DPDK

DATA PLANE DEVELOPMENT KIT

Implementation and Testing of Soft Patch Panel

Yasufumi Ogawa (NTT)

Tetsuro Nakamura (NTT)

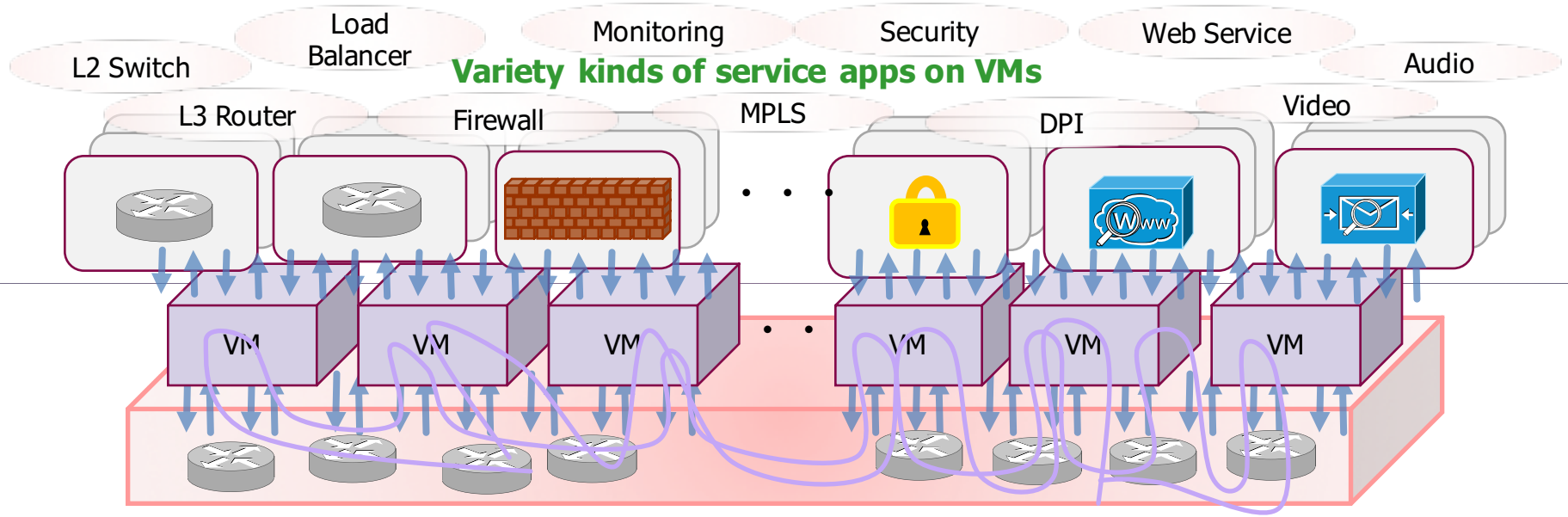
DPDK Summit - San Jose – 2017

Agenda

- Motivation
- SPP (Soft Patch Panel)
  - Design and Network Configuration
  - Implementation
  - Performance Test
- Use-case
  - PoC Implementation
  - Performance Test
- Summary

