# P4 deep dive

Richard Cziva
rcziva@es.net

7th September 2017

## Agenda

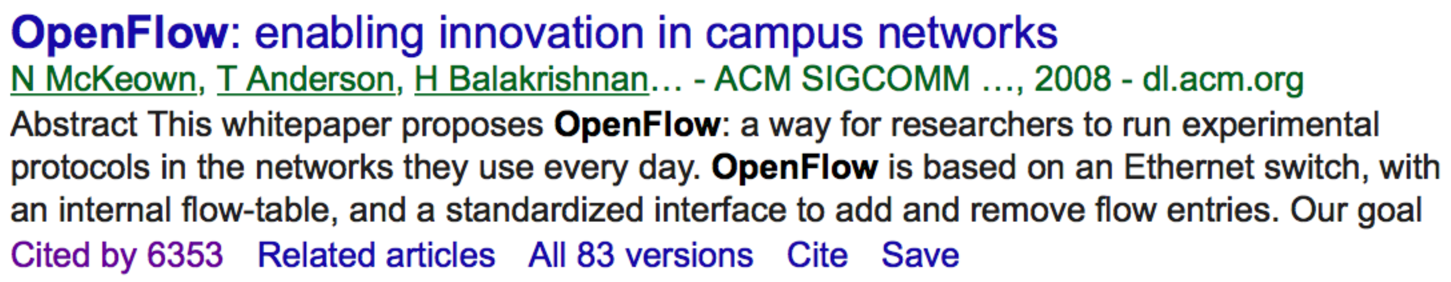
- 9:30 – P4 motivation, language overview
  - OpenFlow vs P4
  - SDN today
- ~10:30 – P4 demos, example applications
  - Ipv4 forwarding switch
  - Calculator (run 1+1 on your router!)
  - Multi-hop route inspection (in-network telemetry)
  - Tracking per-flow TCP retransmissions
- Discussions: data plane programmability @ESnet

## The road to P4

P4 deep dive

# Software-Defined Networking

- Programmable control over networks
- Physically separated control plane the forwarding plane
- Most popular realization of SDN’s data-plane abstraction is **OpenFlow**
  - OpenFlow was born 10 years ago



**OpenFlow**: enabling innovation in campus networks
N McKeown, T Anderson, H Balakrishnan… - ACM SIGCOMM …, 2008 - dl.acm.org
Abstract This whitepaper proposes **OpenFlow**: a way for researchers to run experimental protocols in the networks they use every day. **OpenFlow** is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our goal
Cited by 6353 Related articles All 83 versions Cite Save

