Implementing DPDK based Application Container Framework with SPP

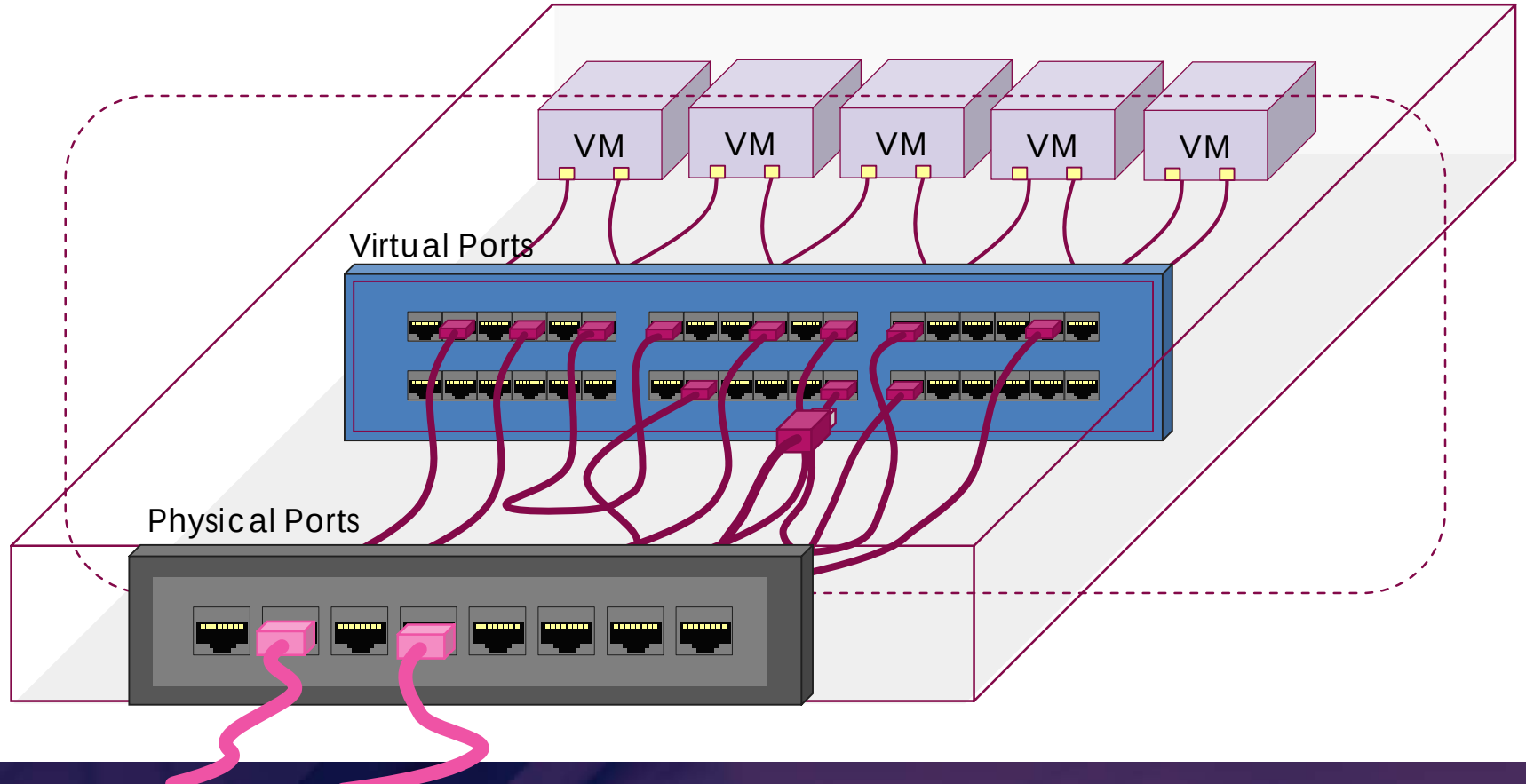
YASUFUMI OGAWA, NTT

Agenda

- Introduction of SPP
- SPP Container
- Containerize DPDK Apps
- SPP Container Tools
- Usecases
- Limitations and Restrictions
- Debugging