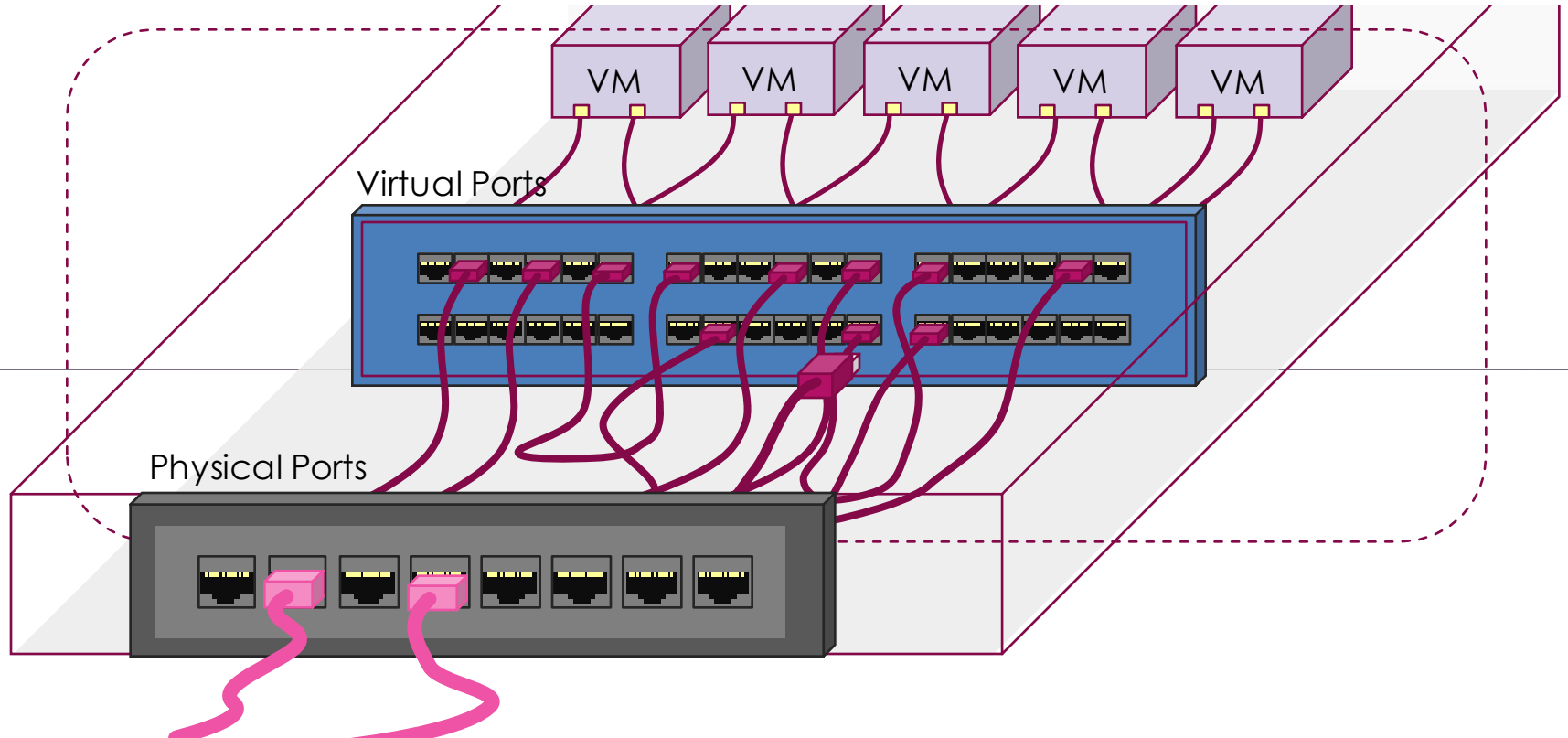
Motivation

- Large-scale cloud for telecom services
- Service Function Chaining for virtual network appliances
- Flexibility, Maintainability and High-Performance



Soft Patch Panel

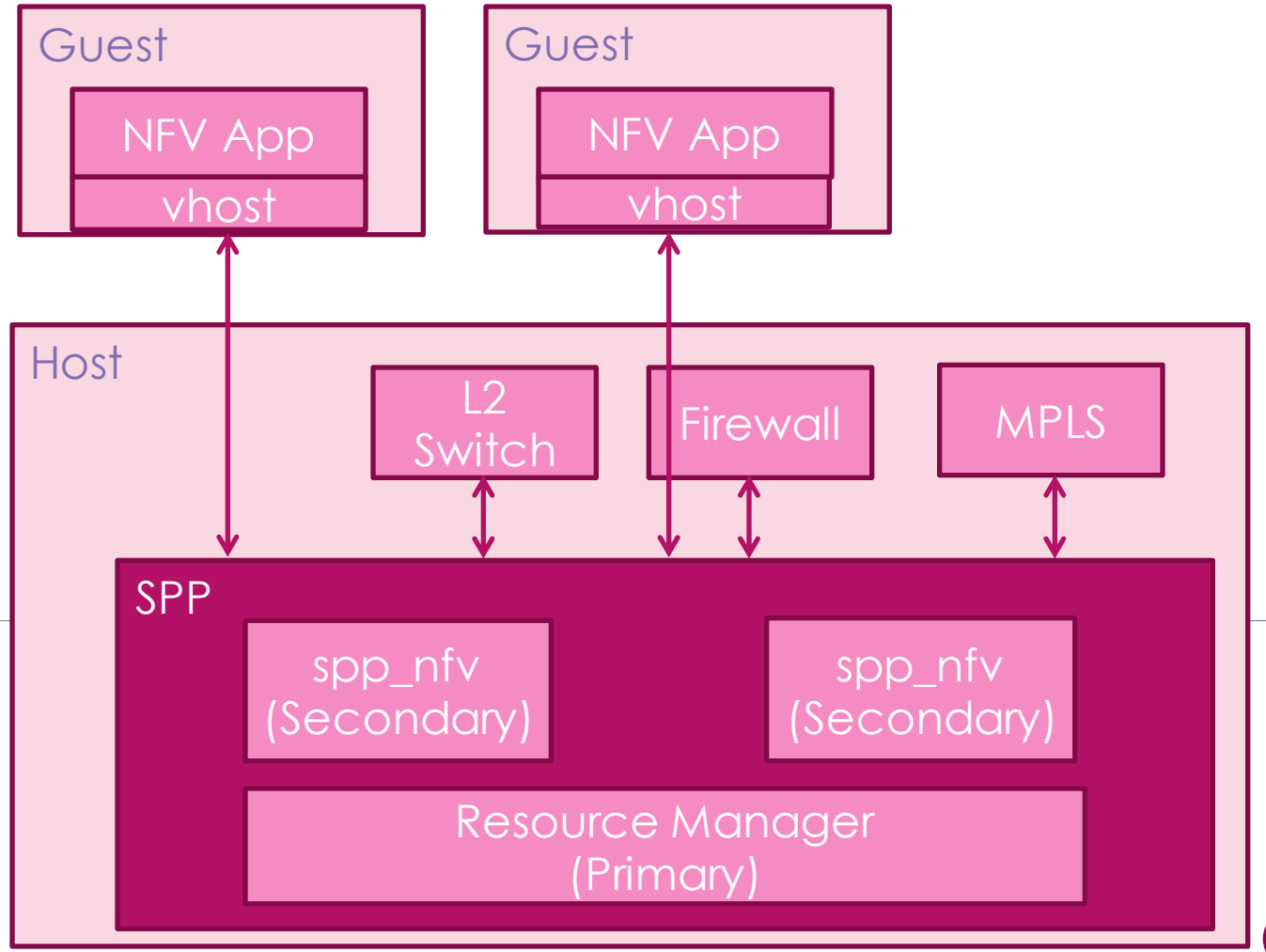
- Change network path with patch panel like simple interface
- High-speed packet processing with DPDK
- Update network configuration dynamically without terminating services



