Data Plane Programmability

Introduction to P4

Tomasz Osiński, PhD

Agenda

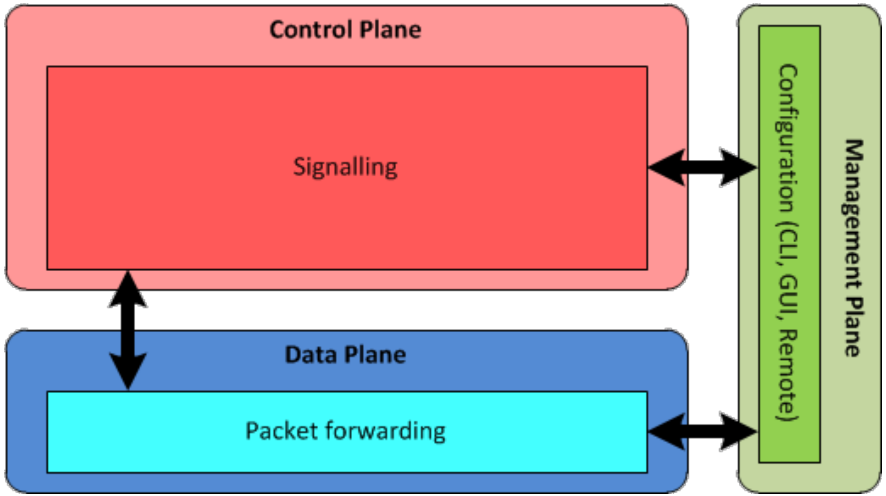
- Introduction to Data Plane Programmability
  - Motivation
  - Concept
  - Relationship with OpenFlow & SDN
- P4
  - Architecture
  - Language
  - Toolchain
  - Demo
- P4Runtime protocol
- Use cases
- Research activities
- Summary

Basics

Telecommunications Architecture (quick reminder)

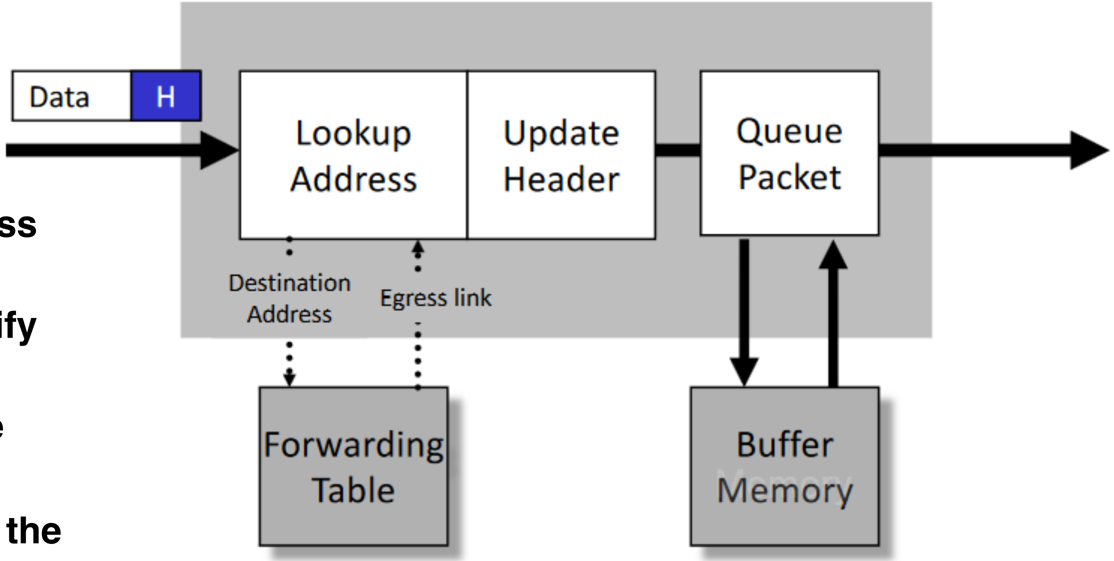
- **Classical architecture consists of 3 planes**
  - **Group of protocols**
- **Differences between planes:**
  - **Kind of traffic**
  - **Performance**
  - **Programming languages**
  - **Different execution environments**

Exercise!



What the Data Plane is?

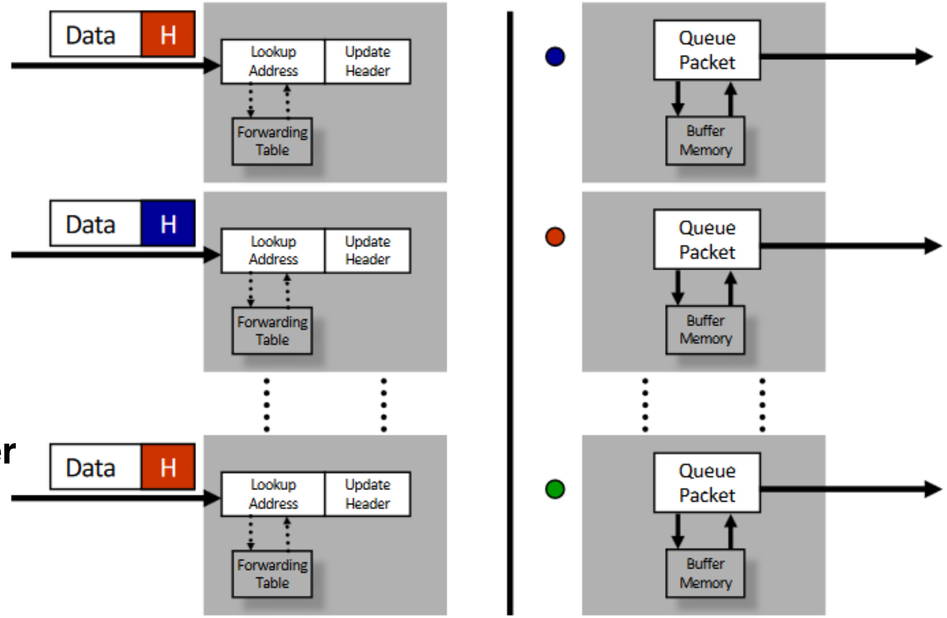
- **The mechanism responsible for packet processing based on the protocol's headers**
- **Operations:**
  1. **Parse – read the packet**
  2. **Lookup – determine how to process the packet**
  3. **Update Header (optionally) – modify packet header's field(s)**
  4. **Switch/Route – send packet to the output port**
  5. **Queue Packet – put the packet on the output queue**
  6. **Deparse – create packet and send on the wire**



Source: CS344, Stanford University

Internet Router – the real-world example

1. **Parse**
   - **Examine IP packet – read the packet's header and determine packet length**
2. **Lookup**
   - **Check how to forward packet based on destination IP address**
3. **Update Header**
   - **Decrement TTL, calculate the IP checksum**
4. **Switch/Route packet**
   - **Forward to the correct egress port**
5. **Queue packet**
   - **Find the Ethernet DA for the next hop router**
   - **Put the packet on the egress port's queue**
6. **Deparse**
   - **Create Ethernet frame and send it**

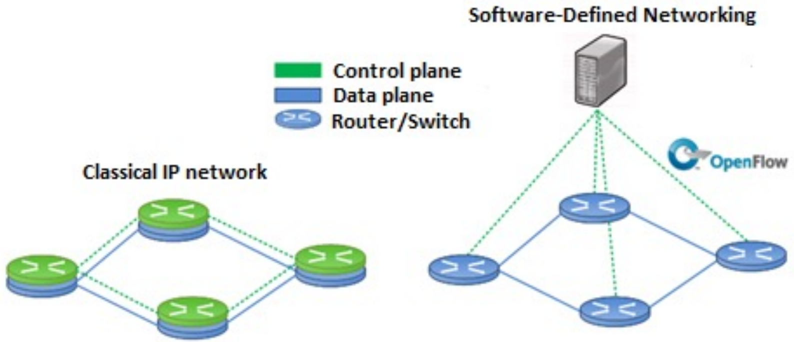


Source: CS344, Stanford University

Introduction to Data Plane Programmability

Software-Defined Networking

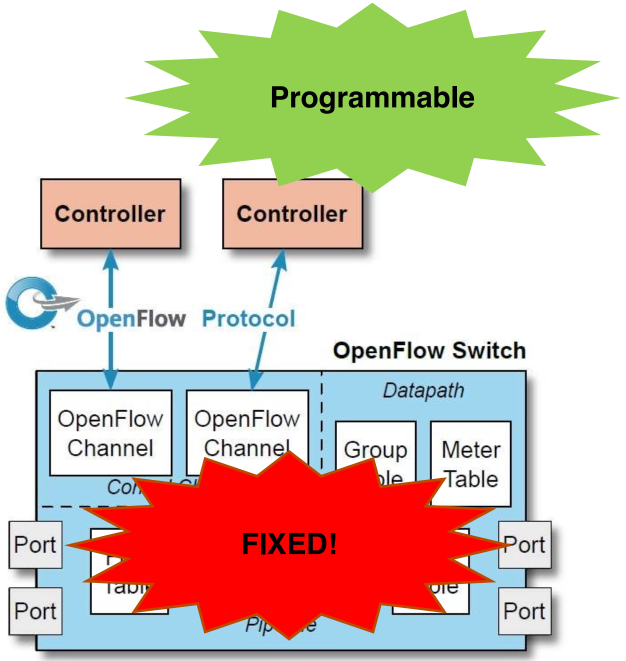
- Software-Defined Networking (SDN)
  - Data plane and control plane decoupling
  - Logically centralized control plane
  - Programmable control plane and Open API to data plane
- OpenFlow protocol
- Issues:
  - <u>Data plane protocol evolution requires changes to standards (e.g. OpenFlow specification)</u>
  - <u>Data plane scalability (due to TCAM)</u>



Why Data Plane Programmability?

