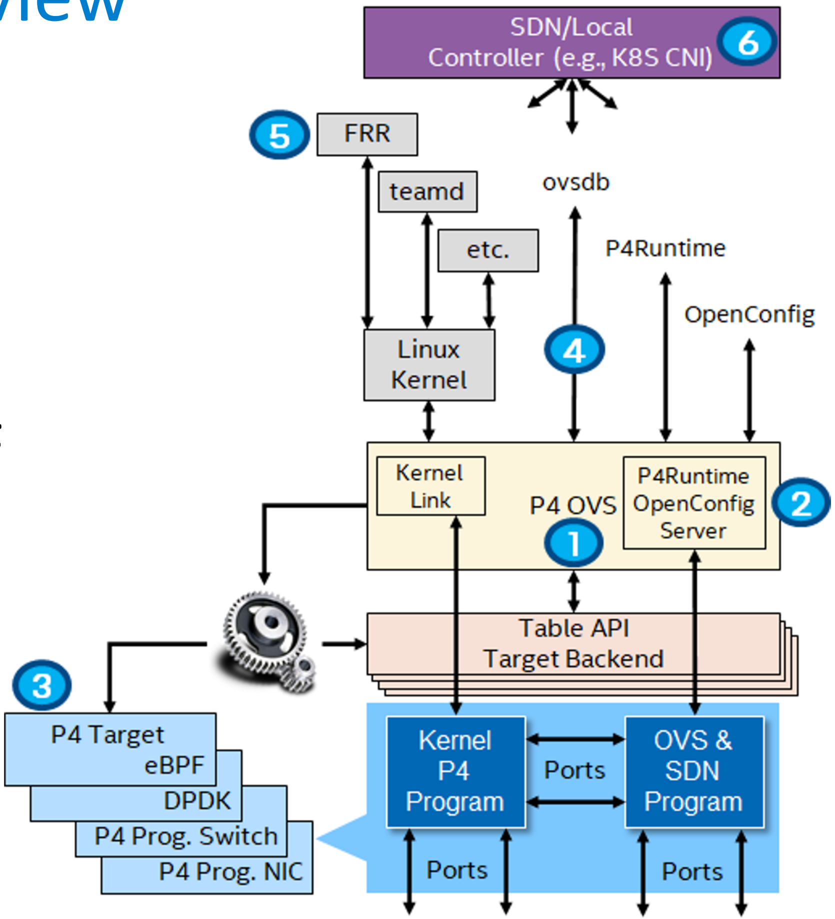
MAY 18-20
2021 P4
Workshop
Hosted by ONF

P4 OVS

Namrata Limaye, Intel
Brian O'Connor, ONF
Tomasz Osinski, ONF
Mateusz Kossakowski, Orange
Gerald Rogers, Intel
Nupur Jain, Intel
Dan Daly, Intel

Overview

1. P4 Enhanced Open vSwitch
2. P4Runtime & OpenConfig on the Host
3. P4 Target Example: PSA-eBPF
4. OVS Virtual Bridging w/ VXLAN
5. Kernel Routing/ECMP
6. K8s CNI Agent



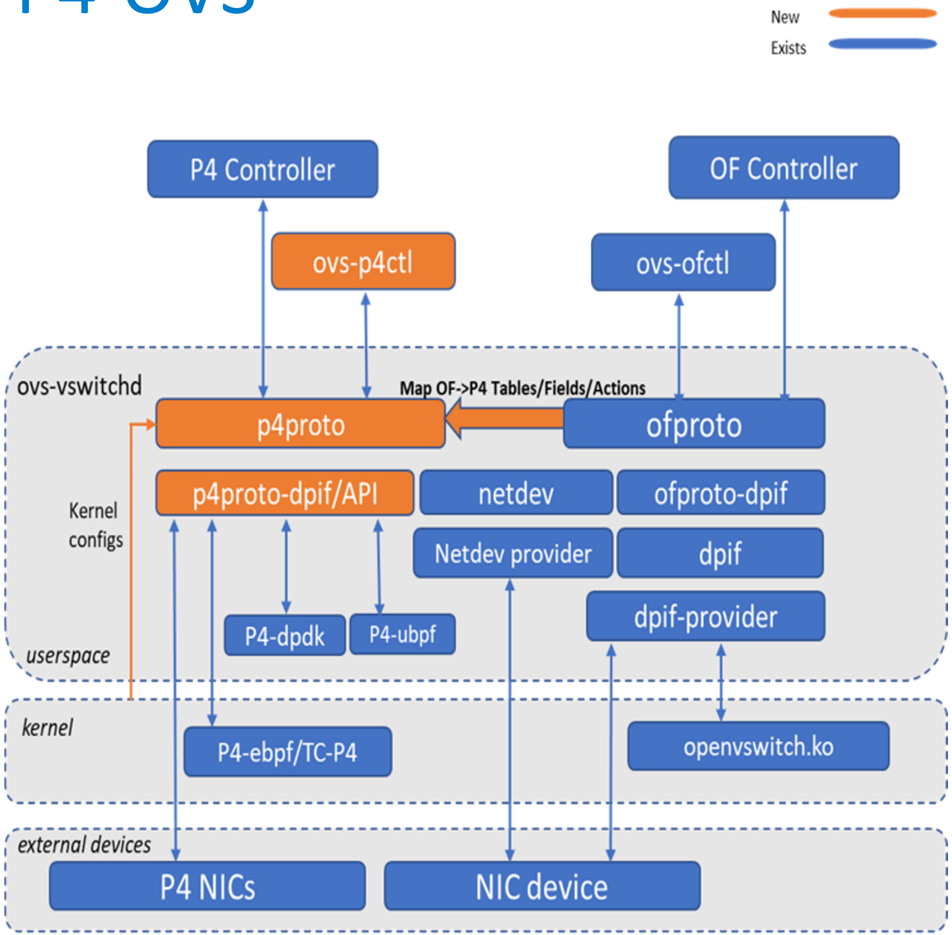
Part 1: P4-OVS

Control planes:

- OVS – Maps OVS configuration to P4 Tables (E.g. Vxlan)
- P4Runtime + Openconfig – Configures P4 tables explicitly (E.g. Container load-balancing)
- Kernel – Maps Kernel configurations (via SAI) to P4 Tables (E.g. ECMP w/ FRR)
- All three control planes can used to program the same P4 target.
- Multiple P4Runtime clients can connect and program different P4 pipelines

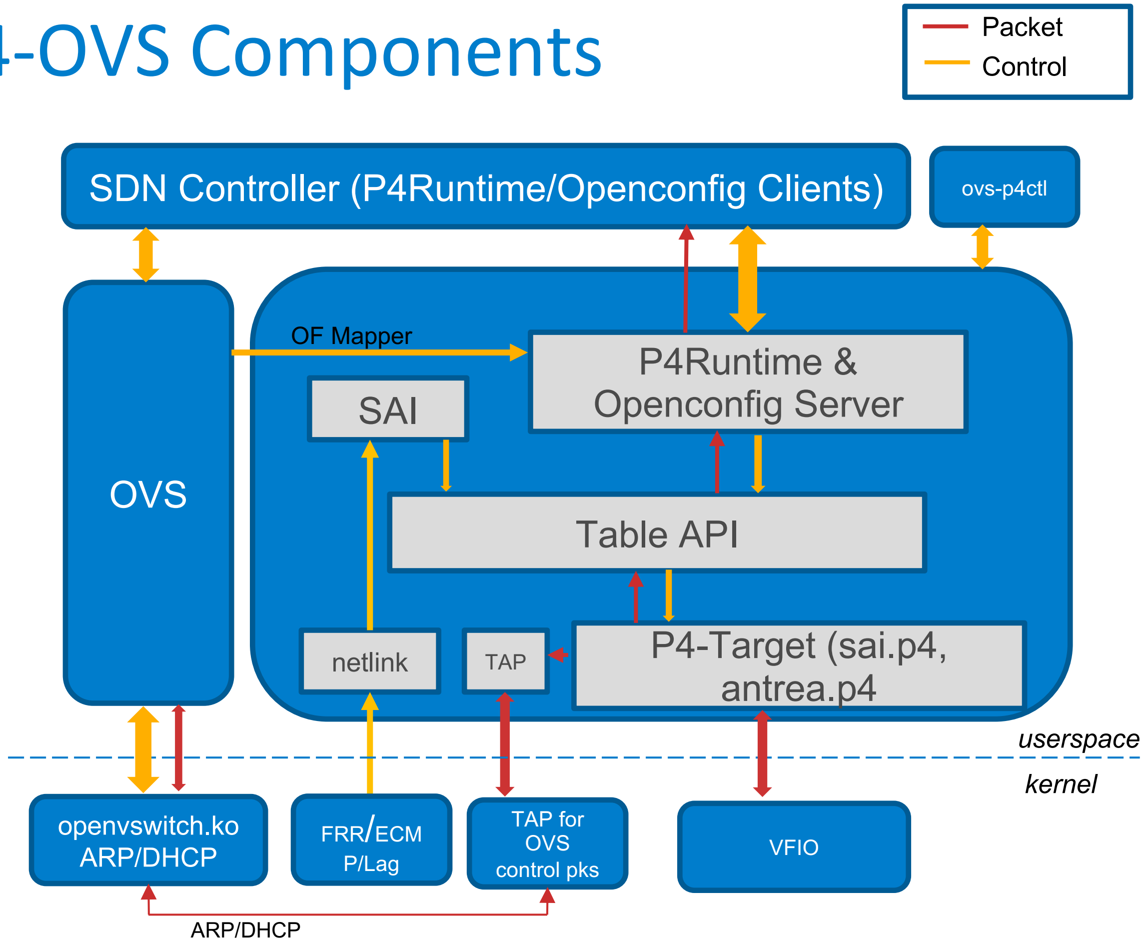
Data Planes:

- Software: PSA-eBPF, P4 DPDK, others
- Hardware: P4 Switches, P4 NICs, etc



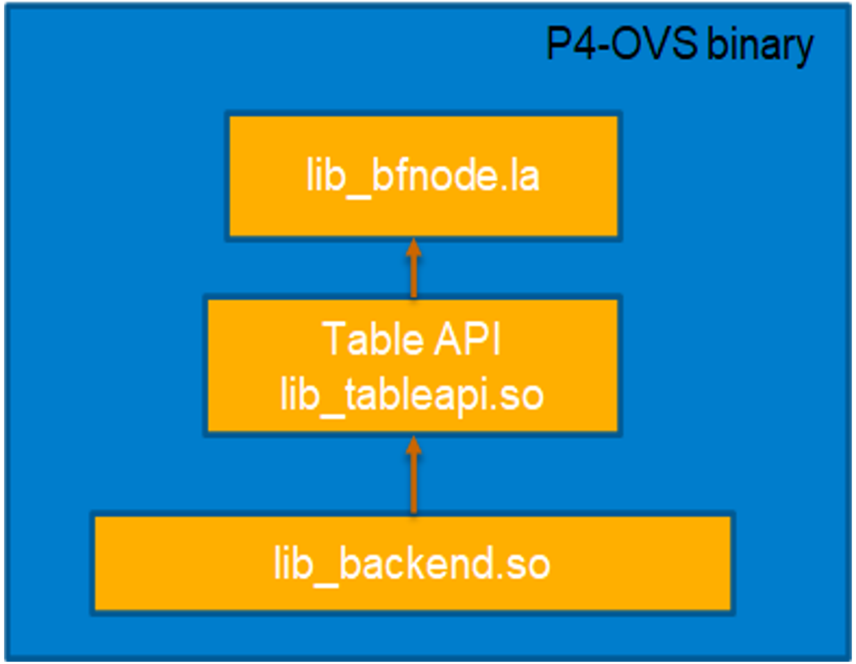
P4-OVS Components

- Table API – Common programming IF for P4Runtime/Openconfig/SAI
- P4-targets are programmed by either of the 3 control planes
- TAP for passing control packets to OVS
- Netlink for receiving configurations from kernel
- Openflow to P4 Mapper

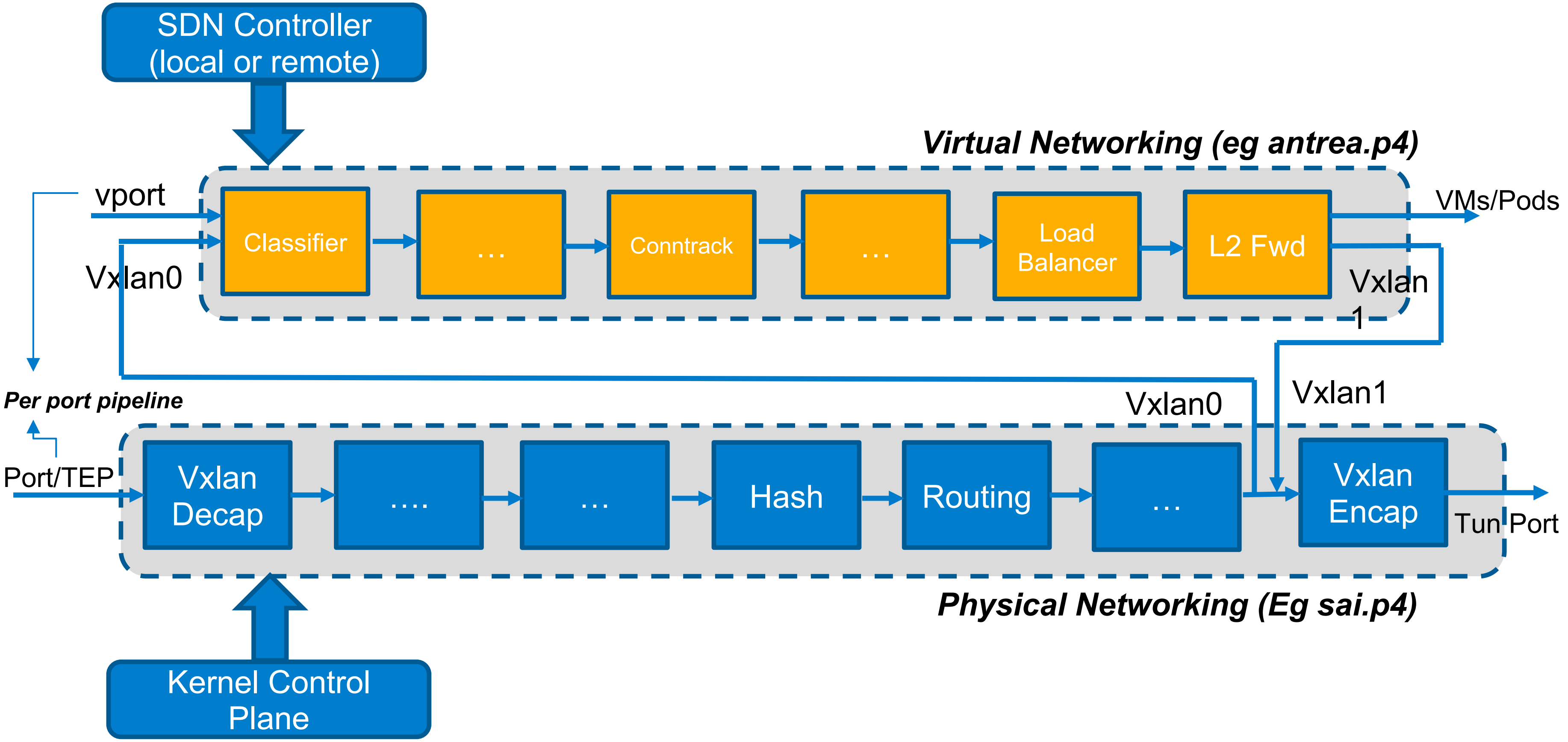


Building p4-OVS

- Coupled targets- Table API builds with the linked target(lib_ebpf.so/lib_p4dpdk.so) dataplane library. Generates a 'lib_tableapi.so'
- Decoupled targets –Table API builds with interface lib_backend.so library. Generates a lib_tableapi.so
- Build BFNode with linked Table API 'lib_tableapi.so' Generates 'lib_bfnode.la'
- Build P4-OVS with linked BFNode 'lib_bfnode.la'
- P4-OVS calls BFNode API to program targets.

