Netmap as a backend for VMs

25th Aug 2017, Vincenzo Maffione
Università di Pisa
Introduction

- Network I/O virtualization is about attaching the Virtual Machines to the network of the hypervisor/host.
- Let different VMs on the same host communicate among them and with the external physical network.
- Hot use-cases → Network Function Virtualization
Outline

1. Traditional Virtual Machine networking
2. Netmap as an hypervisor network back-end
3. VM networking based on Netmap passthrough
VM networking - Front-end

- VM has one or more *emulated* NICs
  - A regular driver is used to access an emulated NIC
- A guest NIC comes with its own *device model*
  - An *emulated commercial NIC* (e.g. e1000, r8169)
  - A *paravirtualized NIC* (e.g. virtio-net, Xen netfront, ptnet)
    - Paravirtualized NICs offer better performance, as drivers are *aware* of running in a virtualized environment
- Device model implemented by a *front-end* module in the hypervisor
  - Emulates device I/O register access
  - Injects PCI interrupts
  - Emulates receive and transmit functionalities, accessing the device *rings*
VM networking - Front-end rings

- A ring is a circular array of descriptors
- A descriptor contains information about a buffer to be used for TX or RX
  - At least the (guest) physical address and length of the buffer
- Different front-ends, different ring(descriptor) formats
  - Virtio-net: VirtQueues, Avail and Used rings, …
  - Xen netfront: I/O rings
  - Commercial NICs: hardware specs Intel, Realtek, … use formats specified in their documentation
VM networking - Back-end

- A **back-end** module allows VM network traffic to leave the hypervisor towards the host network.
- **Front-end** and **back-end** interact to transmit and receive packets to/from the host network.
- Different back-ends usually available:
  - TAP: inject/receive packets from host TCP/IP stack
  - Socket: packets forwarded through a TCP or UDP socket
  - NAT: backend implements NAT (in user-space) to give a VM easy internet access
  - Netmap/DPDK: Packets injected/received from an high performance userspace networking framework
- Different back-ends, different **packet representations**
The journey of a TX/RX packet is **convoluted**
- Guest OS/driver uses sk_buff/mbuf
- Front-end uses a chain of descriptors in a ring/queue
- Backend may use multiple formats, e.g.:
  - TAP and sockets uses sk_bufs/mbufs
  - Netmap uses its API (netmap rings and slots)
- Packet representation conversions is needed at each step
  - Conversions require processing
  - Copies may be needed
  - Front-end and/or back-end usually run in a separate I/O thread
    - Need driver/front-end synchronization and/or front-end/back-end synchronization
A traditional deployment

- **QEMU-KVM** hypervisor
- **virtio-net** front-end
  - With vhost acceleration
- **TAP** back-end attached to either
  - OpenVswitch switch instance, or
  - Standard in-kernel L2 bridge
- Many bottlenecks:
  - read/write to TAP interfaces
    - not batched
  - virtual switch processing
    - not batched
  - driver/front-end conversions and synchronization
Netmap as a back-end

- When the back-end is netmap, the netmap API is used to copy packets between the rings of the emulated NIC (e.g. virtio-net, e1000) and the rings of the netmap port
  - Copies are necessary, as netmap buffers are allocated by the host, while sk_buffs are allocated by the guest OS
  - Front-end and back-end code normally runs in an I/O thread separate from the guest vCPU threads

- The netmap port can be anything:
  - A VALE switch is a good choice to connect several VMs together, and possibly also to a physical NIC.
  - A netmap pipe, to create a fast point-to-point virtual link between two VMs.
  - A physical netmap port, to give the VM exclusive access to an host physical device.
Netmap support in hypervisors

- A netmap back-end is available upstream for
  - QEMU ([http://qemu.org](http://qemu.org)), mainly to be used on Linux with KVM
  - bhyve ([http://bhyve.org](http://bhyve.org)), to be used on FreeBSD hosts