Design

- Multi-process Application
  - Primary process is a resource manager
  - Secondary processes are workers for packet forwarding
- Several Virtual Port Support
  - ring pmd
  - vhost pmd
  - pcap pmd

  etc

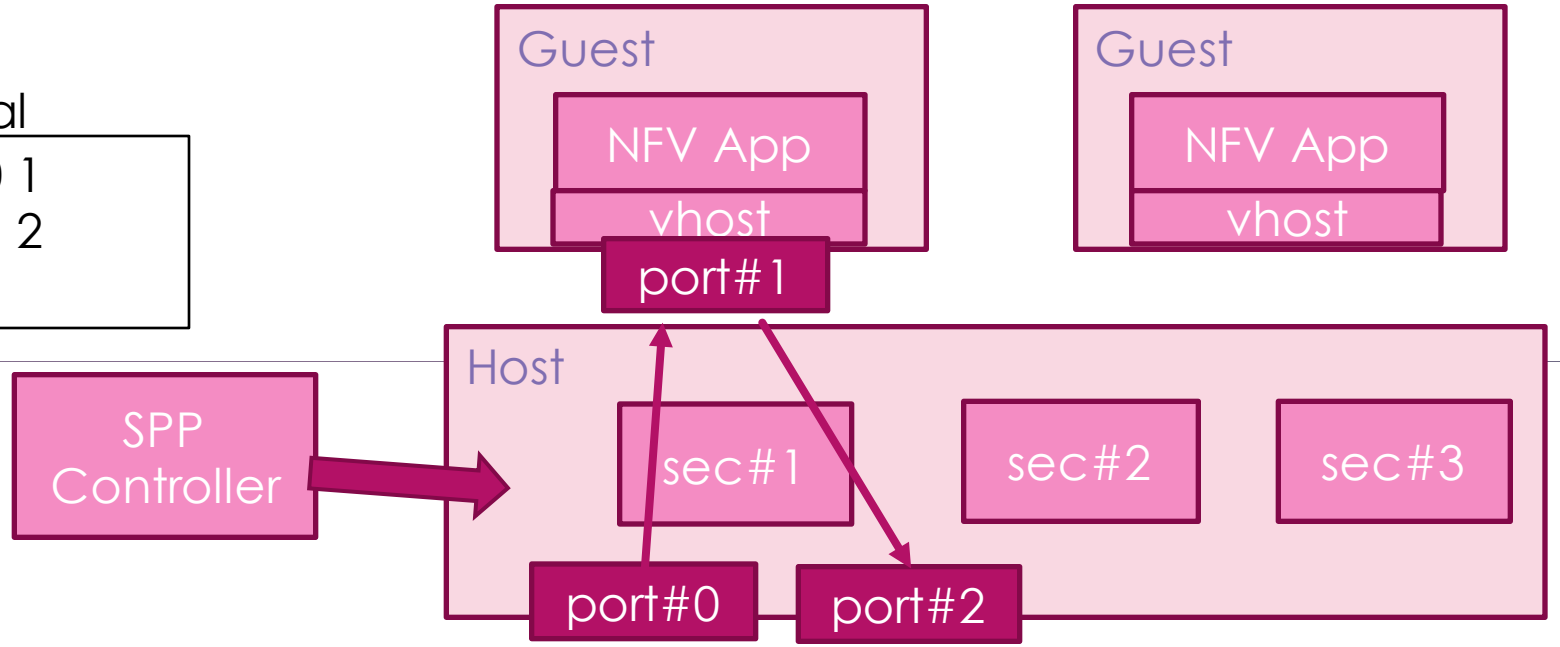


Network Configuration

- SPP Controller manages connection between secondary processes
- Each of secondary has its connection as patch information
- Patch can be updated dynamically while running NFV applications

SPP Controller Terminal

```
spp> sec 1; patch 0 1
spp> sec 1; patch 1 2
spp> sec 1;forward
```