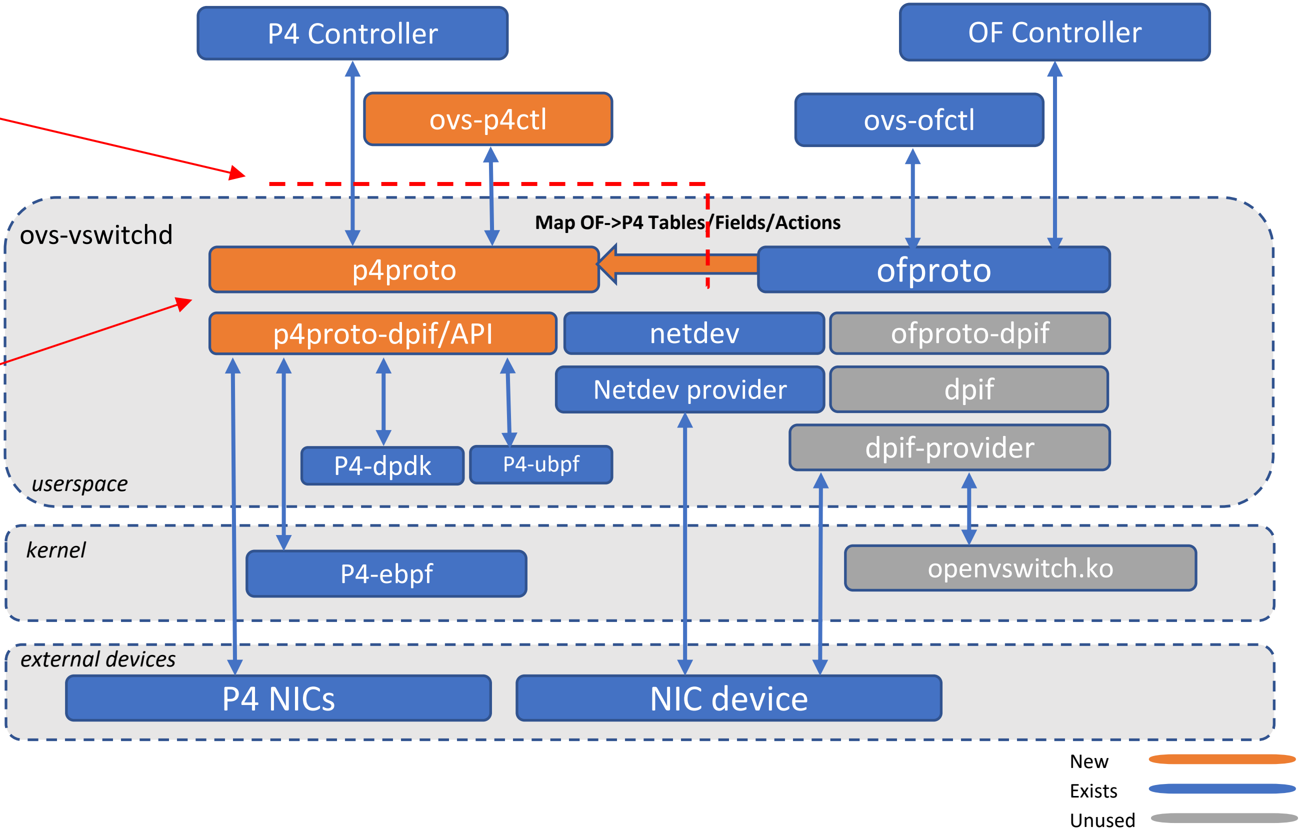


P4 Dataplane Layering

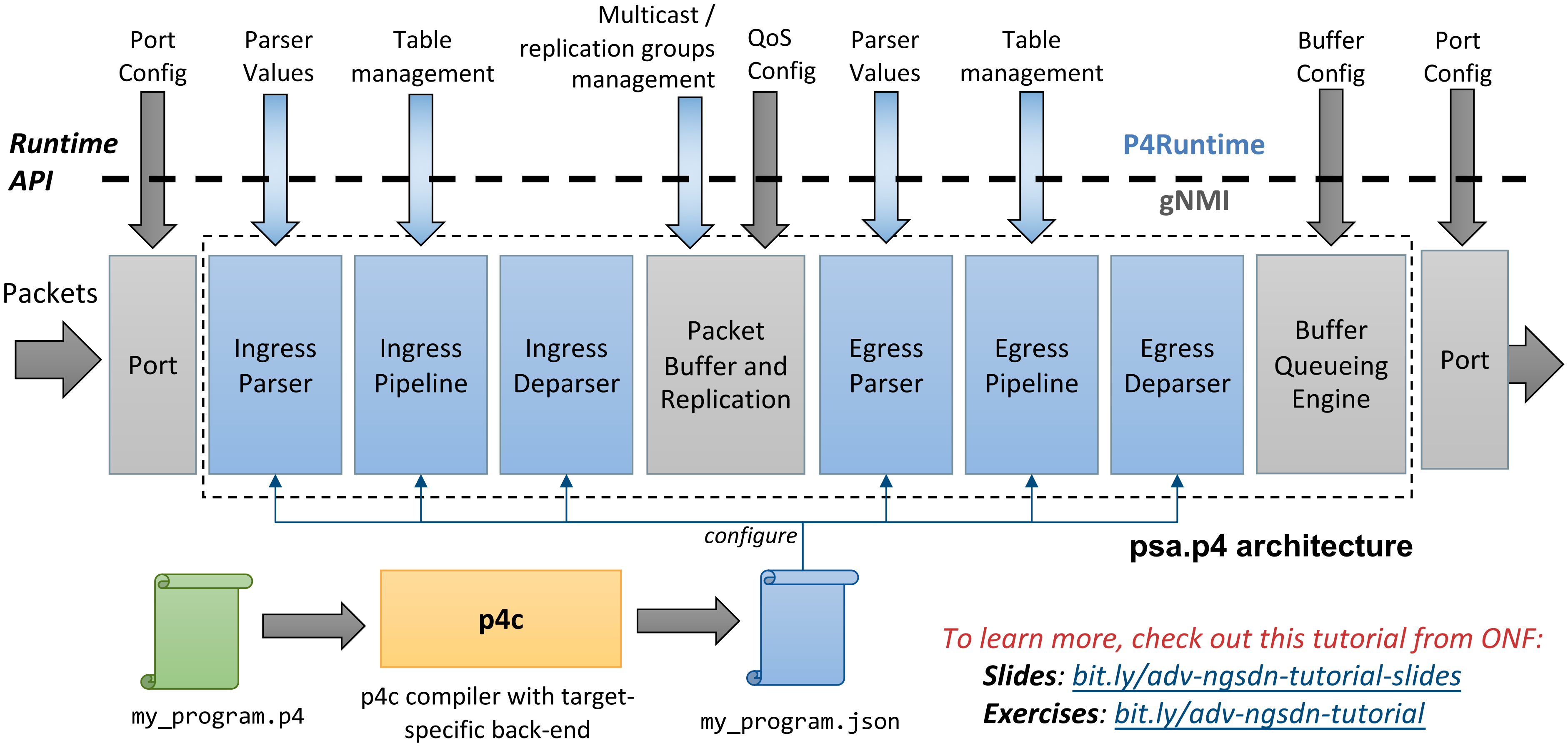


Part 2: P4Runtime & OpenConfig on the Host

1. Identify the new forwarding plane interfaces: **P4Runtime** and **gNMI**
2. Explain how these are mapped to the P4 target interface: **Table API**



P4, P4Runtime, gNMI



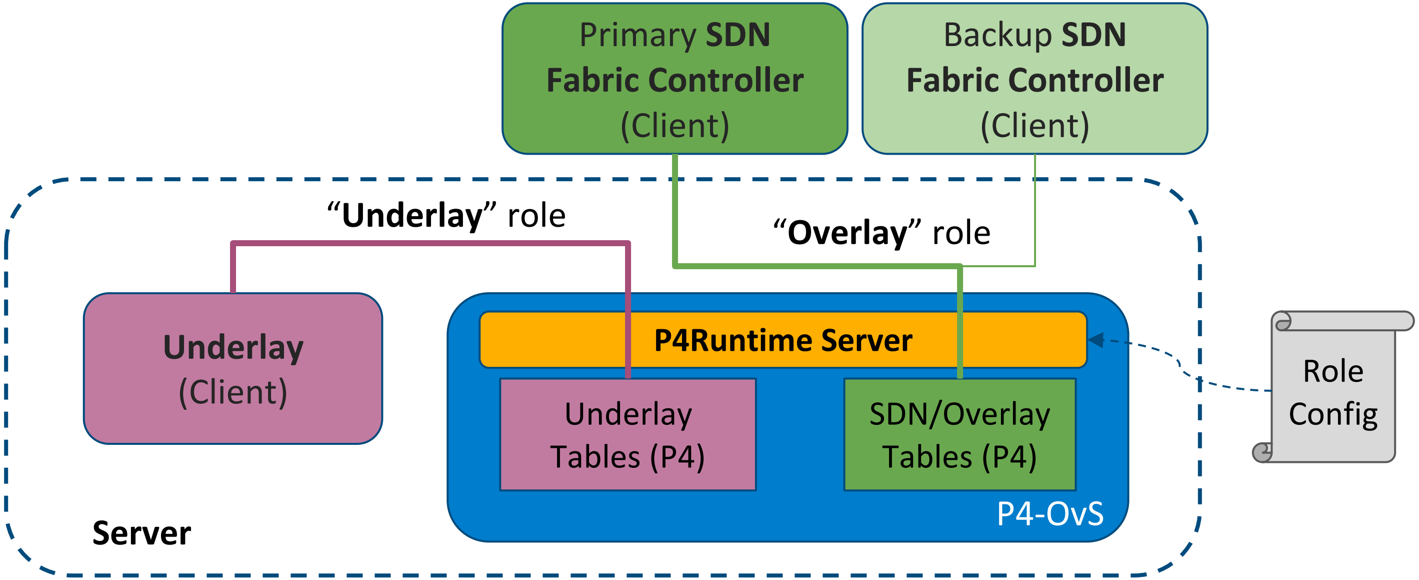
To learn more, check out this tutorial from ONF:

Slides: *bit.ly/adv-ngsdn-tutorial-slides*

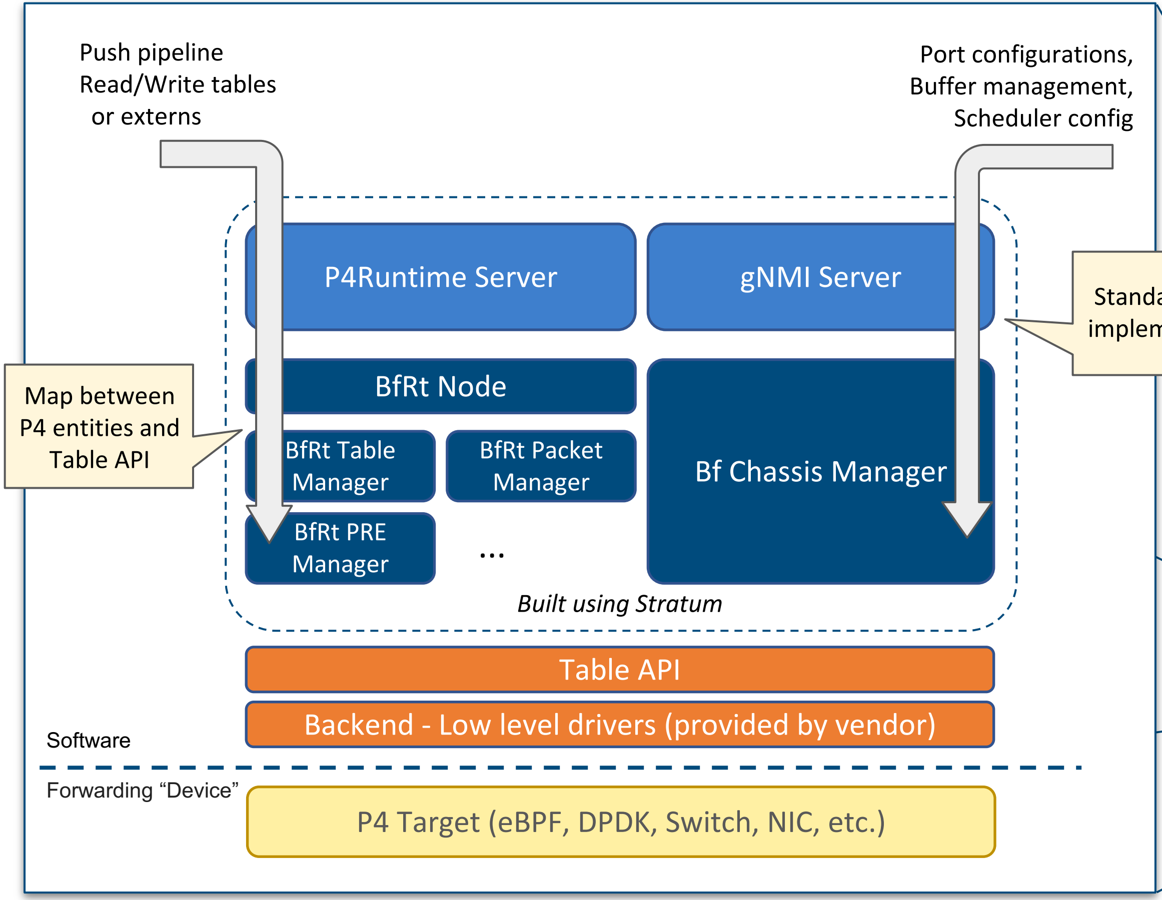
Exercises: *bit.ly/adv-ngsdn-tutorial*

P4Runtime's Multi-client Architecture

- P4Runtime's role configuration allows P4 entities to be divided among different client roles
- Each role has a primary client and may have backup clients for high availability (HA)