- **Limitations of OpenFlow:**
  - **Examples:**
    - **Support for GTP protocol (vEPC)**
    - **Tunnelling protocols (e.g. MPLS over UDP)**
    - **Service Function Chaining (NSH headers)**
- **Vendor lock-in (oops?)**

Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

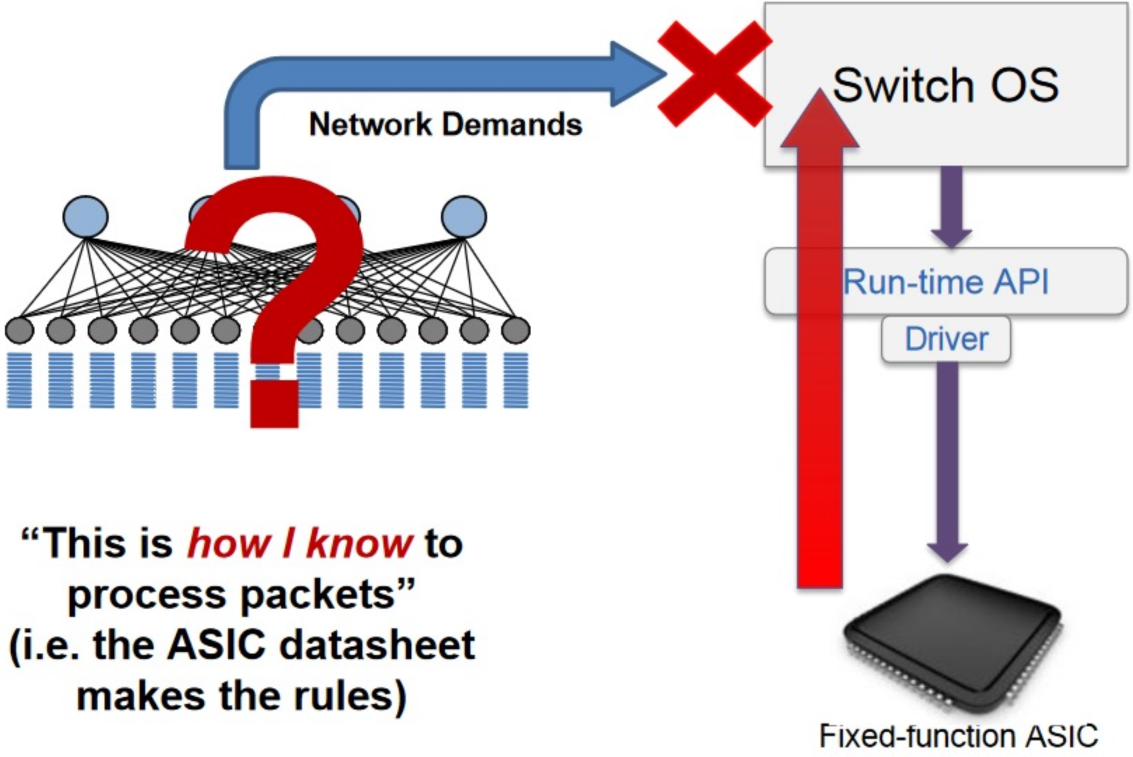


Source: ONF „OpenFlow Switch specification”

Motivation (1/2)

Status quo: Bottom-Up design

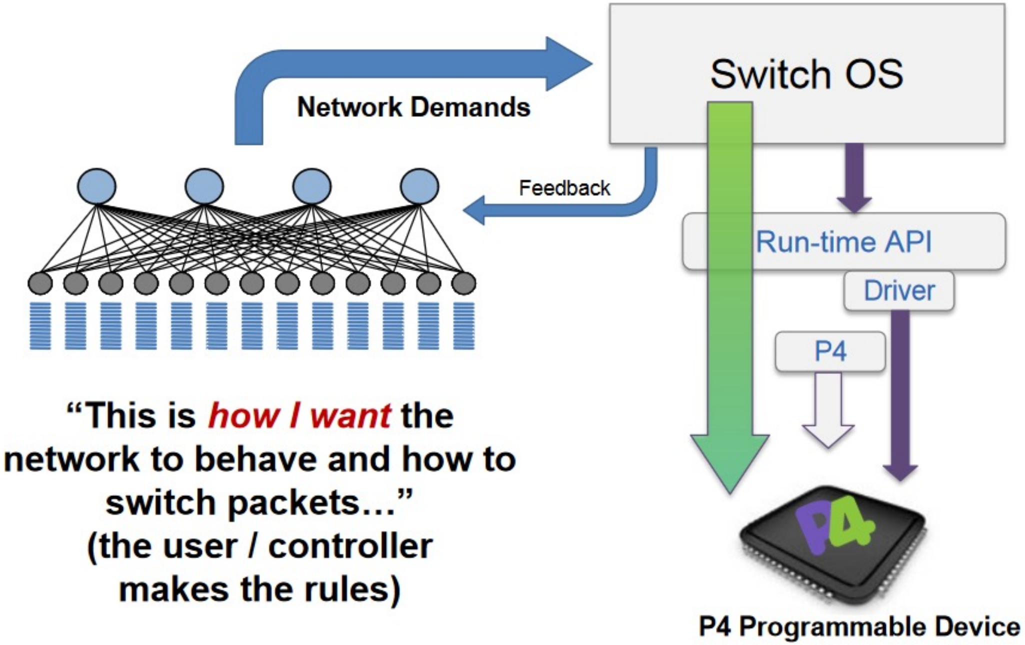
- Slow release cycle (for ASICs ~4 years)
  - VxLAN case study
    - Standard: 2010 by Cisco & VMWare
    - Implementation: 2014



Source: https://p4.org/assets/P4_tutorial_01_basics.gslide.pdf

Motivation (2/2)

A better approach: Top-Down approach

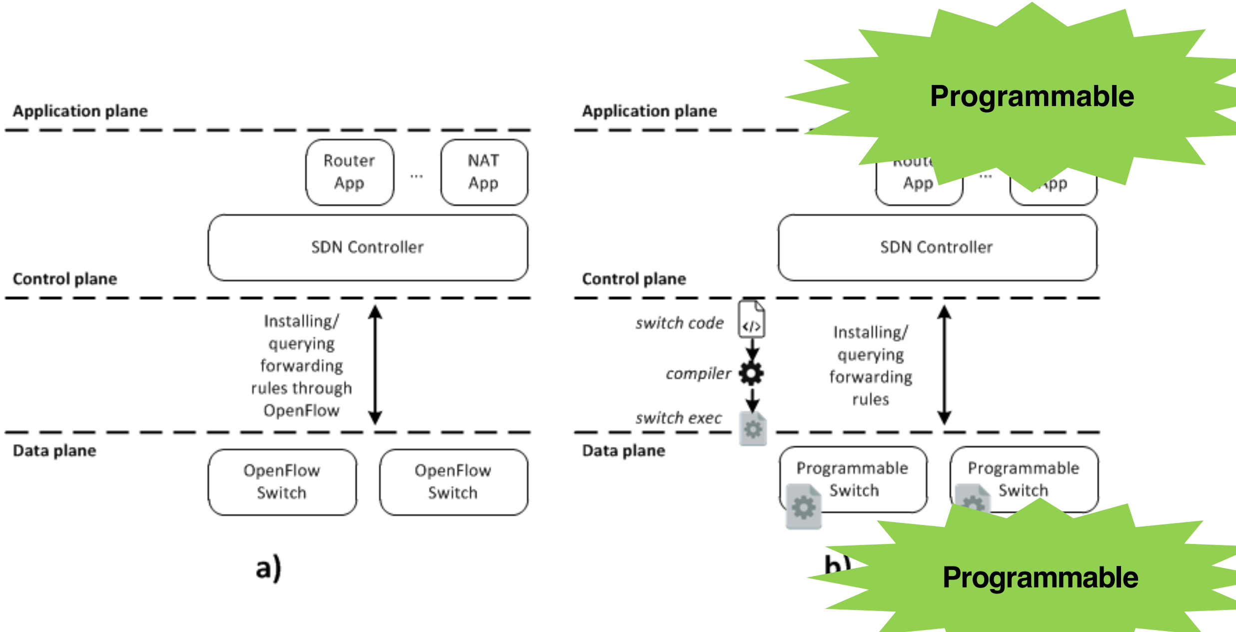


Why Does the Internet Need a Programmable Forwarding Plane by Nick McKeown
https://www.youtube.com/watch?v=zR88NIg3n3g

Source: https://p4.org/assets/P4_tutorial_01_basics.gslide.pdf

What is Data Plane Programmability?

- Allows to define data plane of device in software
- Fixed OpenFlow switch → Programmable Switch
- Open API to program data plane (change software version of data plane device)



Classical OpenFlow **Programmable data plane**

Source: Weverton Luis Costa Cordeíro, Jonatas Adilson Marques, and Luciano Paschoal Gaspary. 2017. Data Plane Programmability Beyond OpenFlow: Opportunities and Challenges for Network and Service Operations and Management. *J. Netw. Syst. Manage.* 25, 4 (October 2017)

Benefits of Data Plane Programmability

- New Features – Fast Time to Market (TTM) for new protocols
- Software development style:
    - Debugging
    - Bug fixing
    - Software re-use (libraries)
    - Software upgrade
    - CI/CD
- Reduced complexity – Remove unused protocols
- Increased reliability and security – reduced risk by removing unused protocols
- New Use Cases – e.g. telemetry
- You keep your own ideas.. – Intellectual Property (IP)

P4 – Programming Protocol-independent Packet Processors

P4 is a high-level programming language for Software-defined Networking (SDN).

It is intended to describe the behavior of the data plane of devices that forwards, modifies or inspects network traffic.

