High-performance vNIC framework for hypervisor-based NFV with userspace vSwitch

Yoshihiro Nakajima, Hitoshi Masutani, Hirokazu Takahashi
NTT Labs.
Outline

- Motivation and background
- Issues on current NFV implementation
- vNIC
  - Design & implementation
  - Performance evaluation
Goal

- To provide novel components to enable high-performance NFV with general propose hardware
- To provide high-performance vNIC with operation-friendly features
How much cycles for one packet?

Short packet 64Byte
14.88 MPPS, 67.2 ns
- 2Ghz: 134 clocks
- 3Ghz: 201 clocks

Computer packet 1KByte
1.2MPPS, 835 ns
- 2Ghz: 1670 clocks
- 3Ghz: 2505 clocks
Data Plane Development Kit (DPDK)

- **x86 architecture-optimized data-plane library and NIC drivers**
  - Memory structure-aware queue, buffer management
  - Packet flow classification
  - Polling mode-based NIC driver
- **Low-overhead & high-speed runtime optimized with data-plane processing**
- **Abstraction layer for hetero server environments**
- **BSD-license 😊**
NFV requirements from 30,000 feet

- **High performance network I/O for all packet sizes**
  - Especially in smaller packet size (< 256 bytes)

- **Low-latency and less-jitter**
  - Network I/O & Packet processing

- **Isolation**
  - Performance isolation between NFV VMs
  - Security-related VM-to-VM isolation from untrusted apps

- **Reliability, availability and serviceability (RAS) function for long-term operation**
What’s matter in NFV

- **Still poor performance of NFV apps**
  - Lower network I/O performance
  - Big processing latency and big jitter

- **Limited deployment flexibility**
  - SR-IOV has limitation in performance and configuration
  - Combination of DPDK apps on guest VM and DPDK-enabled vSwitch is configuration

- **Limited operational support**
  - DPDK is good for performance, but has limited dynamic reconfiguration
  - Maintenance features are not realized
Issues on NFV middleware
Performance bottleneck in NFV with HV domain

- HW emulation needs CPU cycle & VM transition
- Pkt recv / send cause VM transition
- Privileged register accesses for vNIC cause VM transition
- VM transition: 800 CPU cycles

System call cause context switch on guest VM
vNIC strategy for performance & RAS

- **Use para-virtualization NIC framework**
  - No full-virtualization (emulation-based)

- **Global-shared memory-based packet exchange**
  - Reduce memory copy

- **User-space-based packet data exchange**
  - No kernel-userspace packet data exchange
Target NFV architecture with hypervisor

- DPDK apps or legacy apps on guest VM
  + userspace DPDK vSwitch
    - Connected by shared memory-based vNIC
    - Reduce OS kernel implementation

Run in userspace to avoid VM transition and context switch
Existing vNIC for u-vSW and guest VM (1/2)

DPDK e1000 PMD with QEMU's e1000 FV and vSwitch connected by tap

Pros: legacy and DPDK support, opposite status detection
Cons: bad performance, many VM transitions, context switch

DPDK virtio-net PV PMD with QEMU virtio-net framework and vSwitch connected by tap

Pros: legacy and DPDK support, opposite status detection
Cons: bad performance, many VM transitions, context switch

DPDK virtio-net PV PMD with vhost-net framework and vSwitch connected by tap

Pros: legacy and DPDK support, opposite status detection
Cons: bad performance, many VM transitions, context switch
Existing vNIC for u-vSW and guest VM (2/2)

DPDK ring by QEMU IVSHMEM extension and vSwitch connected by shared memory

DPDK virtio-net PV PMD with QEMU virtio-net framework and vSwitch with DPDK vhost-user API to connect to virtio-net PMD.

Pros: Best performance
Cons: only DPDK support, static configuration, no RAS

Pros: good performance, both support of legacy and DPDK
Cons: no status tracking of opposite device
High performance vNIC framework for NFV
vNIC requirements for NFV with u-vSW

**High-Performance**
- 10-Gbps network I/O throughput
- No virtualization transition between a guest VM and u-vSW
- Simultaneous support DPDK apps and DPDK u-vSW

**Functionality for operation**
- Isolation between NFV VM and u-vSW
- Flexible service maintenance support
- Link status notification on the both sides

**Virtualization middleware support**
- Support open source hypervisor (KVM)
- DPDK app and legacy app support
- No OS (kernel) modification on a guest VM
vNIC design

vNIC as an extension of virtio-net framework
- Para-virtualization network interface
- Packet communication by global shared memory
- One packet copy to ensure VM-to-VM isolation
- Control msg by inter-process-communication between pseudo devices
vNIC implementation

