DPDK

DATA PLANE DEVELOPMENT KIT

Tungsten Fabric Optimization by DPDK

ZHAOYAN CHEN

YIPENG WANG

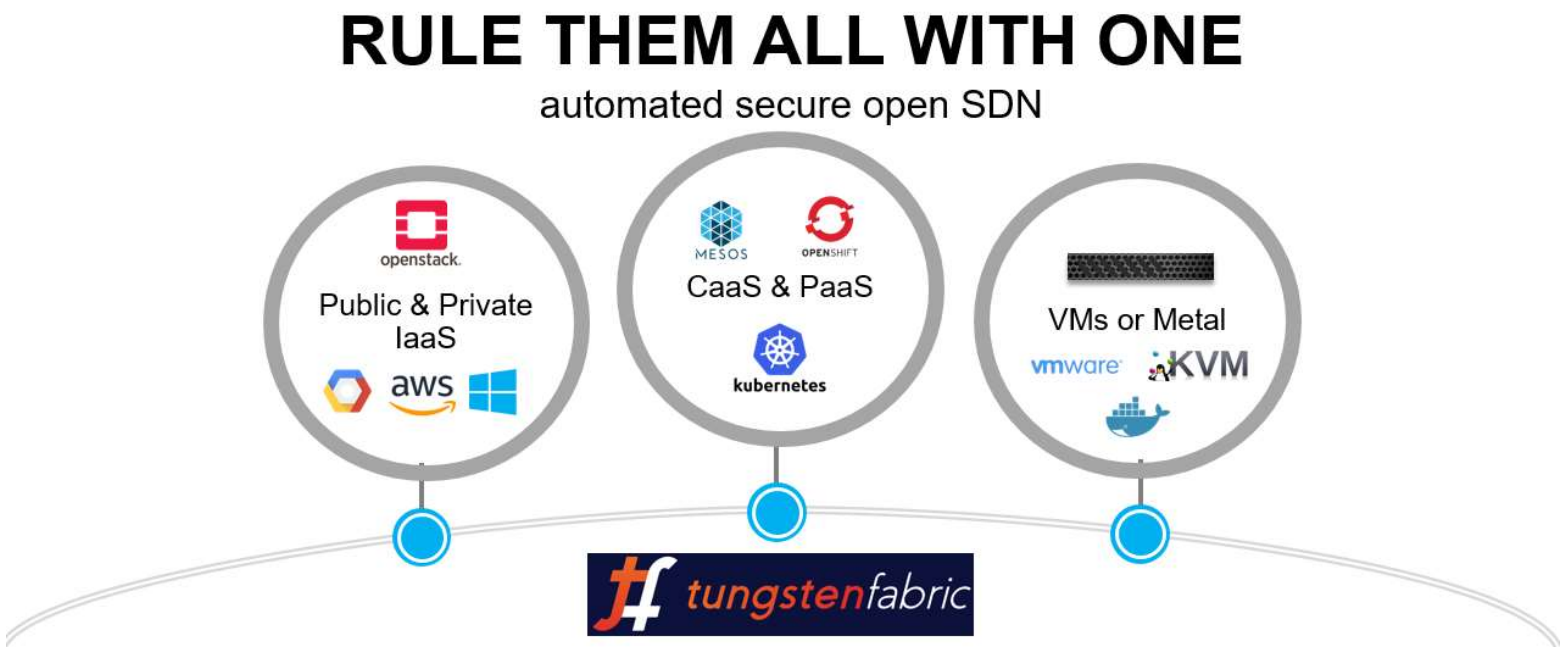
DPDK
DATA PLANE DEVELOPMENT KIT

Agenda

- Introduce Tungsten Fabric
- Support More CPU cores
- MPLS over GRE Optimization
- Hash Table Optimization
- Batch RX for VM and Fabric

What is Tungsten Fabric

- Tungsten Fabric is an open source multi-cloud, multi-stack software-defined networking (SDN) platform, https://tungsten.io
- Previous name is OpenContrail
- All-in-one solution: Controller + vRouter + configuration + analysis + orchestration interface