P4 – a brief history

- **July 2014** – **First paper „P4 – Programming Protocol-Independent Packet Processors”**
- **September 2014** – **First $P4_{14}$ specification**
- **May 2017** – **$P4_{16}$ specification**

http://github.com/p4lang/

P4: Programming Protocol-Independent Packet Processors

Pat Bosshart[†], Dan Daly[*], Glen Gibb[†], Martin Izzard[†], Nick McKeown[‡], Jennifer Rexford[**], Cole Schlesinger[**], Dan Talayco[†], Amin Vahdat[¶], George Varghese[§], David Walker[**]
[†]Barefoot Networks [*]Intel [‡]Stanford University [**]Princeton University [¶]Google [§]Microsoft Research

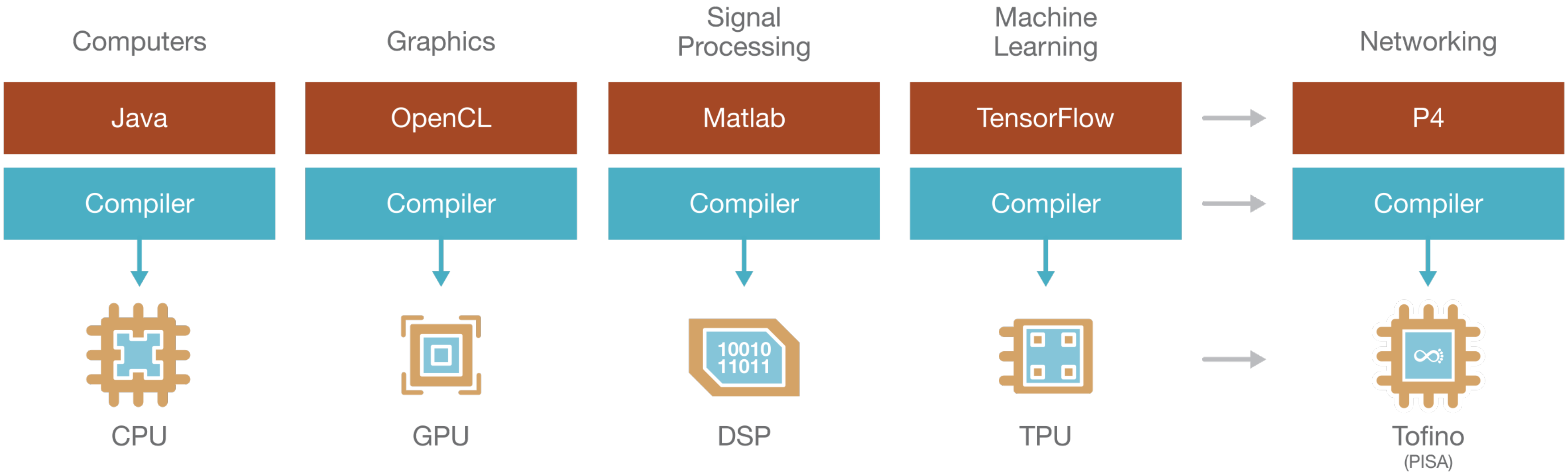
The P4 organization

- **Open source**, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- **Membership is free**: contributions are welcome
- Independent, set up as a California nonprofit

The P4 language - general information

„Our goal is for $P4_{16}$ to enable the same kind of programmability for network data-planes as the CUDA language did for graphics cards"

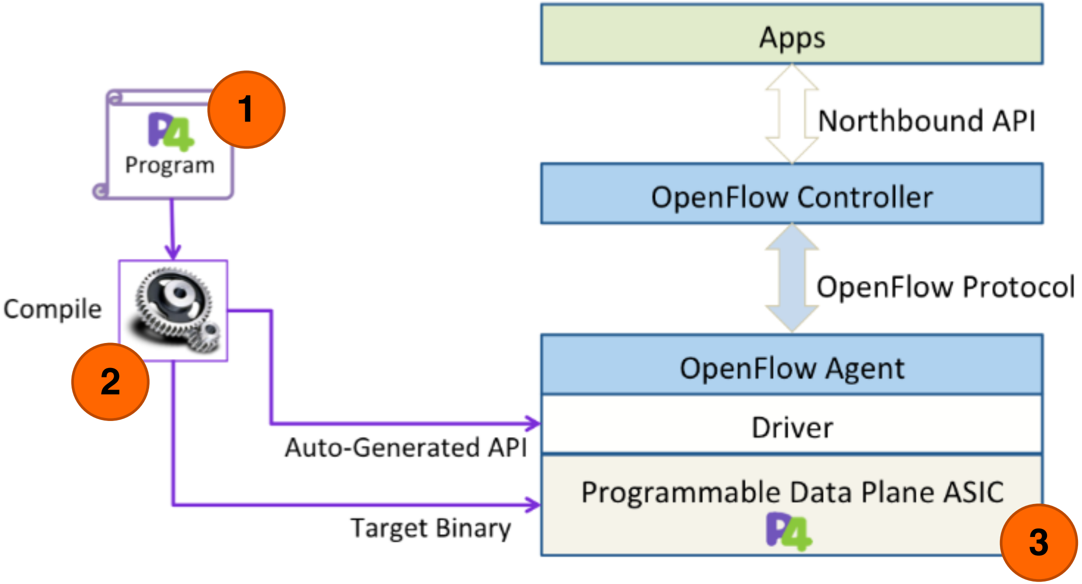
Mihai Budiu, Chris Dodd, „The $P4_{16}$ Programming Language"



Source: https://www.barefootnetworks.com/technology/

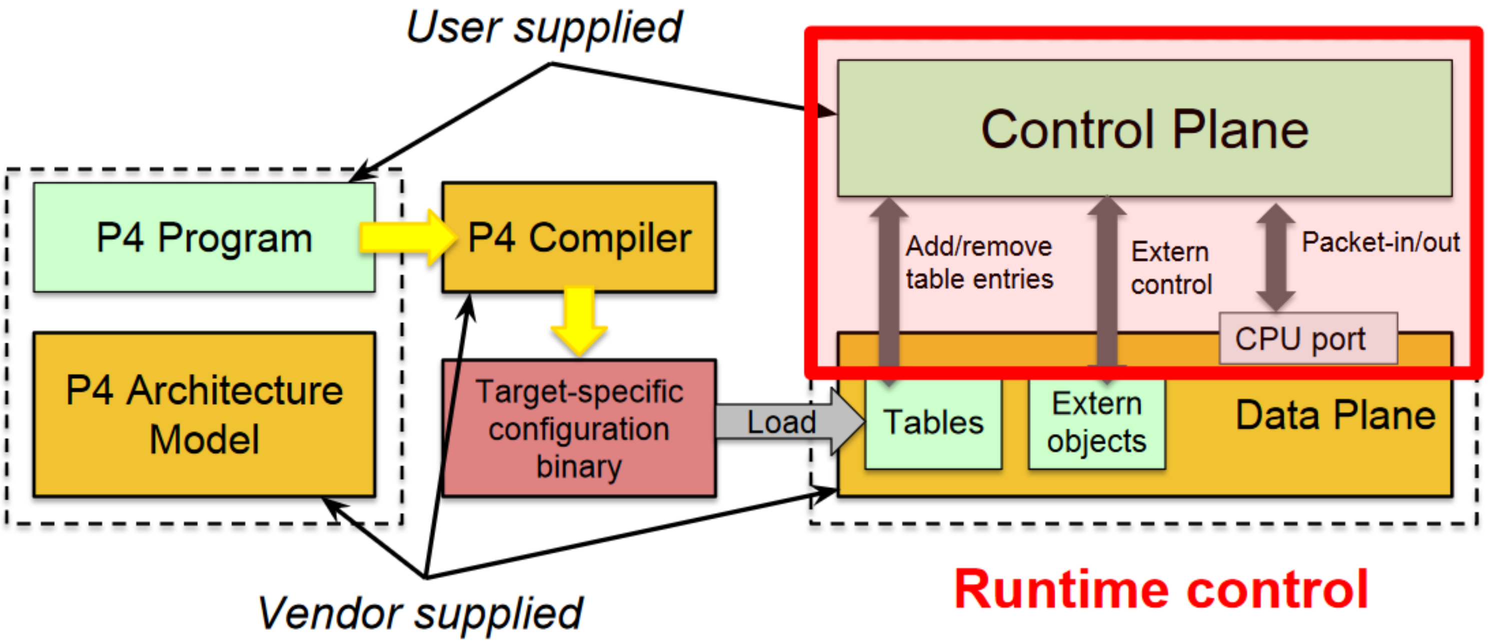
The P4 technology

1. **The P4 language** – **<u>declarative</u> Domain-Specific Language (DSL) to define data plane behavior**
   - **Protocol-independent**
   - **Target-independent**
   - **Field Reconfigurable**
2. **The P4 Compiler**
3. **Abstract forwarding model for the network device**



Source: https://p4.org/p4/clarifying-the-differences-between-p4-and-openflow.html