Introduction of SPP

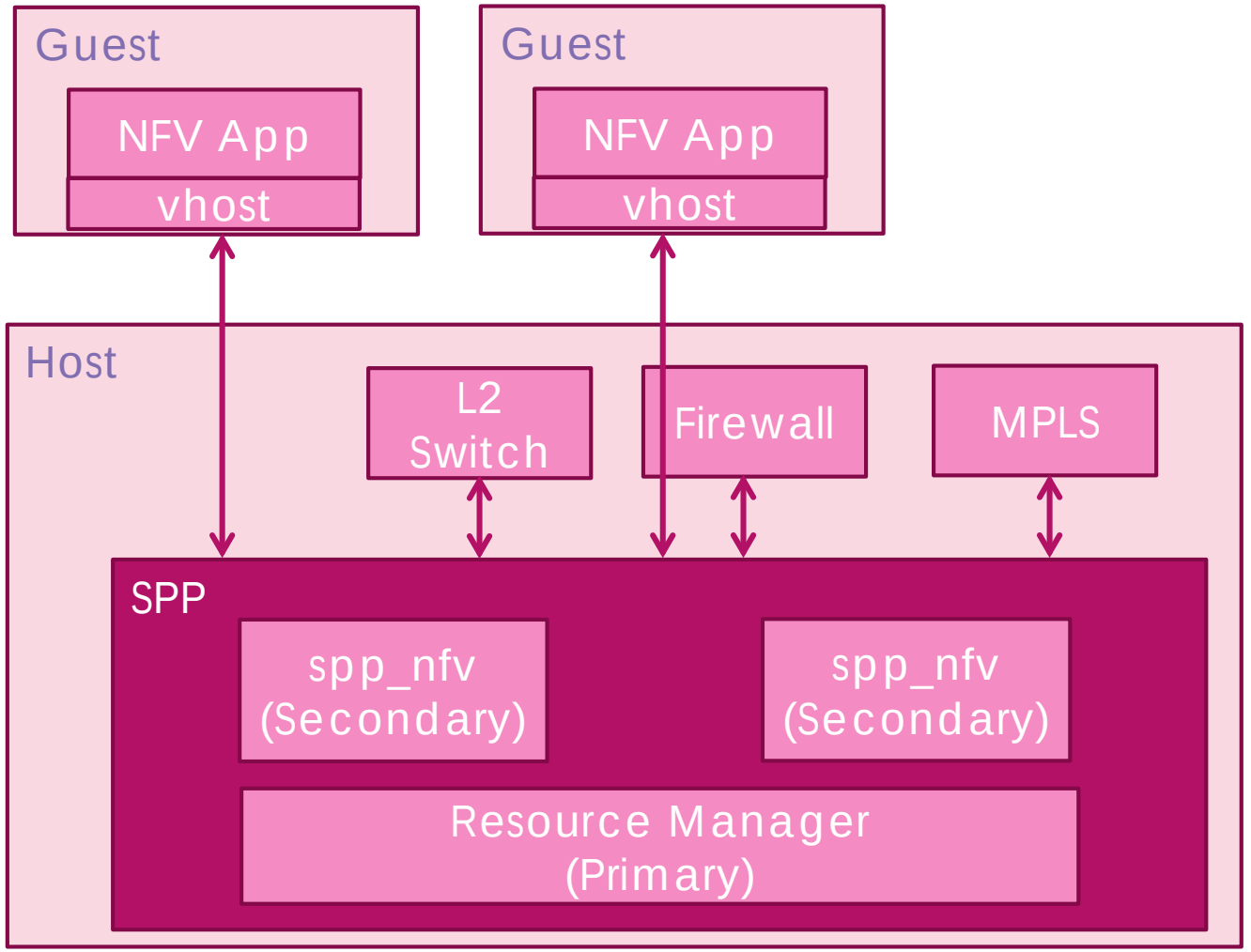
- Change network path with patch panel like simple interface
- High-speed packet processing with DPDK
- Update network configuration dynamically without terminating services



Design

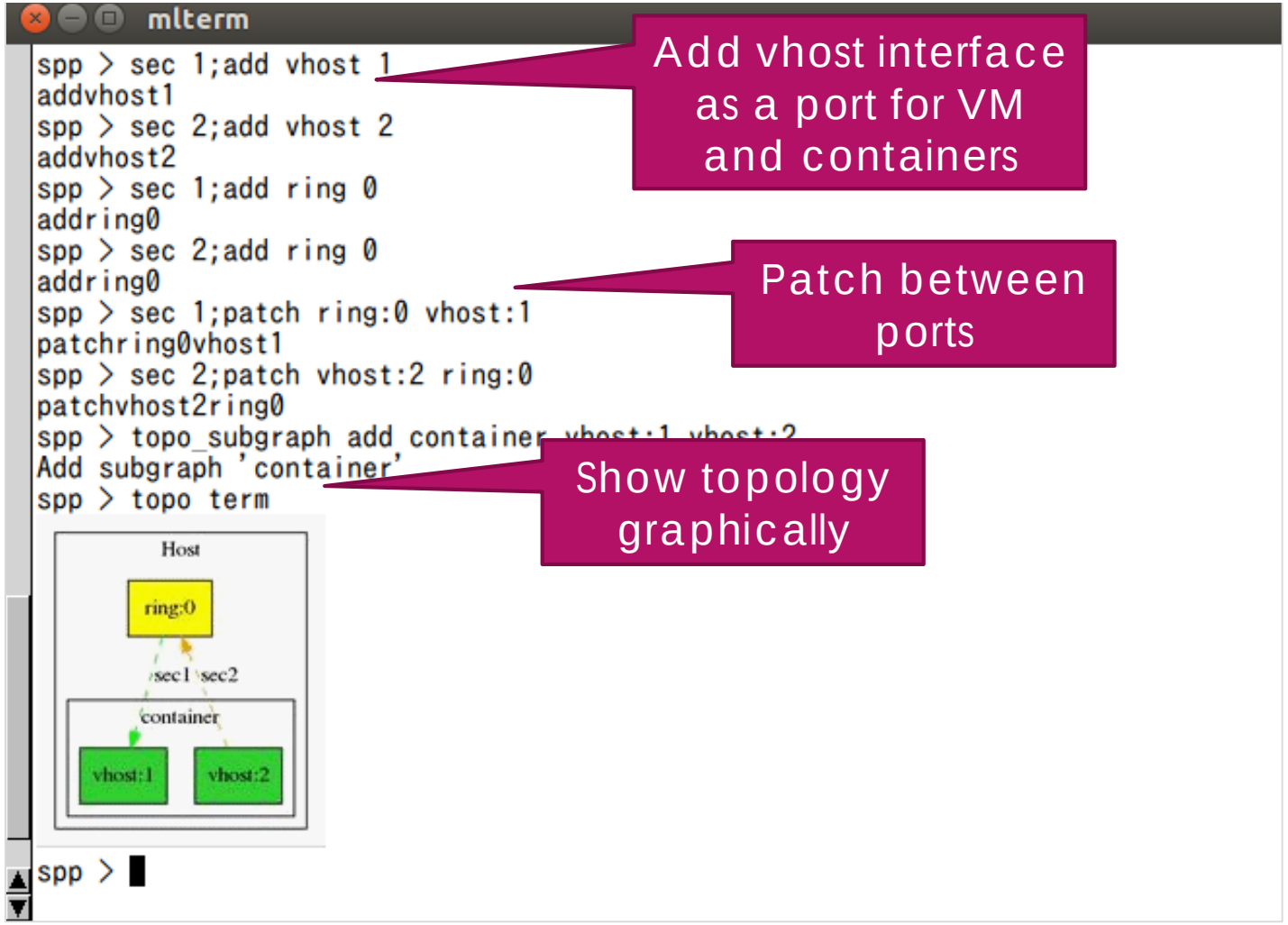
- Multi-process Application
  - Primary process is a resource manager
  - Secondary processes are workers for packet forwarding
    - spp_nfv (Direct forwarding)
    - spp_vf (SR-IOV features)
    - spp_mirror (TaaS)
- Several Virtual Port Support
  - ring pmd
  - vhost pmd
  - pcap pmd

