V4VSockets: low-overhead intra-node communication in Xen.

Anastassios Nanos, Stefanos Gerangelos, Ioanna Alifieraki, Nectarios Koziris

High-Performance Systems and Interconnects (HPSI),
Computing Systems Lab
National Technical University of Athens
Github: `http://github.com/HPSI`
WWW: `http://cslab.ece.ntua.gr/research/`

Apr. 21st, 2015

Overview

1. Communication in Virtualized Environments
   - Basic Concepts
2. I/O access in Virtualized Environments
   - Xen
   - Intra-node communication in Xen
3. V4VSockets
   - Architecture
   - Experimental Evaluation

Introduction

Cloud computing

- application oriented
- fast, flexible, and easy-to-use infrastructure

Consolidation

- $\frac{vCPU}{physicalcores} >> 1$
- multi/many–cores

The number of co-located VMs is drastically increasing

Introduction

Applications

- stand-alone (flexibility)
- distributed (elasticity)
- relatively recent trend: network flows (SDN/NFV)
  - lightweight VMs performing a specific network function
  - network packets go through these VMs forming a flow.

Introduction

Applications

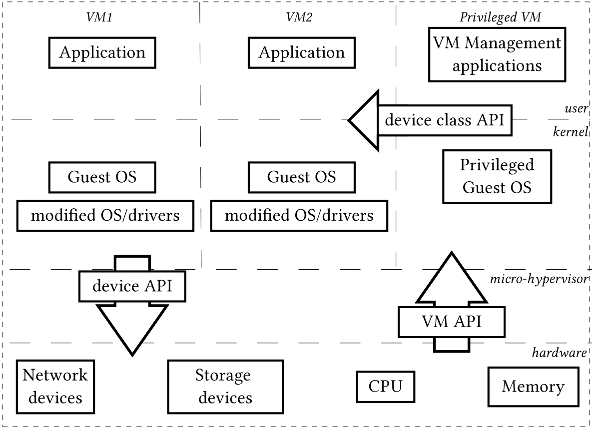
- stand-alone (flexibility)
- distributed (elasticity)
- relatively recent trend: network flows (SDN/NFV)
  - lightweight VMs performing a specific network function
  - network packets go through these VMs forming a flow.

Design and implement V4VSockets:

- efficient message exchange (almost one order of magnitude better than generic approaches)
- isolation
- API compatible (Sockets)

Xen - Architecture

- hypervisor & privileged VM (driver domain) to access hardware



I/O Internals – Xen

Xen basics

- hypervisor – driver domain runs as a Linux guest
- split driver model (frontend/backend)

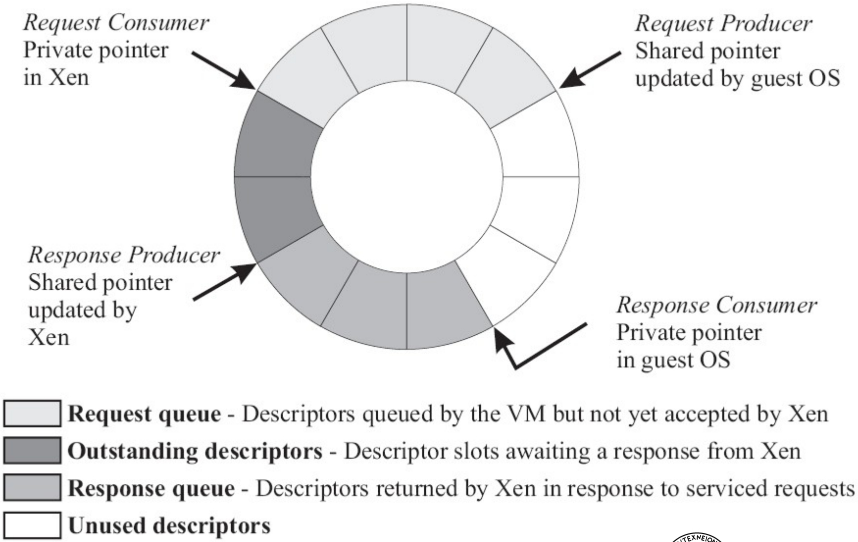
Xen – Event channels

- notify Guest/Host about a pending transaction
- easy to setup – bind to a specific ”port”