Tungsten Fabric – Where is DPDK

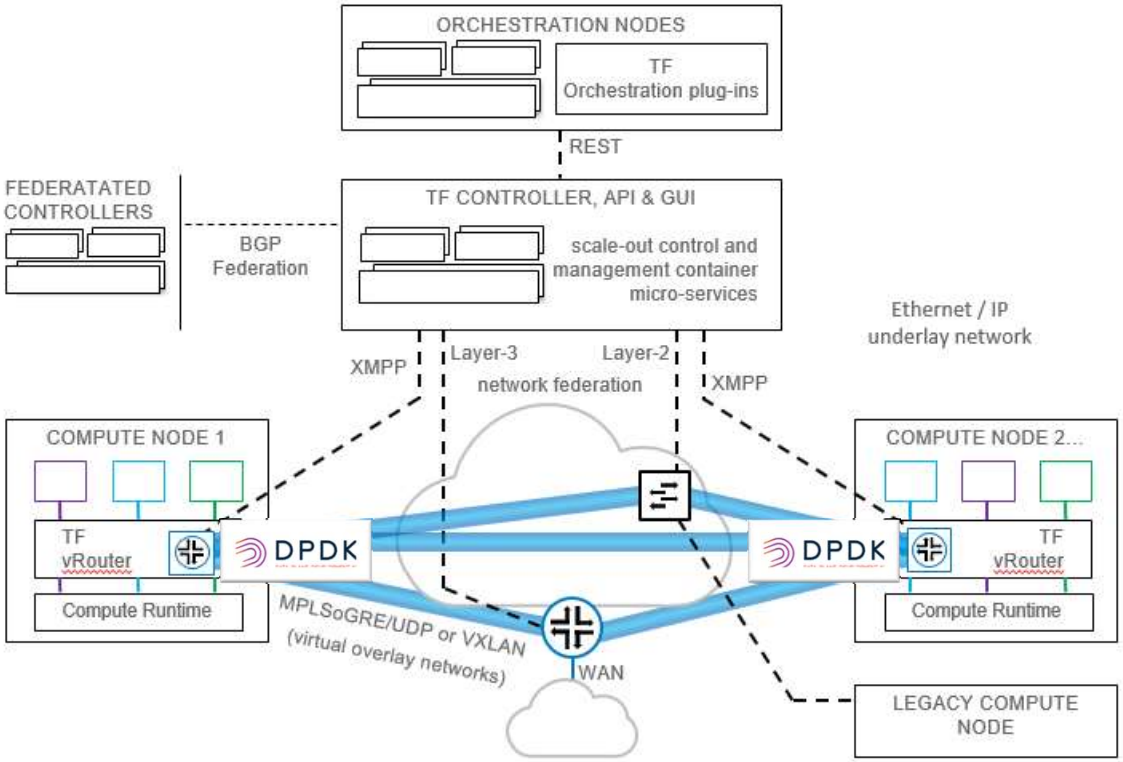
Tungsten Fabric = Controller + vRouters

Interfaces:

- a set of north-bound (REST API's)
- south-bound interfaces (XMPP, BGP+Netconf)
- an east-west interface (BGP)

