DPDK
DATA PLANE DEVELOPMENT KIT

NETCOPE
TECHNOLOGIES

Accelerating DPDK via P4-programmable FPGA-based Smart NICs

Petr Kastovsky
kastovsky@netcope.com

Why FPGAs and P4?

- Goal: Accelerate packet processing
  - Reduce latency and jitter
  - Release cores for application processing rather than network processing
- FPGAs are great for data stream processing = Pipeline
  - Workload specific set of operations executed over a data stream
- FPGAs are hard to program
  - Have you ever seen Verilog or VHDL? It's VERY verbose!
- P4 is target independent and abstract
  - open source, prevents vendor lock-in

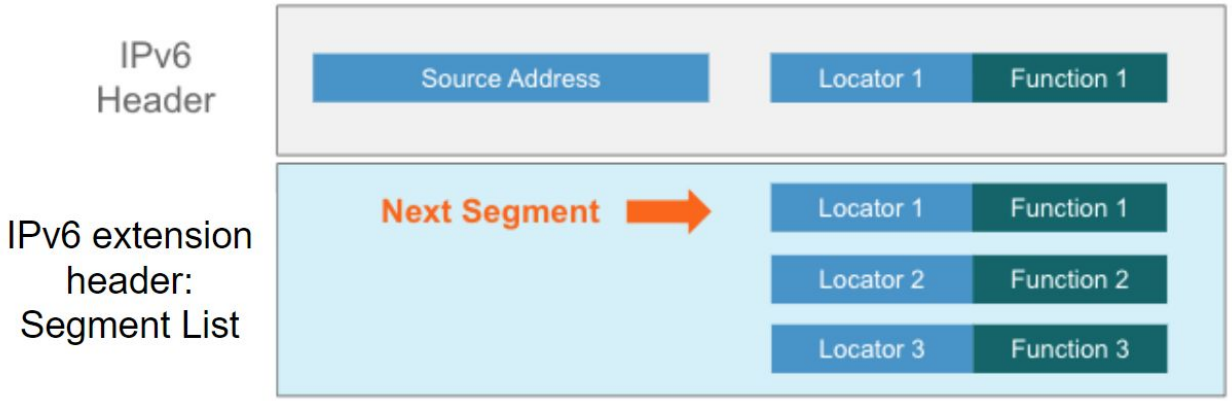
```
-- (this is a VHDL comment)

-- import std_logic from the IEEE library
library IEEE;
use IEEE.std_logic_1164.all;

-- this is the entity
entity ANDGATE is
  port (
    I1 : in std_logic;
    I2 : in std_logic;
    O  : out std_logic);
end entity ANDGATE;

-- this is the architecture
architecture RTL of ANDGATE is
begin
  O <= I1 and I2;
end architecture RTL;
```

Benefits of P4 + FPGAs

- FPGAs are just another software programmable HW
- Deterministic performance, low jitter and low latency
- Dynamic allocation of FPGA resources to P4 pipeline
  - Small number of large tables, large number of small tables
- Parser and deparser of arbitrary complexity
- Scalable FPGA chip offering
  - From 10G, 25G to Nx100G
- Flexible memory configurations
  - On-chip vs off-chip, static vs dynamic
- Extensible through P4 16 externs
  - E.g. payload processing (encryption, pattern matching)

Use case example - segment routing v6

- Segment Routing: Application steers packets through an ordered list of instructions and realizes end-to-end policy.
  - Promoted by Cisco, Microsoft, Bell Canada, Alibaba, SoftBank, ...
- The IPv6 flavor of Segment Routing (SRv6) allows user-defined functions to be associated with segments - network programming
- Needs dataplane support!