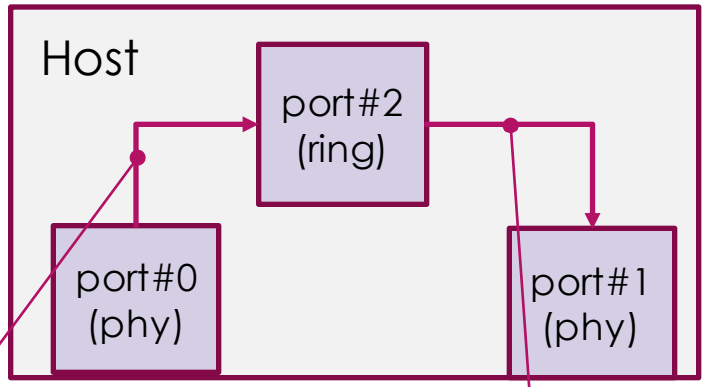


Patch Management

- Each of secondary has patch information and port attributes
- Patch information is managed as an array of port struct, "ports_fwd_array"
- Port ID and type is defined as an array of port_map struct, "port_map"

```
struct port {
  int in_port_id;
  int out_port_id;
  uint16_t (*rx_func)(uint8_t,uint16_t,struct rte_mbuf **,uint16_t);
  uint16_t (*tx_func)(uint8_t,uint16_t,struct rte_mbuf **,uint16_t);
};
```

```
struct port_map {
  int id;
  enum port_type port_type;
  struct stats *stats;
  struct stats default_stats;
};
```




```
ports_fwd_array[0].in_port = -1
ports_fwd_array[0].out_port=2
ports_fwd_array[0].rx_func=rx_burst()
ports_fwd_array[0].tx_func=tx_burst()
```

```
port_map[0].id = 0
port_map[0].port_type = PHY
…
```

```
ports_fwd_array[2].in_port = 2
ports_fwd_array[2].out_port=1
ports_fwd_array[2].rx_func=rx_burst()
ports_fwd_array[2].tx_func=tx_burst()
```

```
port_map[0].id = 2
port_map[0].type = RING
…
```

Port Allocation

ring_pmd

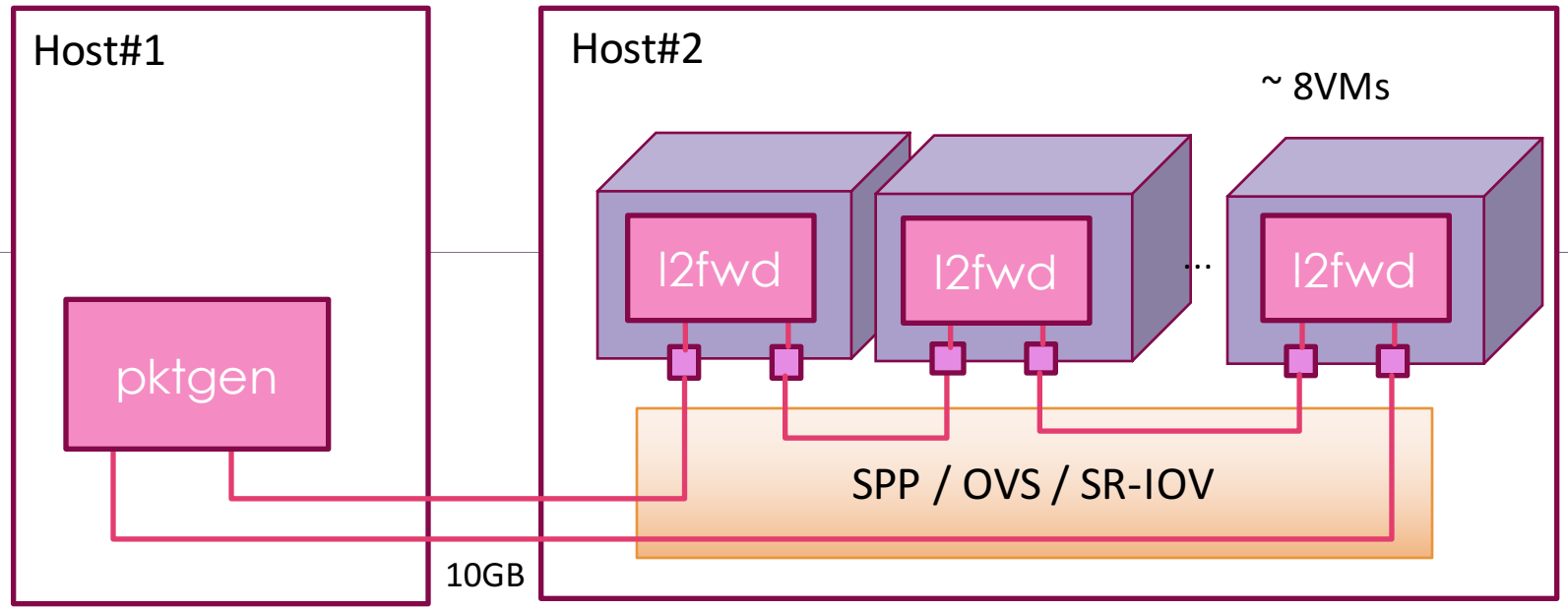
```
414 static int
415 add_ring_pmd(int ring_id)
416 {
417   struct rte_ring *ring;
418   int ring_port_id;
419
420   /* look up ring, based on user's provided id*/
421   ring = rte_ring_lookup(get_rx_queue_name(ring_id));
422   if (ring == NULL) {
423     RTE_LOG(ERR, APP,
424       "Cannot get RX ring - is server process running?\n");
425     return -1;
426   }
427
428   /* create ring pmd*/
429   ring_port_id = rte_eth_from_ring(ring);
430   RTE_LOG(DEBUG, APP, "ring port id %d\n", ring_port_id);
431
432   return ring_port_id;
433 }
```

