DPDK

DATA PLANE DEVELOPMENT KIT

Userspace 2015 | Dublin

DPDK Packet Framework

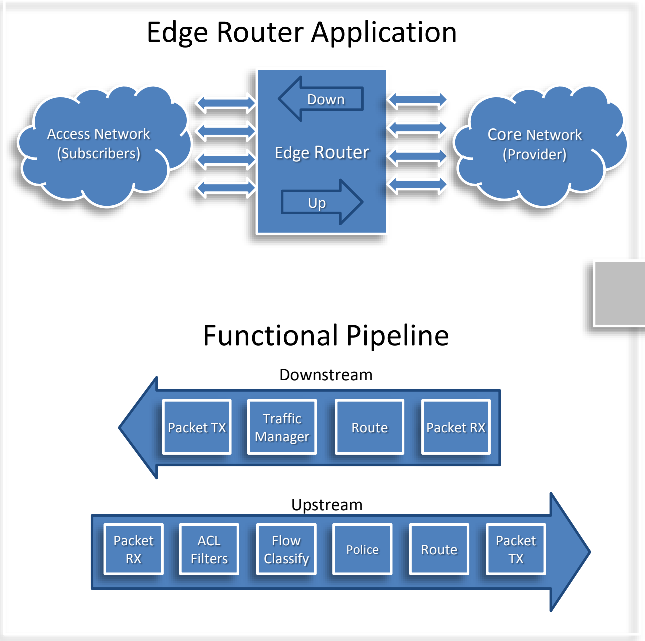
Cristian Dumitrescu, SW Architect at Intel

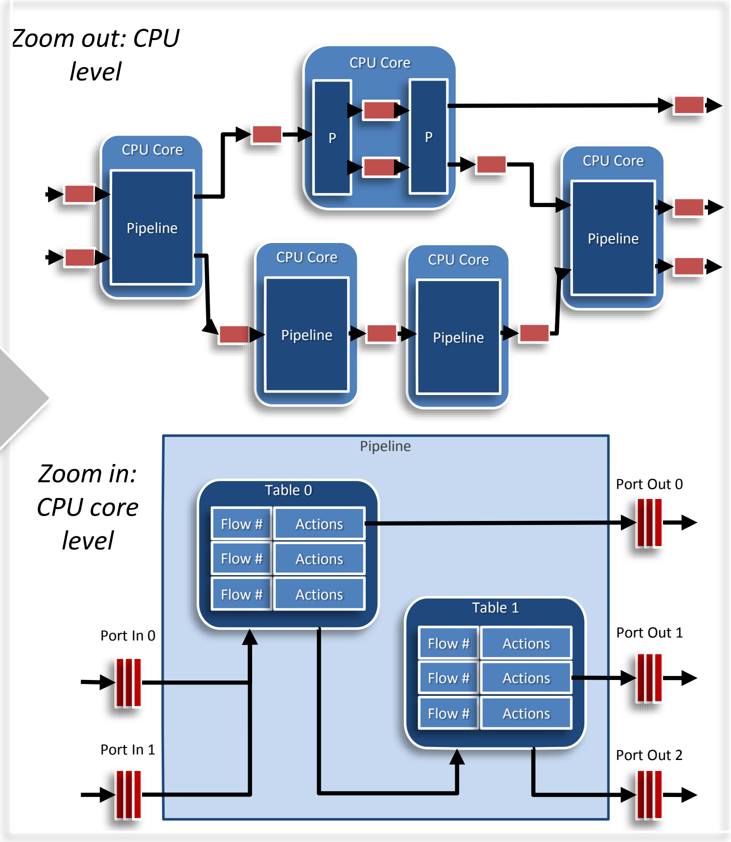
Agenda

1. Motivation
2. DPDK Packet Framework Libraries: librte_port, librte_table, librte_pipeline
3. Application Generator: ip_pipeline