SRv6 - parser definition

```
header_type ipv6_t {
        fields {
                ver         : 4;
                trafClass   : 8;
                flowLab     : 20;
                payLen      : 16;
                nextHead    : 8;
                hopLim      : 8;
                srcAddr     : 128;
                dstAddr     : 128;
        }
 }
```

```
header_type ethernet_t {
        fields {
                dstAddr     : 48;
                srcAddr     : 48;
                etherType   : 16;
        }
}
```

```
#define PROTOCOL_IPV6          0x86dd
#define PROTOCOL_V6EXT         0x2B

header ethernet_t       ethernet_0;
header ipv6_t           ipv6;

// Ethernet parsing
parser parse_ethernet {
        extract(ethernet_0);
        return select(latest.etherType) {
                PROTOCOL_IPV6       : parse_ipv6;
                default             : ingress;
        }
}

// IPv6 parsing
parser parse_ipv6 {
        extract(ipv6);
        return select(latest.nextHead)  {
                PROTOCOL_V6EXT      : parse_ext;
                default             : ingress;
        }
}
```

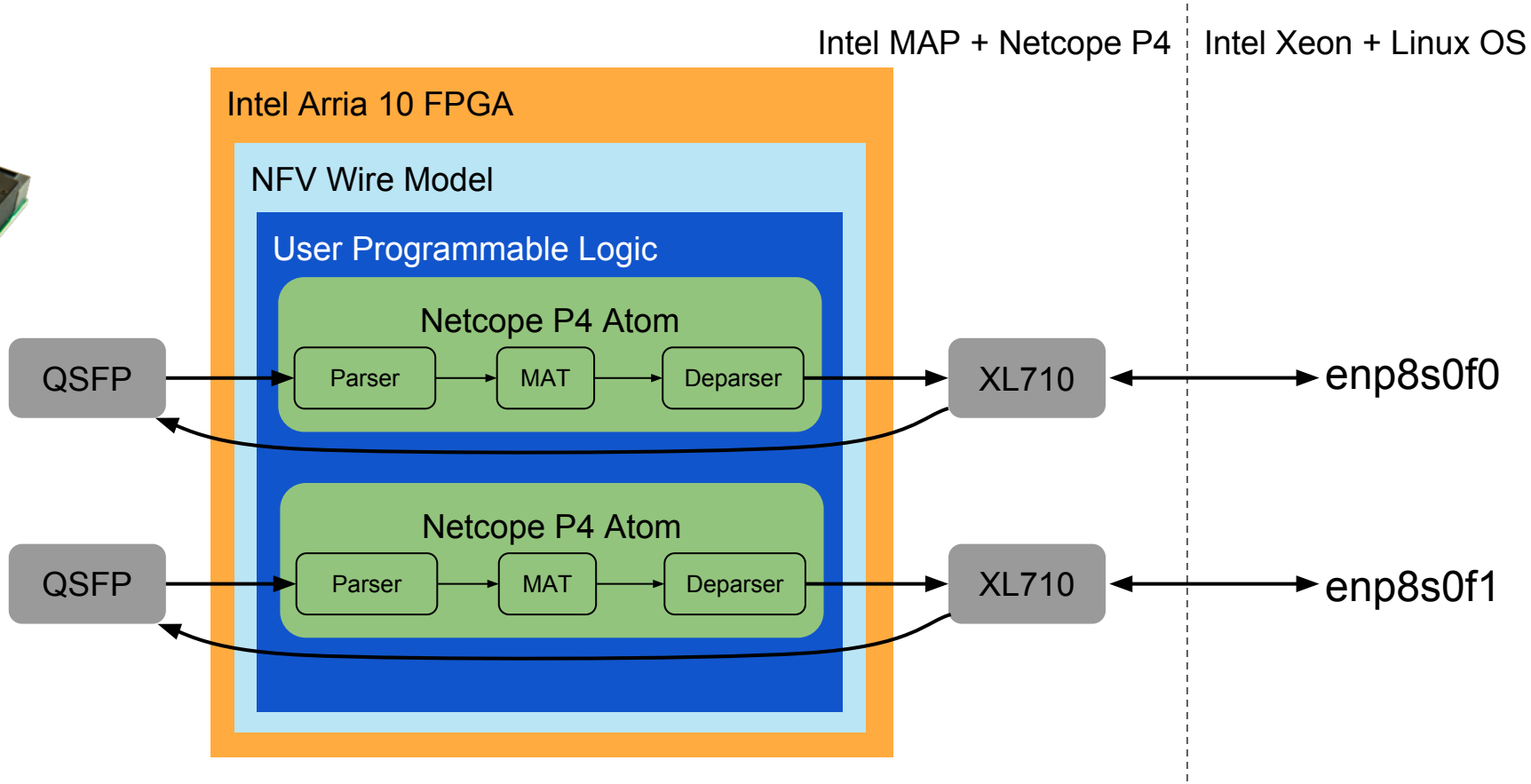
SRv6 - match-action tables

```
// Rewrites destination IPv6 address
action rewrite() {
        // Rewrite the destination IPv6
        // address with the last IPv6 segment
        modify_field(ipv6.dstAddr,lastSeg.segVal);
        add_to_field(ipv6_ext.next_seg,-1)
}
// Only default rule with action rewrite
table tab_rewrite {
        actions {
                rewrite;
        }
}
```

```
control ingress {
        // If any segment was valid
        if(valid(ipv6_seg0)) {
                // Apply the segment routing rewrite
                apply(tab_rewrite);
        }
        // Set destination MAC
        apply(table_set_egress_port);
}
```

