMNet tracks VM membership to identify when a VM is created or destroyed. MMNet connection establishment happens in a totally asynchronous fashion allowing for simultaneous connection establishments. When a VM is created, new MMNet connection channels are established between the new VM and other co-located VMs and new IP routing table entries are added to route traffic to the MMNet ethernet interface. Conversely, VM destruction triggers removal of the routing table entries and tearing

down of the MMNet connection channels.

Data Path: Since MMNet exports a standard ethernet device interface, it needs to operate on OS-specific packet data structures in the data path—for example, `sk_buff` structures in Linux. MMNet provides interoperability between different guest OSes by appropriate translations between OS-specific data structures and an OS-independent shared memory communication format. These translations become non-trivial and complicated because they need to be performed in a zero-copy fashion coupled with OS idiosyncrasies.

3 Evaluation

We have implemented a prototype of MMNet as a dynamically loadable kernel module for the paravirtualized XenLinux guest OS. We have evaluated MMNet based on three metrics: *throughput*, *latency* and *CPU consumption*. We compare MMNet to three configurations: netfront, xenloop and loopback (with 1 and 2 vcpus). All experiments were performed on a machine with a dual socket, dual core AMD Opteron processor with 1MB L2 cache and 8GB memory. We use Xen version 3.1.2 and paravirtualized Linux version 2.6.18. Our VMs were configured with 256MB of memory each.

Figures 2 and 3 present TCP and UDP throughput results for varying message sizes. As the graphs show, MMNet performs better than xenloop and netfront, and comparably with loopback. Figure 4 presents the TCP latency results for varying request sizes. MMNet is two times better than Netfront, but worse than loopback. Unlike xenloop, MMNet latency does not deteriorate with request size. Figure 5 presents the normalized CPU utilization for MMNet. MMNet is better than netfront and xenloop, but much worse than loopback. Further analysis of this is part of future work.

4 Related Work

XenSocket provides a communication mechanism that exports a new socket-like interface. Unlike MMNet, this approach is not transparent. XenLoop [1] achieves inter-VM communication by snooping on every packets and short-circuiting those destined to co-located VMs. This approach is transparent, as well as non-intrusive, but is not performant enough as shown by our evaluation. XWay hooks in at the transport layer in the place of TCP. The transparency of the mechanism is restricted to applications that use TCP. Both IVC and VMware VMCI provide library level solutions that are not generally transparent to the entire system.

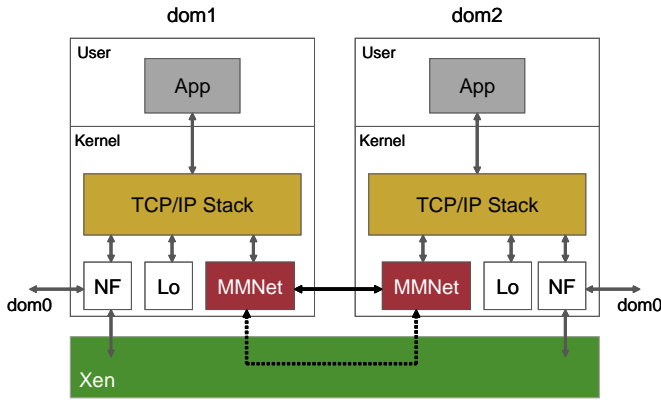


Figure 1: This figure shows where MMNet operates in the guest OS. *dom1* and *dom2* are co-located VMs connected by MMNet. *Lo* represents loopback device and *NF* represents Xen netfront device[3]

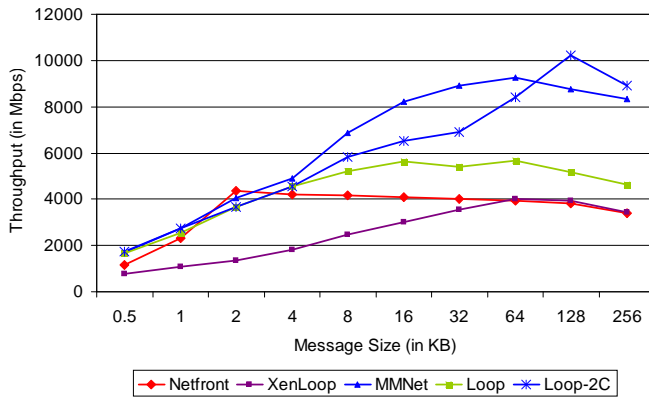


Figure 2: TCP Throughput (TCP_STREAM)

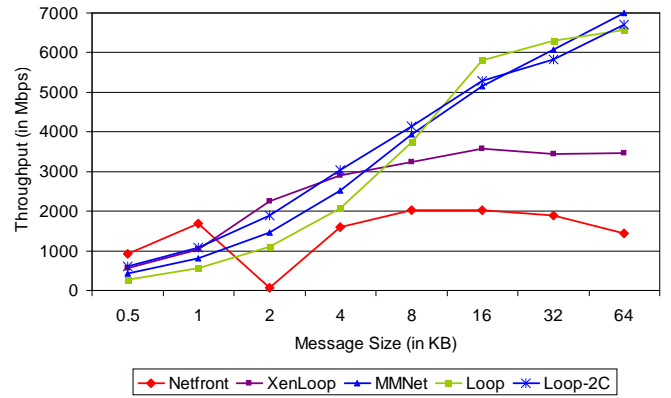


Figure 3: UDP Throughput (TCP_STREAM)

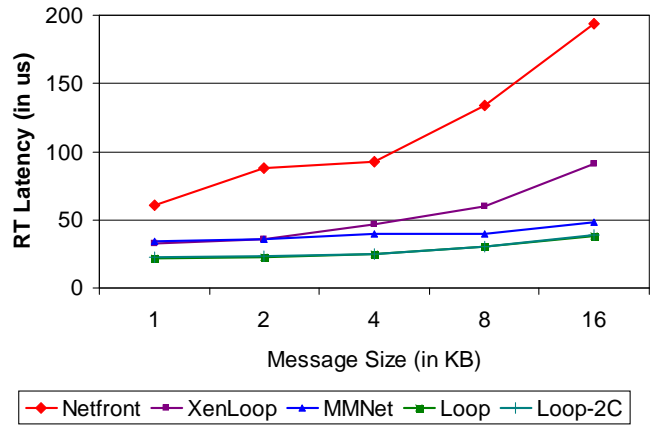


Figure 4: TCP Latency (TCP_RR)

References

- [1] Xenloop: A transparent high performance Inter-VM network loopback. In *Proc. of International Symposium on High Performance Distributed Computing (HPDC)*, June 2008.
- [2] ORACLE Corporation. ORACLE VM. <http://www.oracle.com/technologies/virtualization/index.html>.
- [3] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the Xen virtual machine monitor. In *OASIS*, Oct 2004.

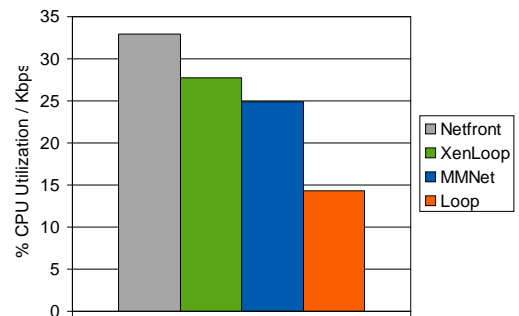


Figure 5: Normalized CPU Consumption (16KB)