Implementing P4Runtime in P4-OvS



- Develop a common server and mapping implementation that can be reused
  - for different switch stacks (including OvS)
  - on different P4 targets

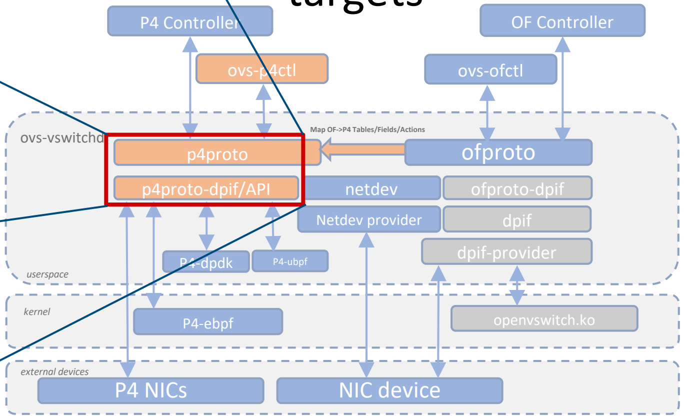


Table API Overview

- Everything is structured as a table object with a **table ID**, **key** and **data**

- Some tables are **generated from the P4 program** by the compiler
- Some table are "**fixed**," i.e. they are the same regardless of the P4 program (e.g. *ports*, *QoS*, *replication*)

Note: Table API used to be called ***BfRt***

Example P4 table (**bfrt.json** from **p4c**)

```
"tables": [
  {
    "id" : 45300881, "name" : "routing_v4",
    "key" : [
      {
        "id" : 1, "name" : "ipv4_dst",
        "match_type" : "LPM",
        "type" : { "type" : "bytes", "width" : 32 }
      }
    ],
    "action_specs" : [
      {
        "id" : 19792090, "name" : "set_next_id",
        "data" : [
          {
            "id" : 1, "name" : "next_id",
            "type" : { "type" : "bytes", "width" : 32 }
          }
        ]
      },
      {
        "id" : 29734112, "name" : "drop",
        "data" : []
      }, ...
```

Example "fixed" table (**port.json**)

```
"tables": [
  {
    "id": 4278255617, "name": "$PORT",
    "key": [{ "id": 1, "name": "$DEV_PORT", }],
    "data": [
      {
        "singleton": {
          "id": 1, "name": "$SPEED",
          "type": {
            "choices": [
              "BF_SPEED_1G",
              "BF_SPEED_10G",
              ...
              "BF_SPEED_400G"
            ]
          }
        }
      },
      {
        "singleton": {
          "id": 2, "name": "$FEC",
          ...
```

Mapping P4 Entity to Table API

```
{
  table_entry {
    table_id: 1000
    match {
      field_id: 1
      exact {
        value: 10
      }
    }
    action {
      action {
        action_id: 2001
        params {
          param_id: 1
          value: 100
        }
      }
    }
    priority: 10
  }
}
```

Pseudo code:

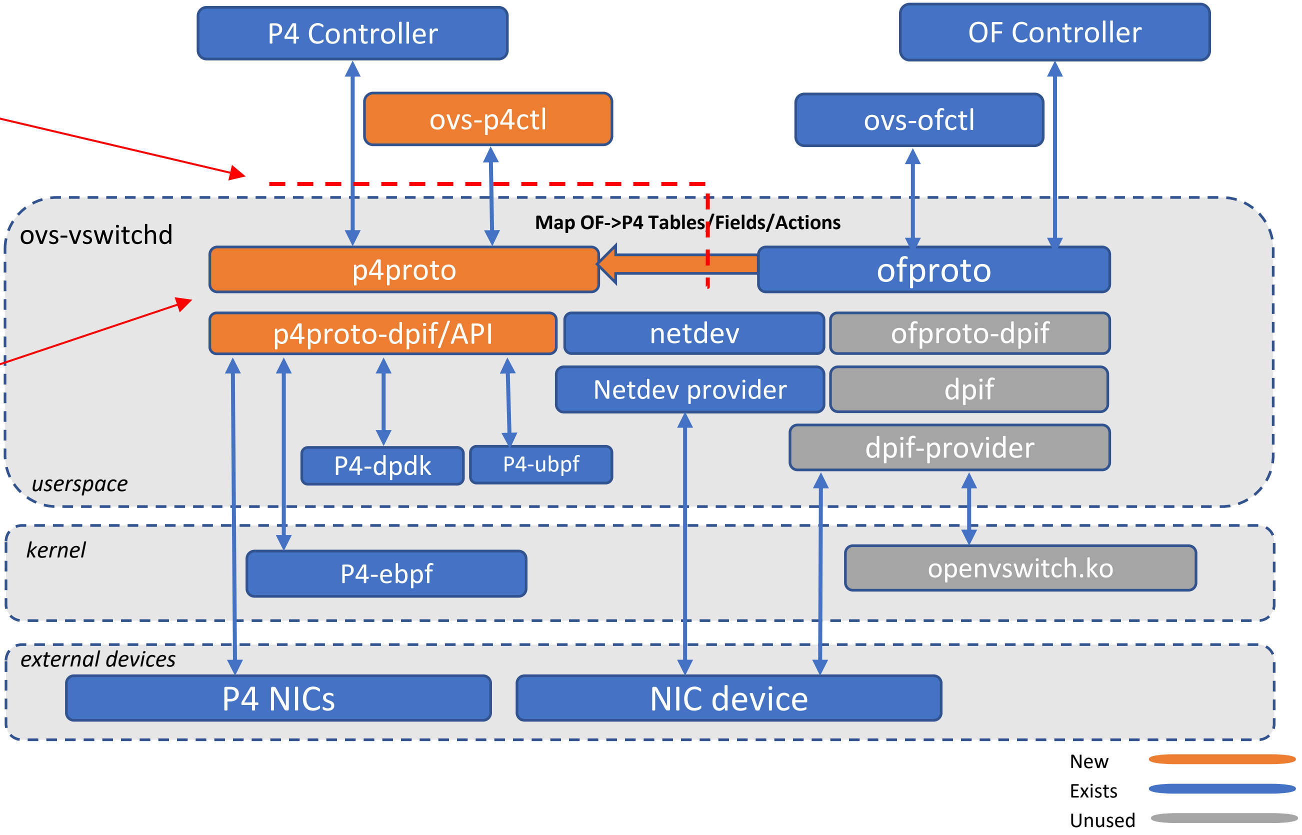
```
BfRtTable* table;
BfRtTableKey* table_key;
BfRtTableData* table_data;

bfrtTableFromIdGet(1000, &table)

// Match
table_key->setValue(1, 10);

// Action
table->dataReset(2001, table_data);
table_data->setValue(1, 100);

// Priority
bf_rt_id_t fid;
table->keyFieldIdGet("$MATCH_PRIORITY", &fid);
table_key->setValue(fid, 0xffffff - 10);
```

P4Runtime uses different priority mechanism

P4Runtime & OpenConfig on the Host

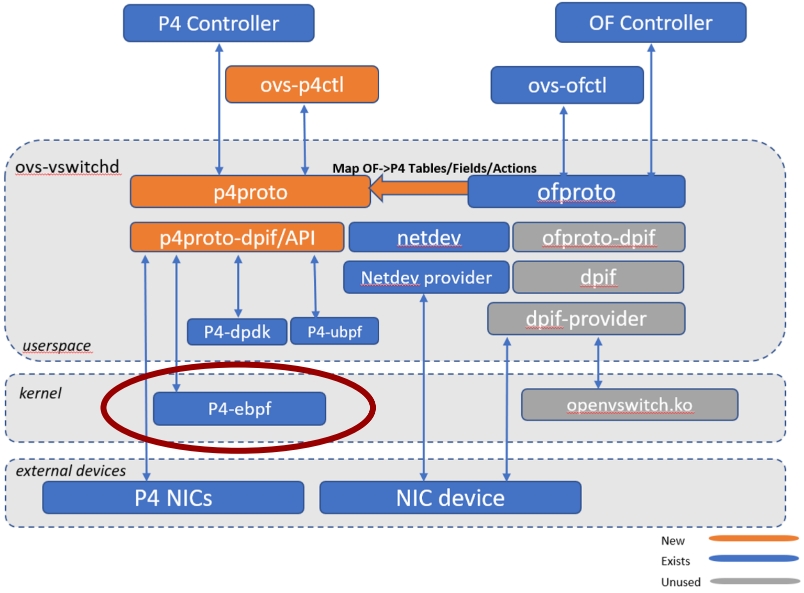
P4Runtime and **gNMI** are new interfaces added to P4-OvS

Common building blocks are leveraged and shared to implement these interfaces and map them to P4 targets



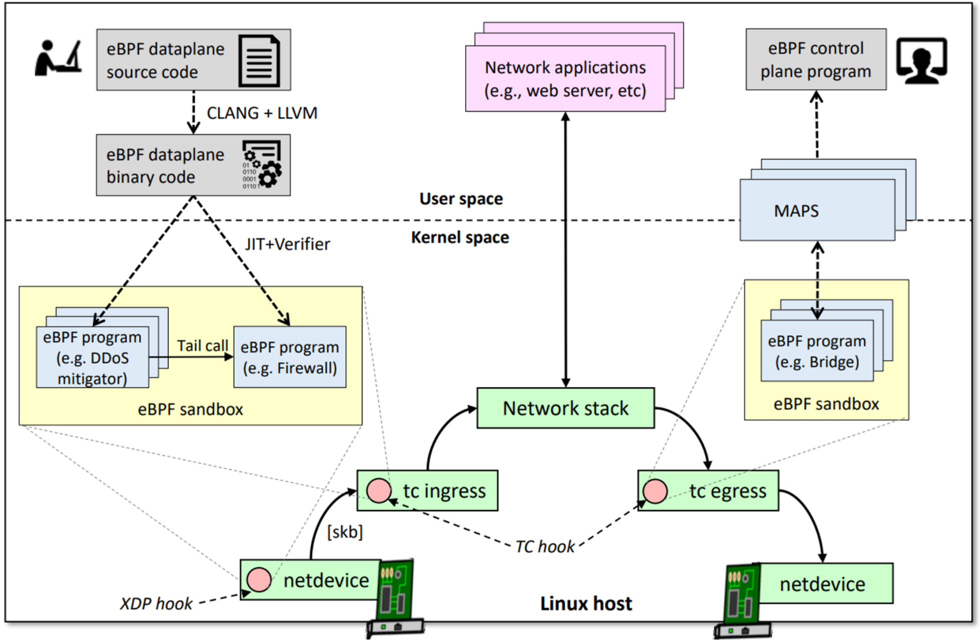
Part 3: PSA-eBPF Target

- eBPF/XDP is planned to be an option for the P4-OVS datapath
- Work-in-progress on the new version of P4 to eBPF compiler
  - Compliant with the PSA architecture
- Main challenges:
  - How to map the PSA architecture into eBPF/XDP?
  - How to implement the P4 objects inside the eBPF subsystem?



Introduction to eBPF

- eBPF provides in-kernel VM for packet filtering
- eBPF allows to execute eBPF bytecode at different hooks
- BPF helpers to handle complex tasks
- BPF maps (e.g. hashmap, arraymap) for stateful processing

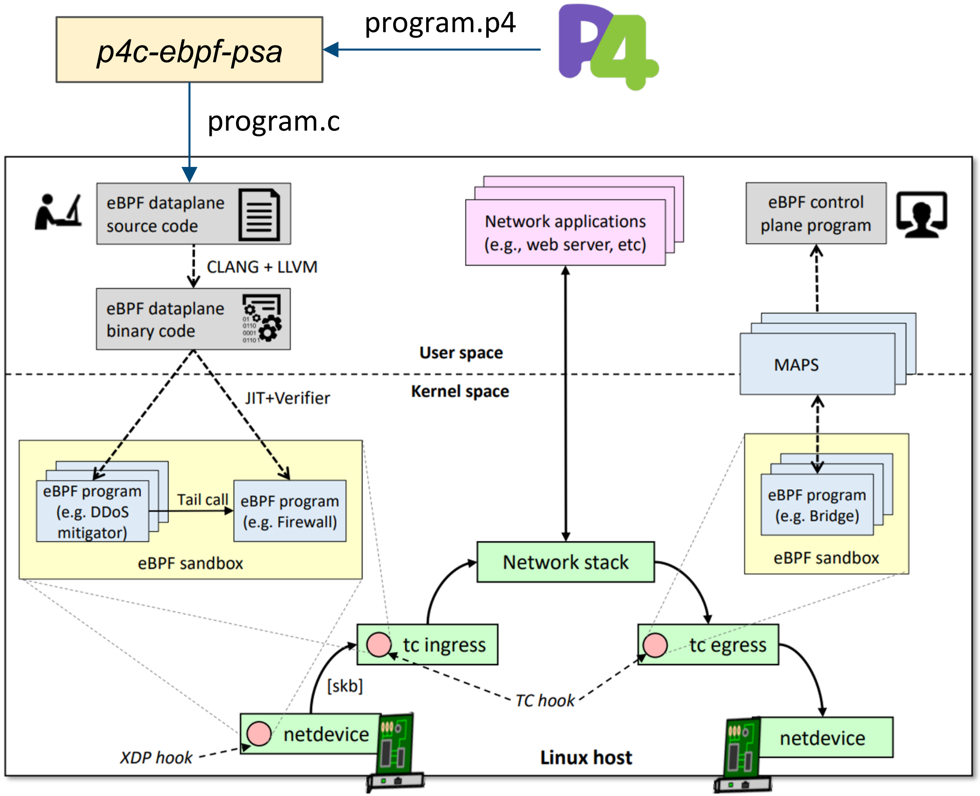


Source: S. Miano, M. Bertrone, F. Risso, M. Tumolo and M. V. Bernal, "Creating Complex Network Services with eBPF: Experience and Lessons Learned," *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, pp. 1-8

Introduction to eBPF

- eBPF provides in-kernel VM for packet filtering
- eBPF allows to execute eBPF bytecode at different hooks
- BPF helpers to handle complex tasks
- BPF maps (e.g. hashmap, arraymap) for stateful processing