vRouter-dpdk

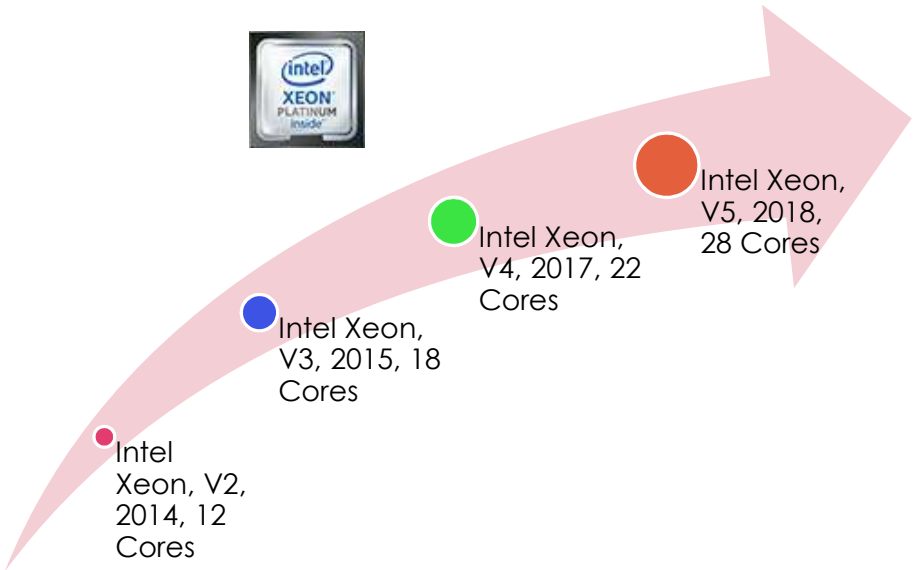
(Tungsten Fabric r5.0.1 integrated DPDK 17.11.3)



Support more CPU cores

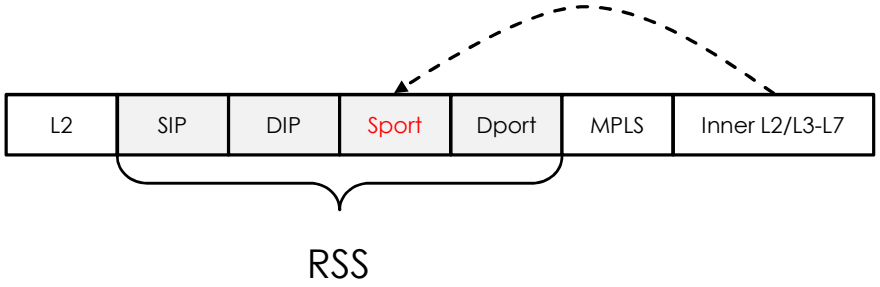
- Increase the limitation of core number for vRouter-DPDK
- ~30% performance improve (16c vs 11c)
- Improve the maximum value of core number
- CPU on NUMA node 1 can be used

	Previous	**Now**
Cores for vRouter	11	**16**
Max value of Core No.	64	**1024**

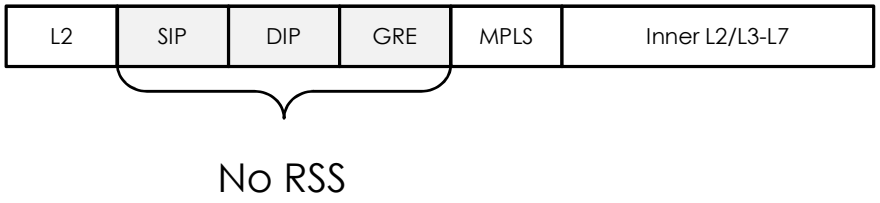


MPLSoUDP and MPLSoGRE

- Tungsten Fabric supports MPLSoUDP, MPLSoGRE, and VxLan overlay network
- MPLS over UDP

