Flow Bifurcation on Intel® Ethernet Controller X710/XL710

Jingjing Wu; Anjali Singhai
DPDK Summit Userspace - Dublin- 2016
Agenda

- Background -- DPDK co-work with Kernel stack
- Flow bifurcation on Intel® Ethernet Controller X710/XL710
- Summary
Kernel Bridging vs. L2Fwd

Kernel bridge throughput is much worse than DPDK L2fwd when processing small packets even the stack doesn't scale.

Throughput

- Kernel bridge with 1 core
- Kernel bridge with 8 cores
- DPDK L2fwd with 1 core
• DPDK is known to build the high performing data plane workload.

• A real world packet processing workload often relies heavily on the Linux kernel and its large stack for the control plane design and implementation. As a known limit, Linux performance is not sufficient for high speed data plane workloads.

• DPDK PMD or kernel driver take over the whole network card, not allowing any traffic on that NIC to reach each other.

• In order to combine the advantages of both, few key technical components are used to achieve the interworking between DPDK and Linux.
  • Exception path: TAP, KNI, AF_Packet
  • A high speed data traffic direction into Linux Kernel and DPDK -- Flow Bifurcation.
Data traffic direction – queue split

User space
- Legacy Network App.
- Socket Lib
- DPK Lib and App.

Kernel space
- TCP/IP Stack
- NIC Kernel Driver
- DPK PMD
- UIO Framework
- UIO Driver

Queue split

NIC

Ingress Traffic
Flow Bifurcation

- SRIOV Based
- Queue split
- Hardware’s Packet classification filtering capability
- kernel driver + DPDK
- Flow director in Intel 82599
- Cloud filter in Intel® X710/XL710
Packet classification filtering on X710/XL710

- To VSI
  - Internal switch filters
- To Queue
  - Ethertype Queue filter
  - Flow director filter
  - MAC/VLAN Queue filter
  - Hash(RSS) filter
Internal Switch - VEB on X710/XL710

- Virtual Ethernet Bridge with Cloud Support (Cloud VEB)

- Cloud VEB Switching Rules
  - Priority 1 filters
  - Priority 2 filters
  - Priority 3 filters

![Diagram of VEB with VMM, PFs, VFIs, and VMs]

- Disallow LOOPBACK: this port won't be allowed to send packets to other virtual ports

- Define which egress ports (VSIs and LAN) will receive a packet received by the VEB.
Priority 1 filters ():
- {Ethertype}
- {MAC, Ethertype}

Priority 2 filters (Cloud Filters):
- {Inner MAC, Inner VLAN}
- {Inner MAC, Inner VLAN, Tenant ID}
- {Inner MAC, Tenant ID}
- {Inner MAC}
- {Outer MAC, Tenant ID, Inner MAC}
- {Inner IP}
- {Inner Source IP, inner destination MAC}

Priority 3 filters:
- {MAC, VLAN}
- {MAC}
- {VLAN}

Control filters: filtering control frame

Cloud filters: used for flow Bifurcation, can be programmed through ethtool

L2 filters: traditional filtering by mac address and VLAN, programmed when mac address or VLAN assigned to device
I40e driver programs classification rule configured by Flow Director typically. But Flow director in i40e filters packets in scope of VSI.
Adapt to Ethtool classification

- If the upper 32 bits of ‘user-def’ are 0xffffffff, then the filter can be used for programming an L3 VEB filter, otherwise the upper 32 bits of ‘user-def’ can carry the tenant ID/VNI if specified/required.

- Cloud filters can be defined with inner mac, outer mac, inner ip, inner vlan and VNI as part of the cloud tuple. It is always the destination (not source) mac/ip that these filters use. For all these examples dst and src mac address fields are overloaded dst == outer, src == inner.

- The filter will direct a packet matching the rule to a vf specified in the lower 32 bits of user-def to the queue specified by ‘action’.

- If the vf id specified by the lower 32 bits of user-def is greater than or equal to max_vfs, then the filter is for the PF queues.
# Create Virtual Functions:
    echo 2 > /sys/bus/pci/devices/0000:01:00.0/sriov_numvfs

# Add udp port offload to the NIC if using cloud filter:
    ip li add vxlan0 type vxlan id 1 group 239.1.1.1 local 127.0.0.1 dev <name>
    ifconfig vxlan0 up

# Enable and setup rules
- Route whose destination IP is 192.168.50.108 to VF 0's queue 0:
    ethtool -N <dev_name> flow-type ip4 dst-ip 192.168.50.108 user-def 0xffffffff00000000 action 0 loc 0
- Route whose inner destination mac is 0:0:0:0:9:0 and VNI is 8 to PF's queue 1:
    ethtool -N <dev_name> flow-type ether dst 00:00:00:00:00:00 m ff:ff:ff:ff:ff \ 
    src 00:00:00:00:09:00 m 00:00:00:00:00:00 user-def 0x80000003 action 1 loc 1
- ......

# start DPDK application without interrupt net device
    testpmd -c 0xff -n 4 -- -i -w 01:10.0 -w 01:10.1 --forward-mode=mac
Platform

- Kernel version: 4.5.5-300.fc24.x86_64
- I40e driver: 1.5.23
- Firmware-version: 5.04
- DPDK: 16.07
- Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz
- Intel® Ethernet Controller XL710 for 40GbE QSFP+ (PCIe Gen 3 x 8)

Mixed traffic flows

- flow_1: IP packets with destination IP address is 192.168.50.109 → kernel bridge
- flow_2: IP packets with destination IP address is 192.168.50.108 → DPDK l2fwd
<table>
<thead>
<tr>
<th>Mixed traffic</th>
<th>Flow1 vs flow 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile_1</td>
<td>100% vs 0</td>
</tr>
<tr>
<td>Profile_2</td>
<td>10% vs 90%</td>
</tr>
<tr>
<td>Profile_3</td>
<td>2% vs 98%</td>
</tr>
<tr>
<td>Profile_4</td>
<td>0 vs 100%</td>
</tr>
</tbody>
</table>

![Flow bifurcation performance measurement graph](image-url)
Advantages

- Support control interface, such as ethtool on PF.
- Flows are split on HW. Without overload, DPDK application’s performance can keep stable.
- Only need kernel driver to enable filters, no DPDK changes are required, and no out-of-tree module is required.
- Security protected by SRIOV and IOMMU.

Disadvantages

- Depends on Hardware’s Packet classification filtering capability. Different NIC has limited filtering capability. Not flexible as SW filtering.
- Is not absolute queue split, depends on PF driver’s supporting.
Questions?

Jingjing Wu
jingjing.wu@intel.com