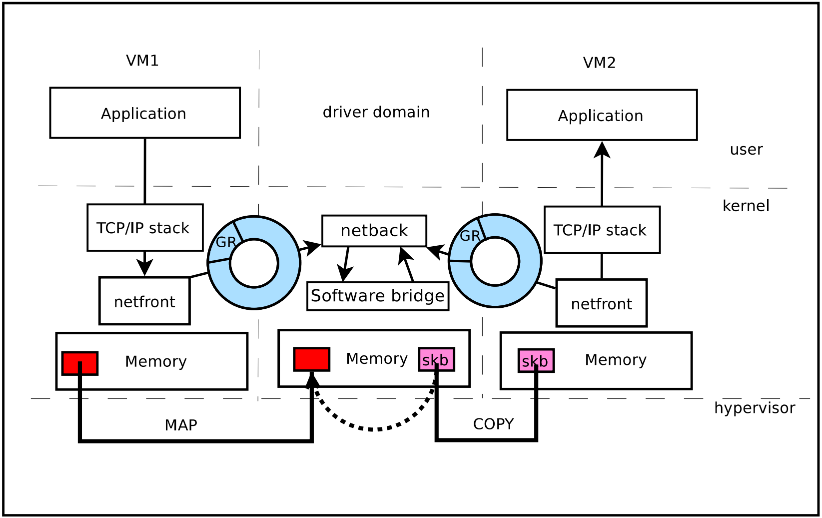
Xen – Grant mechanism

- issue a page grant request
- the other end maps the grant (accept)
- this page is shared across the two domains

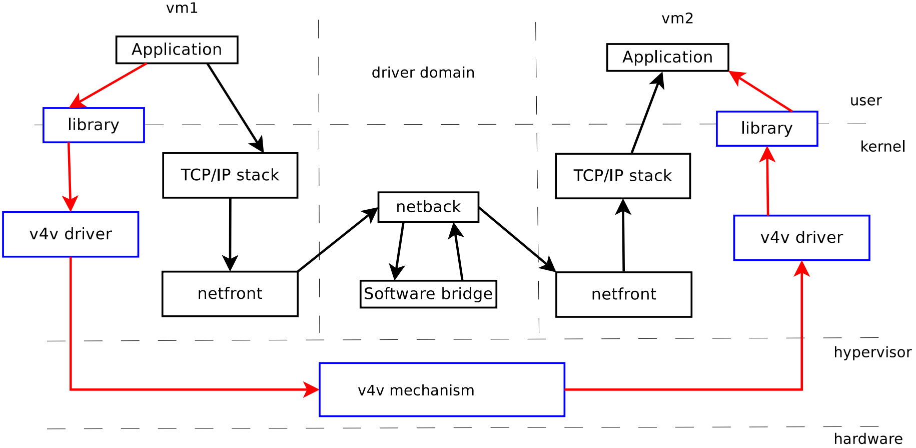
I/O internals – Xen Ring buffers



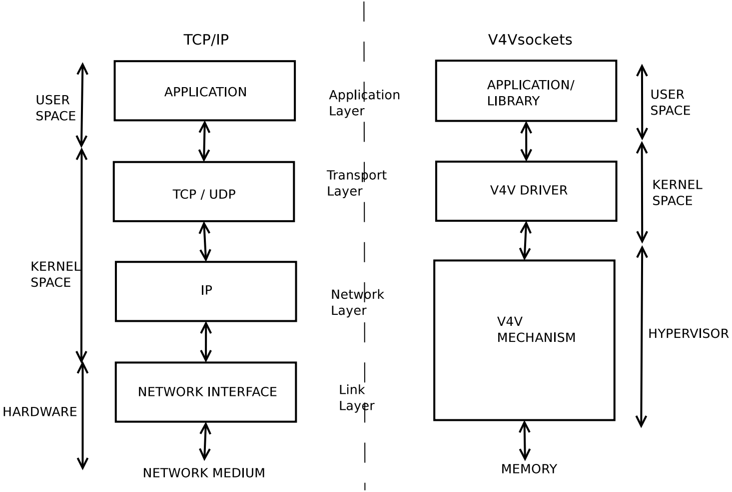
I/O internals – intra-node communication in Xen