Rapid Development

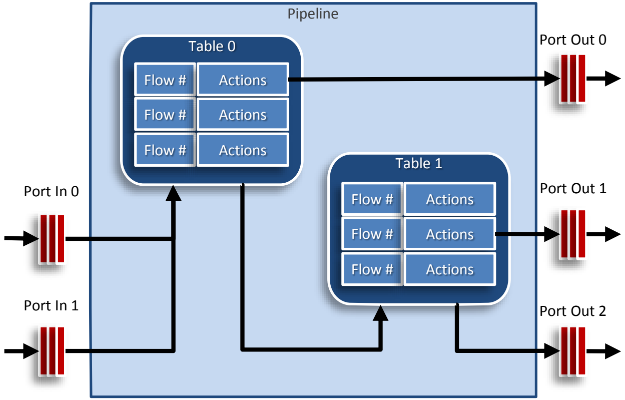
DPDK Packet Framework quickly turns requirements into code



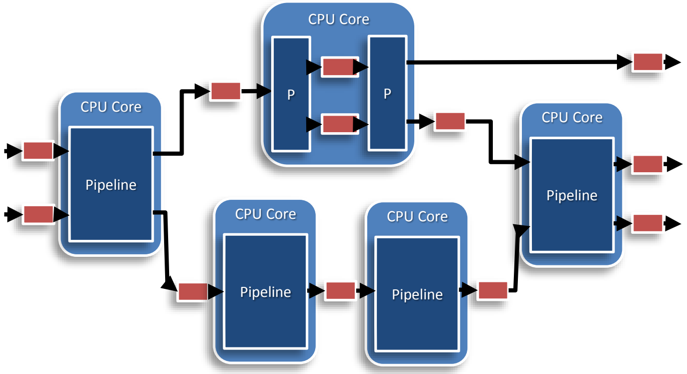


DPDK Packet Framework

Zoom in: Pipeline level



Zoom out: Multi-core application level



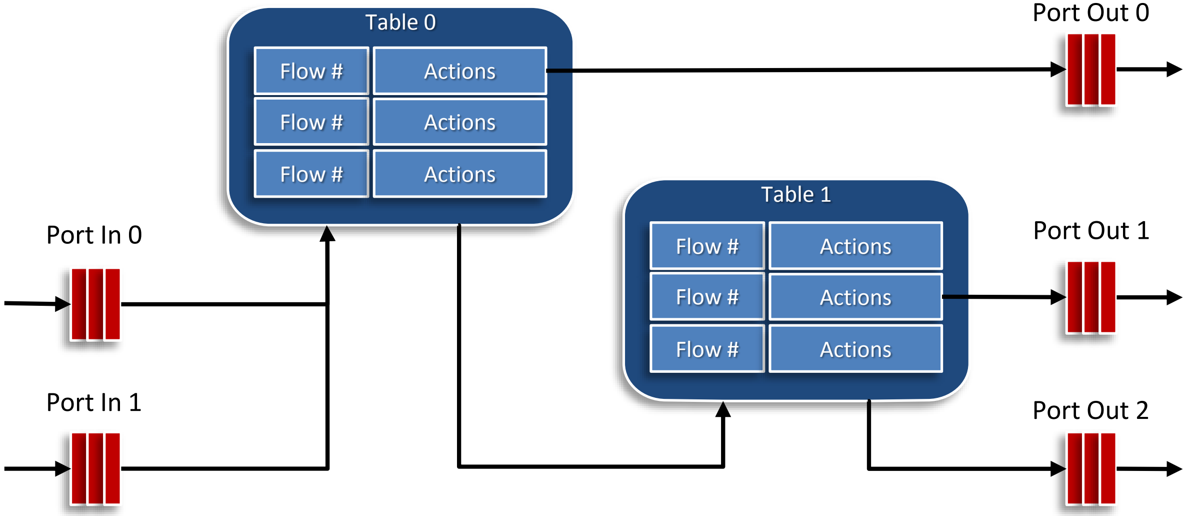
Ports
HW queue
SW queue
IP Fragmentation
IP Reassembly
Traffic Manager
Kernel Network I/F (KNI)
Source/Sink