  etc.

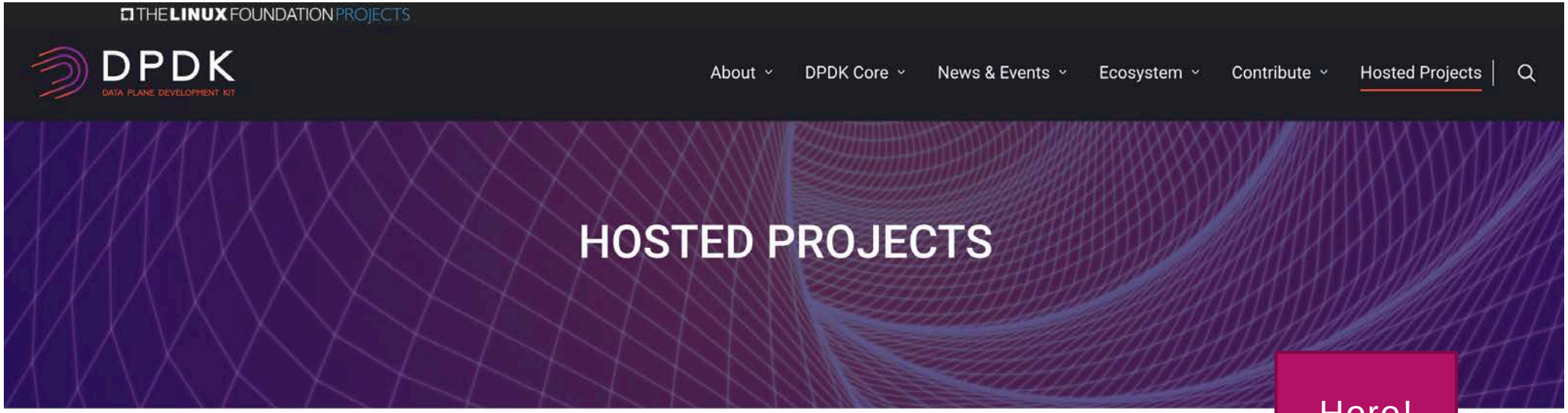


Patch Panel-like Interface

- Simple to add ports and connect them



Hosted Project



Here!