V4Vsockets vs Netfront/Netback



V4VSockets Protocol Stack



V4VSockets Internals

Communication mechanisms in V4VSockets

- system calls
- hypercalls
- event channels - VIRQS

V4V Rings

- circular buffers
- allocated in the VM address space by the VM kernel
- data exchange medium

V4VSockets Internals

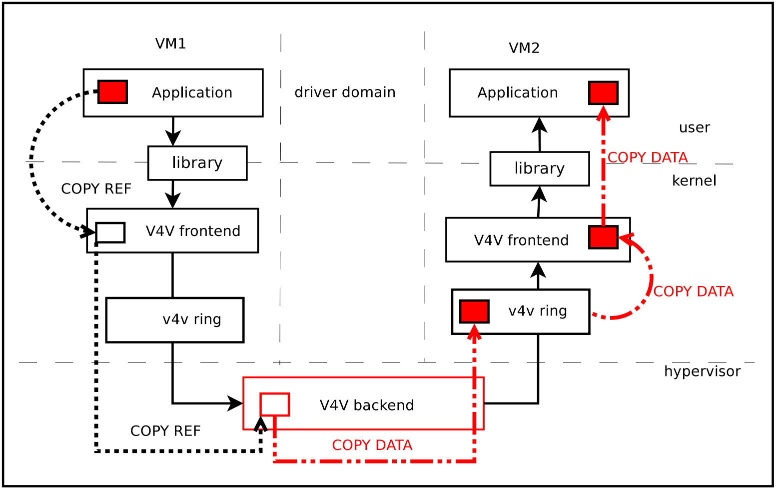
Application/Library layer – user–space

- forwards the relevant actions and arguments to the transport layer.
- kernel-level socket implementation for a new address family (`AF_V4VSOCK`)

Example:

- `socket()` → `v4vsockets_create()`
- `bind(sockaddr)` → `v4vsockets_ring_create(dom_id, port)`
- `sendmsg(msghdr)` → `v4vsockets_sendmsg(msghdr)`

V4VSockets Internals



V4VSockets Internals

Transport layer – V4V frontend driver – VM kernel

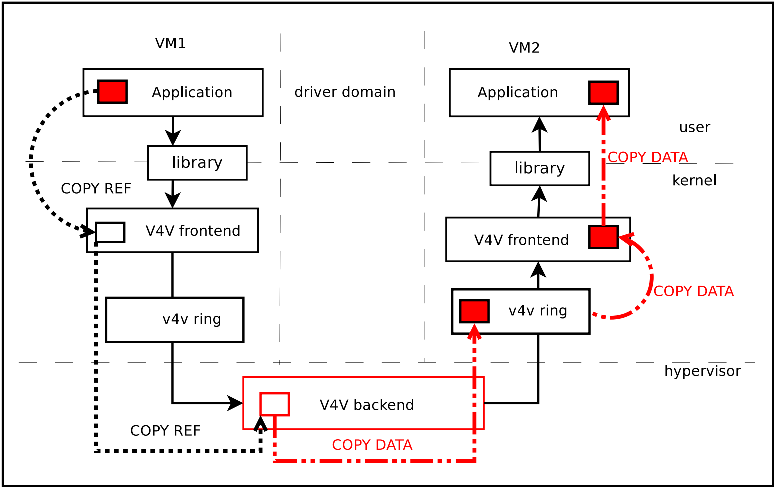
- handles the virtual connection semantics between peer VMs that need to communicate,
- is in charge of fragmenting and sending upper-layer packets by issuing hypercalls to the hypervisor (network layer), and
- provides a notification mechanism to the VM's user-space for receiving packets, as well as error control.

Example:

- `v4vsockets_sendmsg(msghdr)` $\rightarrow$

```
dom_id = v4vsockets_resolve(sockaddr);
while(nr_iovecs)
v4v_send(dom_id, iovec)
```

V4VSockets Internals



V4VSockets Internals

Network/Link layer – hypervisor

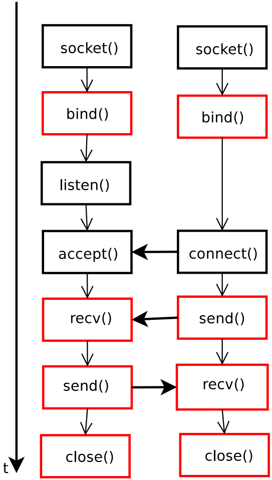
- encapsulation of upper-layer messages to packets that will be transmitted to their destination, according to V4V semantics,
- packet delivery.