vhost_pmd

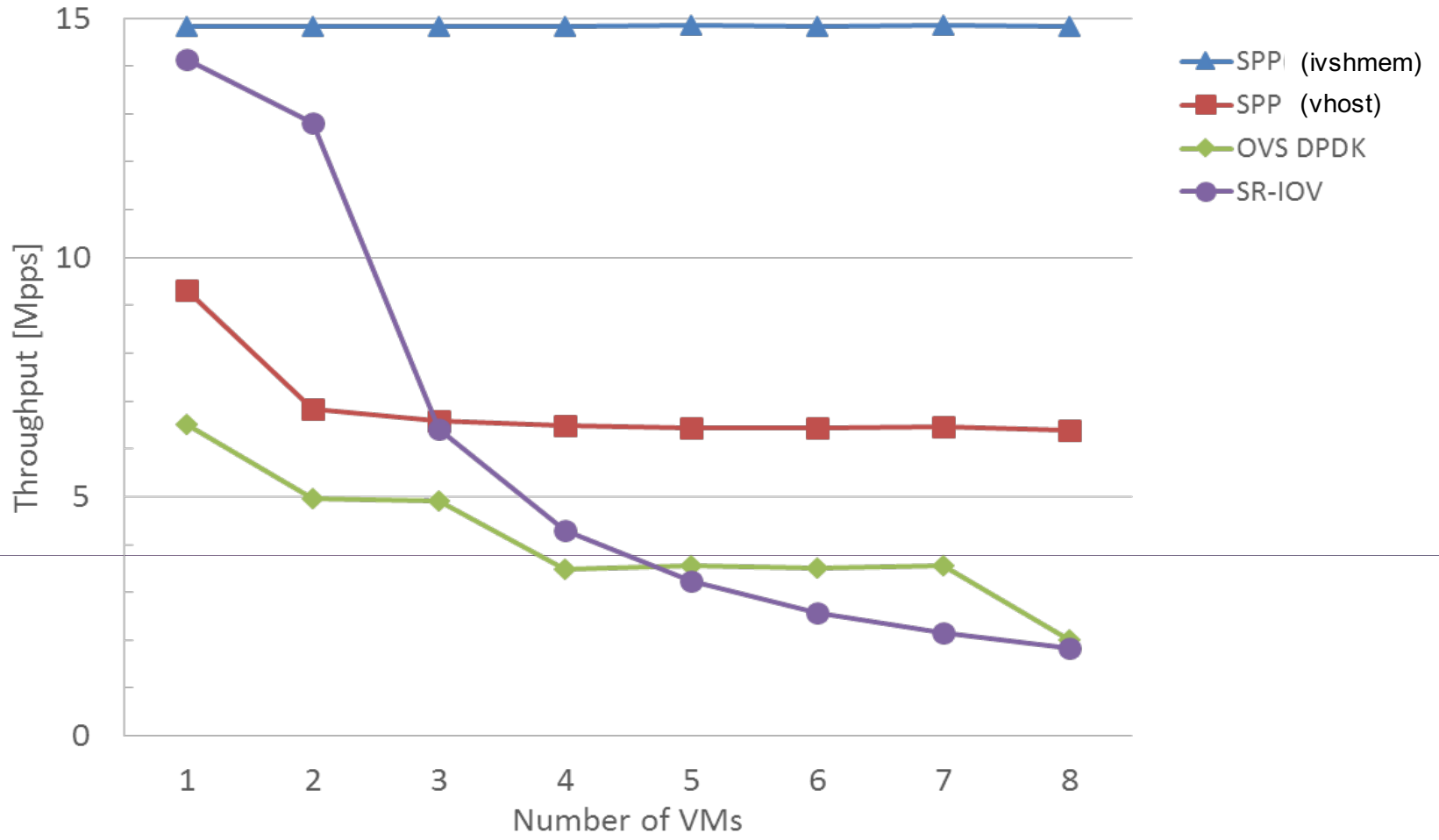
```
435 static int
436 add_vhost_pmd(int index)
437 {
                    :
455   /* eth_vhost0 index 0 iface /tmp/sock0 on numa 0 */
456   name = get_vhost_backend_name(index);
457   iface = get_vhost_iface_name(index);
458
                    :
469   /* Allocate and set up 1 RX queue per Ethernet port. */
470   for (q = 0; q < nr_queues; q++) {
471     ret = rte_eth_rx_queue_setup(vhost_port_id, q, NR_DESCS,
472       rte_eth_dev_socket_id(vhost_port_id), NULL, mp);
473     if (ret < 0)
474       return ret;
475   }
476
477   /* Allocate and set up 1 TX queue per Ethernet port. */
478   for (q = 0; q < nr_queues; q++) {
479     ret = rte_eth_tx_queue_setup(vhost_port_id, q, NR_DESCS,
480       rte_eth_dev_socket_id(vhost_port_id), NULL);
481     if (ret < 0)
482       return ret;
483   }
484
485   /* Start the Ethernet port. */
486   ret = rte_eth_dev_start(vhost_port_id);
487   if (ret < 0)
488     return ret;
489
490   RTE_LOG(DEBUG, APP, "vhost port id %d\n", vhost_port_id);
491
492   return vhost_port_id;
493 }
```

Performance Test

- Compare throughput of SPP with OVS-DPDK and SR-IOV
  - CPU: Xeon E5-2690v3 (12cores/24threads)
  - NIC: Intel X520 DP 10GB DA/SFP+ Server Adapter
  - DPDK v16.07
  - Traffic: 64byte / 10GB
  - Through 1-8 VMs