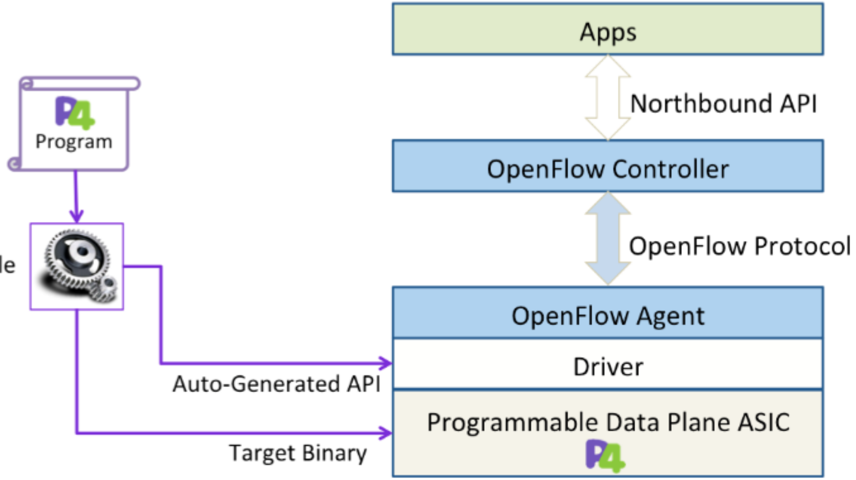
Typical P4 workflow



Source: p4.org

P4 vs. OpenFlow

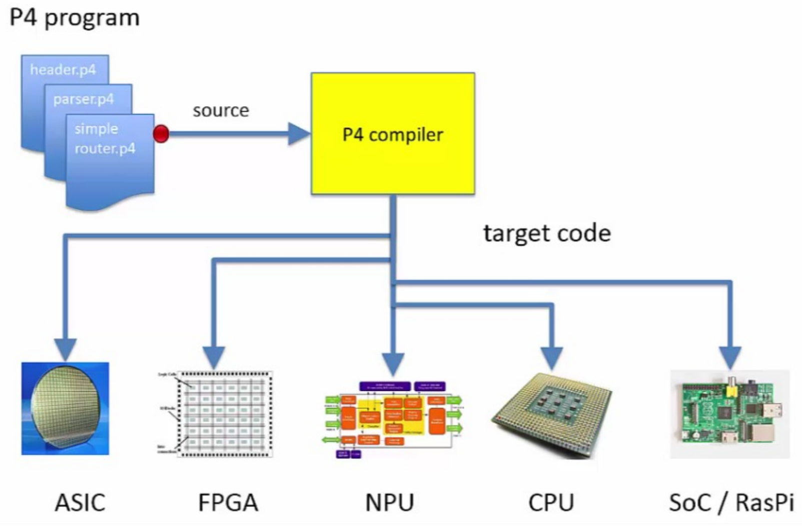
- What is the relationship between P4 and OpenFlow?
  - Both are invented to create open APIs for network devices
  - Serves different roles:
    - OpenFlow gives the way to populate a set of well-known tables
    - P4 gives the way to define forwarding tables
  - OpenFlow is more focused on programmability of control plane, while P4 focues on data plane
  - OpenFlow assumes fixed instruction set for switch, while P4 allows to program that instruction set
- „While OpenFlow is designed for SDN networks in which we separate the control plane from the forwarding plane, P4 is designed to program the behavior of *any* switch or router"
- Question: Why are switch chips fixed-function, so far?



More on: https://p4.org/p4/clarifying-the-differences-between-p4-and-openflow.html

P4: Target-independence

- P4 Target – the platform, where the P4 program is executed on.
    - Target's vendor must provide a corresponding compiler for its platform.
- Target-independence – P4 language is independent from underlaying platform (as Java)
    - Compiler should translate a *target-independent* description (P4 program) into *target-dependent* code (used to configure the switch)



P4: Protocol-independence

- **P4 language can be used to define any network protocol**
  - **P4 has no native support for any protocol (such as Ethernet, IP, etc.)**
  - **The programmer must define all headers and appropriate parsers**

```
typedef bit<48> macAddr_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}
```

Ethernet

```
typedef bit<32> ip4Addr_t;
header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}
```

IPv4

```
header_type icmp_t {
    fields {
        typeCode : 16;
        hdrChecksum : 16;
    }
}
```

ICMP

```
header_type tcp_t {
    fields {
        srcPort : 16;
        dstPort : 16;
        seqNo : 32;
        ackNo : 32;
        dataOffset : 4;
        res : 4;
        flags : 8;
        window : 16;
        checksum : 16;
        urgentPtr : 16;
    }
}
```

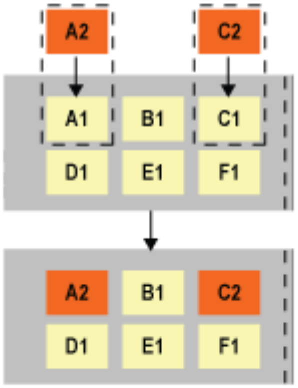
TCP

```
header_type udp_t {
    fields {
        srcPort : 16;
        dstPort : 16;
        length_ : 16;
        checksum : 16;
    }
}
```

UDP

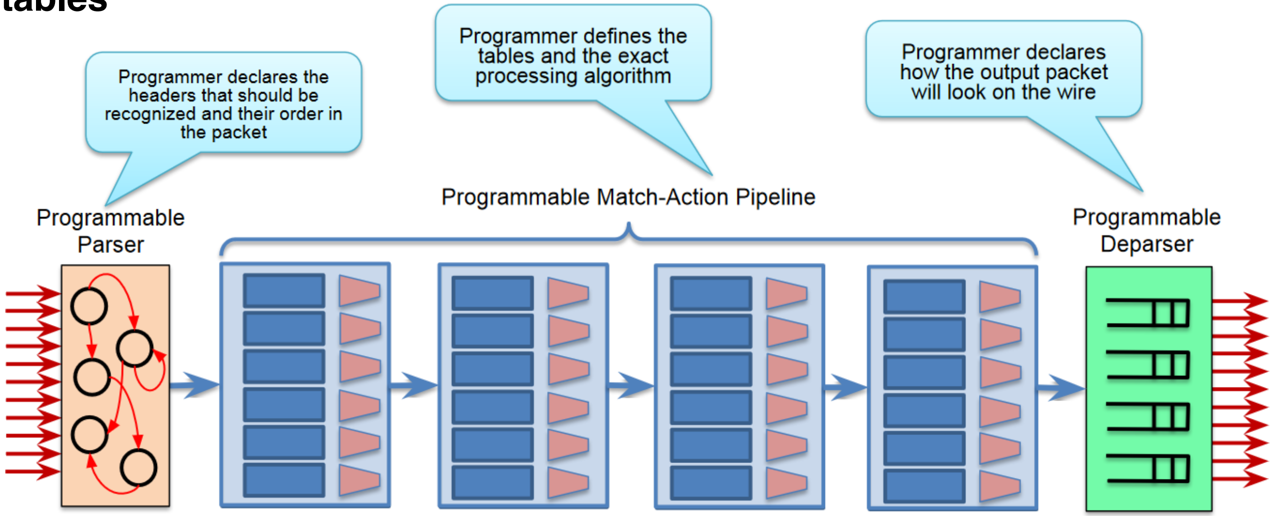
P4: Reconfigurability

- It should be possible to re-define packet processing behavior on-the-fly.
- P4 has been designed to support reconfigurability
    - P4 provides high-level abstract model
    - P4 Target should be able to change the way of processing packets
- <u>However, not all targets are re-configurable (e.g. fixed ASICs)</u>



P4 language – architecture model

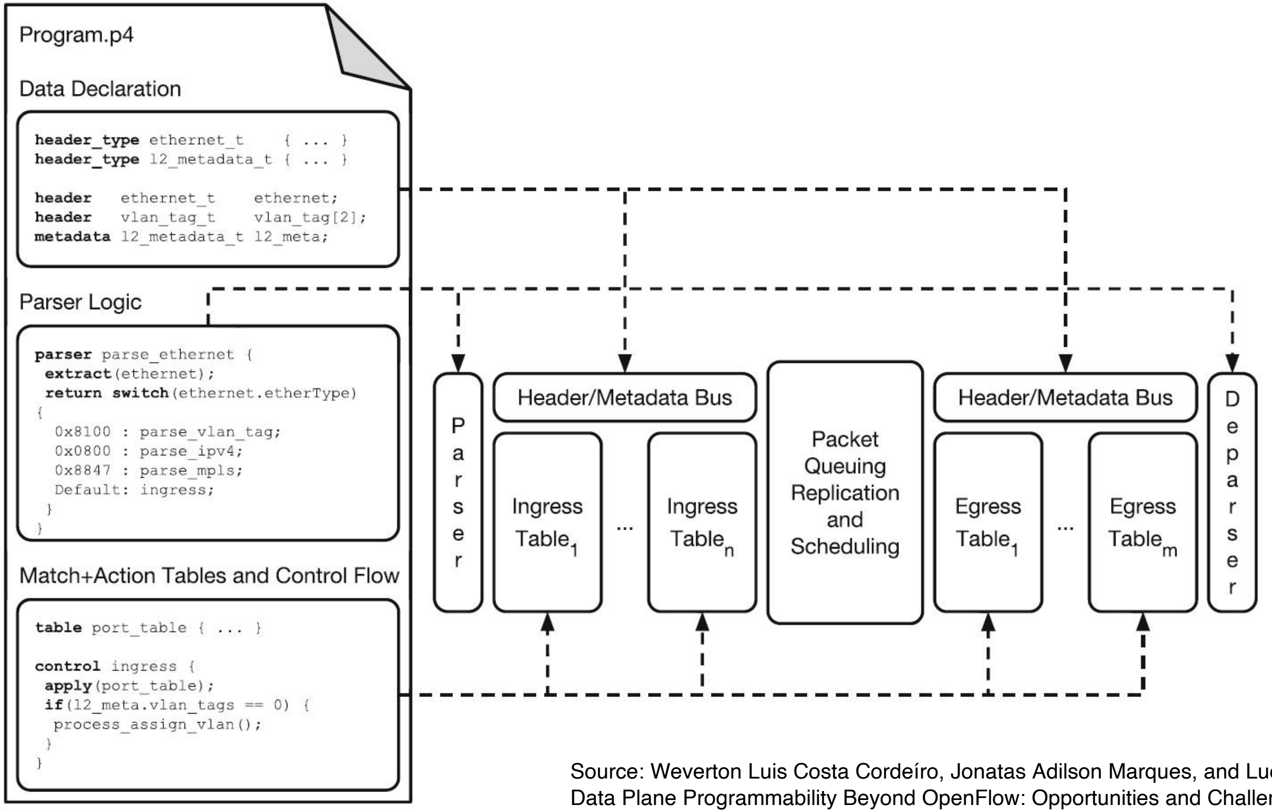
- **Headers** – defintions of packet's headers
- **Parser** – finite state machine, tells how to process incoming bytes and extract headers
- **Deparser** – defines how the output packet will look like on the wire
- **Match-Action tables** – lookup keys + corresponding actions
- **Control flow** - sequence of tables



P4 Switch Architecture - Abstraction Model

Source: p4.org

The P4 switch model



Source: Weverton Luis Costa Cordeíro, Jonatas Adilson Marques, and Luciano Paschoal Gaspary. 2017. Data Plane Programmability Beyond OpenFlow: Opportunities and Challenges for Network and Service Operations and Management. *J. Netw. Syst. Manage.* 25, 4 (October 2017)