Example:

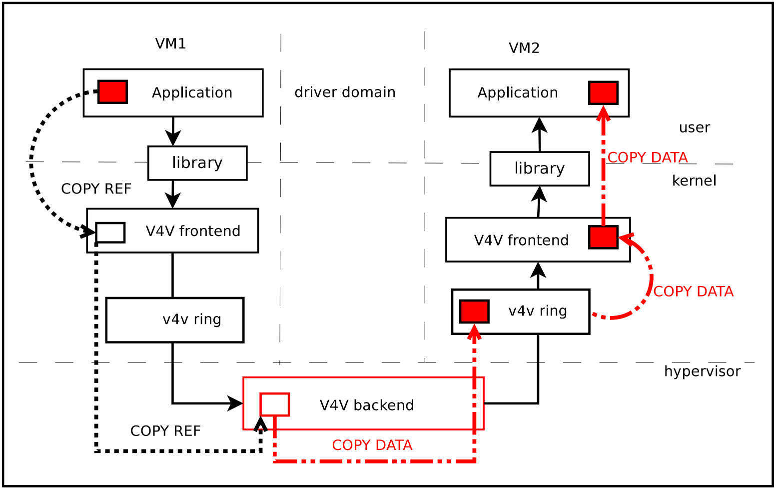
- `v4v_send(dom_id, iovec)` →

```
dom_ring = v4v_resolve(dom_id);
memcpy(dom_ring, iovec_ptr, iovec_len)
```

V4VSockets – Message Exchange



V4VSockets – Message Exchange



Experimental Evaluation

Testbed

- 2x {Intel Xeon @2.4Ghz}, Intel 5520, 48GB memory
- Xen 4.5-unstable, Debian GNU/Linux (Linux kernel 3.14.2)
- generic micro-benchmark: `pingpong`

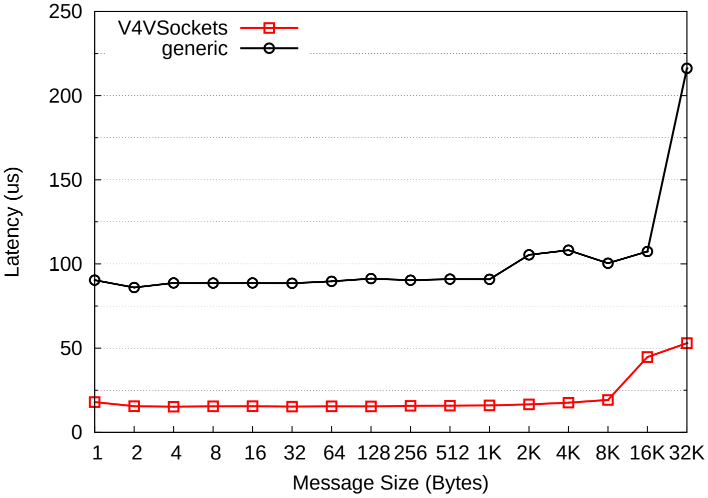
Cases:

- TCP sockets
- V4V Stream

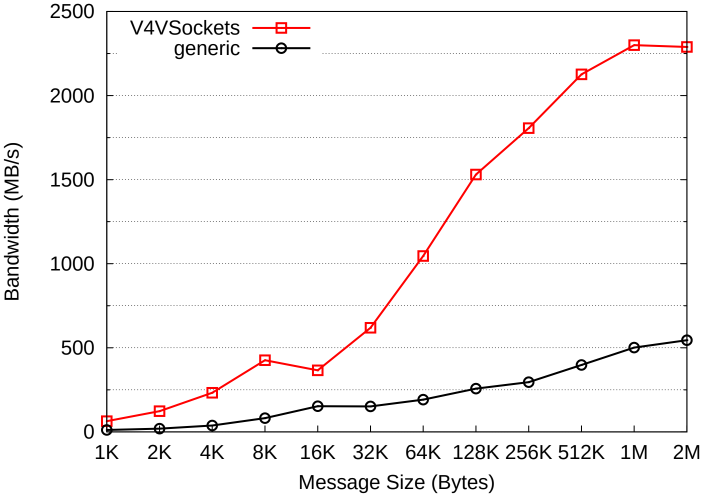
Experiment setup:

- 2 VMs exchanging messages (latency, throughput)
- up to 16 VMs exchanging messages in pairs (latency, throughput)

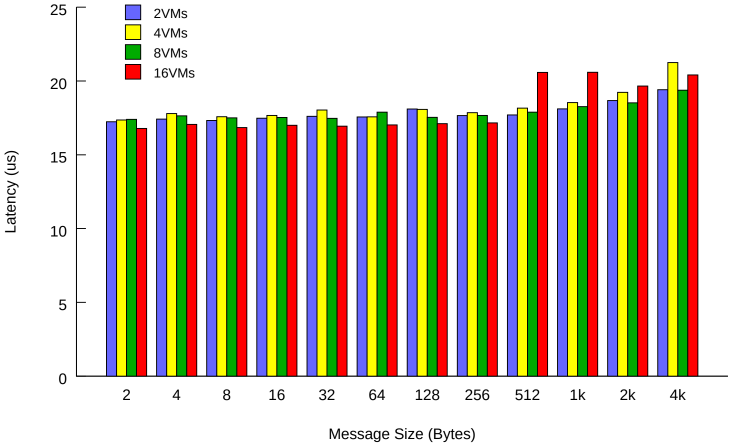
Experimental evaluation – latency



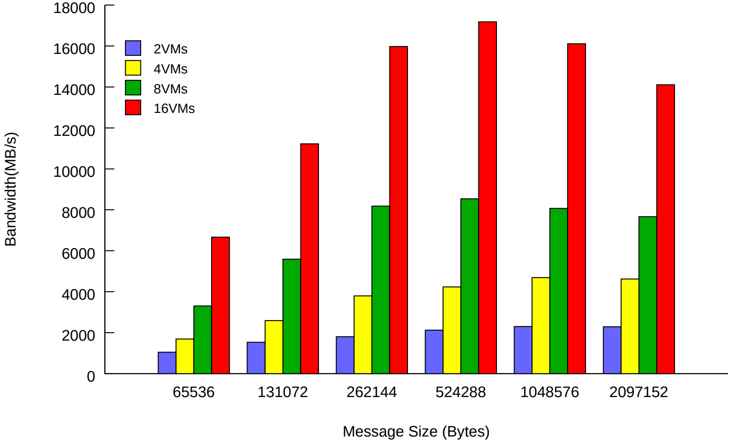
Experimental evaluation – throughput



Experimental evaluation – latency scaling



Experimental evaluation – throughput scaling



Experimental evaluation – GPU stencil

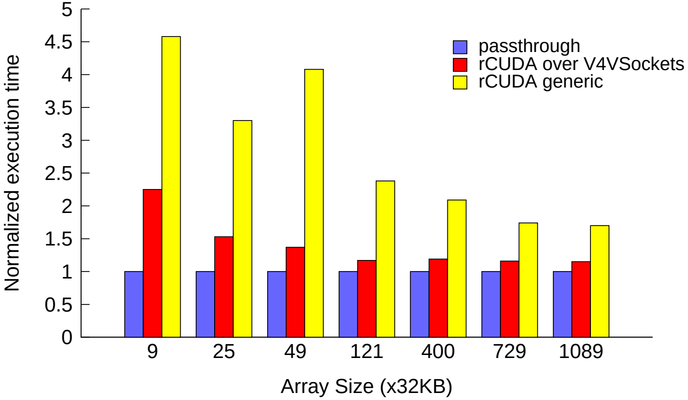
Remote CUDA execution framework (rCUDA)

- A. J. Peña, C. Reaño, F. Silla, R. Mayo, E. S. Quintana-Ortí, and J. Duato. *A complete and efficient cuda-sharing solution for HPC clusters. Parallel Computing*, 40(10):574–588, 2014.
- execute remote CUDA calls through TCP sockets
- direct assignment (PCI passthrough)
- remote calls via TCP sockets and V4VSockets