Pktgen

Traffic generator powered by DPDK

Git Repo | Latest Release | Docs | Mailing List

SPP

Soft Patch Panel – DPDK Resource Management Framework

Git Repo | Latest Release | Docs | Mailing List

DTS

DPDK Test Suite

Git Repo | Latest Release | Docs | Test Plans

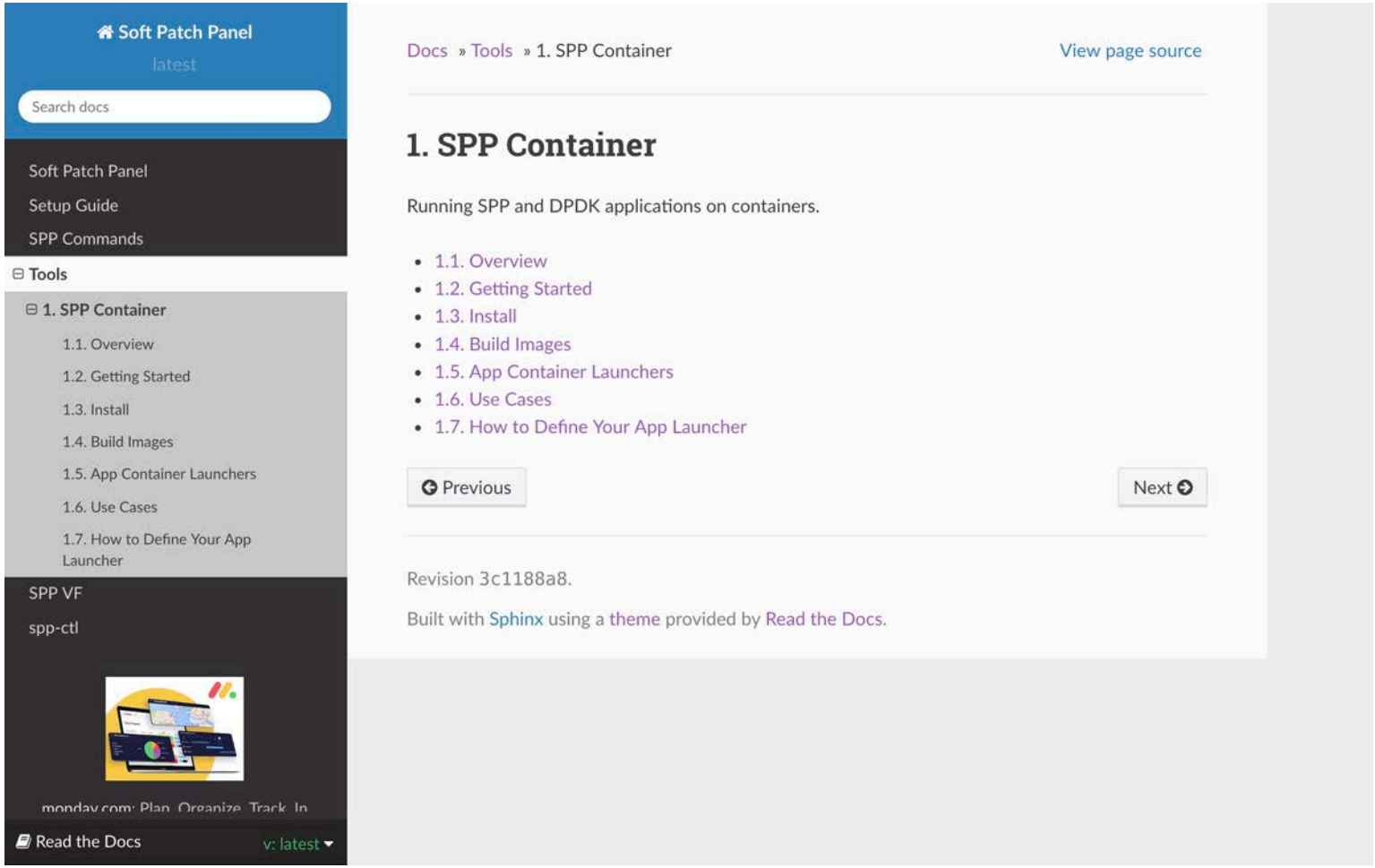
NFF-Go

Network Function Framework for Go

Git Repo

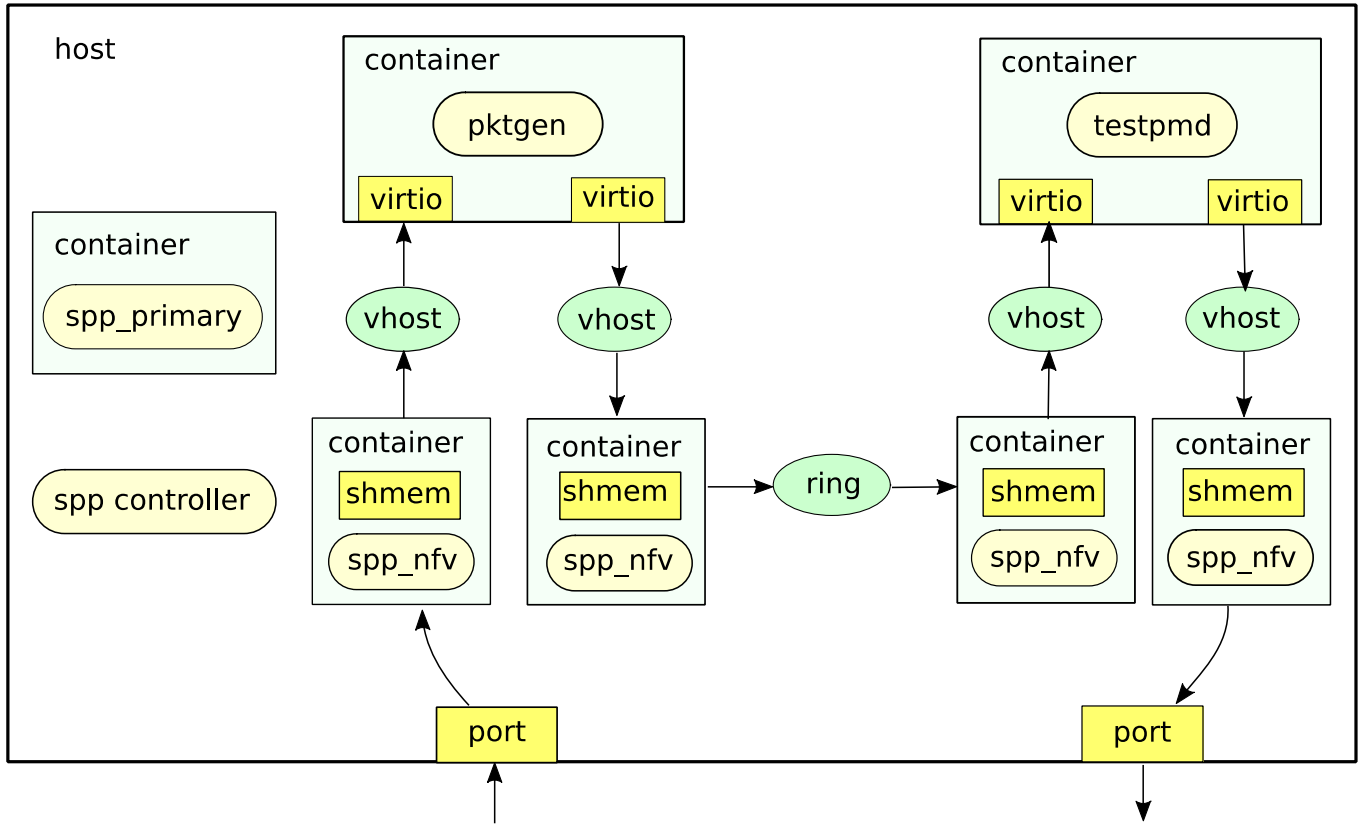
The Latest Documentation

https://spp.readthedocs.io/en/latest/



SPP Container

- Toolset for easy deployment of containerized DPDK apps
- Containerize DPDK apps for running as App containers
- SPP is also launched as App containers for configuring network



Containerize DPDK Apps

- Build container images with dedicated Dockerfiles
- Launch DPDK App via docker command with specifying ...
  - Docker options for resources shared between host and container (socket file , hugepages, etc.)
  - The name of binary
  - Options for DPDK’s EAL and application itself