- However, QEMU support for netmap passthrough is not upstream yet
  - The modified QEMU can currently be found at [https://github.com/vmaffione/qemu.git](https://github.com/vmaffione/qemu.git)
  - We will use this code through the tutorial
  - Compile QEMU with netmap support
    - `$ ./configure --target-list=x86_64-softmmu --enable-kvm
      --enable-vhost-net --disable-werror --enable-netmap --enable-ptnetmap`
Run a QEMU VM with a netmap back-end

● Example to run a VM with a single emulated NIC

```bash
# qemu-system-x86_64 img.qcow2 -enable-kvm -smp 2 -m 2G -vga std
-device virtio-net-pci,netdev=data1,mac=00:AA:BB:CC:0a:0a,iodev=on,mrg_rxbuf=on
-netdev netmap,ifname=vale1:10,id=data1
```

● The `-netdev` option specifies a back-end, while the `-device` option specifies a front-end. The id is used to bind a front-end to a back-end.

● The `ifname` argument specifies the netmap port. Examples:
  ○ “valeXX:YY” → A VALE port named “YY” on a VALE switch called “XX”
  ○ “netmap:eth4” → physical netmap port to access the eth4 host network device
  ○ “netmap:pipe0[1” → master side of a netmap pipe called pipe0
  ○ “netmap:pipe0}1” → slave side of a netmap pipe called pipe0

● Multiple occurrences of `-netdev` and `-device` allow for multiple emulated NICs
VM-to-VM performance experiments

- Two VMs running on the same host, connected through a software switch or a point-to-point link
- We run some benchmarks to measure performance
  - TCP_STREAM test for unidirectional bulk TCP transfer
  - TCP_RR test for request/response short transactions
  - Ping flood to measure average latency
  - pkt-gen tests to measure maximum packet-rate for short packets with guest OS bypass
- A VM runs the sender, while the other VM runs the receiver
VM-to-VM performance experiments

- Two VMs running on the same host, connected through an in-kernel sw bridge

```
$ qemu-system-x86_64 img.qcow2 -enable-kvm -smp 2 -m 2G -vga std -device virtio-net-pci,netdev=data10,mac=00:AA:BB:CC:0a:0a,iomv状元=on,mrg_rxdump=on -netdev tap,ifname=tap1,id=data10,vhost=off
```
VM-to-VM performance experiments

- Two VMs running on the same host, connected through a VALE software switch

```
# qemu-system-x86_64 img.qcow2 -enable-kvm -smp 2 -m 2G -vga std -device virtio-net-pci,netdev=data10,mac=00:AA:BB:CC:0a:0a,iodevfd=on,mrg_rxbuf=on -netdev netmap,ifname=vale1:10,id=data10
```
VM-to-VM performance experiments

- Two VMs running on the same host, connected through a netmap pipe

```
# qemu-system-x86_64 img.qcow2 -enable-kvm -smp 2 -m 2G -vga std -device virtio-net-pci,netdev=data1,mac=00:AA:BB:CC:0a:01,ioeventfd=on,mrg_rxbuf=on -netdev netmap,ifname=netmap:pipe0{1,id=data1
```
# Experiment results

<table>
<thead>
<tr>
<th>Front-end</th>
<th>Back-end</th>
<th>TCP_STREAM</th>
<th>TCP_RR</th>
<th>ping -f</th>
<th>pkt-gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtio-net</td>
<td>TAP</td>
<td>18.2 Gbps</td>
<td>18.4 Ktps</td>
<td>75 us</td>
<td>0.5 Mpps</td>
</tr>
<tr>
<td>virtio-net</td>
<td>netmap VALE</td>
<td>21 Gbps</td>
<td>20 Ktps</td>
<td>55 us</td>
<td>1 Mpps</td>
</tr>
<tr>
<td>virtio-net</td>
<td>netmap pipe</td>
<td>7.9 Gbps</td>
<td>20.5 Ktps</td>
<td>65 us</td>
<td>1 Mpps</td>
</tr>
<tr>
<td>e1000</td>
<td>TAP</td>
<td>3.5 Gbps</td>
<td>9.5 Ktps</td>
<td>80 us</td>
<td>1.3 Mpps</td>
</tr>
<tr>
<td>e1000</td>
<td>netmap VALE</td>
<td>2.8 Gbps</td>
<td>9.5 Ktps</td>
<td>80 us</td>
<td>1.4 Mpps</td>
</tr>
</tbody>
</table>