Tables
Exact Match / Hash
Access Control List (ACL)
Longest Prefix Match (LPM)
Array
Pattern Matching

Actions
Reserved actions: Send to port, Send to table, Drop
Packet edits: push/pop/modify headers
Flow-based: meter, stats, app ID
Accelerators: crypto, compress
Load Balancing

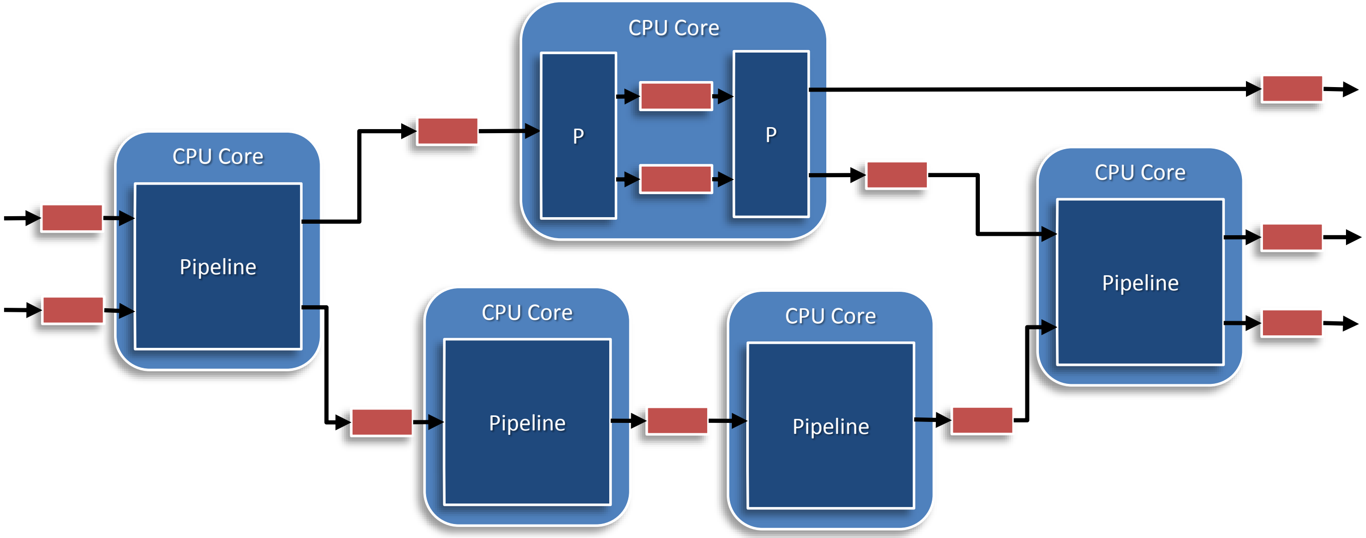
Pipelines
Packet I/O
Flow Classification
Firewall
Routing
Metering
Traffic Mgmt

CPU Core Level (Pipeline)



Rapid ***pipeline*** development out of ***ports***, ***tables*** and ***actions*** based on Open Flow inspired methodology

CPU Level (Application)



Application is made up of multiple pipelines connected together. Several pipelines can be mapped to the same CPU core.

ip_pipeline

Configuration file:

- Defines the application structure by gluing together all pipeline instances. By using different configuration files, different applications are generated
- All the application resources are created and configured through it
- Syntax is “define by reference”: first time a resource name is detected, it is registered with default parameters, which can be refined through dedicated section

