DPDK Based Vswitch Upgrade

Yuanhan Liu

Tencent *yuanhanliu@tencent.com*

Dec 04, 2018

Outline

1. Introduction
2. Make it fast - around 400ms
3. Faster - around 50ms

The key point

We have to upgrade, for

- bug fixes
- new features

Has to be fast, as it impacts VM networking connections

Why multiple process is NOT a good solution?

How multi-process basically works?

- The primary init and configures the port, and store key port infos to struct *nic_private*
- The secondary skips init and config, and get the port infos from the struct *nic_private* filled by the primary process

The fatal issue: **it can't work across versions**

- *nic_private* will likely change across versions
- ABI/API changes

Which makes it painful for switching DPDK versions; therefore, **not good for long run**

What slows the startup down?

The major part goes to the **DPDK hugepage memory init**

- DPDK ($<=$ v18.02) allocates **all** free hugepages (even if we want 1G), then free the rest
- populating one 1G hugepage is time consuming, as it involves memset

```
$ time reserve_hugepage /dev/hugepages/1G-file 1
real    0m0.200s
user    0m0.000s
sys     0m0.197s

$ grep HugePages_Free /proc/meminfo
HugePages_Free:       60

$ time testpmd -l 0 --no-pci --socket-mem 1024,1024 -- --invalid-arg &>/dev/null
real    0m11.962s
user    0m0.001s
sys     0m11.938s
```

One solution to hugepage init: --mem-file option

- populate the hugepages **before** starting DPDK
- introduce --**mem-file** option
    - DPDK memories will only be allocated from given files
    - memory size is retrieved by fstat

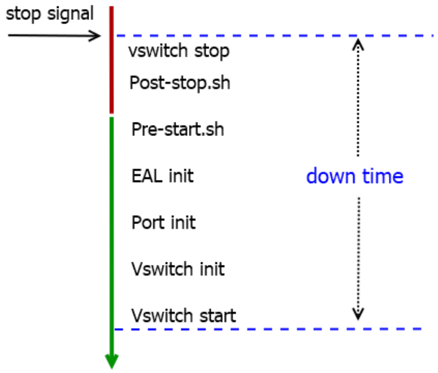
```
$ reserve_hugepage /dev/hugepages/node0 1 0
$ reserve_hugepage /dev/hugepages/node1 1 1

$ time testpmd -l 0 --no-pci --mem-file /dev/hugepages/node0,/.../node1 \
       -- --invalid-arg
...
real    0m0.026s
user    0m0.002s
sys     0m0.009s
```

What else may slow it down?

Just share something interesting:

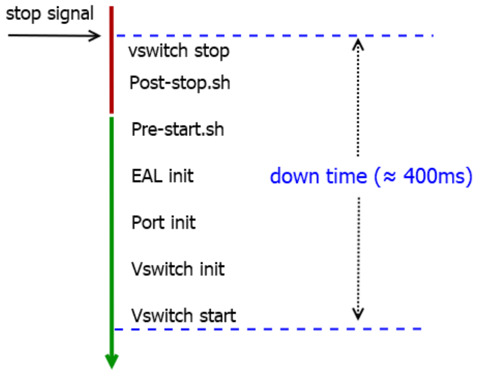
- lspci may take up to 50ms!
- probe a port may take more than 100ms
- configure and start a port may take another 100ms
- mbuf mempool creation is also time consuming: it touches a big chunk of memory to construct all the mbuf instances, which may take few hundreds of ms

How to make it fast?



- make post-stop.sh and pre-start.sh as lightweight as possible
  - put everything possible before sending the stop signal (lspci, hugepages reservation, etc)
- parallel init

How to make it fast?



- make post-stop.sh and pre-start.sh as lightweight as possible
    - put everything possible before sending the stop signal (lspci, hugepages reservation, etc)
- parallel init

Is it good enough?

It's being okay, but far from being perfect!

And it has bottlenecks:

- Use a trick to save few ms may break something else
- Add a new feature will likely make the downtime longer again

How to make it faster?