Build Container Image

```
FROM ubuntu:16.04
ARG rte_sdk
ARG rte_target
ARG home_dir
ARG dpdk_repo
ARG dpdk_branch
ENV PATH ${rte_sdk}/${rte_target}/app:${PATH}
ENV RTE_SDK ${rte_sdk}
ENV RTE_TARGET ${rte_target}
RUN apt-get update && apt-get install -y ¥
    git gcc python pciutils make libnuma-dev gcc-multilib ¥
    libarchive-dev linux-headers-$(uname -r) libpcap-dev pkg-config ¥
    && apt-get clean && rm -rf /var/lib/apt/lists/*
WORKDIR $home_dir
RUN git clone $dpdk_branch $dpdk_repo
# Compile DPDK
WORKDIR $rte_sdk
RUN make install T=$rte_target
RUN make app T=$rte_target
RUN make examples T=$rte_target
# Set working directory when container is launched
WORKDIR ${home_dir}
ADD env.sh ${home_dir}/env.sh
RUN echo "source ${home_dir}/env.sh" >> ${home_dir}/.bashrc
```

```
$ sudo docker build ¥
 --build-arg home_dir=/root ¥
 --build-arg rte_sdk=/root/dpdk ¥
 --build-arg rte_target=x86_64-native-linuxapp-gcc ¥
 --build-arg dpdk_repo=http://dpdk.org/git/dpdk ¥
 --build-arg dpdk_branch= ¥
 -f ./build/ubuntu/dpdk/Dockerfile.16.04 ¥
 -t sppc/dpdk-ubuntu:16.04 ¥
 ./build/ubuntu/dpdk
```

Install packages with apt-get

Compile DPDK and apps

Activate environments

Launch Containerized DPDK App

- Example of launching l2fwd with two vhost ports

```
$ sudo docker run -d --privileged ¥
 -v /tmp/sock5:/var/run/usvhost5 ¥
 -v /tmp/sock6:/var/run/usvhost6 ¥
 -v /dev/hugepages:/dev/hugepages ¥
 sppc/dpdk:16.04 ¥
 /root/dpdk/examples/l2fwd/x86_64-native-linuxapp-gcc/l2fwd ¥
 -l 3,4 ¥
 -n 4 ¥
 -m 1024 ¥
 --proc-type auto ¥
 --vdev virtio_user5,queues=1,path=/var/run/usvhost5 ¥
 --vdev virtio_user6,queues=1,path=/var/run/usvhost6 ¥
 --file-prefix spp-l2fwd-container5 ¥
 -- ¥
 -p 0x03
```

Run with “--privileged” to share sockets and hugepages

Use “--file-prefix” for preparing metadata file for the process

SPP Container Tools

- A set of Python scripts and Dockerfiles for building and launching
- ‘build’ tool is used for creating container images
  - Support several distributions and versions (but only Ubuntu currently)
  - Apps included in DPDK, Pktgen and SPP
- Each of app containers are launched via ‘app’ tools
  - testpmd
  - pktgen
  - DPDK sample apps (l2fwd, etc.)
  - SPP

```
dpdk1805@lancer:~/spp/tools/sppc$ tree
├── app
│   ├── __init__.py
│   ├── l2fwd.py
│   ├── spp-nfv.py
│   ├── ...
│   └── testpmd.py
├── build
│   ├── main.py
│   ├── ...
│   └── ubuntu
│       ├── dpdk
│       │   ├── Dockerfile.16.04
│       │   ├── Dockerfile.18.04
│       │   ├── Dockerfile.latest
│       │   └── env.sh
│       ├── ...
├── conf
│   ├── env.py
│   ├── ...
├── lib
│   ├── __init__.py
│   ├── ...
```

Build tool

- Expand to ‘docker build’ with options for your target environments
- Choose the Dockerfile and define the name of container image from the options

```
$ python ./build/main.py
--dist-name ubuntu ¥
-t dpdk ¥
--dist-ver 16.04 ¥
--dpdk-repo https://github.com/yasufum/dpdk-custom.git
```

```
├── build
│   └── ubuntu
│       ├── dpdk
│       │   ├── Dockerfile.16.04
│       ├── ...
```

```
sudo docker build ¥
 --build-arg home_dir=/root ¥
 --build-arg rte_sdk=/root/dpdk ¥
 --build-arg rte_target=x86_64-native-linuxapp-gcc ¥
 --build-arg dpdk_repo=https://github.com/yasufum/dpdk-custom.git ¥
 --build-arg dpdk_branch= ¥
 -f ./build/ubuntu/dpdk/Dockerfile.16.04 ¥
 -t sppc/dpdk-ubuntu:16.04 ¥
 ./build/ubuntu/dpdk
```

Dockerfile

Container image

App tool

- Expand to ‘docker run’ with options of docker docker, DPDK EAL and the app
- Vhost is simply assinged by giving IDs with ‘-d’ option

```
$ ./app/l2fwd.py --dist-ver 16.04 -p 0x03 -l 1-2 -d 1,2
```

```
sudo docker run ¥
 -d ¥
 --privileged ¥
 -v /tmp/sock1:/var/run/usvhost1 ¥
 -v /tmp/sock2:/var/run/usvhost2 ¥
 -v /dev/hugepages:/dev/hugepages ¥
 sppc/dpdk-ubuntu:16.04 ¥
 /root/dpdk/examples/l2fwd/x86_64-native-linuxapp-gcc/l2fwd ¥
 -l 1-2 -n 4 -m 1024 ¥
 --proc-type auto ¥
 --vdev virtio_user1,queues=1,path=/var/run/usvhost1 ¥
 --vdev virtio_user2,queues=1,path=/var/run/usvhost2 ¥
 --file-prefix spp-l2fwd-container1 ¥
 -- ¥
 -p 0x03
```

