Scalable High-performance Userland Container Networking for NFV

Jianfeng Tan, Cunming Liang, Huawei Xie, Zhihong Wang, Yuanhan Liu, Heqing Zhu
Agenda

• Background
• Userland container networking
• Impact on application development
NFV and challenges

- Virtual Network function
- Virtual machine, Container
VM vs Container based VNFs

• Challenges of VM-based VNFs
  – Provisioning time
  – Runtime performance overhead

• Challenges of container-based VNFs
  – High performance networking
    • For VNFs, like LB, FW, IDS/IPS, DPI, VPN, pktgen, Proxy, AppFilter, etc
  – Security
Challenges of high perf. network

- Forward 1~2 Mpps per core

<table>
<thead>
<tr>
<th>NIC</th>
<th>Time budget for 64B</th>
<th>Time budget for 1518B</th>
</tr>
</thead>
<tbody>
<tr>
<td>10Gb</td>
<td>67.2 ns</td>
<td>1,230 ns</td>
</tr>
<tr>
<td>40Gb</td>
<td>N/A</td>
<td>307 ns</td>
</tr>
<tr>
<td>100Gb</td>
<td>N/A</td>
<td>120 ns</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NIC</th>
<th>Time cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>System call</td>
<td>75 ns/42 ns</td>
</tr>
<tr>
<td>Atomic ops</td>
<td>8.25 ns</td>
</tr>
<tr>
<td>Spinlock lock/unlock</td>
<td>16+ ns</td>
</tr>
<tr>
<td>L3 miss*</td>
<td>~53 ns</td>
</tr>
</tbody>
</table>

* Tested on Intel i7-3770 (Ivy Bridge) by www.7-cpu.com
Other data from [LWN article](http://lwn.net)
Container networking status quo

- Multi-host networking
Userland Container Networking
SR-IOV in Virtualization Technologies

Virtual Appliance

VM kernel

Host Kernel

PF
VF
VF
VF

Virtual Ethernet Bridge & Classifier

VTEP
SR-IOV in Kernel

Host Kernel

PF Driver

Virtual Ethernet Bridge & Classifier

VF Driver

Netns

VTEP
SR-IOV in Userland

Host Kernel

PF Driver

Container

DFPDK

Container

DFPDK

Userland
NIC driver

Virtual Ethernet Bridge & Classifier

VTEP

PF

VF

VF
SR-IOV in Userland

• Pros:
  – Line rate even with small packets
  – Low latency
  – HW-based QoS

• Cons:
  – # of VFs is limited (64 or 128)
  – Not flexible (in need of router or switch with support of VTEP)
Setup: SR-IOV in Userland

- **Prepare VFs**
  - `$ echo 1 > /sys/bus/pci/devices/0000:81:00.0/sriov_numvfs`
  - `$ ./tools/dpdk_nic_bind.py --status`
  - `0000:81:00.0 '82599ES 10-Gigabit SFI/SFP+ Network Connection' if=eth1
drv=ixgbe unused=
 0000:81:10.0 '82599 Ethernet Controller Virtual Function' if=eth5
drv=ixgbevr unused= ...`

- **Bind to vfio driver**
  - `$ modprobe vfio-pci`
  - `$ ./tools/dpdk_nic_bind.py -b vfio-pci 0000:81:10.0`

- **Prepare hugetlbfs**
  - `$ mount -t hugetlbfs -o pagesize=2M,size=1024M none /mnt/huge_c0/`

- **Start container**
  - `$ docker run ... -v /dev/vfio/vfio0:/dev/vfio/vfio0 -v /mnt/huge_c0:/dev/hugepages/ ...`
Container Networking Scenarios

- VF pass-through
- Aggregation
- Host Kernel
- vSwitch
- Linux bridge, OVS bridge
Aggregation Scenario

VF pass-through

Host Kernel

VF Driver

VF

VF pass-through

Aggregation

vSwitch

IPC

DATA PLANE DEVELOPMENT KIT
VIRTIO for IPC
Why virtio for IPC?

• Performance
  – Bypass kernel (shm-based)
  – Smarter notification
  – Cache friendly

• Make best use of what already we have in DPDK
VIRTIO as Unified Interface
VIRTIO in VM

VM

DPDK

VIRTIO PMD

or

Kernel

VIRTIO NET

virtio

vhost

vSwitch

Transport: PCI bus

MMIO or PMIO

Interrupt injection

Hypervisor

Device Emulation

vhost-user adapter

Socket
/tmp/vx.socket

VIRTIO PMD

VIRTIO NET

Hypervisor

Device Emulation

vhost-user adapter
VIRTIO in Container

Container/App

- DPDK
  - ETHDEV
  - VIRTIO PMD
- VIRTIO
  - VIRTIO-USER
  - vhost-user adapter

PCI device

Virtual device

vSwitch

Socket
/tmp/xx.socket
Address translation

VM

Holes

Container

Non-DPDK memory

GPA: Guest Physical Address
BVA: Backend Virtual Address

<table>
<thead>
<tr>
<th>GPA</th>
<th>BVA</th>
<th>LEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPA 0</td>
<td>BVA 0</td>
<td>Len 0</td>
</tr>
<tr>
<td>GPA 1</td>
<td>BVA 1</td>
<td>Len 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>GPA n</td>
<td>BVA n</td>
<td>Len n</td>
</tr>
</tbody>
</table>