- **Virtq-PMD driver: 4K LOC modification**
  - Virtio-net device with DPDK extension
  - DPDK API and PV-based NIC (virtio-net) API
  - Global shared memory-based packet transmission on hugeTLB
  - UNIX domain socket based control message
    - Event notification (link-status, finalization)
    - Pooling-based the opposite device check mechanism

- **QEMU: 1K LOC modification**
  - virtio-net-ipc device on shared memory space
  - Shared memory-based device mapping
Virtq-PMD connection sequence

**Initialization phase**
- **Virtq PMD**
  - Initialization request
  - Response
- **Virtio-net device**
  - Initialization request
- **Virtio-net-ipc device**
  - Initialization request

**Configuration phase**
- **mmap()**
  - Shared-memory config request
  - Response
- **Virtq PMD**
  - Device close

**Run loop (interrupted mode or legacy mode)**
- **Virtq PMD**
  - Device close
  - Finalization request (avail ring queue)
  - Response
- **Virtio-net-ipc device**
  - Finalization request (used ring queue)
- **Virtio-net PMD**
  - Update notification (avail ring queue)
  - Update notification (used ring queue)

**Finalization phase**
- **From virtio-net-ipc**
  - Finalization request (avail ring queue)
  - Response
Performance evaluation
Evaluation setting and memory copy performance

Hardware configuration

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel Xeon E5-2697 v2 (2.7GHz) x 2 (SandyBridge)</th>
</tr>
</thead>
<tbody>
<tr>
<td># of CPU cores</td>
<td>12 (Hyper-thread off) (total 24)</td>
</tr>
<tr>
<td>Memory</td>
<td>DDR3-1866 RDIMM ECC 8GB x 8 (total 64GB)</td>
</tr>
<tr>
<td>Motherboard</td>
<td>Supermicro X9DRH-7TF (Intel C602)</td>
</tr>
<tr>
<td>Intel VT-x</td>
<td>enabled</td>
</tr>
</tbody>
</table>

Software configuration

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Ubuntu 12.04.4 LTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>Linux ubuntu 3.20-58-generic x86_64</td>
</tr>
<tr>
<td>QEMU</td>
<td>version 1.0</td>
</tr>
<tr>
<td>GCC</td>
<td>version 4.8.2</td>
</tr>
<tr>
<td>Isolcpus</td>
<td>2,3,4,5,6,7,8,9,10,11,14,15,16,17,18,19,20,21,22,23</td>
</tr>
<tr>
<td>Hugetlbfs</td>
<td>1 GB page x 40 for KVM testpmd apps</td>
</tr>
<tr>
<td>DPDK</td>
<td>1.6.0</td>
</tr>
<tr>
<td>Guest VM</td>
<td>4 CPU cores &amp; 8 GB memory</td>
</tr>
</tbody>
</table>

CPU assignment

<table>
<thead>
<tr>
<th>Logical CPU core</th>
<th>process assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 3</td>
<td>None</td>
</tr>
<tr>
<td>4 - 7</td>
<td>Host: testpmd</td>
</tr>
<tr>
<td>8 - 11</td>
<td>QEMU &amp; Guest VM: testpmd (using all cores (8-11))</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
</tbody>
</table>
Performance benchmark

- **micro benchmarking tool: Testpmd apps**
  - Polling-based DPDK bridge app that reads data from a NIC and writes data to another NIC in both directions.
  - null-PMD: a DPDK-enabled dummy PMD to allow packet generation from memory buffer and packet discard to memory buffer.
Performance evaluation

- **Virtq PMD achieved great performance**
  - 62.45 Gbps (7.36 MPPS) unidirectional throughput
  - 122.90 Gbps (14.72 MPPS) bidirectional throughput
  - 5.7 times faster than Linux driver in 64B, 2.8 times faster than Linux driver in 1500B

- **Virtq PMD achieved better performance in large packet to vhost app**
Conclusion

**Virtq-PMD: High performance virtio-net-based vNIC framework for NFV with HV domain**

- Achieved 122.90 Gbps (14.72 MPPS) throughput
- DPDK APIs and legacy network I/O API
- Operation-aware extension support
- Same overheads in any packet size.
Please try and fork me 😊

- **DPDK Virtq-PMD**
  - [https://github.com/lagopus/virtio-net-ipc-qemu-1.0](https://github.com/lagopus/virtio-net-ipc-qemu-1.0)
  - [https://github.com/lagopus/virtq-pmd-dpdk-1.6](https://github.com/lagopus/virtq-pmd-dpdk-1.6)

- **Lagopus vSwitch: yet another high-performance SDN/OpenFlow switch**
  - [http://lagopus.github.io/](http://lagopus.github.io/)
  - [https://github.com/lagopus/lagopus](https://github.com/lagopus/lagopus)