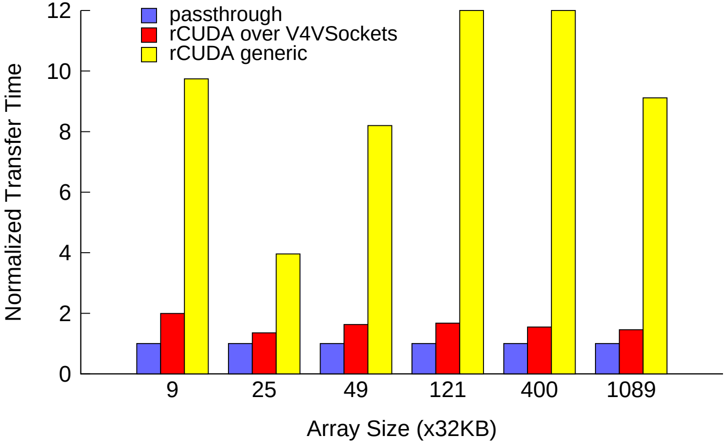
GPU stencil: matrix-matrix product benchmark

- plot total time of execution
- plot transfer time of first input matrix

Experimental evaluation – GPU stencil



Experimental evaluation – GPU stencil



Summary

Intra-node communication in VM environments

- split driver model – generic
- V4VSockets: framework for low-overhead intra-node communication
  - is not based on a driver domain
  - does not use shared memory between guests (map/grant mechanism)
  - uses memory copies, hypercalls and event channels

- better throughput (efficient data path, bypass the complex TCP/IP stack)
- hypercall overheads (small, negligible if correctly finetuned)
- scalability (no privileged guest involved in communication)
- isolation (no shared memory)
- event driven

Future endaevors

- CPU utilization overheads
- NUMA and multihierarchical memory architectures
- map instead of copy (study the systems behavior of providing a shared memory space between VMs)
- GPU sharing evaluation

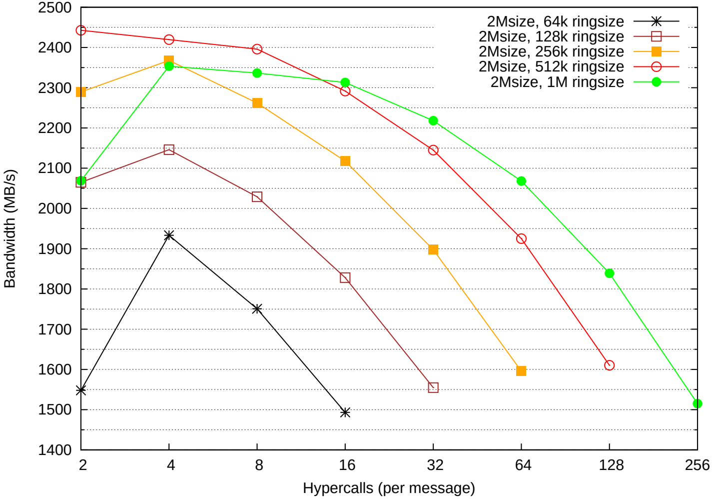
Available online as open-source
`https://github.com/HPSI/V4VSockets`

Thanks!

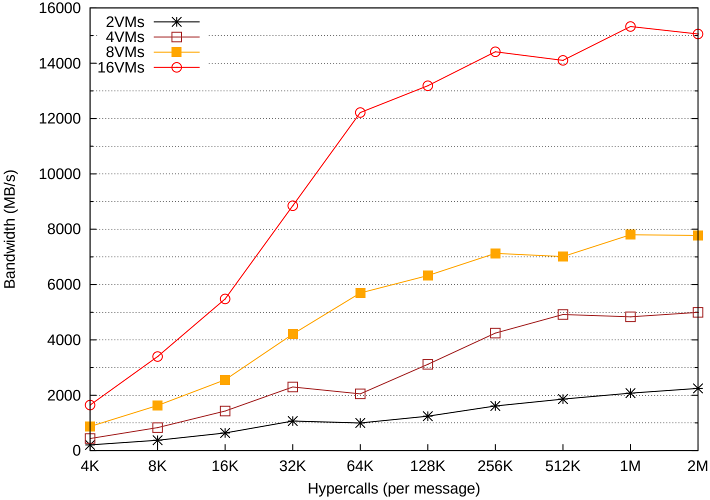
Questions?

Backup

Experimental evaluation – 2M message size vs Hypercalls



Experimental evaluation – Datagram scalability



Experimental evaluation – Hypercalls per message