The inspiration: **live migration**

- While the current (and the old) vswitch is running, start another vswitch instance
- When the new instance is ready, do the switch
- The downtime $\approx$ the switch time: **short** and **constant**

More like master-slave **backup** model though, instead of live migration

backup: two key challenges - I

Start 2 DPDK instances on one NIC port

- create 2 VFs, one per instance
- configure NIC to steer pkts to the right VF

backup: two key challenges - II

vhost-user support: the simple one

- stop vhost-user from the old vswitch
- restart vhost-user from the new vswitch

The drawback: **have to reconnect all vhost devices one by one**

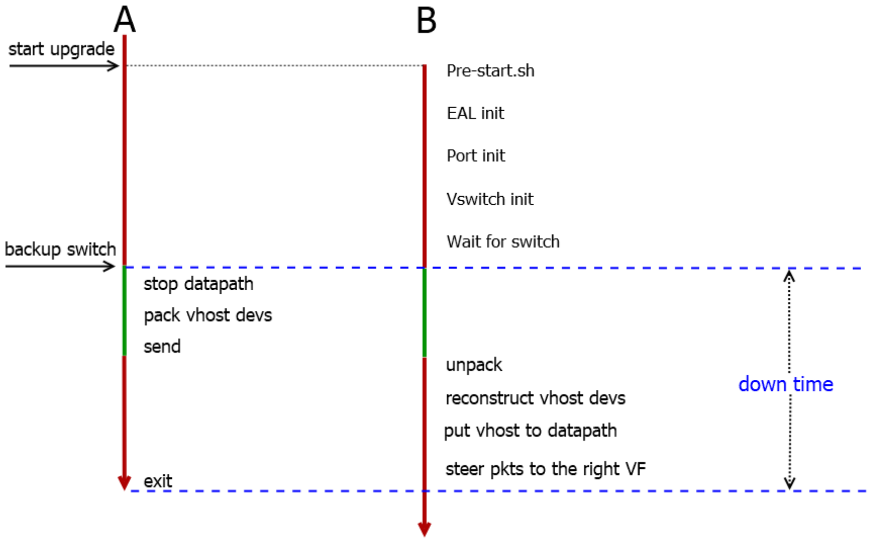
backup: two key challenges - II

vhost-user support: the better one

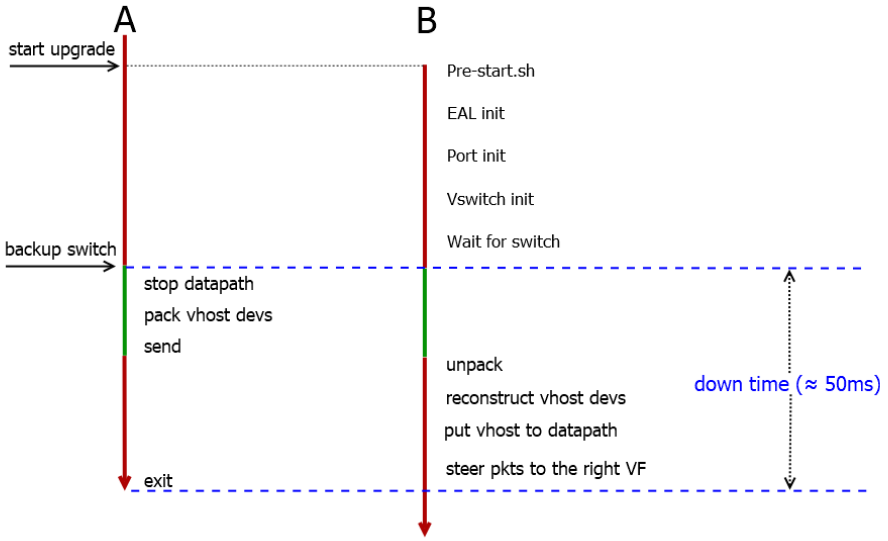
- A: pack all vhost devices information, including features, mem table, vring addr, *the vhost-user socket connection with QEMU*, etc
- A: send them to B by unix socket: fds will be passed (thanks to the ancillary data)
- B: unpack and re-construct all the vhost devices and socket connections
    - allocate new device
    - re-map the vhost memory
    - re-translate the vring addr based on the new mapping
    - *add the socket connection fd back to the event loop*
    - now it's ready

The vhost-user socket connection is well kept: no reconnect required at all

backup: the big picture



backup: the big picture



backup: the possiblity of making it upstream?

- proved to be working fine with vhost-user
- need to prove it may work fine with a PCI device (say, virtio-net)

Summary

The backup method has 2 advantages:

- **short** and **constant** down time
- vhost-user connections are well retained

Thank You!