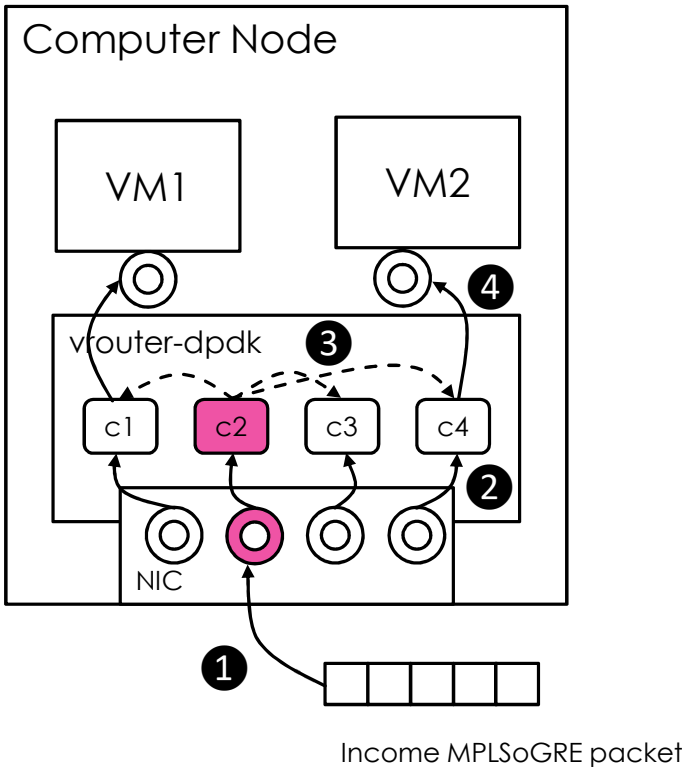


- MPLS over GRE



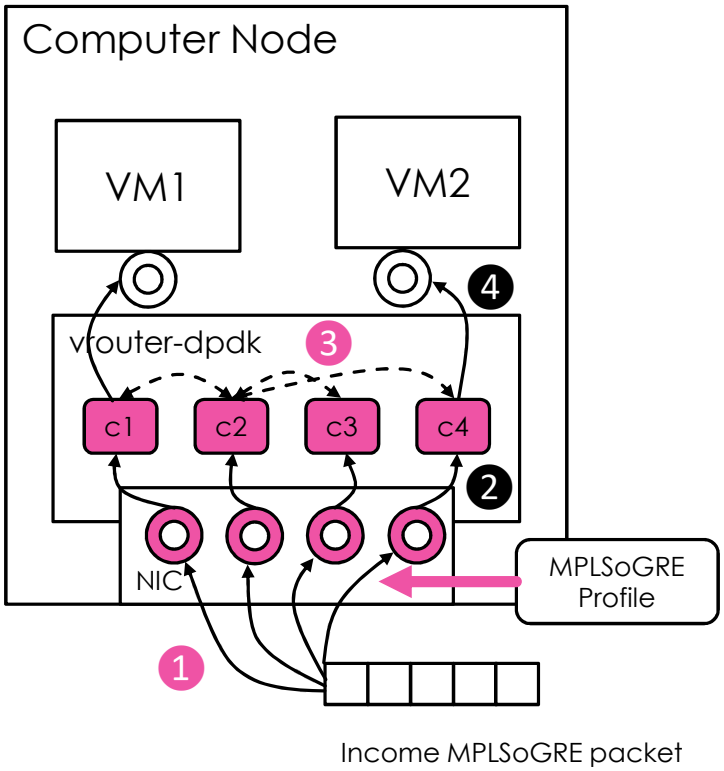
vRotuer Workflow for MPLSoGRE

- Income MPLSoGRE packets go to only 1 hardware RX queue
- vRouter binds each lcore with hardware queue, in this case, only 1 lcore deals with income packet
- vRouter has a distributing mechanism, calculate the RSS by inner packet and distribute the packet to other lcore for transmitting.
- Transmit the packet to VMs



MPLSoGRE Classification

- With NIC supporting MPLSoGRE classification, then vRouter workflow ❶ and ❸ could be optimized
- Dynamic Device Personalization (DDP) profiles for Intel® Ethernet 700 Series is one option for advanced packet classification.
- Based on DDP, we got **~20%** RX performance gain for MPLSoGRE packet



For more info related to DDP, please refer this link:

- https://software.intel.com/en-us/articles/dynamic-device-personalization-for-intel-ethernet-700-series

Flow table optimization

- A linked list based hash table (htable) is used to implement the flow table in TF.