Sock files are mounted as ‘/var/run/usvhost*’ to be shared

Assigned with ‘--vdev’ on the container

Create App Container of Your App

- You can launch your own application by building a container image and install your application
- Define Dockerfile for your application
  - Packages installation
  - Get and compile DPDK and your app from repos
  - Configure env on the container
- Create App Container Script

App Container Script

- To understand how to implement app container script, ‘app/helloworld.py’ is the best example
- There are just three parts should be changed for your app
  - Path of binary of your app
  - File prefix for metadata file
  - Options for your app (not includes EAL opts)
- Docker and DPDK EAL options are setup by helper methods
  - app_helper.setup_docker_opts()
  - app_helper.setup_eal_opts()

```
32 def main():
33     args = parse_args()
34
35     # Check for other mandatory opitons.
36     if args.dev_ids is None:
37         common.error_exit('--dev-ids')
38
39     # Setup for vhost devices with given device IDs.
40     dev_ids_list = app_helper.dev_ids_to_list(args.dev_ids)
41     sock_files = app_helper.sock_files(dev_ids_list)
42
43     # Setup docker command.
44     docker_cmd = ['sudo', 'docker', 'run', '\\']
45     docker_opts = app_helper.setup_docker_opts(
46         args, target_name, sock_files)
47
48     # Setup helloworld run on container.
49     cmd_path = '%s/examples/helloworld/%s/helloworld' % (
50         env.RTE_SDK, env.RTE_TARGET)
51
52     hello_cmd = [cmd_path, '\\']
53
54     file_prefix = 'spp-hello-container%d' % dev_ids_list[0]
55     eal_opts = app_helper.setup_eal_opts(args, file_prefix)
56
57     # No application specific options for helloworld
58     hello_opts = []
59
60     cmds = docker_cmd + docker_opts + hello_cmd + eal_opts + hello_opts
61     if cmds[-1] == '\\':
62         cmds.pop()
63     common.print_pretty_commands(cmds)
64
65     if args.dry_run is True:
66         exit()
67
68     # Remove delimiters for print_pretty_commands().
69     while '\\' in cmds:
70         cmds.remove('\\')
71     subprocess.call(cmds)
```