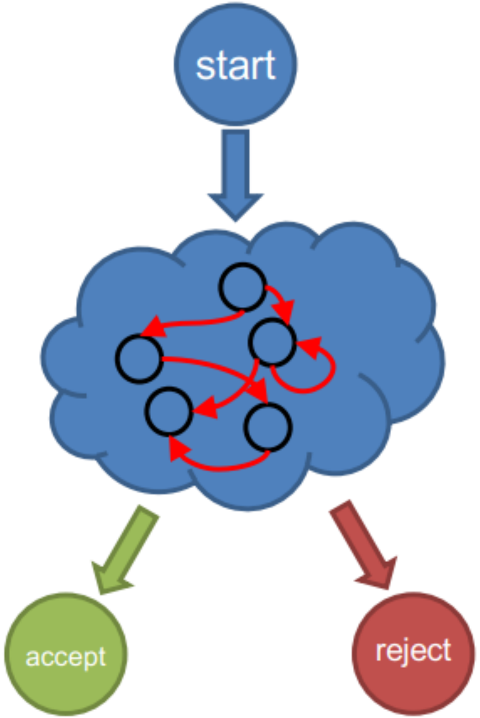
P4 Language – Types & Headers (1/2)

```
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16> etherType;
}
header ipv4_t {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}
```

- **Basic types:**
  - **bit<n> - unsigned integer of size „n”**
  - **int<n> - signed integer of size n (<=2)** // Why?
  - **varbit<n> - variable-length field (e.g. for IPv4 options)**
- **Headers:**
  - **Defines the structure of protocol’s header**
  - **Group of bit<n> fields**
  - **Defined based on specification e.g. RFCs**
  - **Can be valid or invalid**
- **Typedef – the construct to define a new type**

P4 – Protocol parser

- Parser – the construct that map packets into headers and metadata
  - Copy header's values from packet buffer to local memory
  - Finite state machine (FSM)
- Every parser has three predefined states:
  - *start*
  - *accept*
  - *reject*
- If protocol is not supported by parser -> reject packet

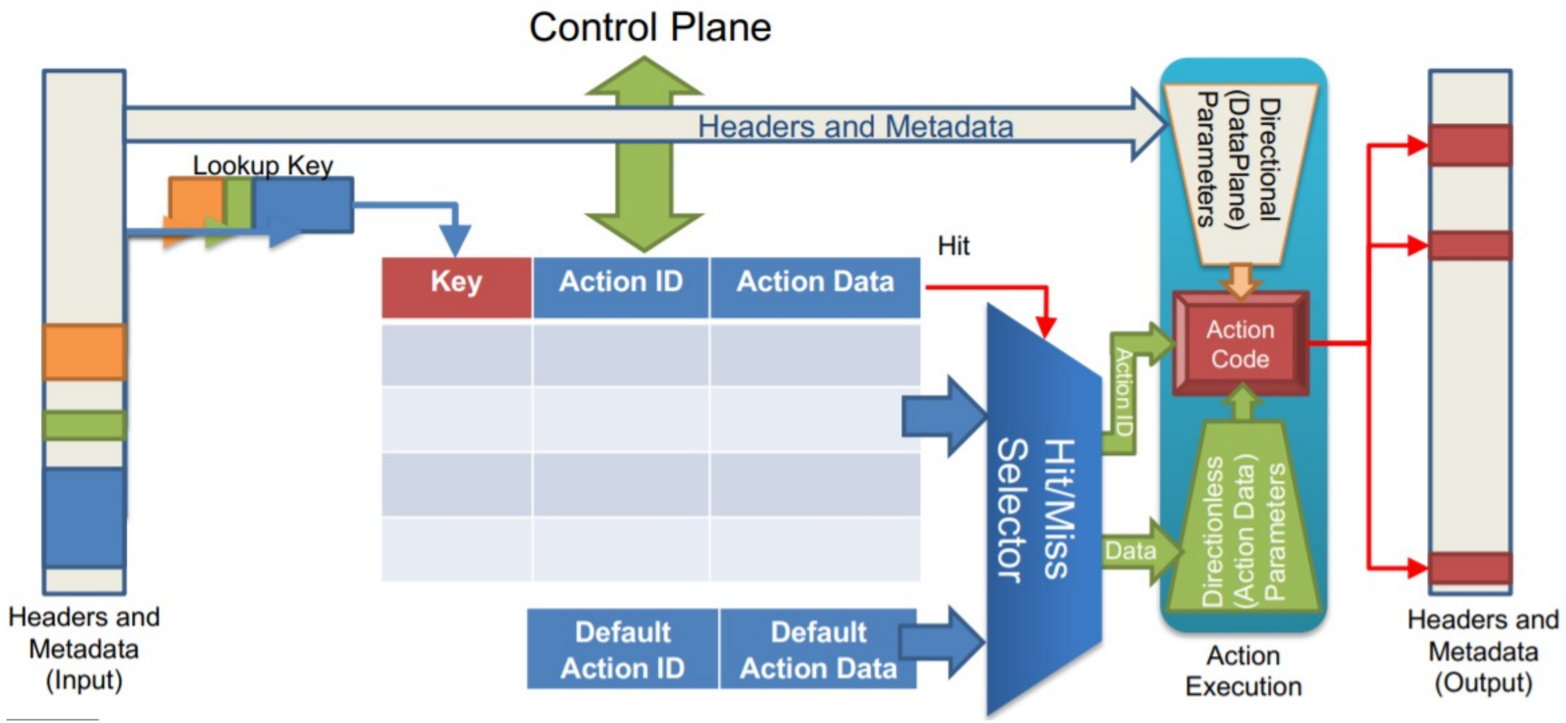


P4 – Protocol parser example

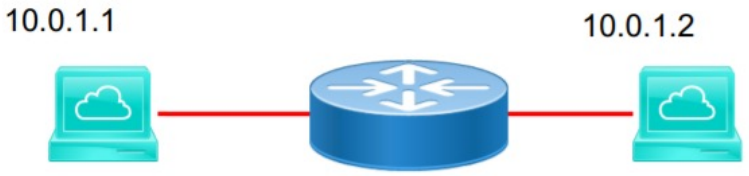
```
parser ParserImpl(packet_in packet,
                  out headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {

  state start {
    transition parse_ethernet;
  }

  state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
      0x0800:  parse_ipv4;
      0x0806:  parse_arp;
      0x86DD: parse_ipv6;
      0x8847:  parse_mpls;
      default:   reject;
    }
  }
}
```

- **Parser keywords:**
  - **select() – used to branch in a parser**
    - ***similar to switch-case statement from C or Java***
  - **transition – used to go to the next parser**
- **Implementing parser requires expertise in network protocols:**
  - **Next stage in parser is determined by the protocol header's field**
  - **For example, etherType is used to determine next protocol for Ethernet frames**

P4 - Match-Action tables (1/3)



P4 - Match-Action tables (2/3)



Key	Action	Action Data
10.0.1.1/32	ipv4_forward	dstAddr=00:00:00:00:01:01 port=1
10.0.1.2/32	drop	
*`	NoAction	

- Data plane (P4) program:
  - Defines the format of the table:
    - Key fields
    - Actions + Action Data
  - Performs lookup
  - Executes the chosen action
- Control plane (IP stack, routing protocols, SDN controller)
  - Populates table entries with specific information based on:
    - Configuration (e.g. BGP routing policy)
    - Automatic discovery (e.g. MAC learning)
    - Protocol calculations (Dijkstra algorithm in OSPF)

P4 - Match-Action tables (3/3)

```
/* Defined in core.p4 */
match_kind {
    exact,
    ternary,
    lpm
}
action NoAction() { }
action drop() {
    mark_to_drop();
}
action MyAction(macAddr_t dstAddr) { … }

table ipv4_lpm {
    key = {
      hdr.ipv4.dstAddr: lpm;
    }
    actions = { ipv4_forward;
                drop;
                NoAction; }
    size = 1024;
    default_action = NoAction();
}
```

- **Table keywords:**
  - ***key*** **– defines set of match fields**
  - ***actions*** **– defines a list of supported actions**
  - ***size*** **– defines a maximum number of entries**
  - ***default_action*** **– defines an action to invoke if there is no match**
- **Different match types:**
  - ***exact***
  - ***lpm*** **(Longest-Prefix Match)**
  - ***ternary***
- **Actions declared as simple functions in other languages (e.g. C, Java, etc.)**

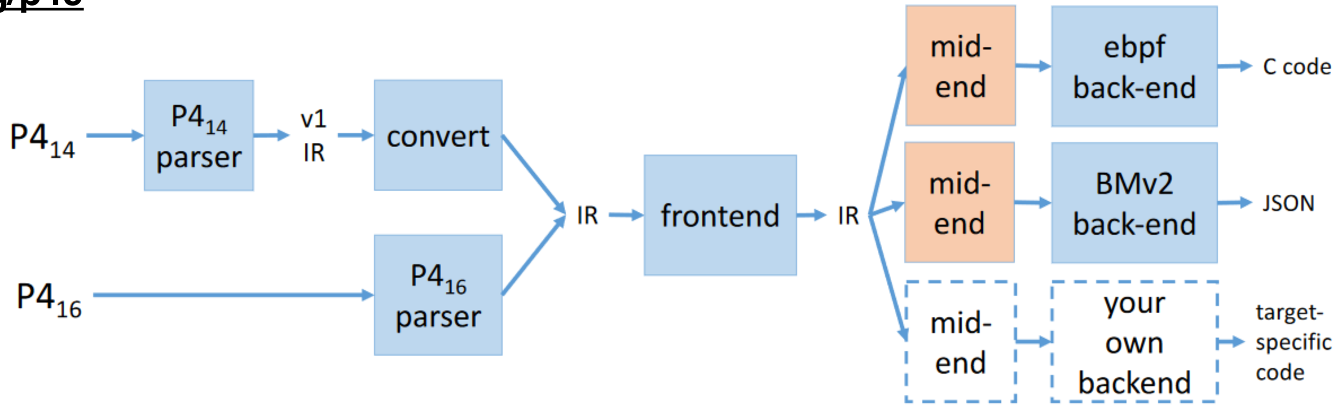
P4 - Deparser

```
control deparser(packet_out packet,
                 in headers hdr) {

  apply {
    packet.emit(hdr.ethernet);
    packet.emit(hdr.ipv4);
  }
}
```