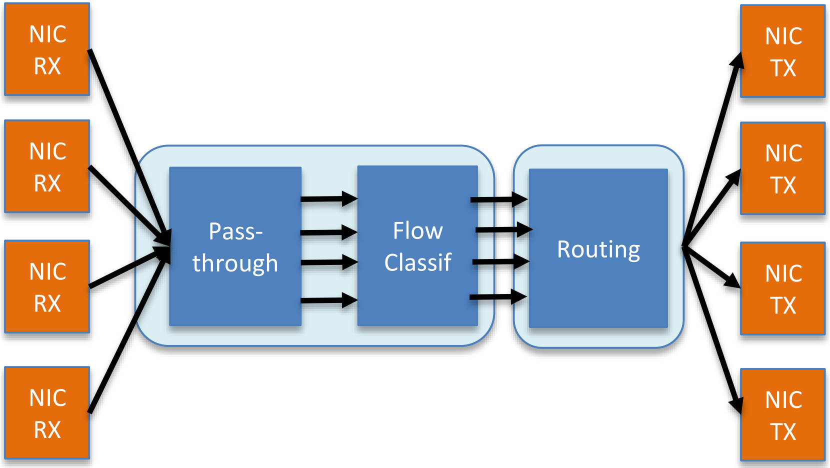
Command Line Interface (CLI):

- Pipeline type specific CLI commands: registered when pipeline type is registered (e.g. route add, route delete, route list, etc for routing pipeline).
- Common pipeline CLI commands: ping (keep-alive), statistics, etc.