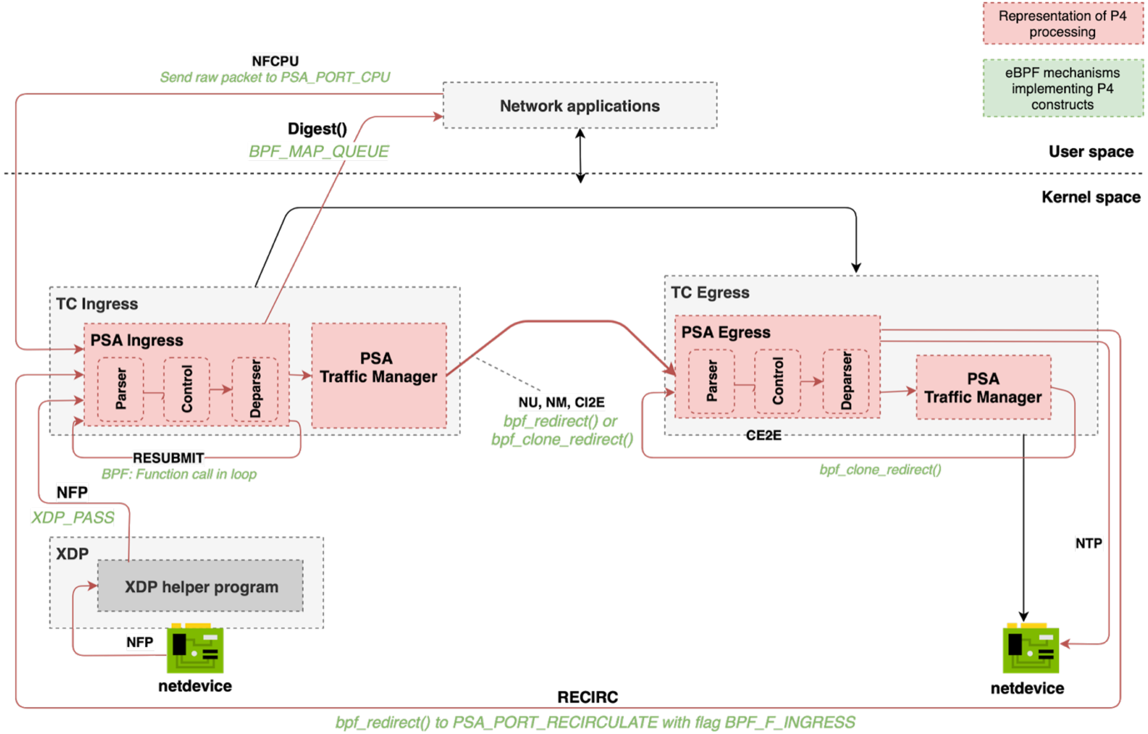
Our goal is to design how the PSA architecture can be mapped to the eBPF subsystem and build a P4 compiler that will translate a PSA program to the eBPF representation.



Source: S. Miano, M. Bertrone, F. Risso, M. Tumolo and M. V. Bernal, "Creating Complex Network Services with eBPF: Experience and Lessons Learned," *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, pp. 1-8

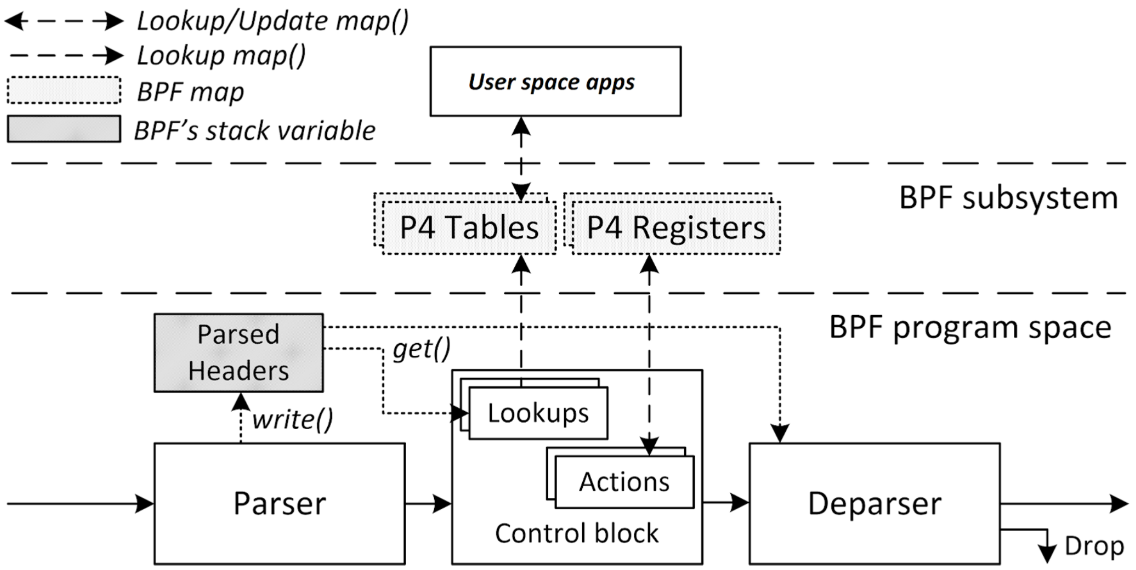
PSA-eBPF: Solution architecture

- PSA architecture decomposed into 3 eBPF programs..
- Generic, TC-based architecture to implement any PSA program
  - XDP does not support packet cloning
- May be adjusted depending on the P4 program's structure



PSA-eBPF Internals: **P4 pipeline to BPF mapping**

- A single P4 pipeline (Parser+Control+Deparser) is translated to a single eBPF program*
- The BPF implementation creates a shared **`Parsed Headers`** structure used to save packet's fields
- BPF maps are heavily used to implement P4 objects (e.g. Tables, Externs, ValueSet)



* The implementation is based on p4c-ebpf (ebpf_model.p4) and p4c-xdp (xdp_model.p4)

PSA-eBPF Internals: **Parser**

```
parser prs(...) {
   state start {
      packet.extract(headers.ethernet);
      transition select(headers.ethernet.etherType) {
         16w0x800 : parse_ipv4;
         0x8847   :   parse_mpls;
         default :    accept;
      }
   }
   state parse_mpls {
         packet.extract(headers.mpls);
         transition ipv4;  // simplified
   }
   state parse_ipv4 {
         packet.extract(headers.ipv4);
         transition accept;
   }
}
```

p4c-ebpf-psa

```
struct Headers_t headers = {
      .ethernet = { .ebpf_valid = 0 },
      .mpls = { .ebpf_valid = 0 },
      .ipv4 = { .ebpf_valid = 0 },
};
goto start;
start: {
   headers.ethernet.dstAddr = load_dword(pkt, ...);
   headers.ethernet.ether_type = load_half(pkt, ...);
   headers.ethernet.ebpf_valid = 1;
   switch (headers.ethernet.etherType) {
         case 0x0800: goto ipv4;
         case 0x8847: goto mpls;
         default: goto accept;
   }
}
ipv4: { /* 'ipv4' parser state */  }
mpls: { /* 'mpls' parser state */  }
accept: { /* Control block */ }
```

PSA-eBPF Internals: **P4 Tables**

```
action mac_learn() {
      local_md.send_mac_learn_msg = true;
      local_md.mac_learn_msg.mac_addr =

                  headers.ethernet.src_addr;
      local_md.mac_learn_msg.port =
                  standard_metadata.ingress_port;
      local_md.mac_learn_msg.vlan_id =
                  headers.vlan_tag.vlan_id;
}
table tbl_mac_learning {
      key = { headers.ethernet.src_addr : exact; }
      actions = { mac_learn;
                  NoAction; }
      size = 100;
      const default_action = mac_learn();
}
apply {
      tbl_mac_learning.apply();
}
```

lpm & *ternary* also supported

p4c-ebpf-psa →

```
struct  {
__uint(type, BPF_MAP_TYPE_HASH);
__uint(key_size, sizeof(struct
           tbl_mac_learning_key));
__uint(value_size, sizeof(struct
           tbl_mac_learning_value));
 __uint(max_entries, 100);
} tbl_mac_learning;
```

```
struct  {
__uint(type, BPF_MAP_TYPE_ARRAY);
__uint(key_size, sizeof(u32);
__uint(value_size, sizeof(struct
           tbl_mac_learning_value);
__uint(max_entries, 1);
} tbl_mac_learning_defaultAction;
```

p4c-ebpf-psa →

```
struct tbl_mac_learning_key key = { .ipv4_dstAddr = headers.ipv4.dstAddr };
struct tbl_mac_learning_value *value = NULL;
value = BPF_MAP_LOOKUP_ELEM(&tbl_mac_learning, &key);
if (value == NULL) {
   hit = 0;
   /* lookup to default action map, will always return non-null value */
   value = BPF_MAP_LOOKUP_ELEM(&tbl_mac_learning_defaultAction, &zero_idx);
} else {
   hit = 1;
   /* run action */
   switch (value->action) {
                    case ACT_TBL_MAC_LEARNING_MAC_LEARN:  { … }
   }
}
```

PSA-eBPF Internals: **P4 Externs**

- PSA-eBPF will support ***all P4 externs defined for the PSA architecture***
  - Supported now: Hash, Checksum, Counter, DirectCounter, Register, Action Profile, Packet Digest
  - In progress: Meters, DirectMeter, Random, Action Selector
- How it is done based on example of:
  - Digest

```
struct mac_learn_digest_t {
    ethernet_addr_t mac_addr;
    PortId_t        port;
    vlan_id_t       vlan_id;
}
...
Digest<mac_learn_digest_t>() mac_learn_digest;
apply {
    if (local_md.send_mac_learn_msg) {

mac_learn_digest.pack(local_md.mac_learn_msg);
    }
}
```

p4c-ebpf-psa

```
/* map definition */
struct bpf_map_def mac_learn_digest_0 = {
    .type = BPF_MAP_TYPE_QUEUE,
    .key_size = 0,
    .value_size = sizeof(struct mac_learn_digest_t),
    .max_entries = 100,
};
...
/* deparser */
if (local_md.send_mac_learn_msg) {
    bpf_map_push_elem(&mac_learn_digest_0,
                      &local_md.mac_learn_msg);
}
```

PSA-eBPF Internals: **Deparser**

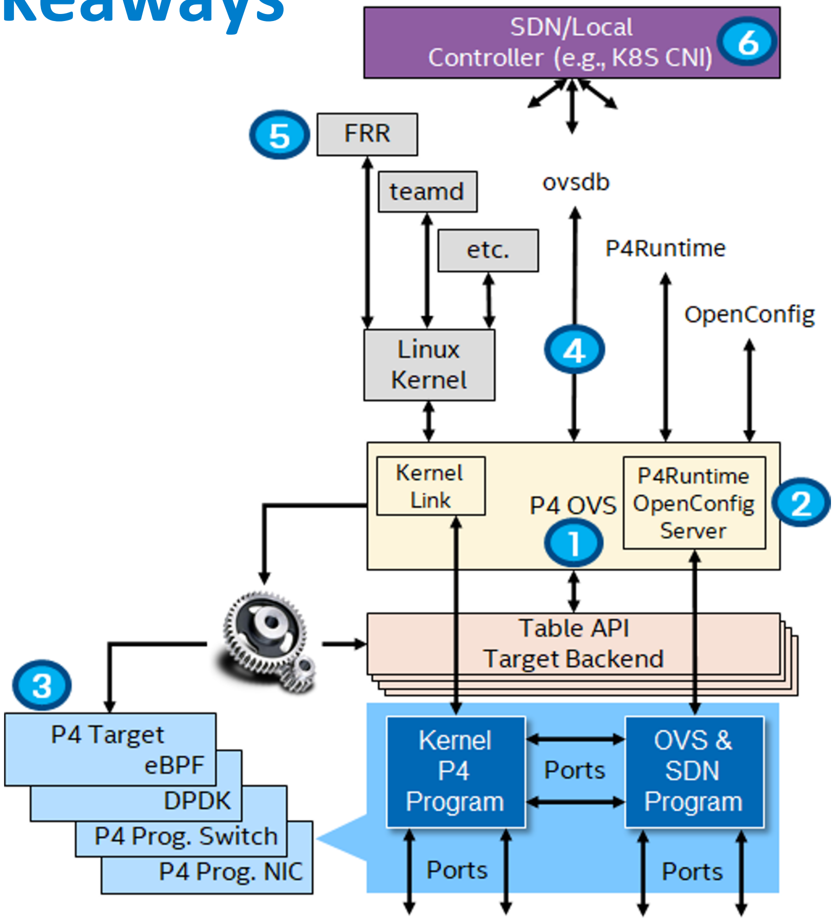
```
control dprs(packet_out packet, Headers_t headers,
                in psa_ingress_output_metadata_t istd,
                ...) {
  apply {
    packet.emit(headers.ethernet);
    packet.emit(headers.ipv4);
  }
}
```

p4c-ebpf-psa

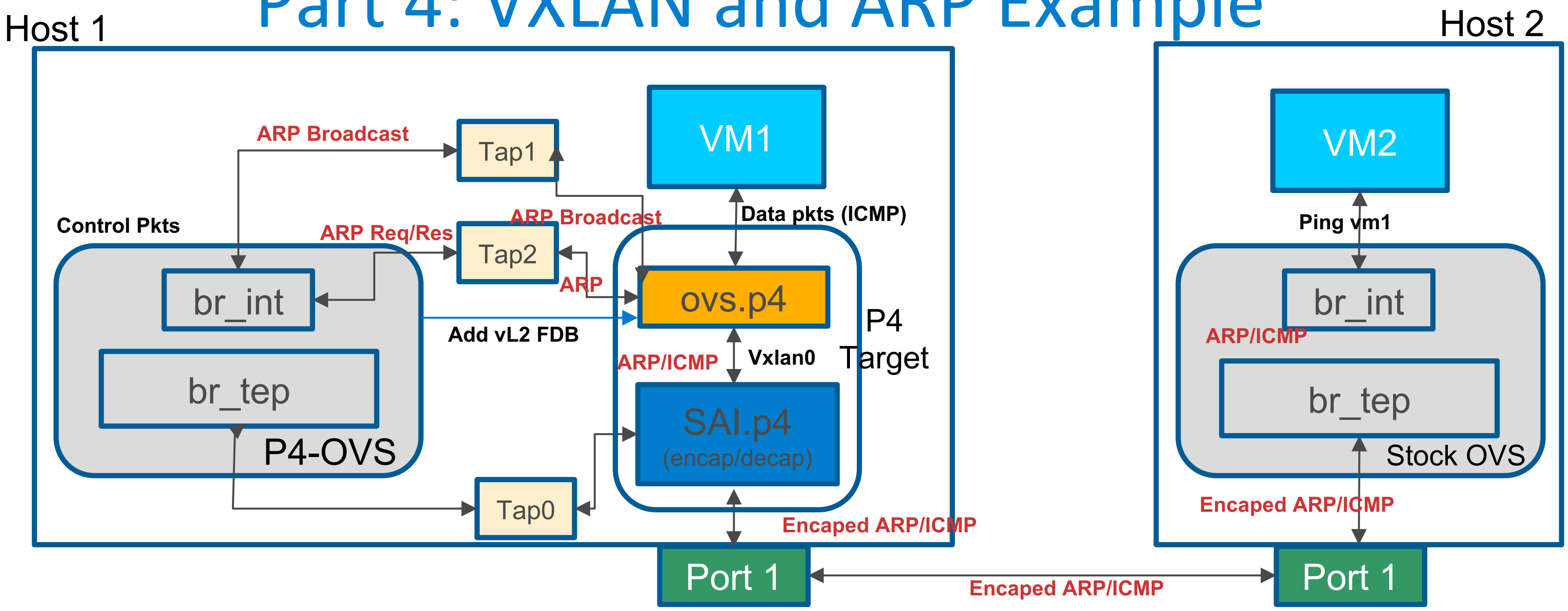
```
if (ostd->clone) {
   do_packet_clones(skb, &clone_session_tbl, istd->clone_session_id,
                            CLONE_I2E);
}
int packetOffsetInBits = 0;
bpf_skb_adjust_room(skb, outHeaderOffset, 1, 0);
if (headers.ethernet.ebpf_valid) {
   ebpf_byte = ((char*)(&headers.ethernet.dstAddr))[0];
   write_byte(pkt, BYTES(packetOffsetInBits) + 0,
                                               (ebpf_byte));
   …
   packetOffsetInBits += 48;
}
if (headers.ipv4.ebpf_valid) {
   …
}
return bpf_redirect(istd->egress_port, 0);
```