- **Assembles the headers back into a well-formed packet**
  - **emit() function serializes the header if it is valid**

The P4 compiler

- **Compiler goals:**
  - **Support current and future P4 versions**
  - **Open-source front-end**
  - **Support for multiple back-ends**
    - **Generate code for ASICs, NICs, FPGAs, software switches and other targets**
- **https://github.com/p4lang/p4c**



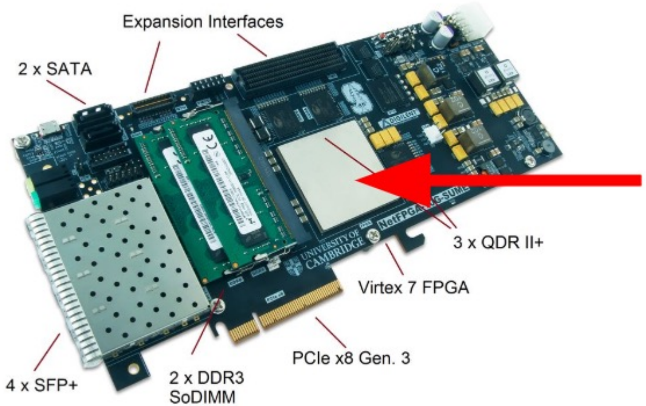
Source: p4.org

P4 targets – hardware platforms

- NetFPGA
  - Prototyping and evaluating P4 program on real hardware
  - https://github.com/NetFPGA/P4-NetFPGA-public/wiki
- Barefoot Tofino
  - Production-ready P4 switch by Barefoot
  - 12.8 Tb/s
- SmartNICs, IPUs, DPUs

Intel IPU E2000

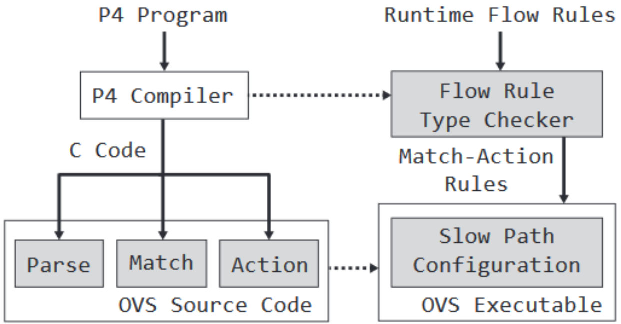
Barefoot Tofino



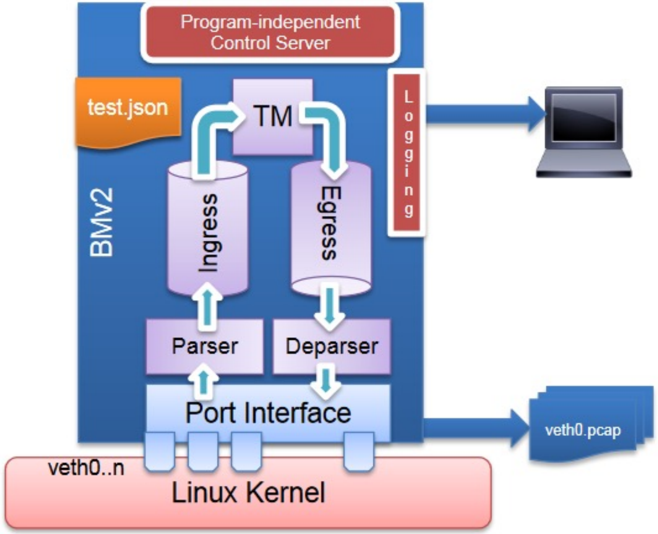
NetFPGA

P4 targets – software platforms

- **BMv2 (Behavioral Model v2)**
  - **The P4 software reference switch**
  - **Up-to-date with P4 specification**
- **PISCES**
  - **P4-compatible Open vSwitch**
  - **P4-to-OVS compiler**
- **P4-to-eBPF**



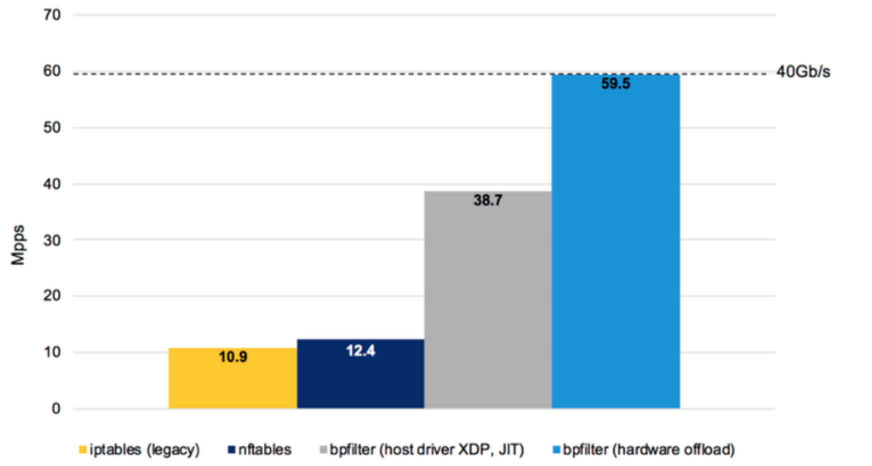
PISCES



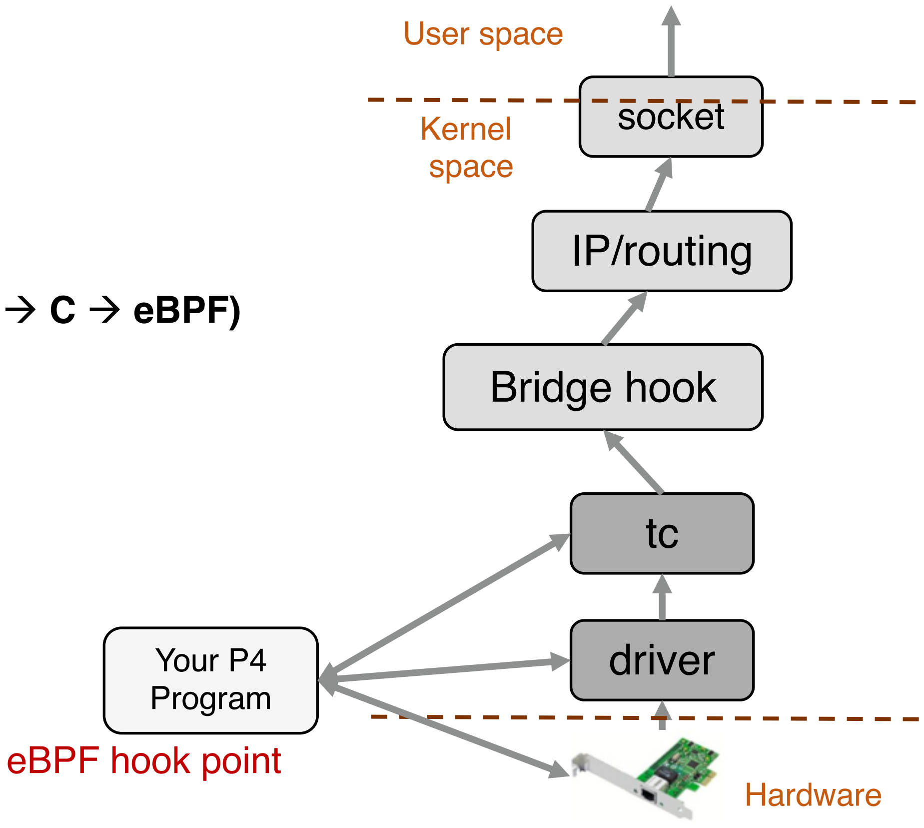
BMv2

P4-to-eBPF

- eBPF (extended Berkeley Packet Filter)
  - In-kernel Virtual Machine for packet filtering
  - Injecting code in the kernel at runtime
- P4-to-eBPF
  - Compiles P4 program to eBPF bytecode (P4 → C → eBPF)



Source: https://www.netronome.com/blog/bpf-ebpf-xdp-and-bpfilter-what-are-these-things-and-what-do-they-mean-enterprise/



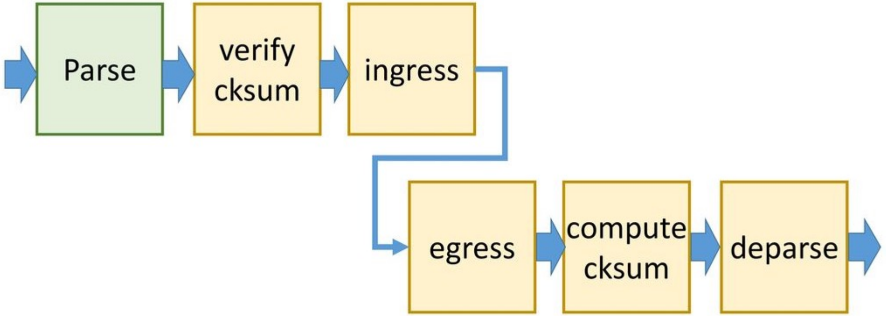
Example of TC+eBPF

Coding break: simple_l3.p4

simple_l3.p4 – Basic structure (template)

```
#include <core.p4>
#include <v1model.p4>

/*** CONSTANTS AND TYPES ***/

/*** HEADER DEFINITIONS ***/

parser MyParser() { … }

control MyIngressControl() { … }

control MyEgressControl() { … }

control MyDeparser() { … }

control MyComputeChecksum() { … }

control MyVerifyChecksum() { … }

V1Switch(MyParser(), MyVerifyChecksum(),
         MyIngressControl(), MyEgressControl(),
         MyComputeChecksum(), MyDeparser()) main;
```