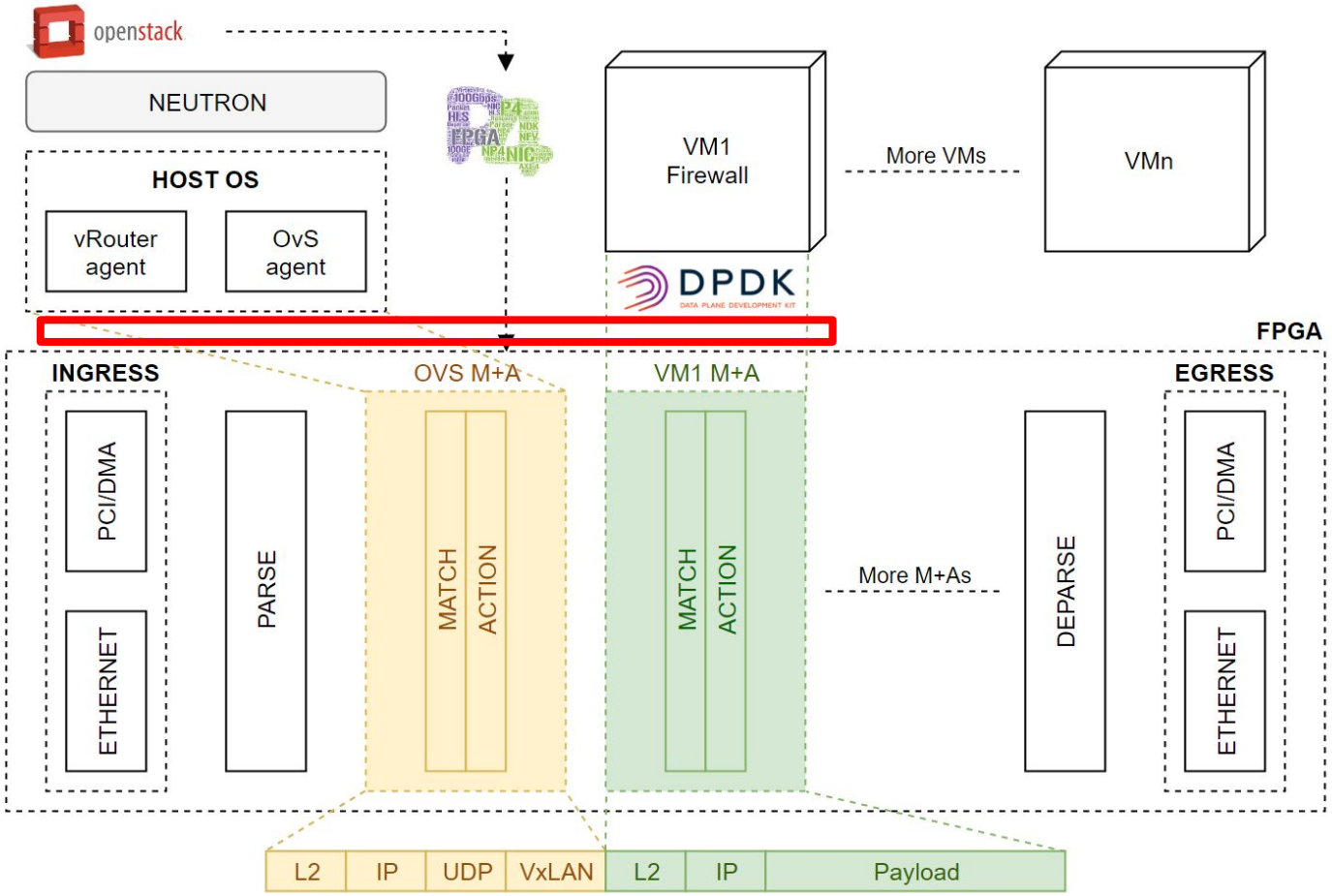
```
// Sets egress port to specific value
action set_egress_port(eport)  {
        modify_field(ethernet_0.dstAddr,eport);
}
// Sets egress port to specific value
action forward_based_on_hash()  {
        modify_field_with_hash_based_offset(
                ethernet_0.dstAddr, 0,ipv6_hash,65536);
}
// Set dst MAC based on destination IPv6
table table_set_egress_port {
        reads {
                ipv6.dstAddr : ternary;
        }
        actions {
                drop_p;
                permit;
                set_egress_port;
                forward_based_on_hash;
        }
        max_size: 15;
}
```