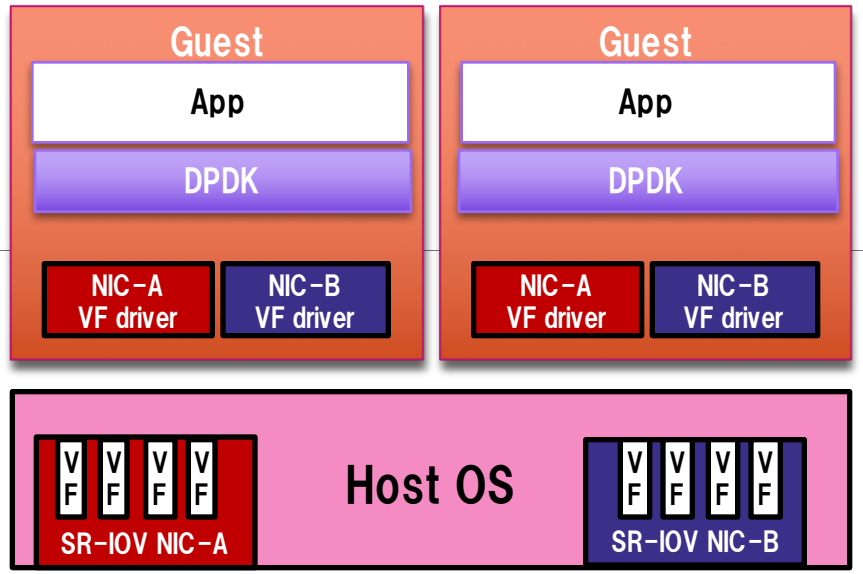
Performance Test



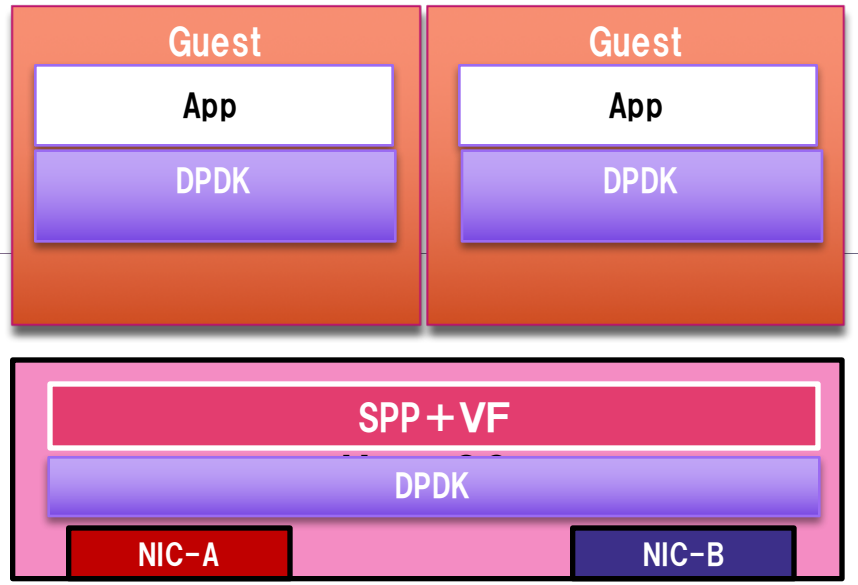
Usecase

- SR-IOV emulation with SPP aiming to improve maintainability
  - Remove dependency for specific hardware
  - Avoid fixed combination of versions of kernel and driver

Before (SR-IOV)

