Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.

Intel may make changes to specifications and product descriptions at any time, without notice.

All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2015 Intel Corporation.
Agenda

• KVM Enhancements for NFV at OPNFV
• Deterministic Execution and Minimal Latency
• Inter-VM communication: vhost-user-shmem
Architecture Framework

Virtual Network Function

NFV Infrastructure

KVM, Containers, ...

Management and Orchestration

Service, VNF and Infrastructure Description
KVM is Crucial to OPNFV

Upstream Projects:

Linux
KVM
openstack
DAYLIGHT
Project: NFV Hypervisors-KVM

1. Minimal Interrupt latency variation for data plane VNFs (Virtual Network Function)
2. Inter-VM Communication
3. Fast Live Migration

Developers from:

https://wiki.opnfv.org/nfv-kvm
Deterministic Execution and Minimal Latency
Causes of Latency Variation

Asynchronous Events
- Interrupts, VM Exits, Cache/TLB Misses

Software
- Spin Locks, Loops, Scheduling, Exit to user-level

Hardware/Firmware
- SMI, Power Management, NIC
Solutions

**Excusive/Static Allocation**
- Soft “Partitioning”, CPU Binding, Huge Pages

**Software**
- PREEMPT-RT Linux, Code inspection, testing/measurements

**Hardware Technologies**
- Cache Allocation Technology, Advanced VT features

**Hardware**
- User-Level
- User-Level
- Intr_Handler
- Linux Kernel
- PREEMPT-RT Configuration
Cache Allocation Technology

- Last Level Cache partitioning mechanism enabling the separation of an application
- VMs can be isolated to increase determinism
- Having limited cache is still better than “unlimited cache and noisy neighbors”

CAT is supported on the following 6 SKUs for Intel Xeon processor E5 v3 family: E5-2658 v3, E5-2658A v3, E5-2648L v3, E5-2628L v3, E5-2618L v3, and E5-2608L v3 and Intel(R) Xeon(R) processor D family.
Latency Data 1: Cyclic Test

Cyclic Test in Guest: Latency (in µS)
- Min: 7
- Avg: 9
- Max: 16

Latency Occurrences

000007 000003
000008 1175756
000009 6781265
000010 159222
000011 069100
000012 011004
000013 000379
000014 000207
000015 00049
000016 00005

99.69% (Total #: 79,810,183)

Histogram

RT Linux Guest on KVM

Host: Linux with RT patches
Latency Data 2: Latency from Periodic External Interrupts

Latency from periodic external interrupt:
• Time delta from interrupt occurrence to invocation of interrupt handler in guest (unit: in µS)
  Min: 3.98
  Avg: 4.42
  Max: 9.10

Expecting even better results with:
• Posted Interrupts and
• CAT (Cache Allocation Technology)
Inter-VM Communication
Communication Models

Specific API
- Shared memory, etc.
- Directly used by particular Processes

Networking API
- Use vSwitch in hypervisor
- Generic

Linux Kernel
- vSwitch
- Network stack

KVM Modules
- Process
- Network stack
- TX, RX queues

Hardware
- NIC

Intel OpenSource Technology Center
Fast Paths: Inter-VM Communication

Specific API
- Need to improve security when using shared memory

Networking API
- Access Destination VM memory
- Use In-VM Switch

Shared Memory (1GB pages)
- In-kernel vSwitch
- TAP
- VM1
- VM2
- VM3

KVM Modules
- Linux Kernel
- Network stack
- Process
- NIC
Implementing Inter-VM Communication: vhost-user-shmem
Goals

- Add fast-paths in VMs as optimized inter-VM communication
- Maintain consistent flow table entries in VMs
- Enable protected access to the destination VM or shared memory
  - Open the Window when needed
  - Close it immediately when done

- Add fast-paths in VMs as optimized inter-VM communication
- Maintain consistent flow table entries in VMs
- Enable protected access to the destination VM or shared memory
  - Open the Window when needed
  - Close it immediately when done
Clean Design Objectives

Extend vhost-user as *transport mechanism* over shared memory/virtqueues:
- Deliver packets to another guest’s virtio device/virtqueue directly
- Provide memory mapping (GPAs), protected access, destination addressing

Build innovative **high-performance networking applications**, e.g:
1. In-VM switch as a fast cached-datapath for the full-blown virtual switch
2. Lightweight and fast Service Function Chaining
3. Next big NFV app you are developing
Shared Memory Using vhost-user Server

vhost-user server (backend) has sufficient info and capability to host shared memory:

- Gather mem info to access virtuques from vhost-user clients (QEMUs)
- It can allocate its own memory for sharing purposes
- E.g. large pages shared by guests (like ivshmem)
Extending it for Inter-VM Communication

- **vhost-user server (backend) becomes a client**
  - Send mem info to QEMUs
  - QEMU extends memory regions
- Allows **vhost-user clients** to access their virtqueues each other
- Provides **vhost-user clients** with shared memory
Simple Example: VM1 and VM2

1. vhost-user from VM2
2. vhost-user for VM2 (multicast)

VM1
- vushmem PCI Detection
- fastpath code
- vhost-user-shmem support
- vhost-user-shmem protocol
- vhost-user socket
- PCI BAR

VM2
- vushmem PCI Detection
- fastpath code
- vhost-user-shmem support
- vhost-user-shmem protocol
- PCI BAR

QEMU

vSwitch
- vhost-user-shmem (vushmem) server
- vushmem control structure

Interrupt (MSI)

Shared

Socket
Adding Protected Access

- **Extends memory** to access fast-path channel or destination VM
- **VMFUNC** instruction in VM w/o VM exit
  - #0 (EAX): Switches EPT (Extend Page Table) Pointers
  - Alternate EPT has additional translation

Diagram:

- VM1, VM2, VM3
- TX(), RX()
- Mapped by Default EPT (Extend Page Table)
- Guest Memory
- Linux Kernel
- KVM Modules
- core
- NIC
Adding EPT Alternate View

Fast Pass Code (Protected Code):
- Upon VMFUNC #0, EPT View is changed
- Access other shared memory and virtqueues of other VMs in protected fashion

KVM ioctl options for QEMU to extend Guest Memory:
1. W/O protection, or
2. W/ protection
   - Extend only in alternate EPT view
Implementing Protected Access

Trampoline Page
- Registered by a trusted entity
- Entry/Exit to/from Trusted Code

Protected Code
- Specified at registration time

EPT Permission
Full (X, W, R)
Write-Protected (X, -, R)
No Access (-, -, -)

start_xmit(*skb, *dev) {
    ...
    send(packets);
}

send(*packet) {
    ...
    VMFUNC #0, EPTP;
    Tx(packets);
    VMFUNC #0, 0
}

Tx(*packet) {
    move_data();
    notifify();
}

Additional pages
Performance Estimate from PoC

Measure cost of VMFUNC and Trampoline Code:
• Transfer 64B packets from virtio-net to another VM (fast path)

65Mpps with 32-packet batching*:
• Same batching size as DPDK

*Intel internal estimation
Summary

1. Minimal Interrupt latency variation for data plane VNFs (Virtual Network Function)
   • On Track
2. Inter-VM Communication
   • Preliminary performance data from PoC with trampoline code
   • Implementation proposal (vhost-user-shmem) based on vhost-user
3. Fast Live Migration
   • Next presentation

Join OPNFV Projects!