Part 3 - **Takeaways**

- eBPF is planned to be an option for the P4-OVS datapath
  - To be published by the end of 2021
- Compliant with the PSA architecture
- Work on performance optimizations (in progress)

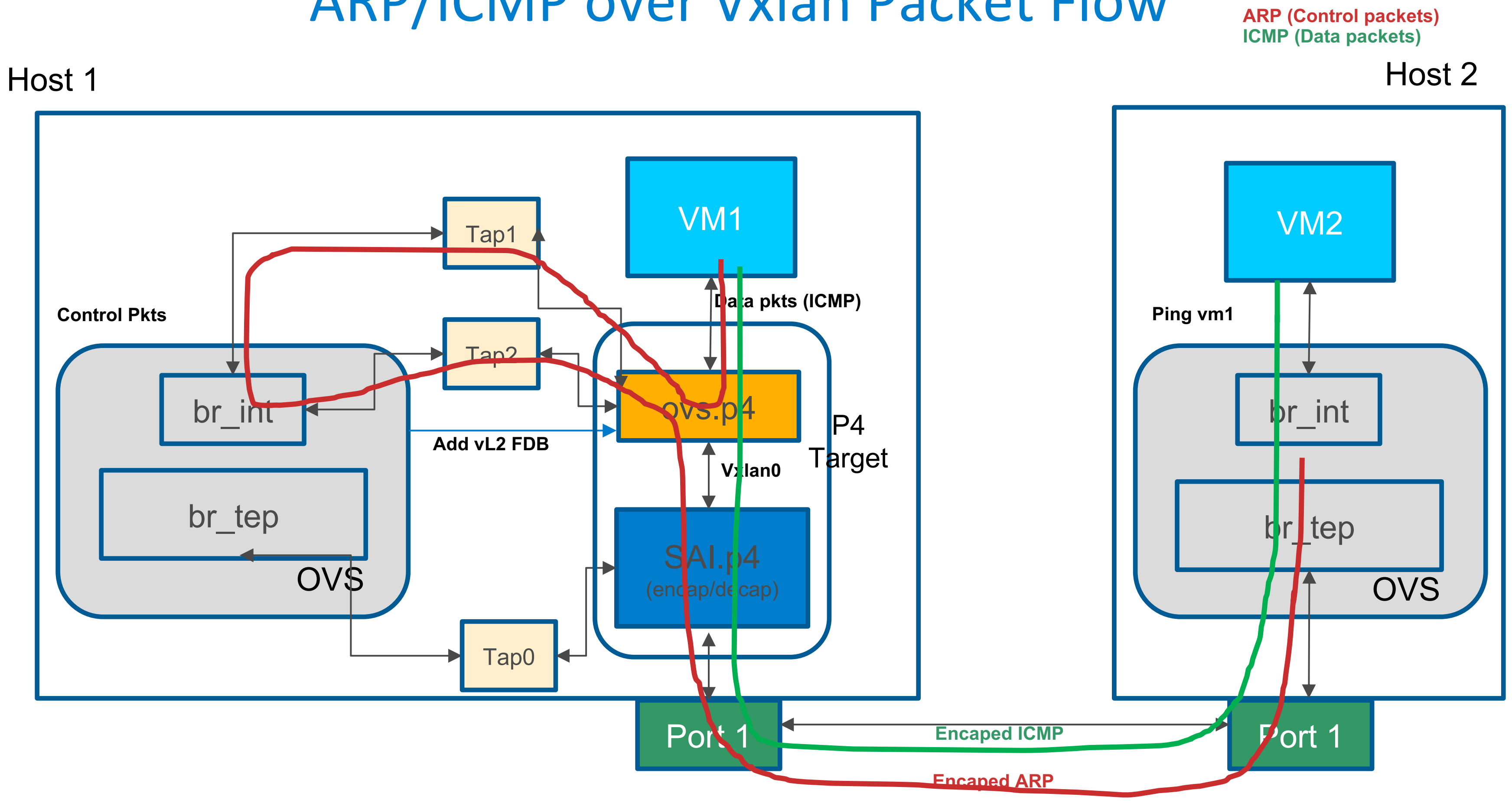


Part 4: VXLAN and ARP Example



- Host 2 – OVS Setup with OVS kernel dataplane. TEP interface with Port 1 as TEP port.
- Host 1 – OVS Setup with OVS kernel dataplane.
  - TEP configured on TAP0.
  - Software P4 target programmed with P4s, connected to OVS using per port TAP.
  - TAP2 used to pass control packets from P4 target to OVS
  - Program ARP entries into ovs.p4 in target via P4-Runtime/Table API

ARP/ICMP over Vxlan Packet Flow



OVS Configs

Host 1

```
[root@host1 ~]# ovs-vsctl show
d1b2062e-efc3-497f-93b4-6a687926b1ac
    Bridge br_int
        Port br_int
            Interface br_int
                type: internal
        Port TAP1
            Interface TAP1
        Port TAP2
            Interface TAP2
        Port vxlan0
            Interface vxlan0
                type: vxlan
                options: {key=flow, remote_ip="70.0.0.1"}
    Bridge br_tep
        Port br_tep
            Interface br_tep
                type: internal
        Port TAP0
            Interface TAP0

[root@host1 ~]# ifconfig br_tep
br_tep: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 70.0.1.1  netmask 255.255.0.0  broadcast 70.0.255.255
```

Host 2

```
[root@host2 ~]# ovs-vsctl show
6d8a2b3e-921d-492f-99e8-542bfd657f40
    Bridge br_int
        Port br_int
            Interface br_int
            type: internal
        Port vm2
            Interface vm2
        Port vxlan0
            Interface vxlan0
                type: vxlan
                options: {key=flow, remote_ip="70.0.1.1"}
    Bridge br_tep
        Port br_tep
            Interface br_tep
                type: internal
        Port port1
            Interface port1

[root@host2 ~]# ifconfig br_tep
br_tep: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 70.0.0.1  netmask 255.255.0.0  broadcast 70.0.255.255
```

Sai.p4 (vxlan encap/decap)

```
action vxlan_encap {
   bit<48> ethernet_dst_addr,  bit<48> ethernet_src_addr, bit<16> ethernet_ether_type,    bit<8> ipv4_ver_ihl, bit<8>
ipv4_diffserv, bit<16> ipv4_total_len, bit<16> ipv4_identification, bit<16> ipv4_flags_offset, bit<8> ipv4_ttl, bit<8>
ipv4_protocol, bit<16> ipv4_hdr_checksum, bit<32> ipv4_src_addr, bit<32> ipv4_dst_addr,  bit<16> udp_src_port, bit<16>
udp_dst_port, bit<16> udp_length, bit<16> udp_checksum, bit<8> vxlan_flags,
bit<24> vxlan_reserved, bit<24> vxlan_vni,  bit<8> vxlan_reserved2,  bit<32> port_out) {

   headers.outer_ethernet.src_addr = ethernet_src_addr;
   headers.outer_ethernet.dst_addr = ethernet_dst_addr;
   headers.outer_ethernet.ether_type = ethernet_ether_type;
   headers.outer_ipv4.ver_ihl = ipv4_ver_ihl;
   headers.outer_ipv4.diffserv = ipv4_diffserv;
   headers.outer_ipv4.total_len = ipv4_total_len;
   headers.outer_ipv4.identification = ipv4_identification;
   headers.outer_ipv4.flags_offset = ipv4_flags_offset;
   headers.outer_ipv4.ttl = ipv4_ttl;
   headers.outer_ipv4.protocol = ipv4_protocol;
   headers.outer_ipv4.hdr_checksum = ipv4_hdr_checksum;
   headers.outer_ipv4.src_addr = ipv4_src_addr;
   headers.outer_ipv4.dst_addr = ipv4_dst_addr;
   headers.outer_udp.src_port = udp_src_port;
   headers.outer_udp.dst_port = udp_dst_port;
   headers.outer_udp.length = udp_length;
   headers.outer_udp.checksum = udp_checksum;
   headers.vxlan.flags = vxlan_flags;
   headers.vxlan.reserved = vxlan_reserved;
   headers.vxlan.vni = vxlan_vni;
   headers.vxlan.reserved2 = vxlan_reserved2;
   ostd.egress_port = (PortId_t)port_out;
   csum.add({headers.outer_ipv4.hdr_checksum, headers.ipv4.total_len});
   headers.outer_ipv4.hdr_checksum = csum.get();
   headers.outer_ipv4.total_len = headers.outer_ipv4.total_len + headers.ipv4.total_len;
    headers.outer_udp.length = headers.outer_udp.length + headers.ipv4.total_len;
}
```

```
action encap {
   vxlan_encap;
   send_to_port1;
}
action vxlan_decap {
   headers.outer_ethernet.setInvalid();
   headers.outer_ipv4.setInvalid();
   headers.outer_udp.setInvalid();
   headers.outer.tunnel.setInvalid();
}
action decap {
   vxlan_decap;
   send_to_ovspipe;
}

table vxlan_encap {
   key = {
      headers.ethernet.dst_addr: exact;
   }
   actions = {
      encap;
      drop;
   }
   const default_action = drop;
}

table vxlan_decap {
   key = {
      headers.ethernet.dst_add: exact;
      headers.outer.ipv4.dst_ip: exact;
      headers.outer.ipv4.src_ip: exact;
   }
   actions = {
      decap;
      drop;
   }
   const default_action = send_to_tap0;
}
```

```
apply {
      if (metadata.isTunnel == TRUE) {
         vxlan_decap.apply();
      }
      if (metadata.ovs == TRUE)
         vxlan_encap.apply();
      }
   }
```

ovs.p4

```
table tap_table {
   key = {
      metadata.in_port;
   }
   actions = {
      send_to_vm(port_no);
   }
   const default_action = noaction;
}

table ovs_table {
   key = {
      headers.ethernet.dst_addr: exact;
   }
   actions = {
      send_to_vm(port_no);
      drop;
   }
   const default_action = drop;
}
```

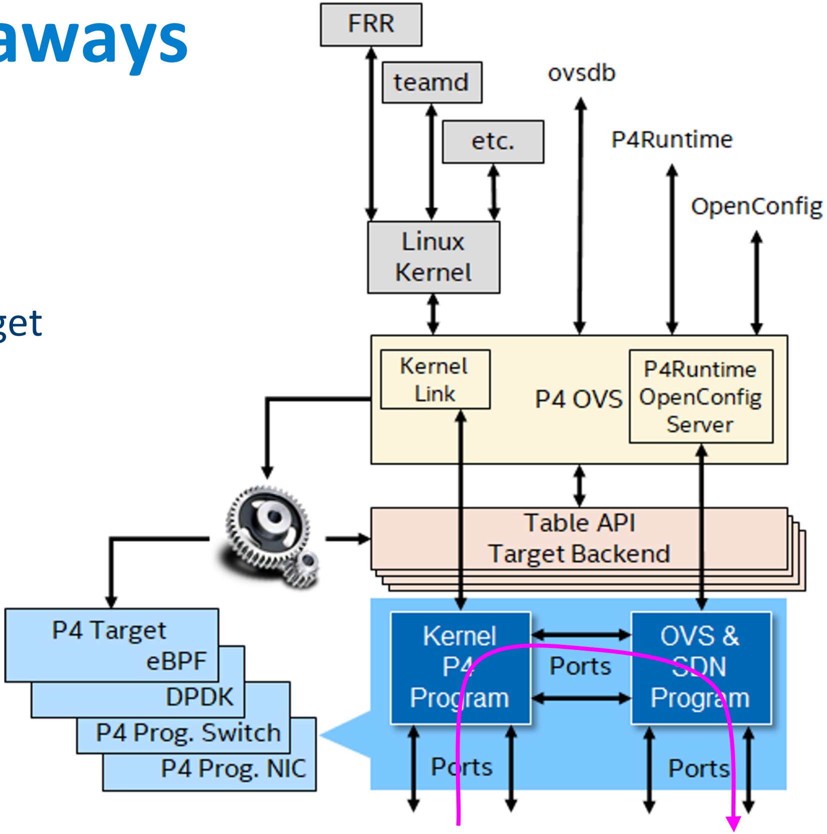
```
apply {
     if (header.arp.isValid())
        send_to_tap2;
     } else if (metadata.isTap)
        tap_table.apply();
     } else if (metadata.isTap2 || metadata.inport
== vxlan0)
        ovs_table.apply();
     }
  }
```

Part 4 - **Takeaways**

- Main OVS config changes to use P4-OVS:
  - Different netdev port handles
- Kernel & OVS dataplane migrates to the P4 target pipeline

Left -> Right:

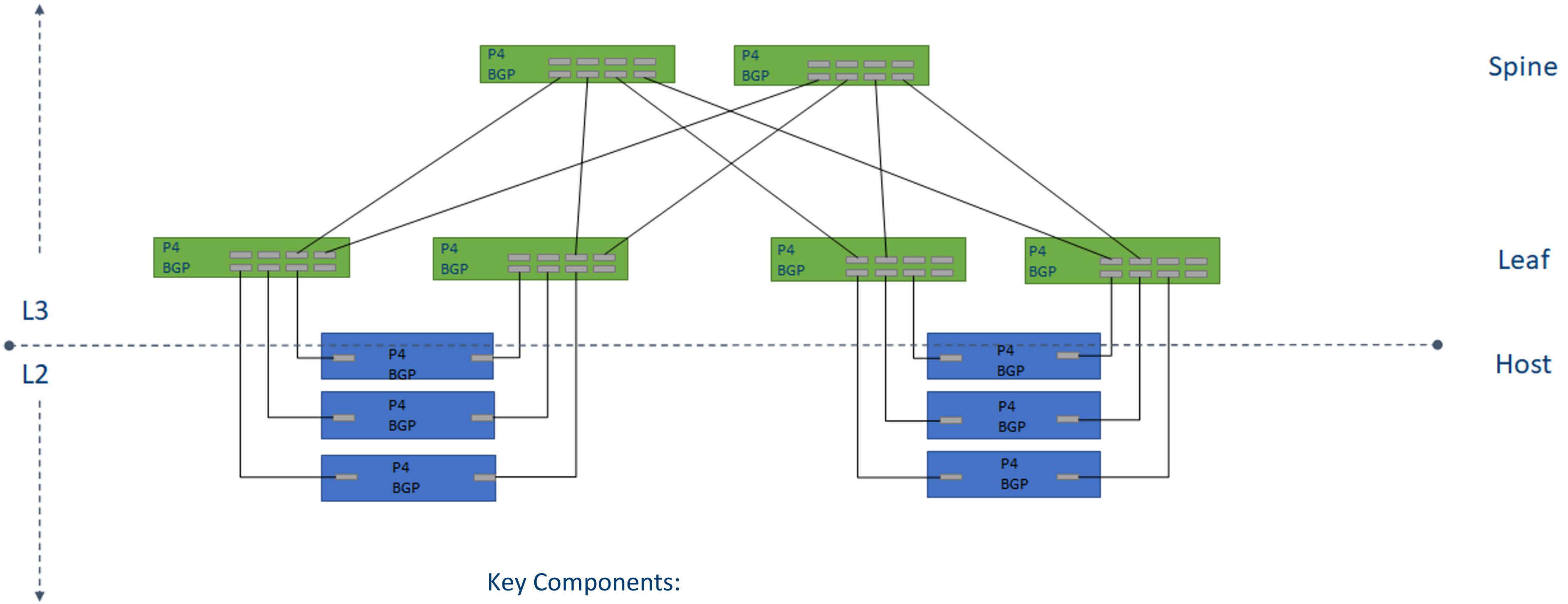
1. SAI.p4 (Kernel P4 Program) receives a packet
2. SAI.p4 decaps packet and sends on tunnel port
3. OVS.p4 matches on the packet
4. OVS.p4 forwards to the virtual port (VM, container, etc. inside the host)



Physical Ports: Encapsulated Packets

Virtual Ports: Un-encap'd Packets

Part 5: Link Redundancy with BGP ECMP and P4

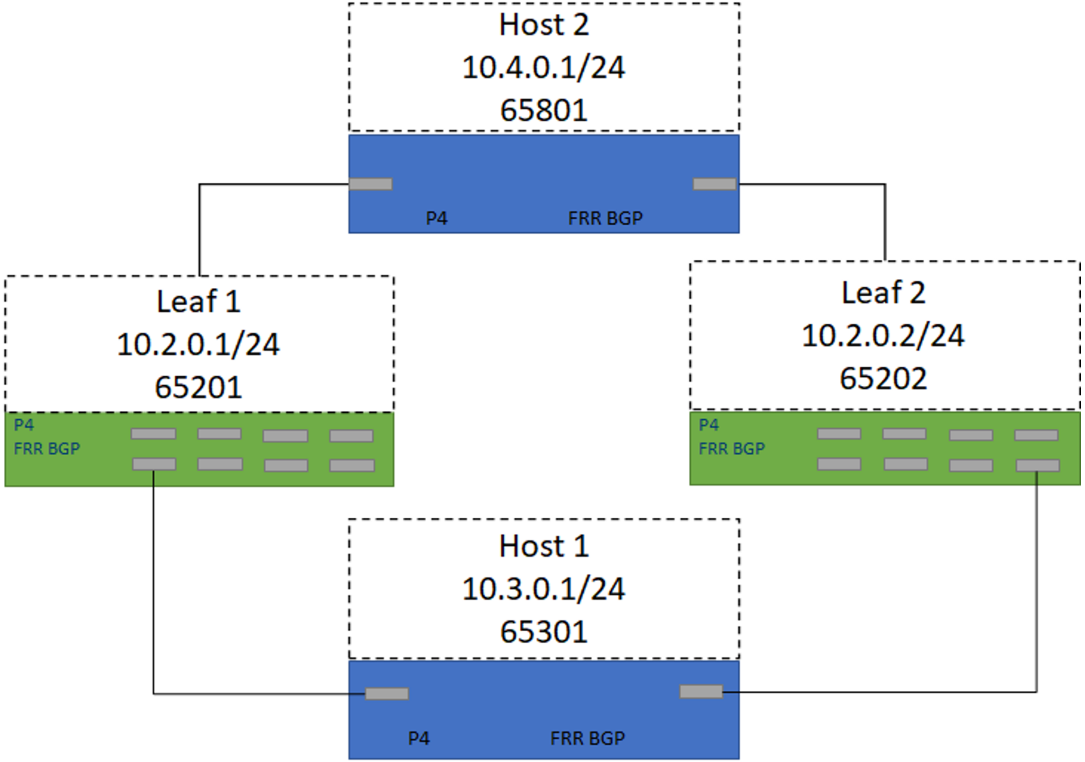


Key Components:

P4 programmable dataplane

Unnumbered BGP for Redundancy and Failover

Demo Setup

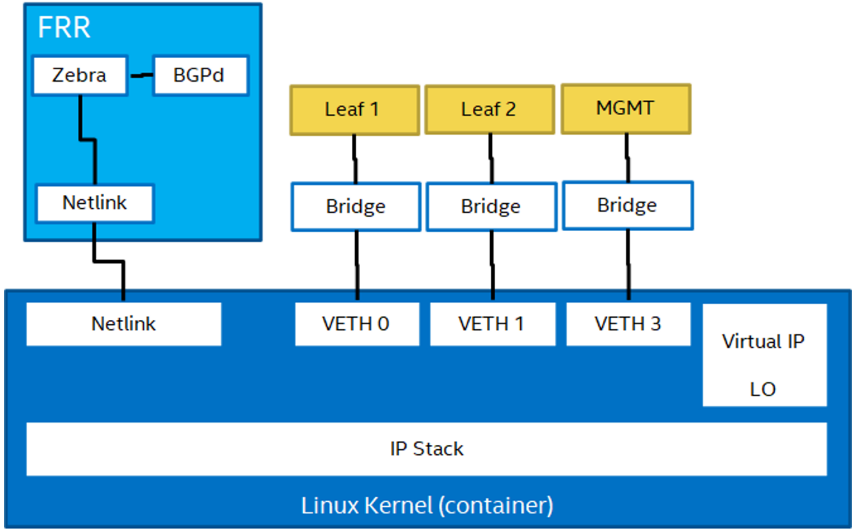


Unnumbered BGP Setup

- IPv6 Link Local addresses
- Link Redundancy via BGP routing
- ECMP Multipath routing

- BGP propagates routes and optimal path is chosen
- Failure on leaf 1 results in rerouting of traffic to leaf 2 and vice versa.
- Setup is emulated with 4 containers each with 3 VETH interfaces each, interconnected via bridges.
- FRR provides the BGP routing support.
- Linux kernel routing and Linux bridges
- Virtual IP address on loopback port

Linux Kernel (Standalone)



Linux Kernel Functionality

- Linux kernel provides all routing support
- Routing tables will forward based upon BGP priorities packets to the Virtual IP assigned to loopback interface.

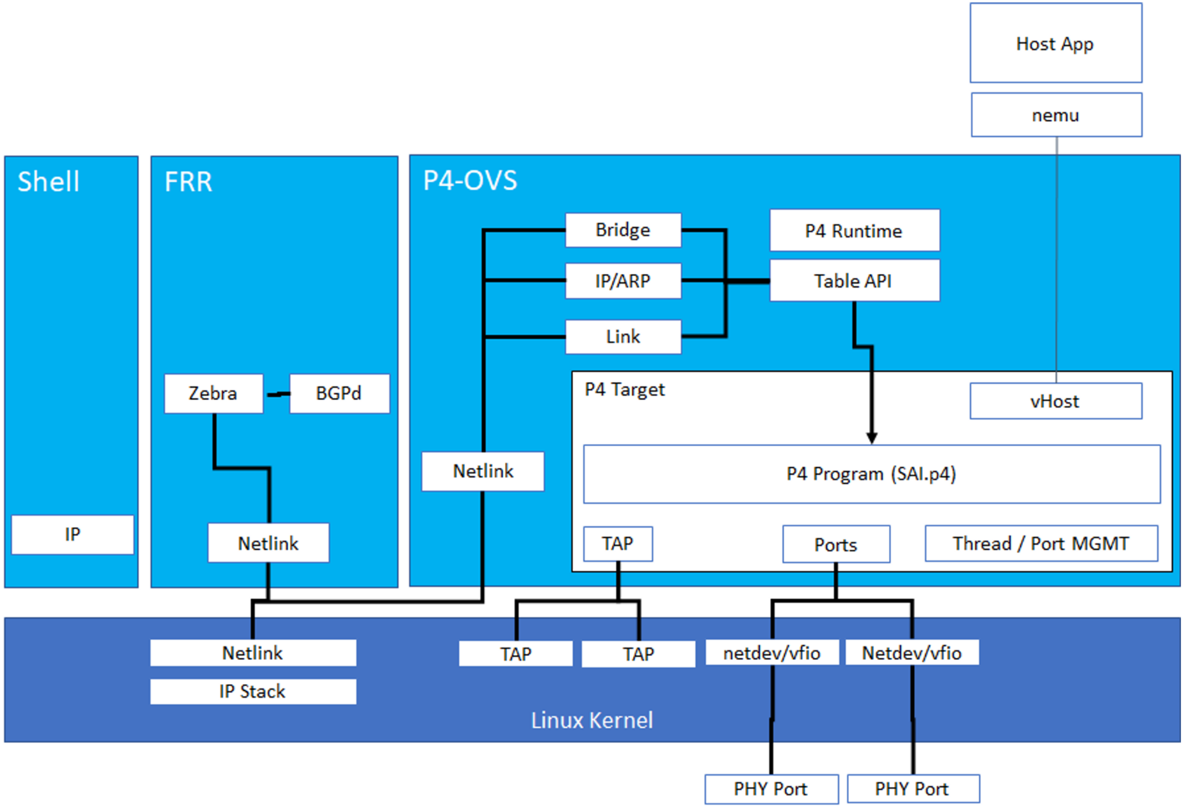
Control Plane

- FRR is utilized to provided unnumbered BGP support and program the Linux kernel routing tables.
- Node BGP Config (configuration to be provided)

Demo Summary

- Demonstrate BGP Unnumbered ECMP routing with Link Redundancy and Failover using Linux kernel stack.
- Show how P4 coupled with Linux kernel stack hooks via netlink will provide equivalent functionality.
- Netlink, P4 Runtime, and Table API integration to be released soon

Linux Kernel + P4 OVS



Control Plane :

- FRR and Linux Stack

Forwarding Plane

- P4 Target
- SAI.P4 program

Summary Operation

- Configuration via Linux shell commands and FRR BGPd
- Network state reflected via Netlink into P4 Runtime environment
- P4 Target executes P4 program SAI.P4 to forward traffic between physical and logical ports.

ECMP Action Selector

```
control ipv4_fib(inout H hdr, inout M meta) {

  action hit(nexthop_id_t nexthop) {
    meta.nexthop_id = nexthop;
  }
  action miss() {
    meta.drop = true;  // Drop packet.
  }

  ActionSelector(PSA_HashAlgorithm_t.CRC16, 16, 10) as;
  ...
```

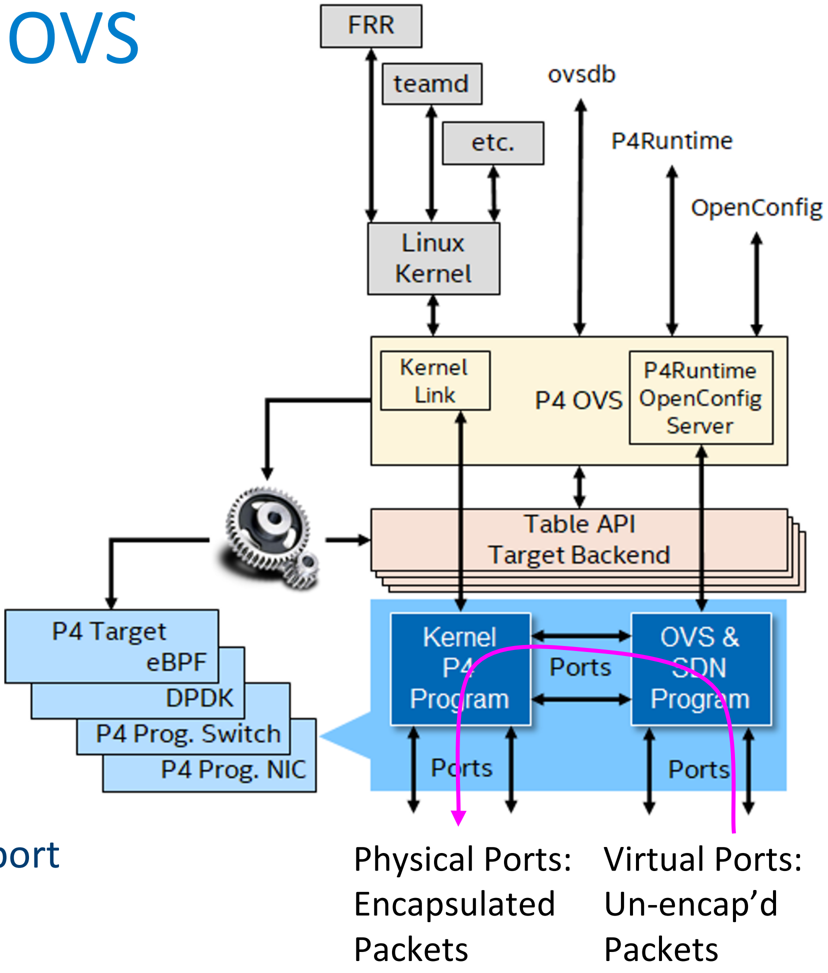
```
  ...
  table ipv4_route {
    key = {
      meta.router : exact;  //Virtual Router
      hdr.ipv4.dst_addr : lpm;  //Dest IP
      meta.l4_src_port : selector;
      meta.l4_dst_port : selector;
      hdr.ipv4.src_addr : selector;
      hdr.ipv4.dst_addr : selector;
    }
    actions = { hit; miss; }
    default_action = miss;
    psa_implementation = as;
  }

  apply {
    if (hdr.ipv4.isValid()) { ipv4_route.apply(); }
    ...
}
```

ECMP + OVS

- Table API supports multiple pipelines, each with separate control planes
- Interconnection of data-planes is accomplished via logical internal ports