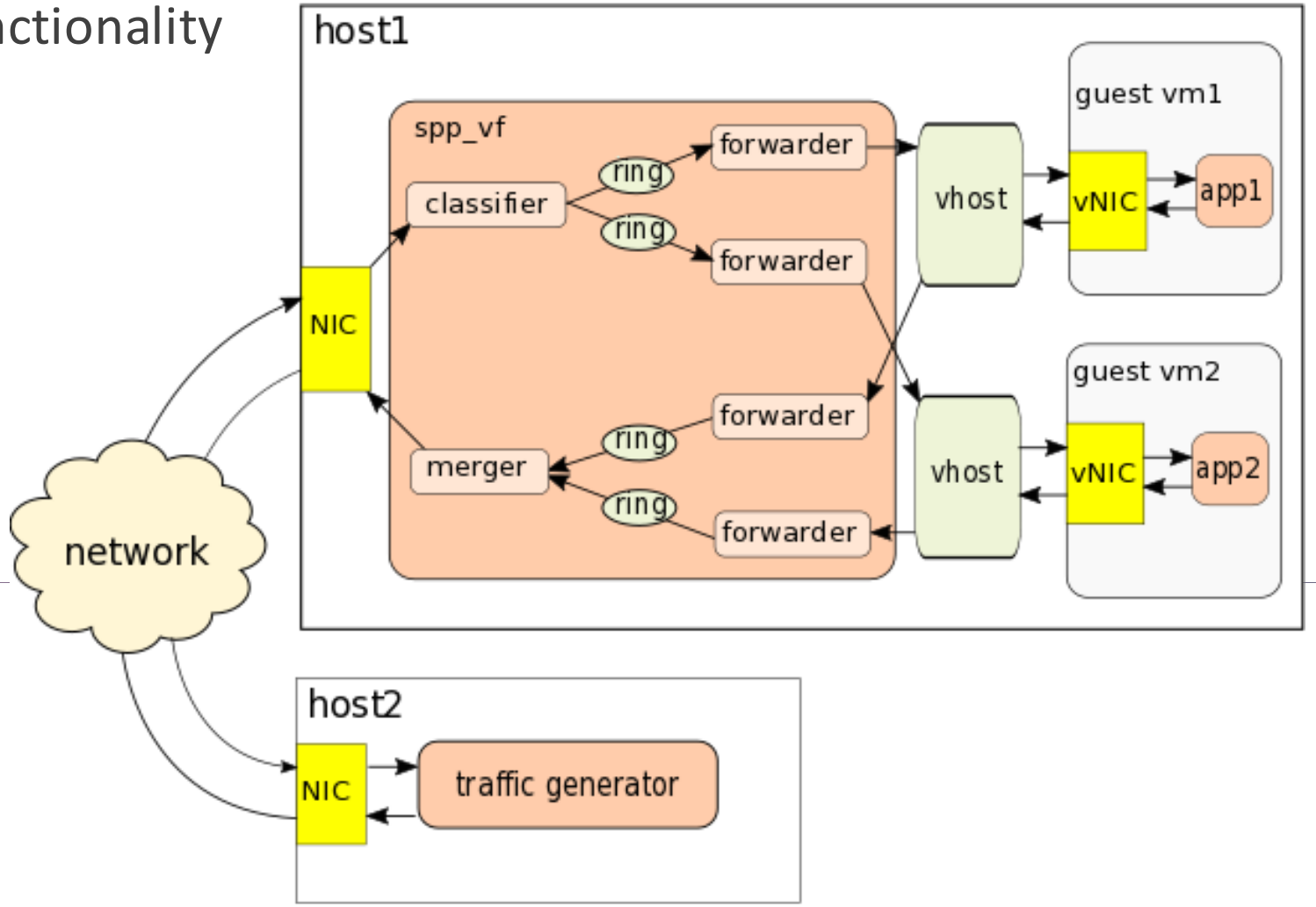


After (SPP)



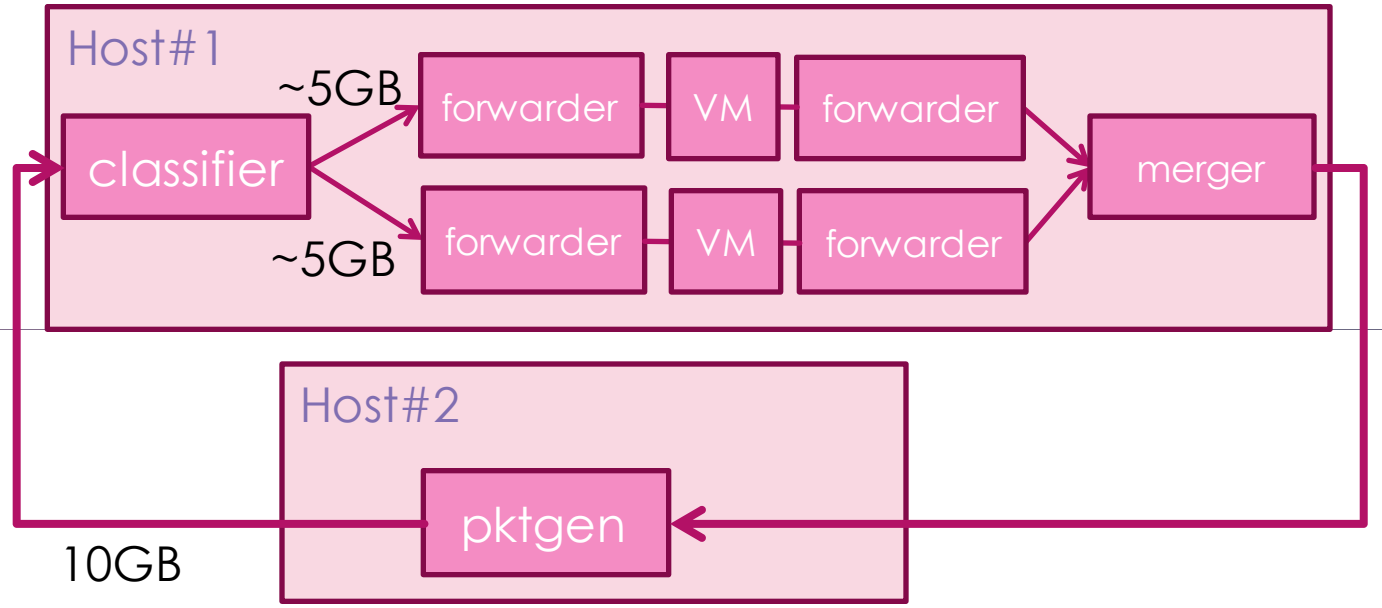
SPP VF

- SPP VF is a pseudo SR-IOV functionality
  - L2 switching
  - MAC address filtering
  - Multicast address
- Three types of workers
  - Forwarder
  - Classifier
  - Merger
- OpenStack Plugin Support
  - Neutron ML2 plugin