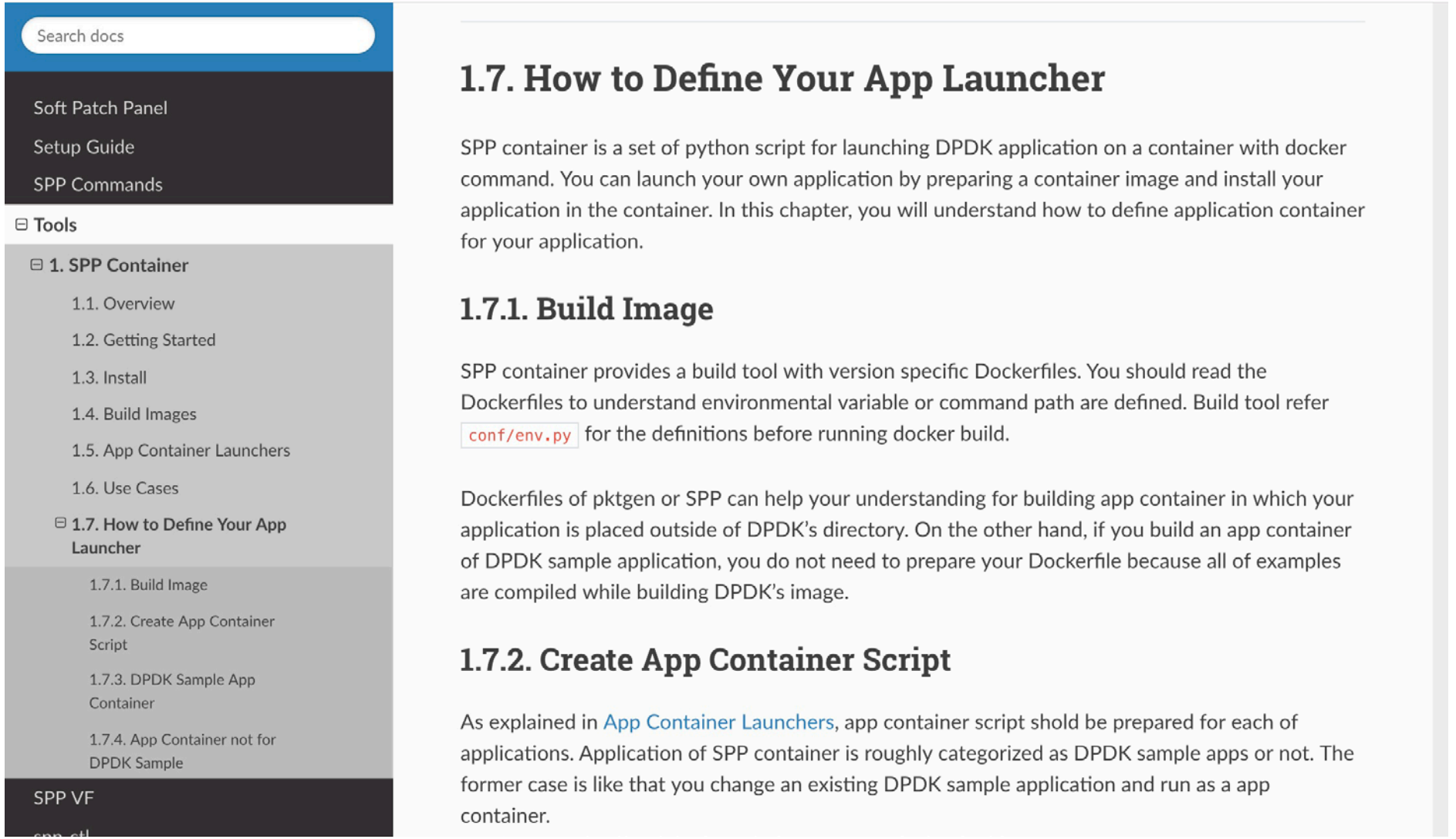
Path of binary of your app

File prefix

Options for your app

Create App Container of Your App

https://spp.readthedocs.io/en/latest/tools/sppc/howto_launcher.html



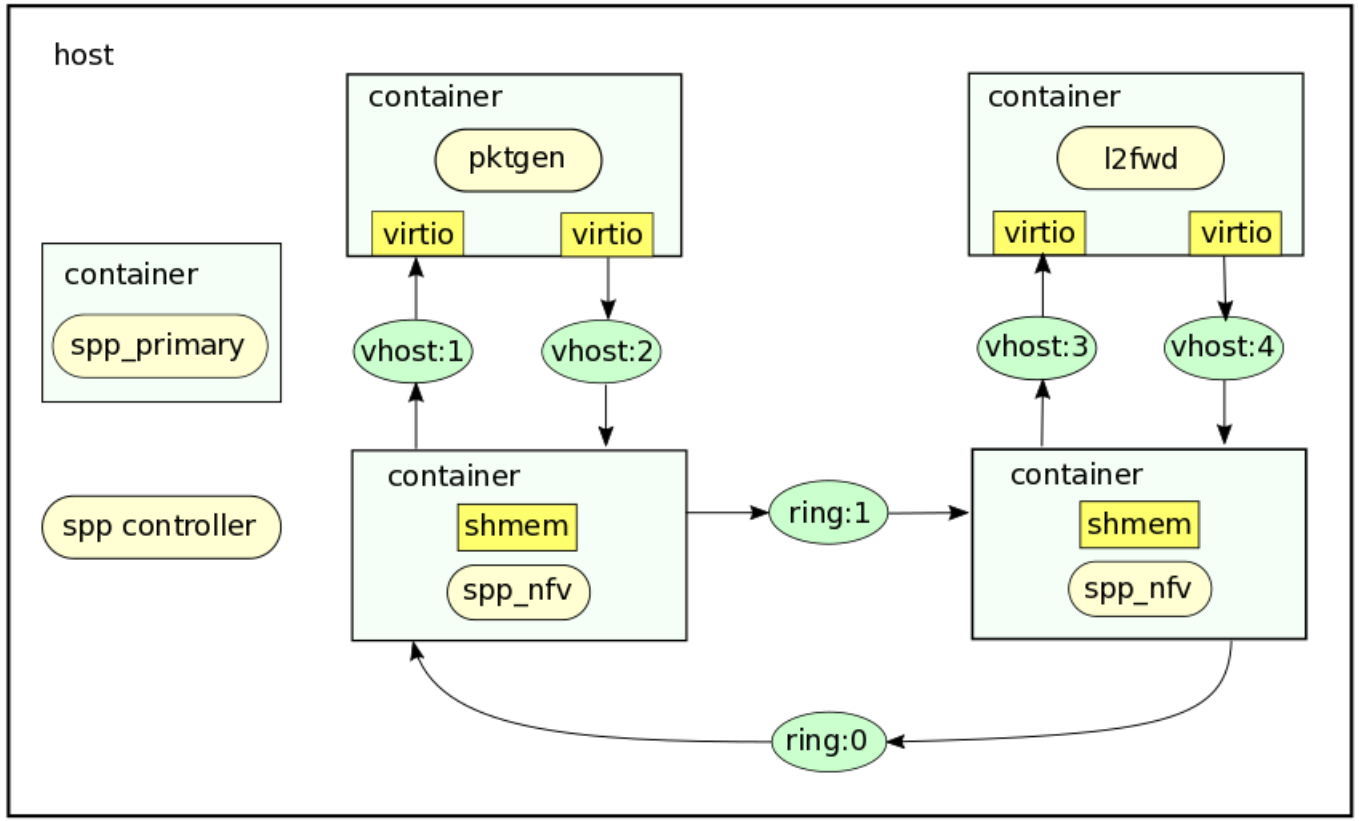
Usecases

- Several usecases suitable for using containerized DPDK apps
  - NFV’s service function chaining by using fewer resources than VMs
    - Service entities running on containers are instantiated and removed cleanly
    - Less time to launch container than VM and start service
  - High performance packet forwarding via shared memory
    - Enable to zero-copy packet forwarding between each of containers of a multi-process application
    - It is still insufficient performance of vhost for some of telco’s requirements

Usecase 1

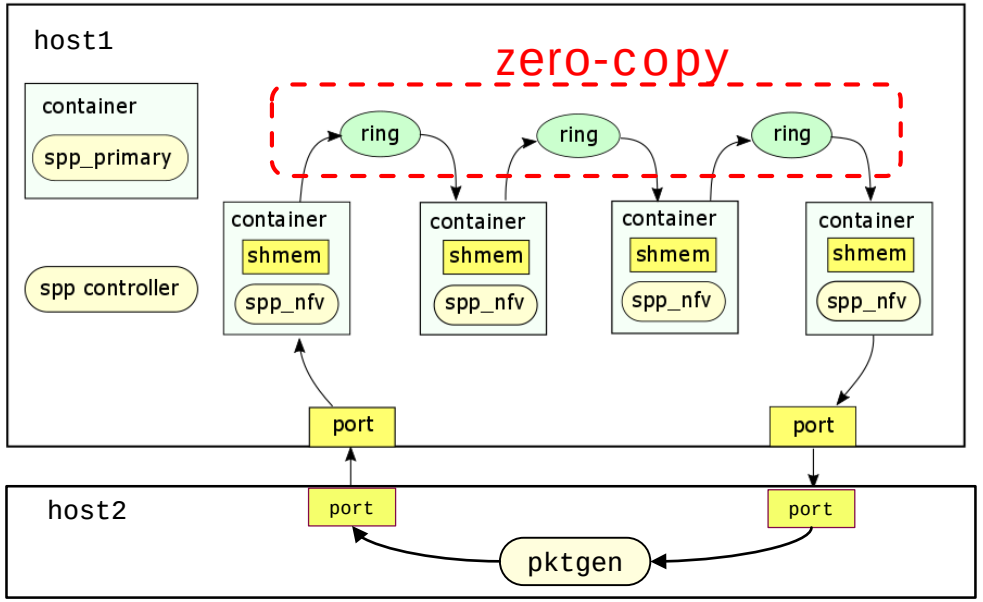
Testing two NFV apps as simple service function chaining