- **Virtio-net + TAP**
  - Good netperf performance because of TSO/checksum offloadings
- **Virtio-net + VALE**
  - Good netperf/ping numbers, because of TSO/checksum offloadings
  - Slightly better than virtio-net + TAP because of slightly reduced latency
  - However, performance gain is not evident because there is *no batching*
## Experiment results

<table>
<thead>
<tr>
<th>Front-end</th>
<th>Back-end</th>
<th>TCP_STREAM</th>
<th>TCP_RR</th>
<th>ping -f</th>
<th>pkt-gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtio-net</td>
<td>TAP</td>
<td>18.2 Gbps</td>
<td>18.4 Ktps</td>
<td>75 us</td>
<td>0.7 Mpps</td>
</tr>
<tr>
<td>virtio-net</td>
<td>netmap VALE</td>
<td>20 Gbps</td>
<td>20 Ktps</td>
<td>55 us</td>
<td>1 Mpps</td>
</tr>
<tr>
<td>virtio-net</td>
<td>netmap pipe</td>
<td>7.9 Gbps</td>
<td>20.5 Ktps</td>
<td>65 us</td>
<td>1 Mpps</td>
</tr>
<tr>
<td>e1000</td>
<td>TAP</td>
<td>3.5 Gbps</td>
<td>9.5 Ktps</td>
<td>80 us</td>
<td>1.3 Mpps</td>
</tr>
<tr>
<td>e1000</td>
<td>netmap VALE</td>
<td>2.8 Gbps</td>
<td>9.5 Ktps</td>
<td>80 us</td>
<td>1.4 Mpps</td>
</tr>
</tbody>
</table>

- **e1000 + TAP:**
  - Low performances in general, as e1000 emulation is slow and there is no batching

- **e1000 + VALE/pipe**
  - Similar bad performance for netperf tests
  - Good pkt-gen number → batching is actually possible because of netmap support for e1000
Offloadings to boost TCP performance

- virtio-net header (metadata prepended to each Ethernet frame) is the key to boost TCP performance:
  - TCP Segmentation Offloading
  - TCP/UDP checksum offloading

- Mechanism, in short:
  - VMs on the same host can exchange 32K/64K TCP packets without ever performing TCP segmentation or computing TCP checksum
  - If a TSO packet needs to leave the host system, segmentation and checksumming can be offloaded to real NIC hardware

- Header supported by the VALE switch ports and TAP interfaces
Pass-through of host netmap ports
Towards an alternative approach

- **A netmap port** is accessed through the netmap API
  - Hardware-independent rings and buffers are mapped into the userspace application address space
  - TX/RX batching is a key point to remove I/O bottlenecks
- **Various port types**, depending on the backing I/O
  - Physical ports (NICs)
  - VALE (virtual L2 switch) ports
  - Pipes
  - Monitors
- **Software running in the guest can access the front-end rings**
  - So far it **could not access the (back-end) netmap rings** directly.
Netmap pass-through (ptnetmap)

- What if we map netmap rings and buffers of an host port inside a VM?
- The VM could directly access the host port!
- A special pass-through netmap port is used to do the trick
  - I/O registers used to ask the host to TX/RX sync
  - TX/RX sync operations performed by host kernel I/O threads, or directly in the context of the I/O access
  - Shared memory (Communication Status Block) used to synchronize guest ring indices with host ring indices

[ANCS 2015] - Virtual device passthrough for high speed VM networking
The *ptnet* front-end

- A device model is needed to expose the passed-through port to the guest OS
- *ptnet* is a paravirtualized NIC which uses the netmap API as the underlying device model
- No format conversions necessary between front-end and back-end
- Guest can access the back-end port directly, using netmap passthrough
- The front-end is used only for
  - **Configuration**: number of available queues
  - **Control**: start/stop kernel threads
  - **Synchronization**: kick/interrupt
- … but it’s not part of the datapath
Socket API and netmap API