Right -> Left:

1. OVS receives a packet from a virtual port
2. OVS vL2 forwards to the tunnel port
3. SAI.p4 (Kernel P4 Program) receives a packet
4. SAI.p4 Encaps packet
5. SAI.p4 uses routing table to pick nexthop
6. Action Selector picks nexthop MAC & physical port

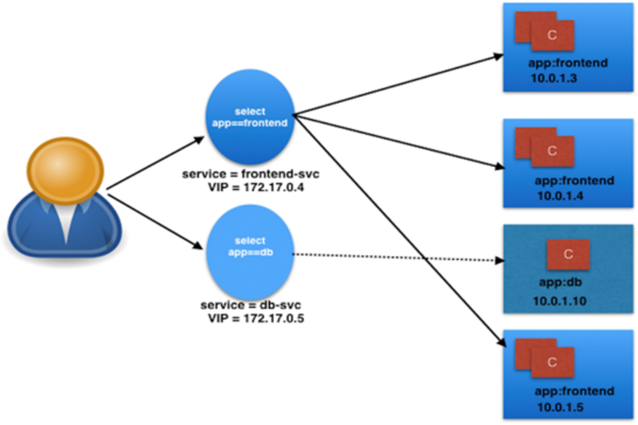


Part 6: Kubernetes Service Intro

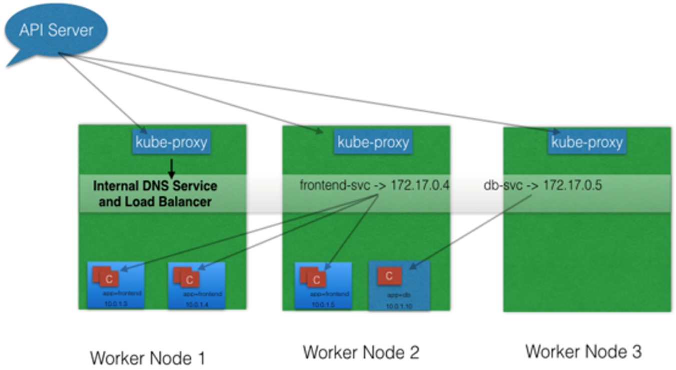
A sample backend service

- Deployment using a yaml file
- Listens on port 80. Has a selector
- A service Object to send traffic to multiple backend replicas
- A Service object creates a persistent IP and a DNS name
- Persistent IP is routable only inside the cluster, known as ClusterIP (VIP)
- A Service uses selectors to find the Pods that it routes traffic to
- Endpoints (c) may come and go.

```
kind: Service
apiVersion: v1
metadata:
  name: frontend-svc
spec:
  selector:
    app: frontend
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
```

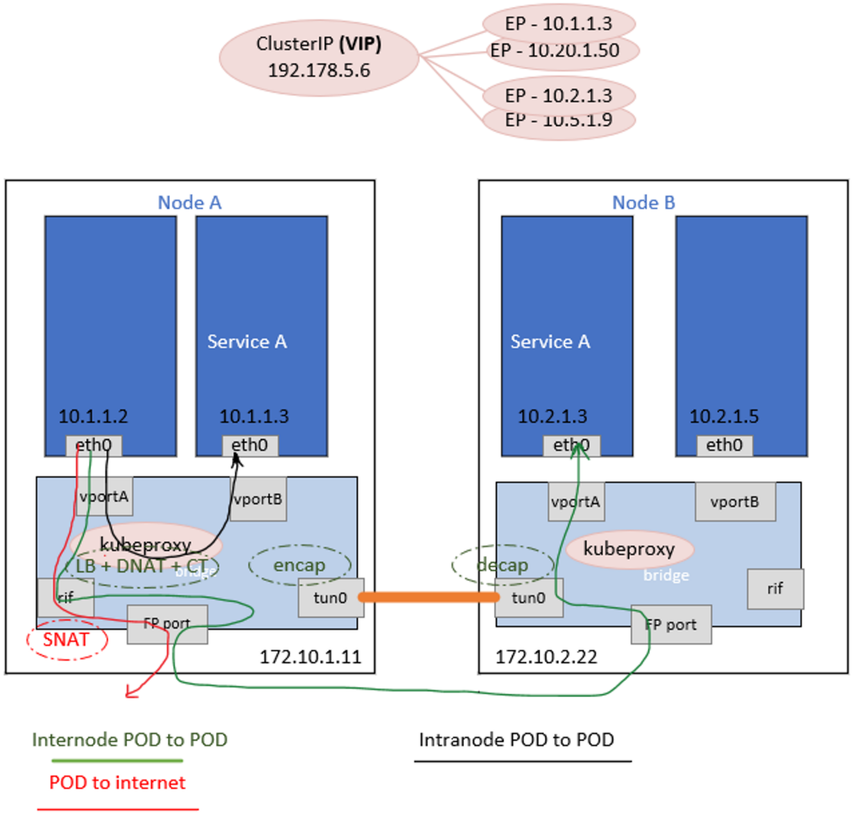


Grouping of Pods using the Service object



K8s Node Flows

- Services
  - Port, protocol, Namespace
  - Service Endpoints - IP
  - Load balancing - VIP
- Flow Affinity
  - Connection tracking
- NAT
  - SNAT, DNAT
- Ingress/Egress Policy Rules
  - IP group, IP/port/proto
- Connectivity
  - Routing, Forwarding, tunneling



P4-Antrea Agent – Networking Flows

Antrea sets up OVS & the kernel for OVS bridging over tunnels.

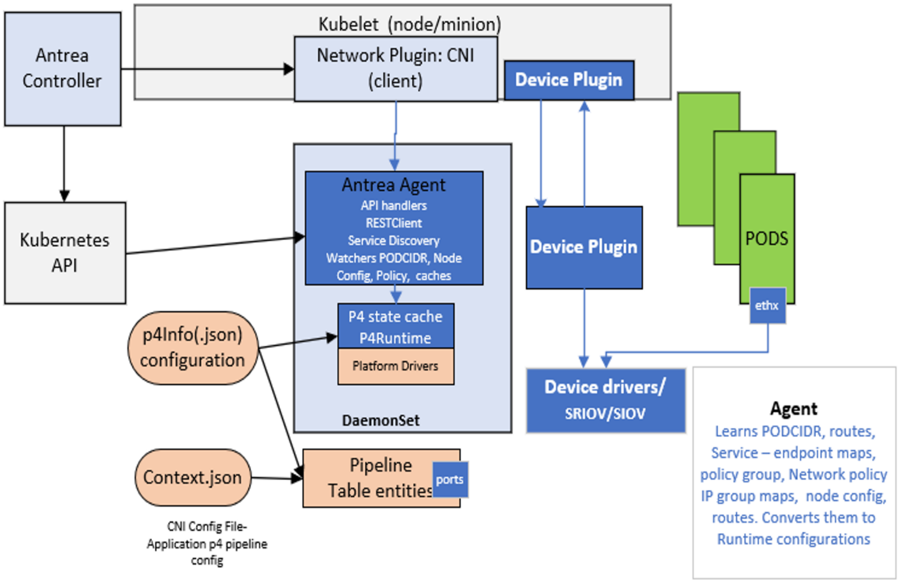
P4-Antrea to use P4 instead of OpenFlow for mappings and store caches for programmable flows

Bridge Setup using P4-OVS

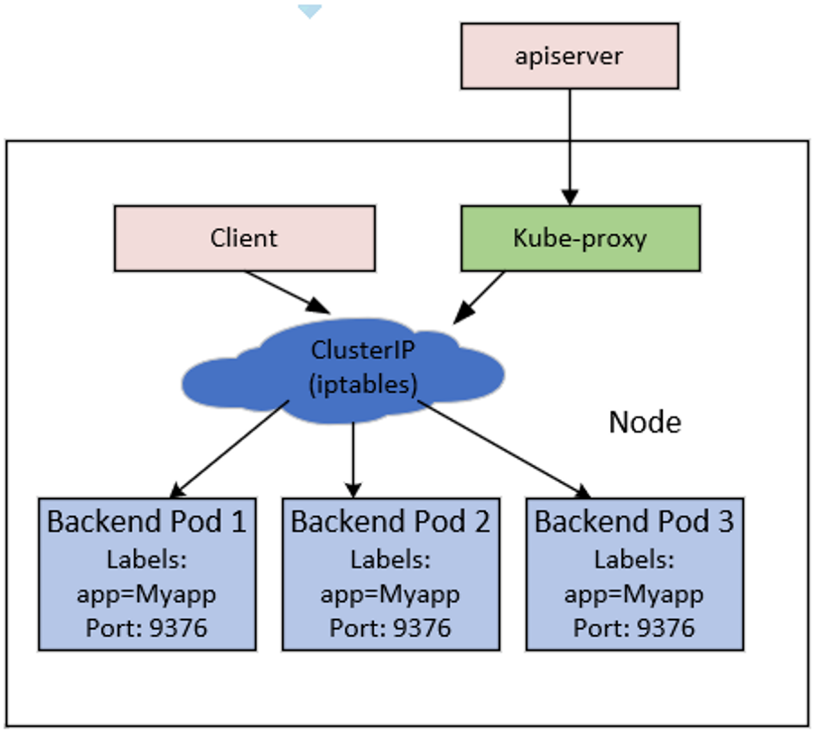
- Ports, interfaces, pod vports, conf port

P4Flow Pipeline

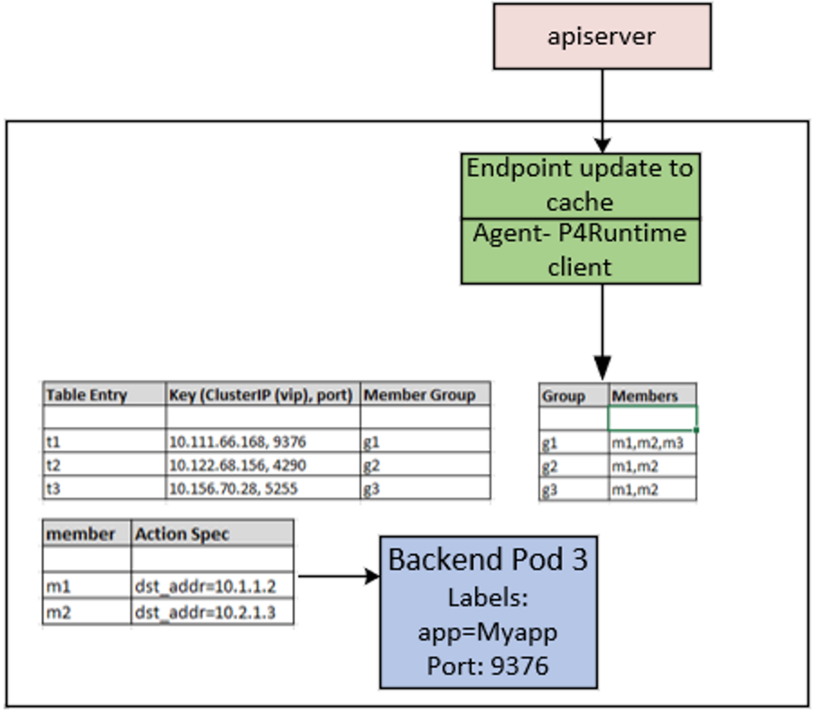
- ClusterServiceFlows – kubeproxy, nat, flow affinity
- GatewayFlows - l2, l3, PODCIDR routing, arpresponder
- PolicyFlows and security



IPTables Proxy Mode

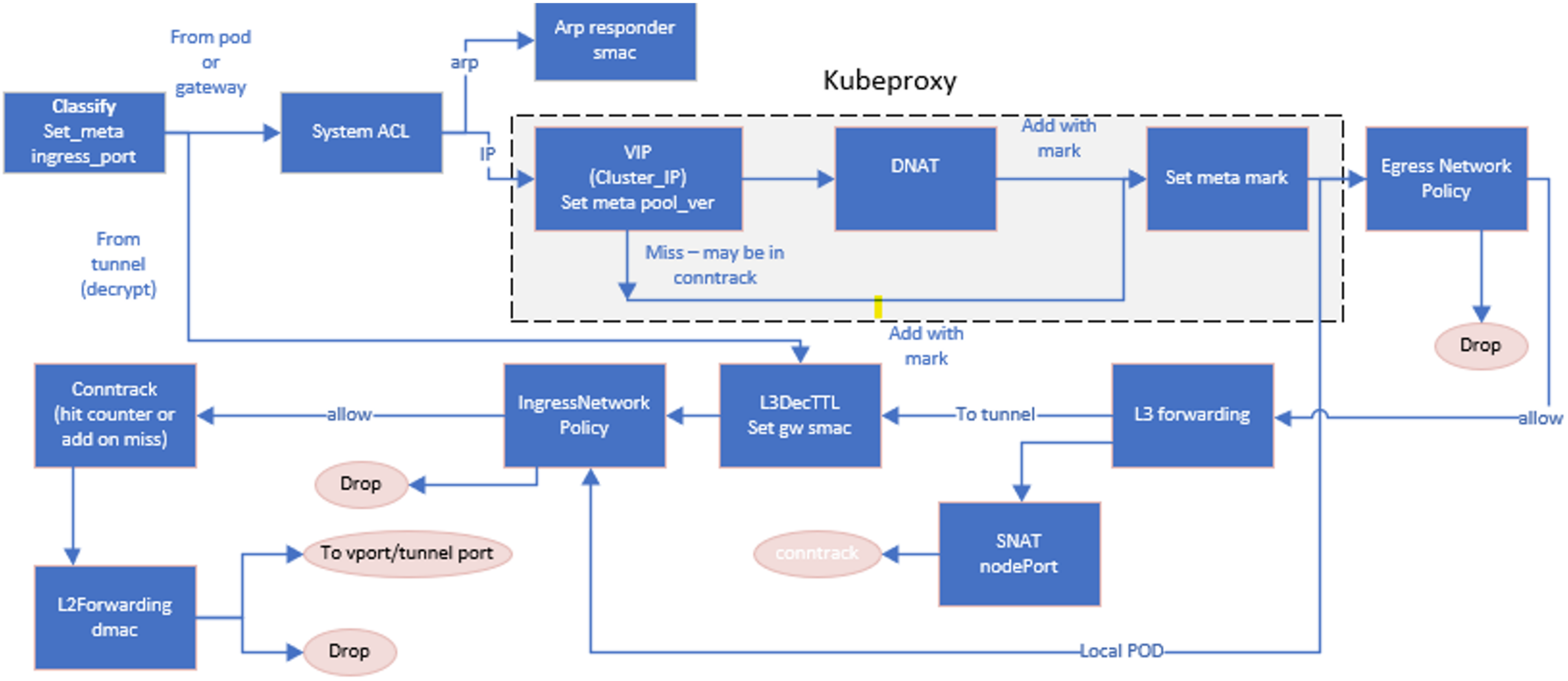


P4 ActionSelector



Action Selectors dynamically select the action specification to apply upon matching a table entry