Firmware architecture



Many more use cases

- Tunnel processing
- Traffic statistics
- Load balancing
- Traffic shaping
- Virtual switching
- Inband network telemetry
- ...



DPDK API

- DPDK RTE FLOW - API to expose P4 pipeline to a user

Attributes

Attribute	Value
Group	2
Priority	0
Traffic	Egress

+

Matching pattern

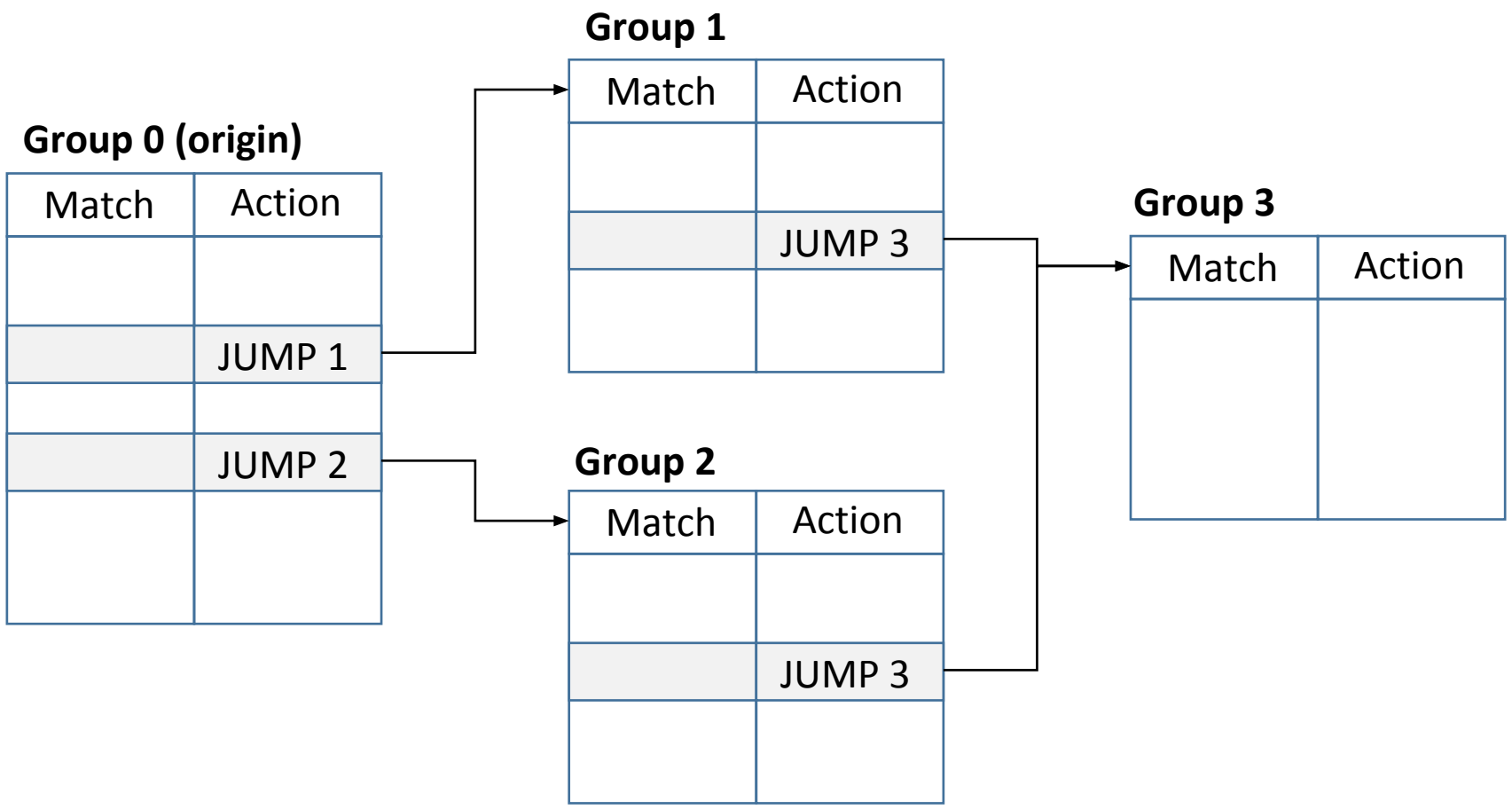
Index	Item
0	Ethernet
1	IPv4
2	TCP
3	END