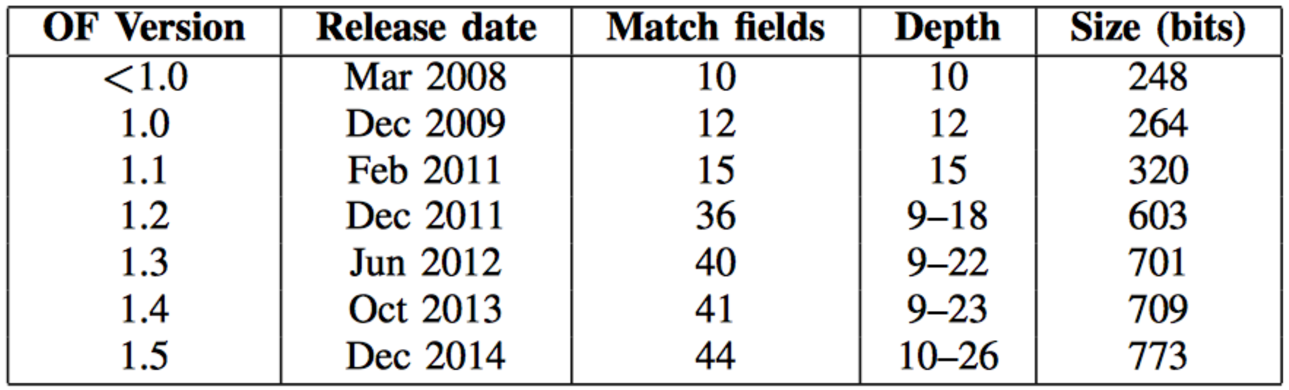
  - And a lot has changed

## Software-Defined Networking

- OpenFlow became the **de facto standard** for network programming
- Originally it was targeting LAN and DC networks
- Lately they kept extending OpenFlow to support new protocols (mostly encapsulation, network virtualization)
  - VxLAN (2010)
  - STT (Stateless Transport Tunnel) (2012)
  - NSH (2014)
  - …

## OpenFlow match fields

- Match fields keep growing and growing: **hard for vendors to keep up**



| OF Version | Release date | Match fields | Depth | Size (bits) |
|---|---|---|---|---|
| <1.0 | Mar 2008 | 10 | 10 | 248 |
| 1.0 | Dec 2009 | 12 | 12 | 264 |
| 1.1 | Feb 2011 | 15 | 15 | 320 |
| 1.2 | Dec 2011 | 36 | 9–18 | 603 |
| 1.3 | Jun 2012 | 40 | 9–22 | 701 |
| 1.4 | Oct 2013 | 41 | 9–23 | 709 |
| 1.5 | Dec 2014 | 44 | 10–26 | 773 |

**Network operators can have specific requirements!**
**Can we let them implement it?**

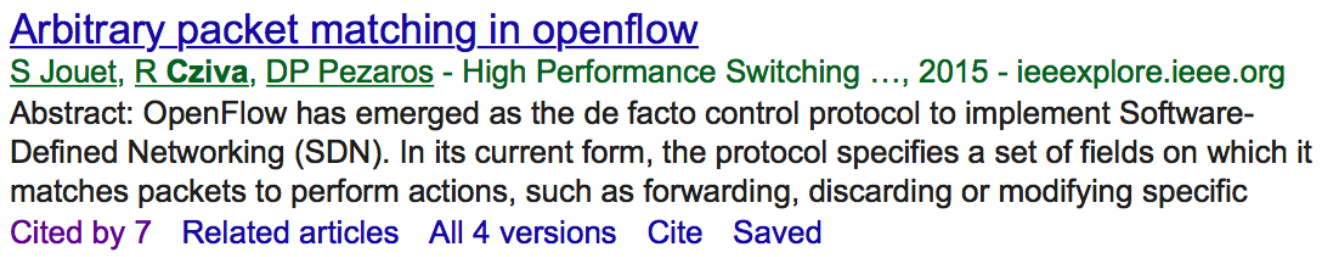
## A partial solution: BPF matching

- Instead of fixed OpenFlow match fields, why don't we use **a Berkeley Packet Filters** (BPF)?
  - (The original BPF paper was written by Steven McCanne and Van Jacobson in 1992 while at Lawrence Berkeley Laboratory[9][10])
- BPF can be used to describe a packet matching as an directed acyclic graph

```
BPFProgram((
 ( 0x20, 0, 0, 0x00000004 ), # load word
 ( 0x15, 0, 3, 0x00000002 ), # in_port: 2
 ( 0x28, 0, 0, 0x00000014 ), # load half word
 ( 0x15, 0, 1, 0x00000800 ), # IPv4 (0x800)
 ( 0x6,  0, 0, 0x0000ffff ), # MATCH
 ( 0x6,  0, 0, 0x00000000 ), # MISS
))
```

## A partial solution: BPF matching

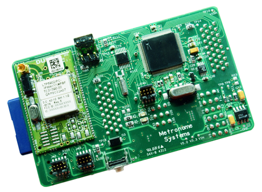
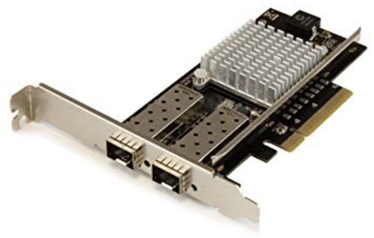
- A BPF program was small enough to be places to an OpenFlow packet (OXM field)
- BPF can be executed on any platform (platform-independence)
- BPF relies on simple instructions (load, compare) -> easy to implement on hardware