P4 pipeline with kubeproxy

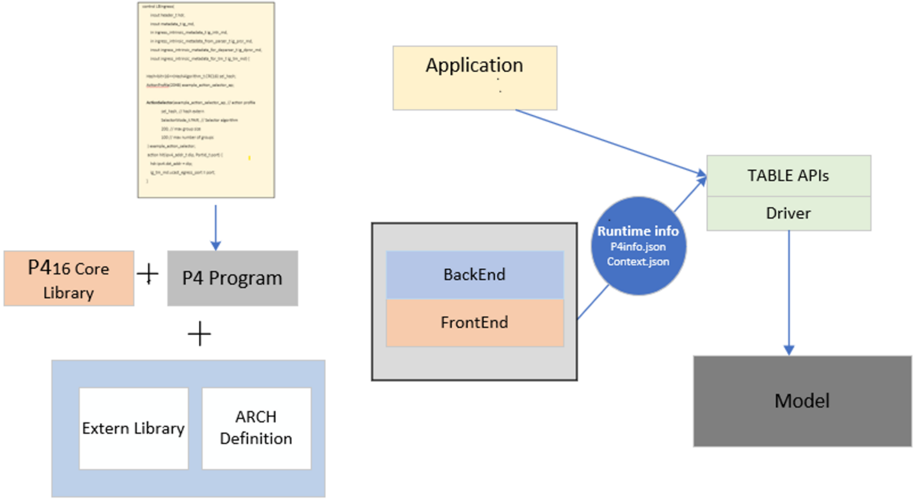


K8s Load Balancer Action Selector

```
control LBIngress(inout headers_t hdr,
                  inout local_metadata_t meta) {

  action hit(ipv4_addr_t dip, PortId_t port) {
    hdr.ipv4.dst_addr = dip;
    meta.oport = port;
  }
  action set_md(bit<16> idx) {
    meta.md = idx;
  }
  action miss() {
    meta.drop = true;  // Drop packet.
  }

  ActionSelector(PSA_HashAlgorithm_t.CRC16, 2048, 16) as;
  ...
```

```
  ...
  table indirect_with_selection {
    key = {
      meta.bridge : exact;  //Virtual Bridge
      meta.l4_port : exact;  //Srv Port
      hdr.ipv4.dst_addr : exact;  //ClusterIP
      hdr.ethernet.src_addr : selector;
      hdr.ethernet.dst_addr : selector;
      hdr.ipv4.src_addr : selector;
      hdr.ipv4.dst_addr : selector;
    }
    actions = { hit; set_md; miss; }
    default_action = miss;
    psa_implementation = as;
  }

  apply { indirect_with_selection.apply(); }
}
```

Example Setup - ActionSelector

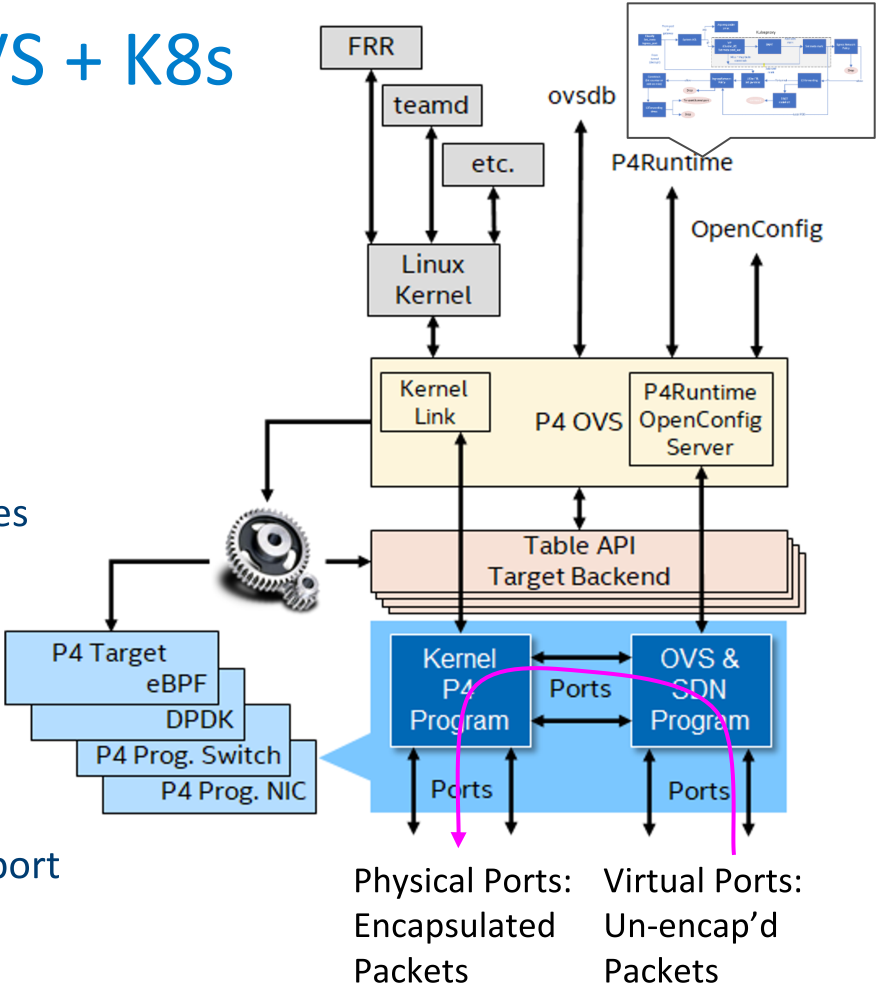



```
05-12 01:02:04.084352:    :0xbc3:-:<0,0,0>:Ingress Parser state 'parse_ipv4.$split_0'
05-12 01:02:04.084387:        :0xbc3:-:<0,0,0>:Ingress Parser state 'parse_ipv4.$split_0': extracted 0xf979a126, up
05-12 01:02:04.084413:        :0xbc3:-:<0,0,->:Ingress Headers:
05-12 01:02:04.084431:        :0xbc3:-:<0,0,->:Header hdr.ethernet.$valid is valid
05-12 01:02:04.084444:        :0xbc3:-:<0,0,->:Header hdr.ipv4.$valid is valid
05-12 01:02:04.084470:    :0xbc3:-:<0,0,->:----------- Stage 0 -----------
05-12 01:02:04.084737:    :0xbc3:-:<0,0,0>:Ingress : Table SwitchIngress.forward is hit
05-12 01:02:04.084762:        :0xbc3:-:<0,0,0>:Key:
05-12 01:02:04.084783:        :0xbc3:-:<0,0,0>:        ig_intr_md.ingress_port[8:0] = 0x1
05-12 01:02:04.084803:        :0xbc3:-:<0,0,0>:Execute Action: SwitchIngress.set_md
05-12 01:02:04.084832:        :0xbc3:-:<0,0,0>:Action Results:
05-12 01:02:04.084849:        :0xbc3:-:<0,0,0>:        ----- ModifyFieldPrimitive -----
05-12 01:02:04.084863:        :0xbc3:-:<0,0,0>:        Operation:
05-12 01:02:04.084875:        :0xbc3:-:<0,0,0>:        set
05-12 01:02:04.084888:        :0xbc3:-:<0,0,0>:        Destination:
05-12 01:02:04.084901:        :0xbc3:-:<0,0,0>:        ig_md.md[15:0] = 0x73F
05-12 01:02:04.084914:        :0xbc3:-:<0,0,0>:        mask=0xFFFF
05-12 01:02:04.084926:        :0xbc3:-:<0,0,0>:        Source 1:
05-12 01:02:04.084938:        :0xbc3:-:<0,0,0>:        idx=action_param
05-12 01:02:04.084952:        :0xbc3:-:<0,0,0>:Next Table = tbl_tna_action_selector140
05-12 01:02:04.084970:    :0xbc3:-:<0,0,->:----------- Stage 1 -----------
05-12 01:02:04.085213:    :0xbc3:-:<0,0,1>:Ingress : Table tbl_tna_action_selector140 is miss
05-12 01:02:04.085237:        :0xbc3:-:<0,0,1>:Key:
05-12 01:02:04.085255:        :0xbc3:-:<0,0,1>:Execute Default Action: tna_action_selector140
05-12 01:02:04.085282:        :0xbc3:-:<0,0,1>:Action Results:
05-12 01:02:04.085299:        :0xbc3:-:<0,0,1>:        ----- ModifyFieldPrimitive -----
05-12 01:02:04.085312:        :0xbc3:-:<0,0,1>:        Operation:
05-12 01:02:04.085325:        :0xbc3:-:<0,0,1>:        set
05-12 01:02:04.085338:        :0xbc3:-:<0,0,1>:        Destination:
05-12 01:02:04.085350:        :0xbc3:-:<0,0,1>:        ig_intr_md_for_tm.bypass_egress[0:0] = 0x1
05-12 01:02:04.085362:        :0xbc3:-:<0,0,1>:        mask=0x1
05-12 01:02:04.085375:        :0xbc3:-:<0,0,1>:        Source 1:
05-12 01:02:04.085387:        :0xbc3:-:<0,0,1>:        Val=0x1
05-12 01:02:04.085399:        :0xbc3:-:<0,0,1>:Next Table = SwitchIngress.set_dest
05-12 01:02:04.085416:    :0xbc3:-:<0,0,1>:Ingress : Table SwitchIngress.set_dest is hit
05-12 01:02:04.085430:        :0xbc3:-:<0,0,1>:Key:
05-12 01:02:04.085447:        :0xbc3:-:<0,0,1>:        ig_md.md[15:0] = 0x73F
05-12 01:02:04.085465:        :0xbc3:-:<0,0,1>:Execute Action: SwitchIngress.hit
05-12 01:02:04.085499:        :0xbc3:-:<0,0,1>:Action Results:
05-12 01:02:04.085518:        :0xbc3:-:<0,0,1>:        ----- ModifyFieldPrimitive -----
05-12 01:02:04.085531:        :0xbc3:-:<0,0,1>:        Operation:
05-12 01:02:04.085544:        :0xbc3:-:<0,0,1>:        set
05-12 01:02:04.085556:        :0xbc3:-:<0,0,1>:        Destination:
05-12 01:02:04.085568:        :0xbc3:-:<0,0,1>:        ig_intr_md_for_tm.ucast_egress_port[8:0] = 0xC
05-12 01:02:04.085580:        :0xbc3:-:<0,0,1>:        mask=0x1FF
05-12 01:02:04.085593:        :0xbc3:-:<0,0,1>:        Source 1:
05-12 01:02:04.085605:        :0xbc3:-:<0,0,1>:        port=action_param
05-12 01:02:04.085618:        :0xbc3:-:<0,0,1>:Next Table = --END_OF_PIPELINE--
05-12 01:02:04.085636:    :0xbc3:-:<0,0,->:----------- Stage 2 -----------
```

ECMP + OVS + K8s

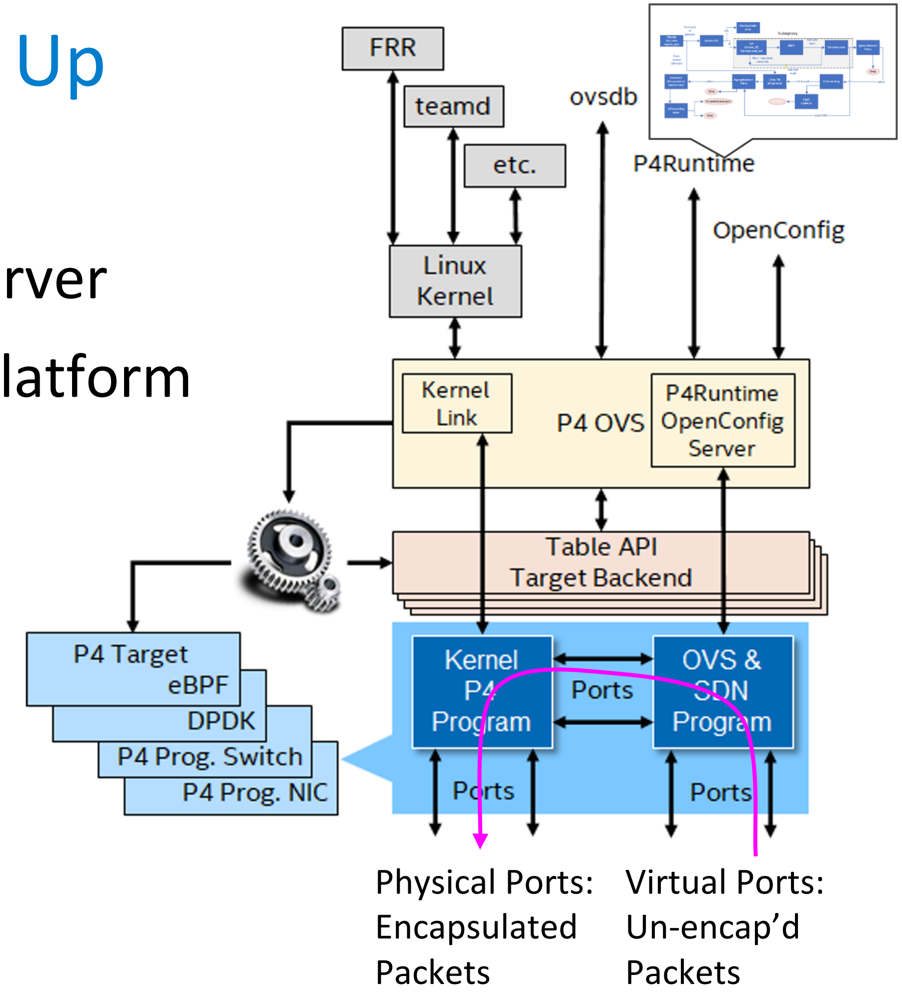
Right -> Left:

1. OVS receives a packet from a virtual port
2. Antrea P4 is run that applies egress ACLs, applies the kubeproxy load balancer, and ingress ACLs
3. Antrea P4 forwards to the tunnel port
4. SAI.p4 (Kernel P4 Program) receives a packet
5. SAI.p4 Encaps packet
6. SAI.p4 uses routing table to pick nexthop
7. Action Selector picks nexthop MAC & physical port



Wrap Up

- P4 Programmable Standard Server
- Network as a Programmable Platform
- P4 programmability is:
  1. Used to satisfy the existing control plane requirements
  2. Used to customize & enhance on top of existing software



MAY 18-20 2021 P4 Workshop

Hosted by ONF

Thank You

Namrata Limaye, Intel

Brian O'Connor, ONF

Tomasz Osinski, ONF

Mateusz Kossakowski, Orange

Gerald Rogers, Intel

Nupur Jain, Intel

Dan Daly, Intel