Library of reusable pipeline types

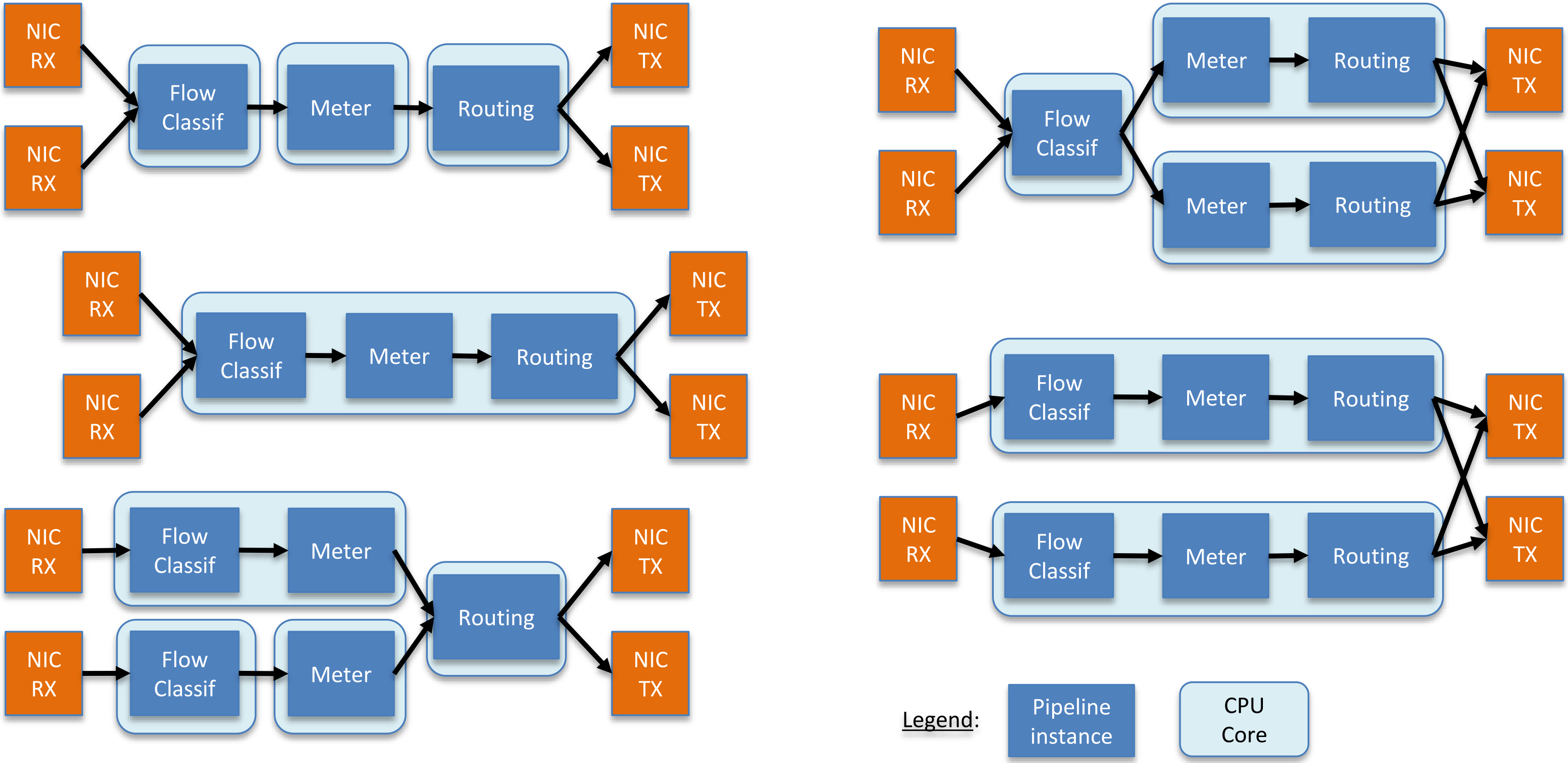
ip_pipeline

```
[PIPELINE0]
type = MASTER
core = 0

[PIPELINE1]
type = PASS-THROUGH
core = 1
pktq_in = RXQ0.0 RXQ1.0 RXQ2.0 RXQ3.0
pktq_out = SWQ0 SWQ1 SWQ2 SWQ3
dma_size = 8
dma_dst_offset = 0
dma_src_offset = 140; headroom (128) + 1st ethertype offset (12) = 140
dma_src_mask = 00000FFF00000FFF; qinq
dma_hash_offset = 8; dma_dst_offset + dma_size = 8

[PIPELINE2]
type = FLOW_CLASSIFICATION
core = 1
pktq_in = SWQ0 SWQ1 SWQ2 SWQ3
pktq_out = SWQ4 SWQ5 SWQ6 SWQ7
n_flows = 16777216; n_flows = 65536
key_size = 8; dma_size = 8
key_offset = 0; dma_dst_offset = 0
hash_offset = 8; dma_hash_offset = 8
flow_id_offset = 64
```



```
[PIPELINE3]
type = ROUTING
core = 2
pktq_in = SWQ4 SWQ5 SWQ6 SWQ7
pktq_out = TXQ0.0 TXQ1.0 TXQ2.0 TXQ3.0
n_routes = 4096
l2 = mpls
mpls_color_mark = yes
ip_hdr_offset = 150; headroom (128) + ethernet header (14) + qinq (8) = 150
color_offset = 68
```

ip_pipeline



NIC RX
NIC RX
Flow Classif
Meter
Routing
NIC TX
NIC TX

NIC RX
NIC RX
Flow Classif
Meter
Routing
NIC TX
NIC TX

NIC RX
NIC RX
Flow Classif
Meter
Flow Classif
Meter
Routing
NIC TX
NIC TX

NIC RX
NIC RX
Flow Classif
Meter
Routing
Meter
Routing
NIC TX
NIC TX

NIC RX
NIC RX
Flow Classif
Meter
Routing
Flow Classif
Meter
Routing
NIC TX
NIC TX

Legend: Pipeline instance CPU Core

ip_pipeline

Pipeline type:

- Functional block: flow classification, routing, etc
- Back-end (packets) + front-end (run-time config)
- Can be instantiated several times in the same app

Pipeline instance:

- Each instance configured independently
- Each instance has its own set of packet Qs (back-end) and message Qs (front-end)
- Each instance mapped to a single CPU core

CPU core:

- Each CPU core can run one or several pipeline instances (of same or different type)
- Pipeline instances mapped to same CPU core are essentially time-sharing threads
- Each pipeline instance can be dynamically remapped from one CPU core to another