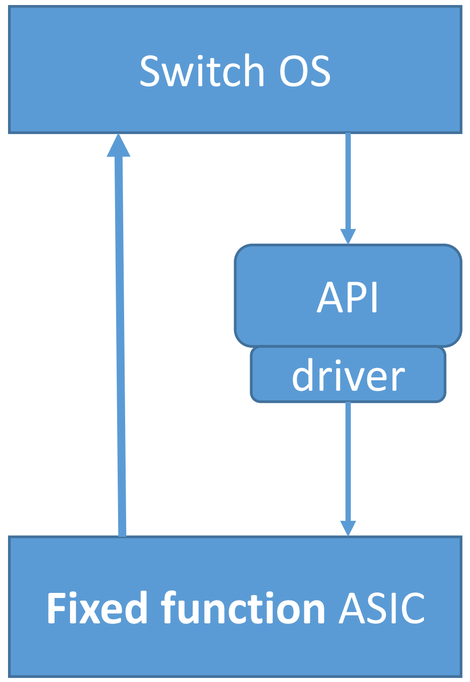


Arbitrary packet matching in openflow
S Jouet, R **Cziva**, DP Pezaros - High Performance Switching …, 2015 - ieeexplore.ieee.org
Abstract: OpenFlow has emerged as the de facto control protocol to implement Software-Defined Networking (SDN). In its current form, the protocol specifies a set of fields on which it matches packets to perform actions, such as forwarding, discarding or modifying specific
Cited by 7 Related articles All 4 versions Cite Saved

# Network data-plane today

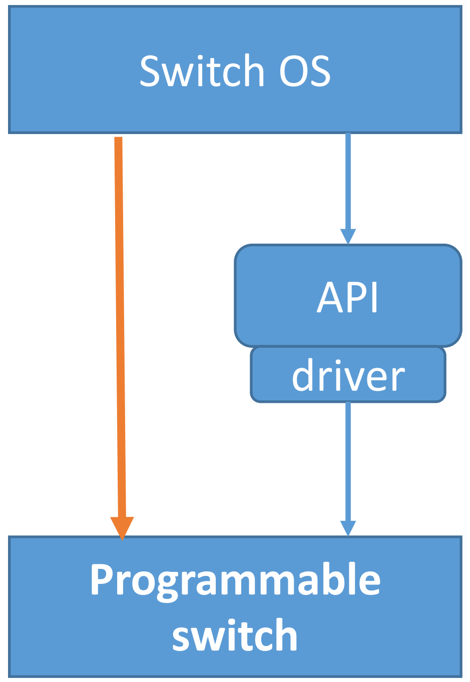
- The ASIC defines how can we handle packets
- ASICs are all closed, we can't extend them if required
- Networks are designed using a bottom-up design



## Networking tomorrow

- We want to tell the device how to behave (top to bottom approach)
- Programmable network devices:
  - PISA (Flexible Match+Action ASICs): Intel Flexpipe, Barefoot Tofino
  - NPU: Netronome
  - CPU: DPDK, eBPF, Open vSwitch
  - FPGA: Xilinx NetFPGA
- P4 defines a language to program some of these devices