FVA: Frontend Virtual Address
BVA: Backend Virtual Address

<table>
<thead>
<tr>
<th>FVA</th>
<th>BVA</th>
<th>LEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>FVA 0</td>
<td>BVA 0</td>
<td>Len 0</td>
</tr>
<tr>
<td>FVA 1</td>
<td>BVA 1</td>
<td>Len 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>FVA n</td>
<td>BVA n</td>
<td>Len n</td>
</tr>
</tbody>
</table>
Setup: VIRTIO in container

- Add a bridge and a vhost-user port in ovs-dpdk
  
  $ ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
  $ ovs-vsctl add-port br0 vhost-user-1 -- set Interface vhost-user-1 type=dpdkvhostuser

- Prepare hugetlbfs
  
  $ mount -t hugetlbfs -o pagesize=2M,size=1024M none /mnt/huge_c0/

- Run container
  
  $ docker run ... \
  -v /usr/local/var/run/openvswitch/vhost-user-1:/var/run/usvhost \n  -v /mnt/huge_c0:/dev/hugepages/ \n  ... 
  -c 0x4 -n 4 --no-pci --vdev=virtio-user0,path=/var/run/usvhost \n  ...
Performance Evaluation - throughput

Case I: Native Linux

Case II: SR-IOV

Case III: VIRTIO

10Gb Line rate

Container
- pcap lib

Kernel
- ixgbe kernel

Container
- ixgbe PMD

Container
- virtio PMD

vSwitch
- vhost PMD
- ixgbe PMD
Performance Evaluation - latency

• For native Linux, ms level
• For the other two, us level
  – Polling mode
  – Batching
  – SIMD
Use case

Container
  ovsdb-server
  ovs-vswitchd
  Bridge

Virtual Appliance
  virtio

VM
  Container
  virtio

Container
  virtio

App
  virtio

VF
Is everything OK to run DPDK in container?
DPDK efforts towards container

- Hugetlb initialization process
  - sysfs is not containerized, and DPDK allocates all free pages
    - Addressed by [here](#), avoid to use \texttt{-m} or \texttt{--socket-mem}

- Cores initialization
  - When/how to specify cores for DPDK?
    - Addressed by [here](#), avoid to use \texttt{-c} or \texttt{-l} or \texttt{--lcores}

- Reduce boot time
  - Addressed by [here](#) and [here](#)
Run DPDK in Container securely

• Dedicated hugetlbfs for each container
• Run without --privileged
  – DPDK needs privilege to do VA2PA translation
    • Use VA instead of PA (see [here](#))
• DMA attack
  • Use VA as the IOVA for IOMMU table
Deterministic Environment

- Deterministic CPU env
  - Boot-time: disable timer / task scheduler
    - ... default_hugepagesz=1G isolcpus=16-19 ...
    - Reduce scheduling-clock ticks: adaptive-tick mode
  - Run-time: core-thread affinity
    - cpuset tool: taskset / numactl
    - cgroup.cpuset: cset / docker run ... --cpuset-cpus ...
  - BIOS setting: if necessary, disable Hyper-Threading
Deterministic Environment

• Deterministic cache env
  – Data Direct I/O (DDIO) technology
  – Cache Allocation Technology (CAT)
• An example from here

<table>
<thead>
<tr>
<th>CAT</th>
<th>Throughput (Mpps)</th>
<th>LLC Occupancy (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Present</td>
<td>9.8</td>
<td>4.5</td>
</tr>
<tr>
<td>Present</td>
<td>15</td>
<td>13.75</td>
</tr>
</tbody>
</table>

* DPDK IP Pipeline Application (Packet size = 64 Bytes, Flows = 16 Millions)
Impact on application development
How to Develop Apps?

- Type I: DPI, FW, LB, vSwitch/vRouter, ...
- Type II: Applications in need of TCP/UDP stack

From scratch:
- mTCP
- LwIP
- TLDK

Ported:
- Libuinet
- NUSE (libos)
- Linux Kernel Library
Transform Middleboxes with VNFs

Logic Network 0

Logic Network 1

Middleboxes

Gateway

Userland container networking
Full Stack in Container

• Full stack
  – Hardware resource
  – Driver (network)
  – Under layer network facilities (optional)
  – Dependencies and application itself

• Benefits
  – Better isolation
  – Convenient for live migration

Snort-DPDK
TrEx
Vortex
6WINDGate
Contrail
ScyllaDB
Future work

• Interrupt mode of virtio (to scale)
• Long path to handle VF interrupts in userland (low latency)
• Integrate with popular orchestrators
Summary

• Use DPDK to accelerate container networking
  – Userland SR-IOV
  – Userland virtio-user
• Compared to traditional ways, it provides
  – High throughput
  – Low latency
  – Deterministic networking
Q & A
Backup- How DPDK improves net?

- CPU affinity
- Hugepages
- UIO
- Polling
- Lockless
- Batching
- SSE/AVX
- High-throughput
- Low-latency
- Deterministic