+

List of actions

Index	Action
0	OF_DEC_NW_TTL
1	OF_PUSH_VLAN
2	PHY_PORT
3	END

Attribute Group



Matching patterns supported protocols

- Packet headers

ETH	**UDP**	**NVGRE**	**GENEVE**	**ICMP6_ND_NS**
VLAN	**TCP**	**MPLS**	**VXLAN-GPE**	**ICMP6_ND_NA**
IPV4	**SCTP**	**GRE**	**ARP_ETH_IPV4**	**ICMP6_ND_OPT**
IPV6	**VXLAN**	**GTP, GTPC, GTPU**	**IPV6_EXT**	**ICMP6_ND_SLA_ETH**
ICMP	**E_TAG**	**ESP**	**ICMP6**	**ICMP6_ND_TLA_ETH**

- Special

ANY	Matches any protocol in place of the current layer
RAW	Matches a byte string of a given length at a given offset
FUZZY	Approximate Matching

Actions

- Support of actions depends on the available hardware

END	End marker for action lists
VOID	Used as a placeholder for convenience
PASSTHRU	Leaves traffic up for additional processing by subsequent flow rules
MARK	Attaches an integer value to packet
FLAG	Flags packets. Similar to Action: MARK without a specific value
DROP	Drop packet
SECURITY	Encrypt packet payload based on configuration (IPSec)

Actions II

- Packet redirection

JUMP	Redirects packets to a group
QUEUE	Assigns packets to a given queue index
RSS	Spread packets among several queues according to the parameters
PF	Directs matching traffic to the physical function (PF)
VF	Directs matching traffic to a given virtual function
PHY_PORT	Directs matching traffic to a given physical port index
PORT_ID	Directs matching traffic to a given DPDK port ID

- Packet statistics

COUNT	Adds a counter action to a matched flow
METER	Applies a stage of metering and policing

Actions III

- “OpenFlow” Actions

OF_SET_MPLS_TTL	OF_COPY_TTL_OUT	OF_SET_VLAN_VID
OF_DEC_MPLS_TTL	OF_COPY_TTL_IN	OF_SET_VLAN_PCP
OF_SET_NW_TTL	OF_POP_VLAN	OF_POP_MPLS
OF_DEC_NW_TTL	OF_PUSH_VLAN	OF_PUSH_MPLS

- Packet encap/decap

VXLAN_ENCAP	NVGRE_ENCAP
VXLAN_DECAP	NVGRE_DECAP

RTE_FLOW and P4

- RTE_FLOW
  - enables very dynamic scenarios
  - follows open flow
- P4
  - successor to open flow
  - protocols defined in the code
  - data plane operations defined in the code
- P4 for FPGA is a valid hardware backend for RTE_FLOW
  - Let’s work to propose API to fully exploit flexibility of P4 for FPGAs

NETCOPE TECHNOLOGIES

Petr Kastovsky
kastovsky@netcope.com

Become **flexible** with **Netcope P4**
- paradigm shift in **FPGA programming.**