V4V.Sockets: low-overhead intra-node communication in Xen.

Anastassios Nanos, Stefanos Gerangelos, Ioanna Aliferaki, 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

- \( \frac{vCPU}{physical\ cores} \gg 1 \)
- 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.

Design and implement V4VSockets:
- efficient message exchange (almost one order of magnitude better than generic approaches)
- isolation
- API compatible (Sockets)
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
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

Request Consumer
Private pointer in Xen

Request Producer
Shared pointer updated by guest OS

Response Producer
Shared pointer updated by Xen

Response Consumer
Private pointer in guest OS

- Request queue - Descriptors queued by the VM but not yet accepted by Xen
- Outstanding descriptors - Descriptor slots awaiting a response from Xen
- Response queue - Descriptors returned by Xen in response to serviced requests
- Unused descriptors
I/O internals – intra-node communication in Xen
V4V sockets vs Netfront/Netback

```
vm1
  Application
    library
      v4v driver

TCP/IP stack
  netfront
    Software bridge
      netback
        v4v mechanism

vm2
  Application
    library
      v4v driver

TCP/IP stack
  netfront

driver domain
```

CloudDP '15, Bordeaux
A. Nanos @ CSLab, NTUA
V4VSockets Protocol Stack

TCP/IP

APPLICATION

TCP / UDP

IP

NETWORK INTERFACE

NETWORK MEDIUM

V4Vsockets

APPLICATION/LIBRARY

V4V DRIVER

V4V MECHANISM

USER SPACE

KERNEL SPACE

HARDWARE
V4V Sockets Internals

Communication mechanisms in V4V Sockets

- system calls
- hypercalls
- event channels - VIRQS

V4V Rings

- circular buffers
- allocated in the VM address space by the VM kernel
- data exchange medium
**V4V Sockets 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)`
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)` →
  `dom_id = v4vsockets_resolve(sockaddr);
  while(nr_iovecs)
    v4v_send(dom_id, iovec)`
V4V Sockets Internals

VM1
- Application
  - library
  - V4V frontend
  - v4v ring
  - COPY REF

VM2
- Application
  - library
  - V4V frontend
  - v4v ring
  - COPY DATA

COPY DATA

driver domain

user

kernel

hypervisor

V4V backend

COPY REF

COPY DATA
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:

- \texttt{v4v\_send(dom\_id, iovec)} →
  
  \texttt{dom\_ring = v4v\_resolve(dom\_id)};
  
  \texttt{memcpy(dom\_ring, iovec\_ptr, iovec\_len)}
V4V Sockets – Message Exchange

VM1

Application

library

V4V frontend

v4v ring

COPY REF

driver domain

VM2

Application

library

V4V frontend

v4v ring

COPY DATA

user

copy data

kernel

COPY DATA

hypervisor

V4V backend

COPY REF

COPY DATA
## 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

![Graph showing latency vs message size for V4VSockets and generic protocols. The graph illustrates a comparison between the two protocols, with V4VSockets showing consistently lower latency across various message sizes.]
Experimental evaluation – throughput

![Graph showing bandwidth and message size relationship for V4V Sockets and generic](image)

- **Bandwidth (MB/s)**
- **Message Size (Bytes)**

- V4V Sockets
- Generic

---

Apr. 21st, 2015
CloudDP ’15, Bordeaux
A. Nanos @ CSLab, NTUA
Experimental evaluation – latency scaling

Latency (us) vs. Message Size (Bytes) for 2VMs, 4VMs, 8VMs, and 16VMs.
Experimental evaluation – throughput scaling
Experimental evaluation – GPU stencil

Remote CUDA execution framework (rCUDA)

- execute remote CUDA calls through TCP sockets
- direct assignment (PCI passthrough)
- remote calls via TCP sockets and V4VSocket

GPU stencil: matrix-matrix product benchmark

- plot total time of execution
- plot transfer time of first input matrix
Experimental evaluation – GPU stencil

![Graph depicting normalized execution time against array size for different GPU stencil implementations. The x-axis represents the array size in x32KB, and the y-axis represents the normalized execution time. The graph compares three implementations: passthrough, rCUDA over V4VSockets, and rCUDA generic. The bars show the execution time for array sizes 9, 25, 49, 121, 400, 729, and 1089, respectively. The graph indicates that rCUDA over V4VSockets generally has the lowest execution time, followed by passthrough, with rCUDA generic being the highest.]
Experimental evaluation – GPU stencil

![Normalized Transfer Time](array_size_x32KB)

- passthrough
- rCUDA over V4VSockets
- rCUDA generic

Array Size (x32KB):
- 9
- 25
- 49
- 121
- 400
- 1089

Apr. 21st, 2015 CloudDP ’15, Bordeaux A. Nanos @ CSLab, NTUA 26
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 endeavors

- CPU utilization overheads
- NUMA and multihierarchicaal 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/V4VSSockets
Thanks!

Questions?
Backup
Experimental evaluation – 2M message size vs Hypercalls

![Graph showing bandwidth (MB/s) vs hypercalls (per message) for different message sizes and ring sizes.]

- 2Msize, 64k ringsize
- 2Msize, 128k ringsize
- 2Msize, 256k ringsize
- 2Msize, 512k ringsize
- 2Msize, 1M ringsize

Apr. 21st, 2015 CloudDP ’15, Bordeaux A. Nanos @ CSLab, NTUA
Experimental evaluation – Datagram scalability

Bandwidth (MB/s) vs. Hypercalls (per message) for different numbers of VMs (2VMs, 4VMs, 8VMs, 16VMs). The graph shows an increase in bandwidth with an increase in the number of hypercalls and the number of VMs.
Experimental evaluation – Hypercalls per message

![Hypercalls vs Bandwidth Graph]

- 64ksize, 64k ringsize
- 128ksize, 128k ringsize
- 256ksize, 256k ringsize
- 512ksize, 512k ringsize
- 1Msize, 1M ringsize
- 2Msize, 2M ringsize

Bandwidth (MB/s)

Hypercalls (per message)