- Support for applications running in netmap mode
  - They directly access the back-end netmap port
- Support for traditional socket applications
  - Conversion between sk_buff/mbuf and netmap slots is performed, including a packet copy
  - The driver behaves as an application running in netmap mode
  - Support for TSO and checksum offloadings
ptnetmap for NIC passthrough

- The VM sees all the queues of the physical NIC
- Netmap performance preserved in the guest
  - 14.88 Mpps TX/RX with a single core
- No PCI passthrough support needed in the hypervisor
  - Netmap code is reused
ptnetmap with VALE

- VALE as *software switch* between VMs and the NIC
- VALE ports are passed through to the VMs
- Up to 20 Mpps between different VMs
  - When using the netmap API
ptnetmap with netmap pipes for NVF

- Netmap pipes allow for zero-copy packet forwarding
- They make it easy to set up high-speed NVF chains
- Netmap applications run in the VMs
- How does it compare to traditional VirtIO deployments?
ptnetmap vs VirtIO: in-kernel pkt-gen

- Even traditional network stack is able to benefit from ptnet batching

- Why the Mpps drop? [ANCS 2016] A Study of Speed Mismatches Between Communicating Virtual Machines
ptnetmap vs VirtIO: socket applications

Guest-to-guest TCP/UDP latency tests

Guest-to-guest UDP throughput tests
Netmap pass-through in a nutshell

- Flexible → the same tool provides multiple functions
  - NIC passthrough
  - Hypervisor software switch
  - Support for novel packet processing applications

- Easy to port
  - Code reuse is maximized, most of code is part of netmap
  - 500-1000 lines of code to add support for new hypervisors

- Good performance
  - Comparable with VirtIO + vhost-net with socket applications
  - Native netmap performance for applications using the netmap API

- ptnetmap guest drivers and host support are available for both Linux and FreeBSD
  - Not yet upstream in QEMU
VM-to-VM experiments with ptnetmap

- VALE ports are passed-through to the VMs with ptnetmap

```
# qemu-system-x86_64 img.qcow2 -enable-kvm -smp 2 -m 2G -vga std
   -device ptnet-pci,netdev=data10,mac=00:AA:BB:CC:0a:0a
   -netdev netmap,ifname=vale1:10,id=data10,passsthrough=on
```
VM-to-VM experiments with ptinetmap

- The two ends of the pipe are passed-through to the VMs with ptinetmap

```
# qemu-system-x86_64 img.qcow2 -enable-kvm -smp 2 -m 2G -vga std
  -device ptnet-pci,netdev=data1,mac=00:AA:BB:CC:0a:01
  -netdev netmap,ifname=netmap:pipe0{1,id=data1,passsthrough=on
```
# Experiment results

<table>
<thead>
<tr>
<th>Front-end</th>
<th>Back-end</th>
<th>TCP_STREAM</th>
<th>TCP_RR</th>
<th>ping -f</th>
<th>pkt-gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtio-net</td>
<td>vhost/TAP</td>
<td>27.2 Gbps</td>
<td>35.5 Ktps</td>
<td>35 us</td>
<td>1 Mpps</td>
</tr>
<tr>
<td>ptnet</td>
<td>netmap VALE</td>
<td>21.7 Gbps</td>
<td>46.7 Ktps</td>
<td>28 us</td>
<td>19 Mpps</td>
</tr>
<tr>
<td>ptnet</td>
<td>netmap pipe</td>
<td>7.4 Gbps</td>
<td>47 Ktps</td>
<td>26 us</td>
<td>46 Mpps</td>
</tr>
</tbody>
</table>

- **Virtio-net + vhost/TAP**
  - Excellent netperf performance because of TSO/checksum offloadings and in-kernel vhost
- **ptnet + VALE**
  - Good netperf performance because of TSO/checksum offloadings and in-kernel processing, but has to pay an additional copy
  - Excellent pkt-gen numbers because of the batching provided by netmap API
- **ptnet + netmap pipe**
  - Excellent pkt-gen and ping numbers, but no support for offloadings
Hands on exercises