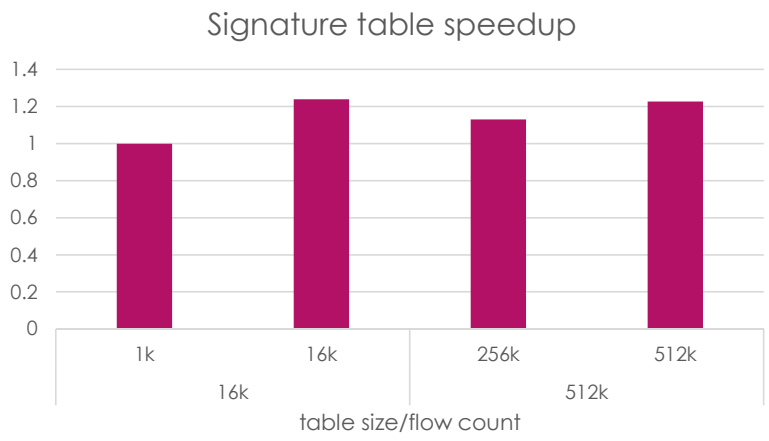
- Lookup Bottlenecks
  - There is no advanced algorithms to reduce key collision in main table: e.g. cuckoo hash used in both OVS and VPP.
  - Linked list traversal when overflow the main table. When this happens, table lookup could take ~50% total forwarding time.
  - No bulk lookup to hide memory access latency.
  - Scalar key comparison is slow for misses.
  - Multicore competing for shared variables.

Hash table with linked list for flow lookup

Function Stack	CPU Time: Total
core_fwd_rxtx	96.9%
_lcore_rxqs_vroute	96.9%
_dpdk_lcore_vroute	92.5%
eth_rx	90.0%
vr_fabric_input	88.0%
vr_l3_input	85.3%
vr_ip_input	65.3%
vr_flow_forward	64.4%
vr_do_flow_lookup	63.8%
vr_inet_flow_lookup	63.6%
vr_flow_lookup	56.2%
vr_find_flow	46.3%
vr_htable_find_hentry	45.5%

Flow table optimization (cont.)

- Optimizations for DPDK path – Borrow the goodness from DPDK/OVS flow table design:
  - Add signature table -> eliminates full key comparison for misses.
  - Software pipelining to hide memory latency with batching processing (requires batch processing code path).
  - Double the number of entries per bucket to reduce the hash collisions -> less linked list traversal.
  - Using Intel® AVX to vectorize signature comparison, using CRC32 for hash value calculation (requires CPU ID check).
  - Using more advanced hashing algorithm, e.g. replace htable with DPDK rte_hash (requires heavily modifying the datapath).



- POC results (with simple IP packet forwarding):
  - signature table + larger bucket size= up to ~20% speedup
  - Using x86 CRC32 instruction to replace software hash = up to ~6% speedup
  - using rte_hash to replace htable (future work).

Batching processing (WIP)

- OVS/DPDK/VPP all use batching processing:
  - Reduce the number of function calls
  - Prefetch next packets while processing the current ones
  - Better icache locality
  - Take advantage of Intel AVX instruction
- Batching for vrouter DPDK data path:
  - Keep the original function API
  - Extract original function to be inline functions
  - Create a bulk version for these functions
  - Enable these bulk* functions in vrouter-dpdk, and enable inline functions for vrouter kernel path
- Expected ~10% end-to-end performance gain

Pseudo code:

```
vm_rx_bulk(struct vr_interface *vif, struct vr_packet **pkts,
        unsigned short *vlan_ids, uint32_t n)
{
    ....
for (i = 0; i < n - 1; i++) {

        /* Prefetch next packet */
        rte_prefetch0(&fmds[i + 1]);
        rte_prefetch0(pkts[i + 1]);
        rte_prefetch0(pkt_data(pkts[i + 1]) - 64);
        rte_prefetch0(pkt_data(pkts[i + 1]));

        /* Process current packet */
        ret = vm_rx_inline(vif, pkt[i], vlan_ids[i], fmds[i]);
        if (ret != 0)
            continue;

        /* Prepare packet array for next stage */
        new_fmds[k] = fmd;
        new_pkts[k] = pkt;
        new_vlan_ids[k] = vlan_id;
        k++;
    }
    ....
}
```

DPDK
DATA PLANE DEVELOPMENT KIT

Thank You