- **We will use v1model.p4**
- **Write code for all P4-programmable components, allowed by the architecture model**
  - ***Your* Headers and Metadata**
  - ***Your* Parser**
  - ***Your* control blocks**
  - ***Your* logic to verify & compute checksum**
  - ***Your* Deparser**
- **Assemble everything in the V1Switch package**



simple_l3.p4 – Constants, types and headers

```
#include <core.p4>
#include <v1model.p4>

/*** CONSTANTS AND TYPES ***/

const bit<16> TYPE_IPV4 = 0x0800;

typedef bit<9> egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

/*** HEADER DEFINITIONS ***/

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16> etherType;
}
header ipv4_t {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}

struct headers {
    ethernet_t ethernet;
    ipv4_t      ipv4;
}

struct routing_metadata_t { ip4Addr_t nhop_ipv4; }

struct metadata { routing_metadata_t routing; }
```

simple_l3.p4 – Parser & Deparser

```
parser RouterParser(packet_in packet,
                    out headers hdr,
                    inout metadata meta,
                    inout standard_metadata_t std_meta)
{
    state start {
        transition parse_ethernet;
    }
    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4:  parse_ipv4;
            default:    reject;
        }
    }
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }
}
```

```
control RouterDeparser(packet_out packet,
                       in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }

}
```

- **Parser:**
  - **Parse Ethernet -> IPv4**
- **Deparser:**
  - **deparse packets in reversed order**

simple_l3.p4 – Ingress control block

```
action drop() { mark_to_drop() );
control ingress(…) {
    action ipv4_forward(ip4Addr_t nextHop,
                                egressSpec_t  port) {
        meta.routing.nhop_ipv4 = nextHop;
        standard_metadata.egress_spec = port;
        hdr.ipv4.ttl = hdr.ipv4.ttl – 1;
    }

    table routing_table {
        key = { hdr.ipv4.dstAddr: lpm; }
        actions = { ipv4_forward;
                    drop;
                    NoAction; }
        default_action = NoAction();
    }

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

- **The role of Ingress control block is to implement L3 functionality:**
  - **Match packets based on IPv4 destination address (LPM)**
  - **Determine IP next hop (an egress port)**
  - **Decrement TTL**
- **Egress port must be determined in the Ingress pipeline**
- **meta.routing.nhop_ipv4 is used to store next hop IPv4 (to be used in the Egress pipeline)**
- **Specifics of v1model / BMv2:**
  - **standard_metadata_t.egress_spec stores the output port**

simple_l3.p4 – Egress control block

```
control egress(…) {
    action set_dmac(macAddr_t dstAddr) {
        hdr.ethernet.dstAddr = dstAddr;
    }
    action set_smac(macAddr_t mac) {
        hdr.ethernet.srcAddr = mac;
    }
    table switching_table {
        key = { meta.routing.nhop_ipv4 : exact; }
        actions = { set_dmac; }
        default_action = NoAction();
    }
    table mac_rewriting_table {
        key = { standard_metadata.egress_port: exact; }
        actions = { set_smac; }
        default_action = drop();
    }
    apply {
        switching_table.apply();
        mac_rewriting_table.apply();
    }
}
```

- **The role of Egress control block is to implement L2 functionality**
    - **Set source MAC address equal to the egress port**
    - **Set destination MAC address equal to the MAC address of next-hop interface**
- **meta.routing.nhop_ipv4 is used to lookup the destination MAC address of the next hop**

simple_l3.p4 – Verify & compute checksum