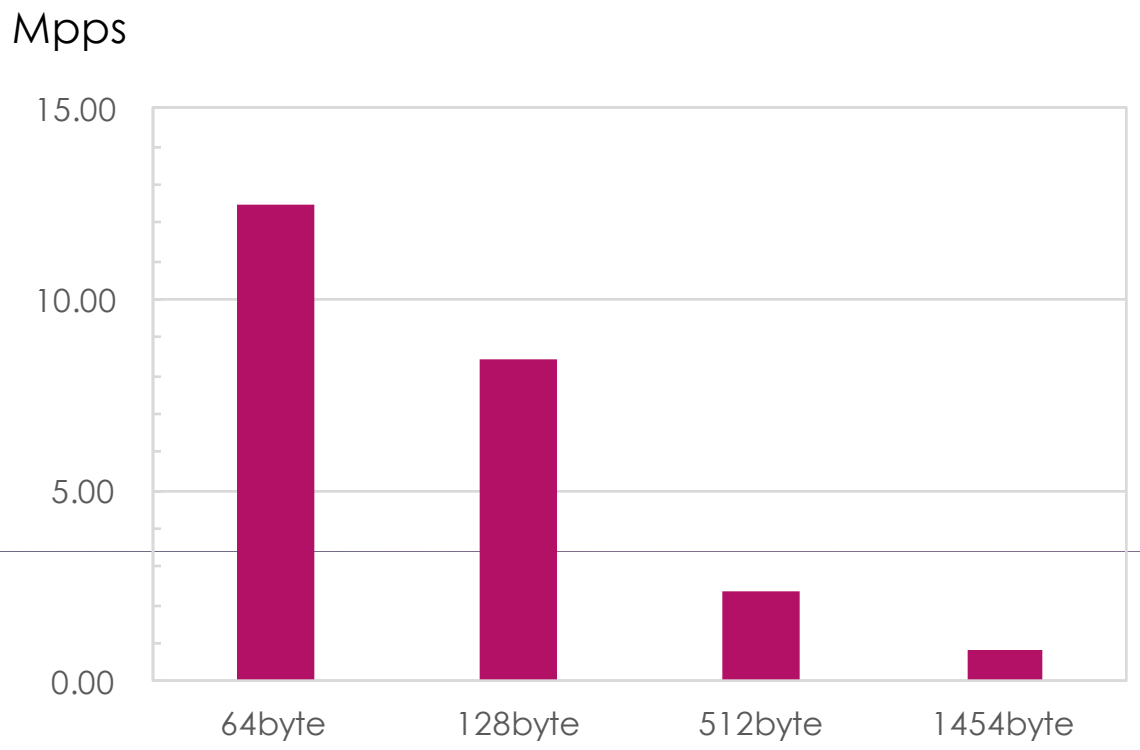


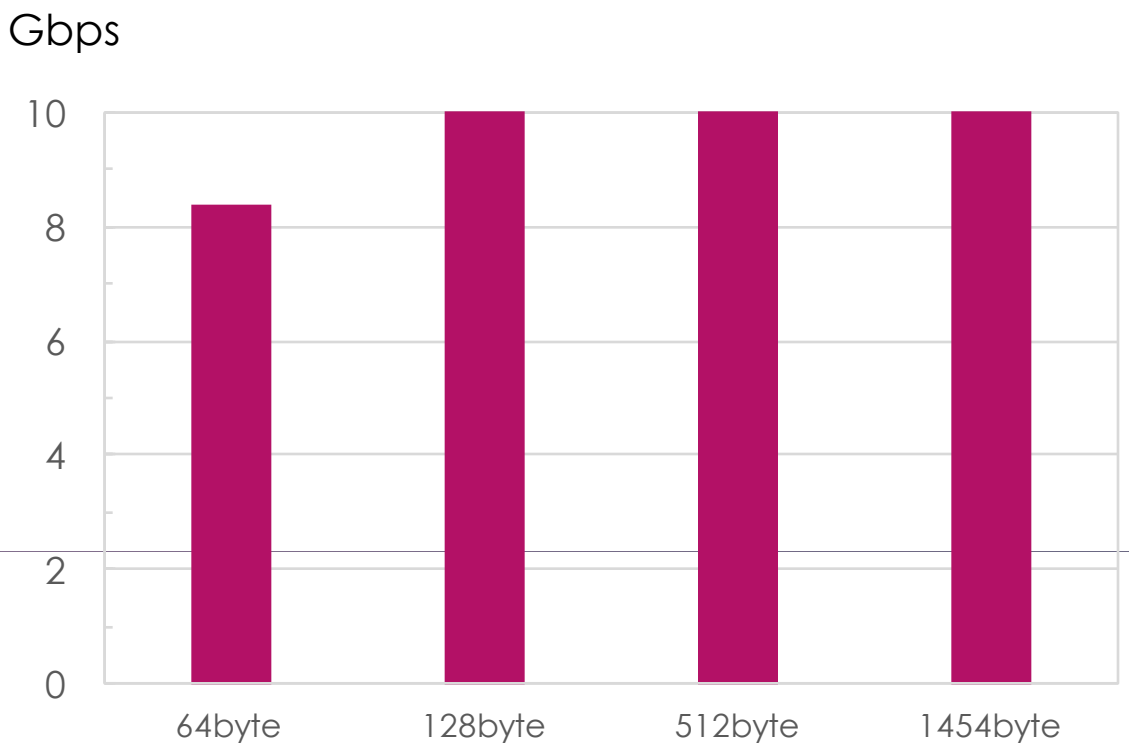
Performance Test of SPP VF

- Benchmarking for classification of MAC addressing
  - CPU: Xeon E5-2690v3 (12cores/24threads)
  - NIC: Intel X520 DP 10GB DA/SFP+ Server Adapter
  - DPDK v17.08
  - Traffic: 64 - 1454byte / 10GB



Performance Test of SPP VF





Summary

- SPP is a framework for Service Function Chaining for telecom services
- SPP consists of Resource Manager and Workers for updating network configuration dynamically
- SPP VF and OpenStack ML2 plugin to reduce complexity of maintenance

SPP

http://dpdk.org/browse/apps/spp/

SPP VF + OpenStack plugin (planed to be merged to SPP repo)

https://github.com/ntt-ns/Soft-Patch-Panel

Questions?

Yasufumi Ogawa (NTT)

<ogawa.yasufumi@lab.ntt.co.jp>

Tetsuro Nakamura (NTT)

<tetsuro.nakamura@lab.ntt.co.jp>