- Totally 7 lcores are required for this usecase
  - One lcore for spp_primary container.
  - Three lcores for four spp_nfv containers.
  - Two lcores for pktgen container.
  - One lcores for l2fwd container.



Usecase 2

High performance packet forwarding via shared memory



	Supermicro Mini Tower Intel Xeon D-1587
CPU	Intel Xeon-D-1587 (1.7 GHz, 32 cores)
Memory	32GB
SSD	Intel SSDSC2BB240G6
OS	Linux (Ubuntu 16.04 LTS)
DPDK	v18.02
pktgen-dpdk	v3.4.9
SPP	v18.02

Limitations

There are several limitations in ‘Virtio_user for Container Networking’

- Cannot work with –huge-unlink option. As we need to reopen the hugepage file to share with vhost backend.
- Cannot work with –no-huge option. Currently, DPDK uses anonymous mapping under this option which cannot be reopened to share with vhost backend.
- Cannot work when there are more than VHOST_MEMORY_MAX_NREGIONS(8) hugepages. If you have more regions (especially when 2MB hugepages are used), the option, –single-file-segments, can help to reduce the number of shared files.
- Applications should not use file name like HUGEFILE_FMT (“%smap_%d”). That will bring confusion when sharing hugepage files with backend by name.
- Root privilege is a must. DPDK resolves physical addresses of hugepages which seems not necessary, and some discussions are going on to remove this restriction.

https://doc.dpdk.org/guides/howto/virtio_user_for_container_networking.html#limitations

Restrictions

- Containerized DPDK apps, including multi-process app work fine with DPDK v18.02

- For v18.05, it works in some of few cases but ...
  - Two or more secondary processes cannot be launched
  - Vhost networking does not work possibly

- Does not work for v18.08

Launch Multiple Secondary Processes

- The reason for several sec process cannot be launched is a change of initializing memseg files.
  - The name of memseg file is defined with PID

```
// lib/librte_eal/linuxapp/eal/eal_memalloc.c
static int
secondary_msl_create_walk(const struct rte_memseg_list *msl,
                void *arg __rte_unused)
{
        struct rte_mem_config *mcfg = rte_eal_get_configuration()->mem_config;
        struct rte_memseg_list *primary_msl, *local_msl;
        char name[PATH_MAX];
        ...
        /* create distinct fbarrays for each secondary */
        snprintf(name, RTE_FBARRAY_NAME_LEN, "%s_%i",
                primary_msl->memseg_arr.name, getpid());
```

  - It cannot ensure unique name because PID is assigned from 1 in each of containers

```
rte_fbarray_init(): couldn't lock /var/run/dpdk/rte/fbarray_memseg-1048576k-0-0_1:
Resource temporarily unavailable
```

  - To avoid this error, need to change to use unique ID for containers

→ I succeeded to launch multiple secondaries, and going to send a patch for DPDK v18.11

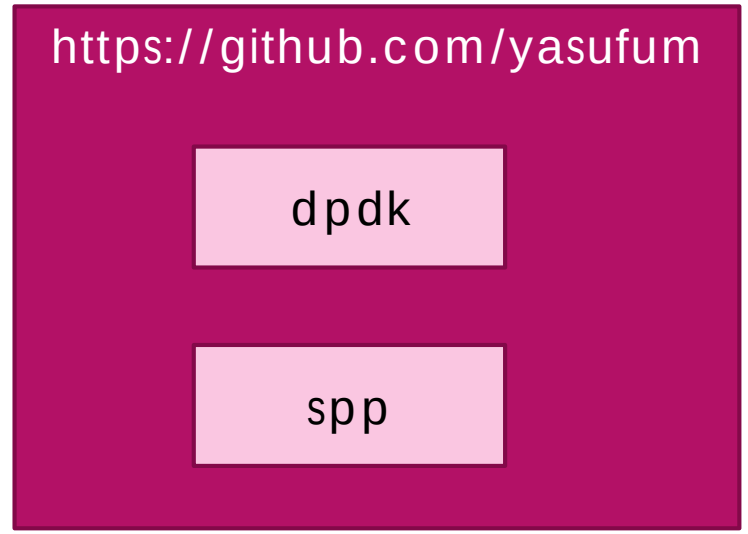
Debugging

- How to debug SPP container

1. Create dev version of DPDK and SPP repos on github
2. Create container image of SPP dev

```
$ python ./build/main.py
-t spp ¥
--dist-ver 16.04 ¥
--dpdk-repo https://github.com/yasufum/dpdk.git
--spp-repo https://github.com/yasufum/spp.git
...
... waiting for long time ...
```

3. Debug and update source code
4. Push the changes to github, and return to the 2nd step



... Need to be improved for more efficient way

Thank you

YASUFUMI OGAWA

ogawa.yasufumi@lab.ntt.co.jp