```
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
  apply {
    update_checksum(
      hdr.ipv4.isValid(),
      { hdr.ipv4.version,
        hdr.ipv4.ihl,
        hdr.ipv4.diffserv,
        hdr.ipv4.totalLen,
        hdr.ipv4.identification,
        hdr.ipv4.flags,
        hdr.ipv4.fragOffset,
        hdr.ipv4.ttl,
        hdr.ipv4.protocol,
        hdr.ipv4.srcAddr,
        hdr.ipv4.dstAddr },
      hdr.ipv4.hdrChecksum,
      HashAlgorithm.csum16);
  }
}
```

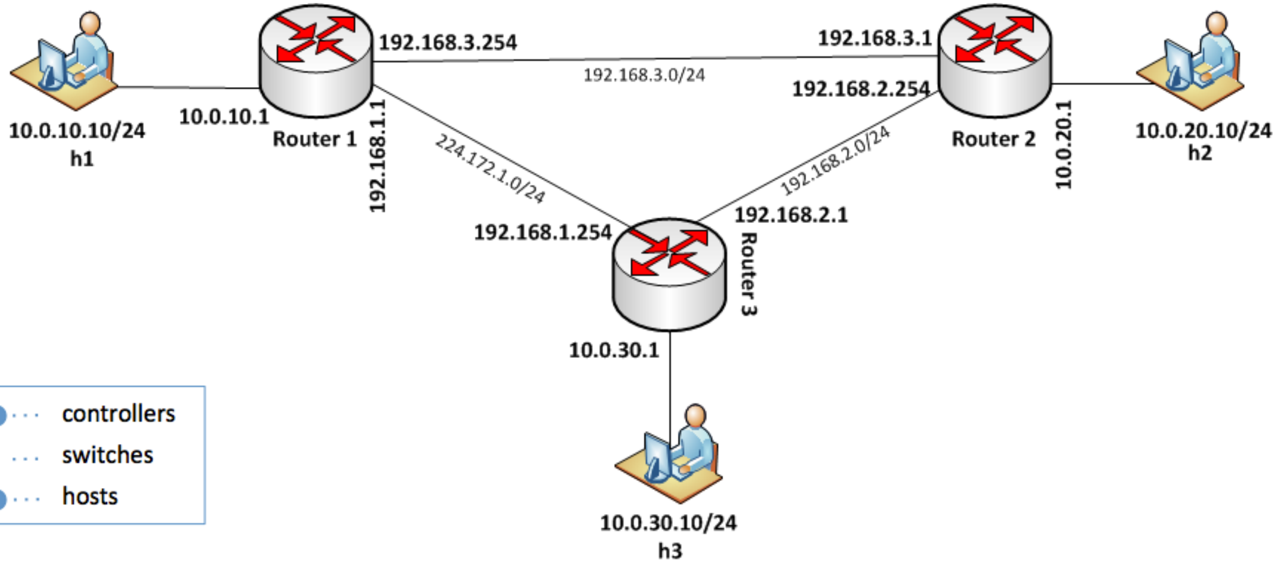
```
control MyVerifyChecksum(inout headers hdr,
                         inout metadata meta) {
  apply { }
}
```

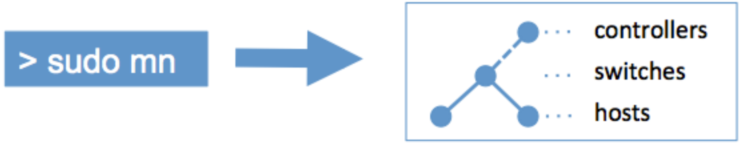
- **Verify Checksum:**
  - **Empty, don't verify checksum**
- **Compute Checksum:**
  - **Built-in function – update_checksum()**
    - ***1st arg*: condition to compute checksum**
    - ***2nd arg*: fields to compute checksum from**
    - ***3rd arg*: where to store checksum**
    - ***4th arg*: what algorithm to use (provided**

Demo

- **P4**
- **Mininet**
- **BMv2 (P4 software switch)**

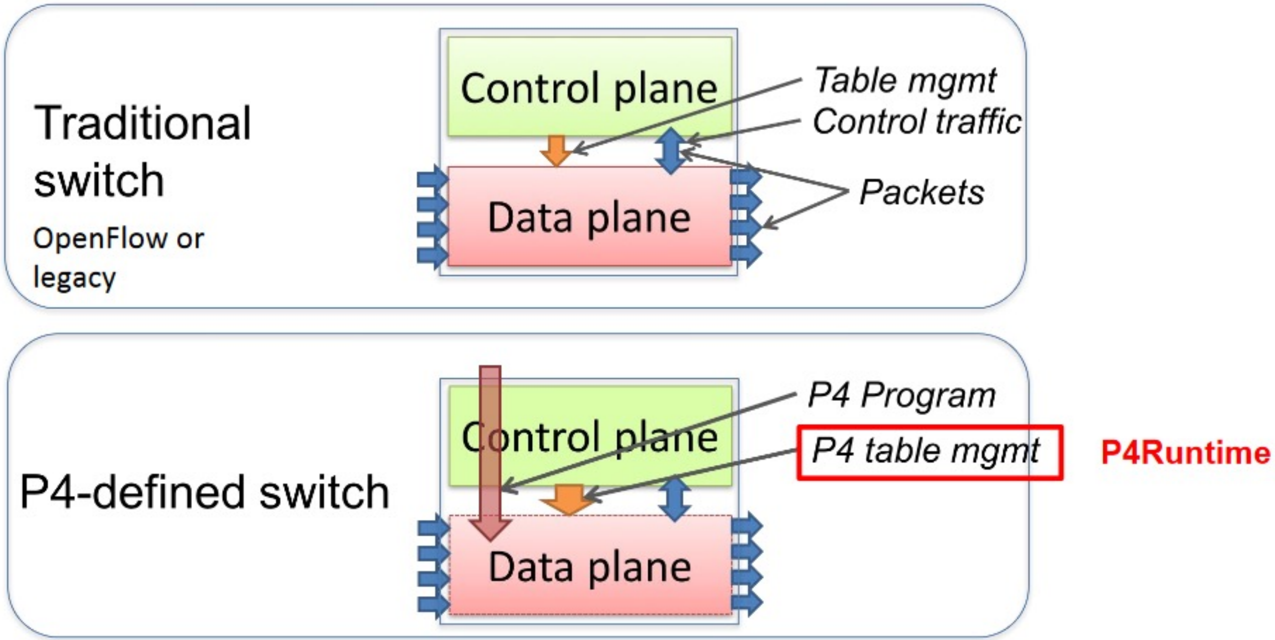
https://github.com/P4-Research/p4-demos/tree/master/ip-routing





P4Runtime – a control plane for P4

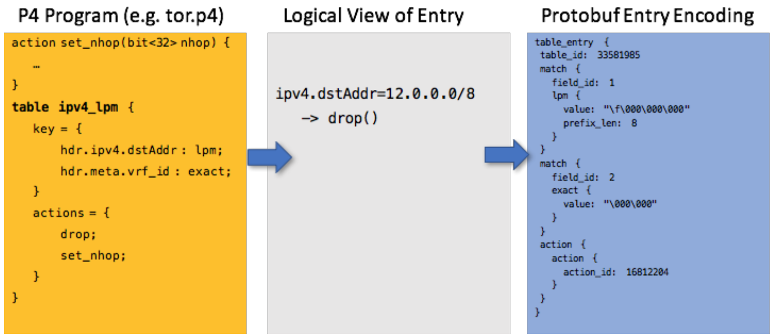
Traditional/OpenFlow vs. P4 paradigm

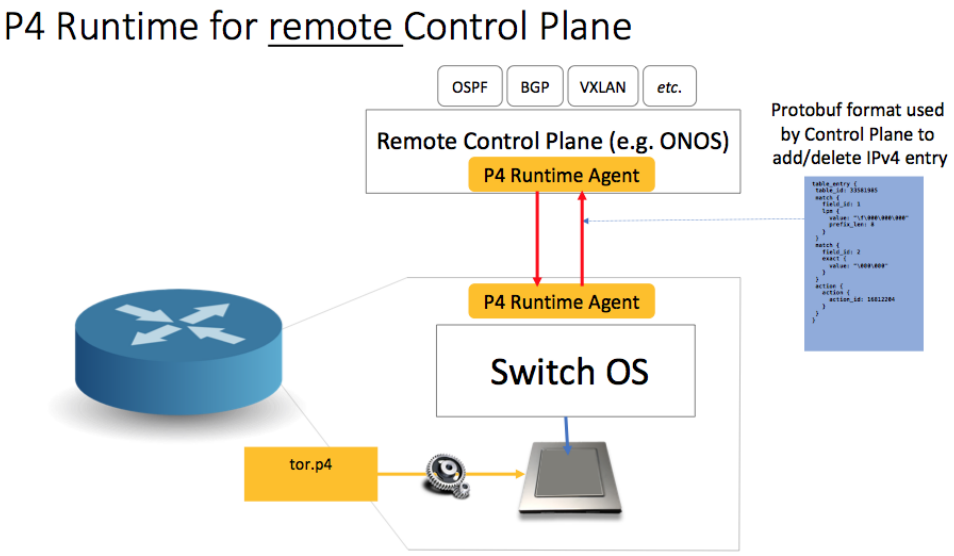


Source: p4.org

P4Runtime – how it is implemented?

- SDN-like control plane for P4 switches
  - Data plane and control plane decoupling
  - Logically centralized
  - Open API to data plane
- Technologies:
  - Google Protocol Buffers
  - gRPC

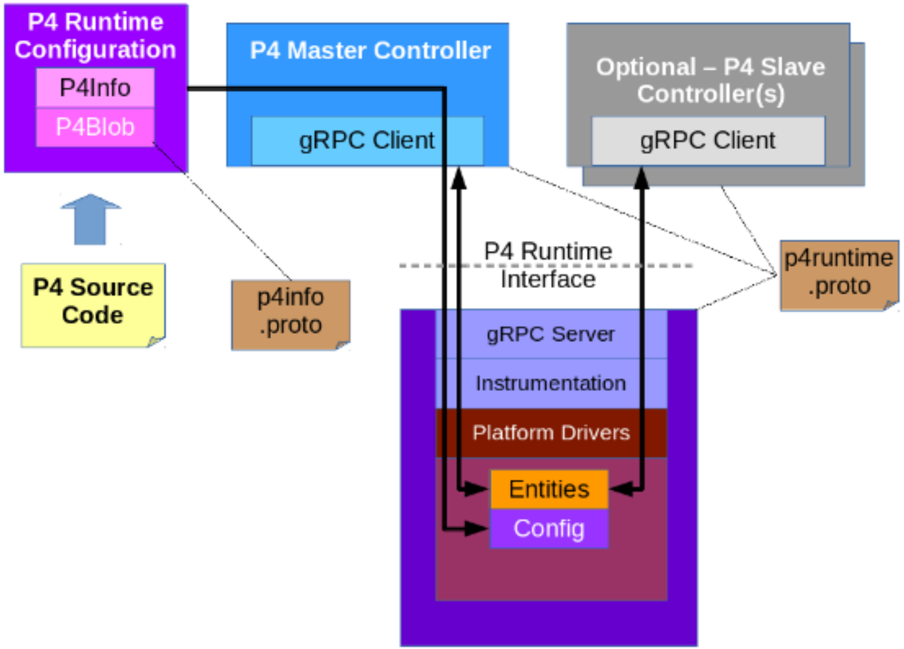




https://p4.org/api/p4-runtime-putting-the-control-plane-in-charge-of-the-forwarding-plane.html

The P4Runtime protocol

- The P4 compiler produces:
  - ForwardingPipelineConfig
  - P4Info metadata – describes the structure of the P4 program (Tables, Actions, etc.)
- The P4Runtime controller can:
  - Set/Get ForwardingPipelineConfig
  - Write/Read Entities: TableEntry, Counter, etc.
- P4Runtime's operations (RPCs):
  - SetForwardingPipelineConfig()
  - GetForwardingPipelineConfigRequest()
  - Write()
  - Read()



https://s3-us-west-2.amazonaws.com/p4runtime/docs/master/P4Runtime-Spec.pdf

Support for P4Runtime in SDN controllers

- Open Network Operating System (ONOS)
  - PoCs with BMv2, Barefoot Tofino, Mellanox Spectrum switch
  - https://wiki.onosproject.org/display/ONOS/P4+brigade
  - https://wiki.onosproject.org/display/ONOS/P4Runtime+support+in+ONOS
- OpenDayLight
  - https://wiki.opendaylight.org/view/P4_Plugin:Main (led by ZTE)

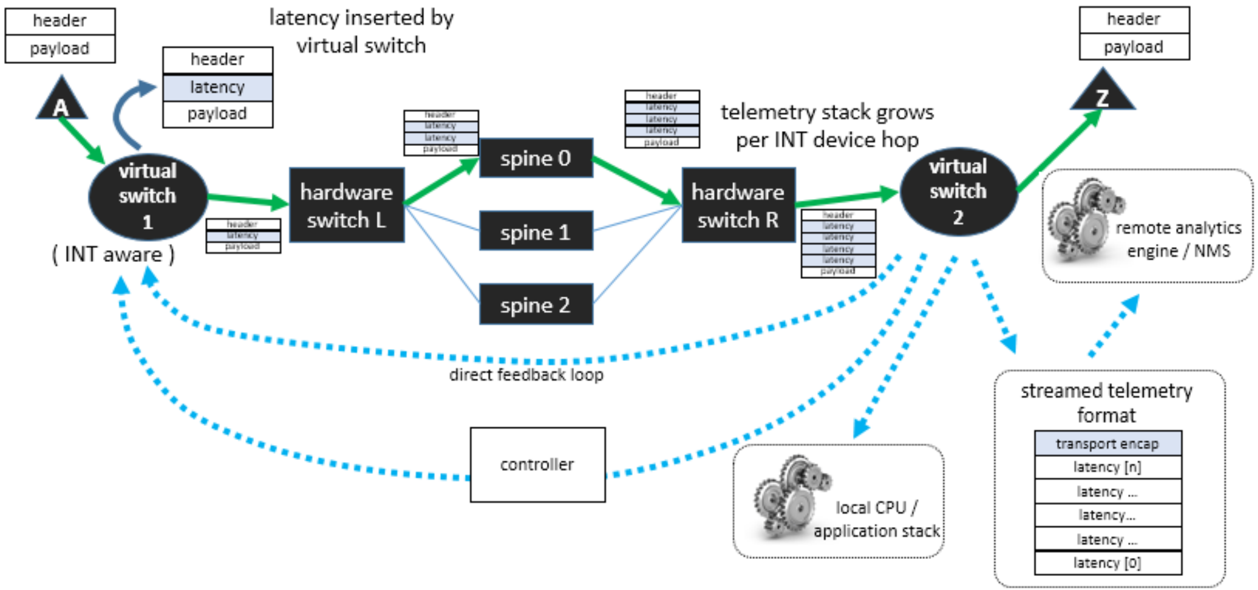
ONOS
Open Network Operating System

OPEN DAYLIGHT

Use cases

Use case study – In-band Network Telemetry (INT)

- The new monitoring tool
- Why P4 helps?



Source:
p4.org

Sample P4 applications

- In-band Network Telemetry (INT)
- Network applications:
  - NAT, L3/L4 Load Balancing
  - Firewall
  - Segment Routing
  - Many others…
- VNF offloading
  - S-/P-GW (LTE)
  - Residential BNG (Broadband Network Gateway) with PPPoE termination

Research activities

- P4 Architecture Working Groups
- Programmable QoS scheduling & traffic management
- New use cases for P4
  - Telemetry & Monitoring
  - VNF’s implementations (NFV Acceleration)
  - New architectures for Data Center
  - Information-Centric Networking
  - Security
  - 5G (Network Slicing)
- More on:
  - https://open-nfp.org/projects/

Summary

Summary

- Network innovation (evolution) is limited due to OpenFlow’s limitations
  - Fixed data plane
  - New data plane protocol requires changes to specification & implementation
- Data plane programmability becomes a next step in the SDN evolution
  - Allows to re-configure data plane functions on-the-fly
- P4 is an enabler for data plane programmability. It provides:
  - Architecture for programming data plane
  - High-level language to describe data plane behavior
  - Abstract model for programmable switch
- P4 is promising, but still under development!

Questions?