## Benefits of top→bottom approach

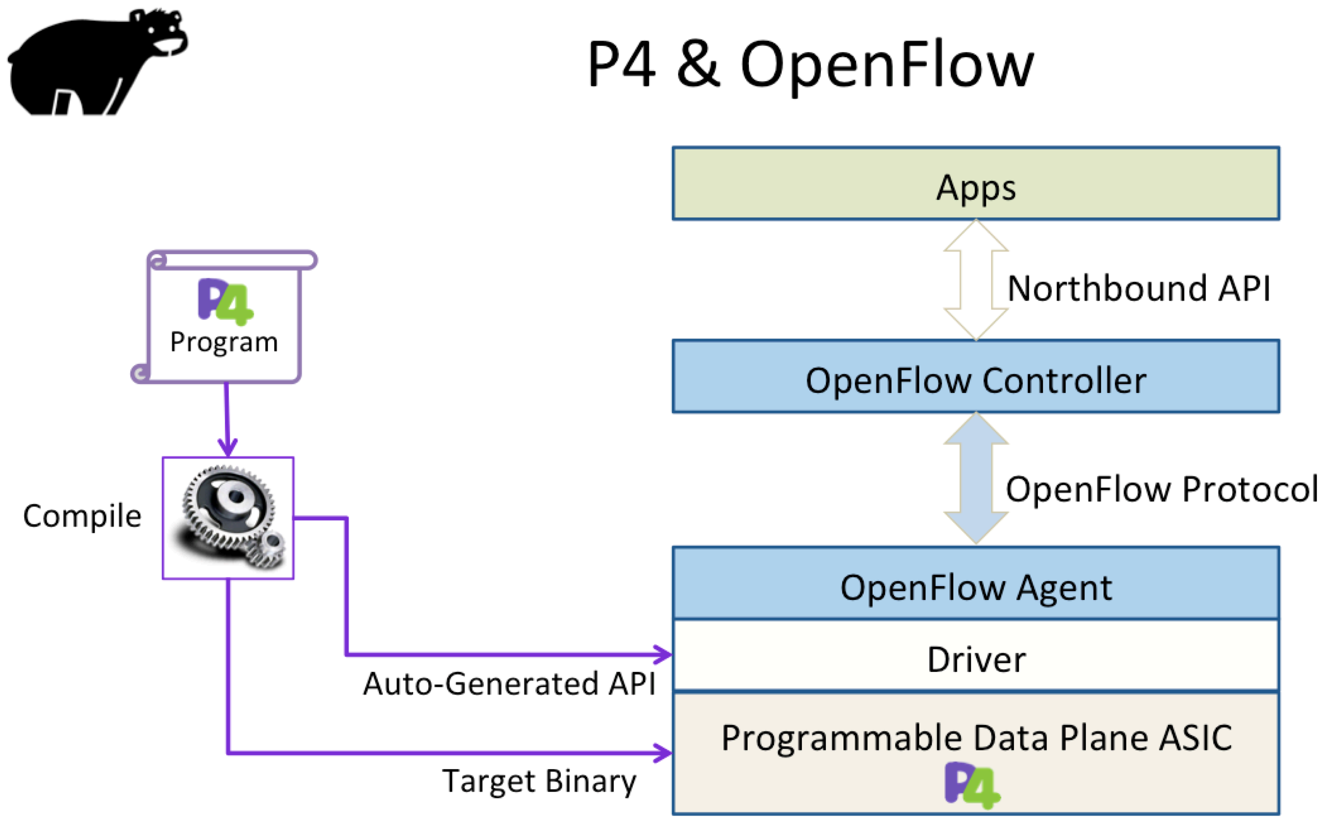
- To network providers
  - Eliminating ”black boxes” and “magic behavior”
  - Developers can program, test and debug network devices all the way down
  - Custom routing / switching protocols can be kept “in-house”
- To vendors
  - Extremely fast iteration and feature release
  - New business models for selling software and hardware separate
  - Vendors can fix data-plane bugs after devices have been deployed
  - Software development practices used in every phase

# A general language: P4

- P4: a language designed to allow top to bottom programmability
- P4 was proposed at SIGCOMM 2013:

  *Bosshart, Pat, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger et al. "P4: Programming protocol-independent packet processors." ACM SIGCOMM Computer Communication Review 44, no. 3 (2014): 87-95.*
- P4 consortium: over 80 vendors, many Universities, individuals…

# P4 vs OpenFlow



## P4 in very high-level

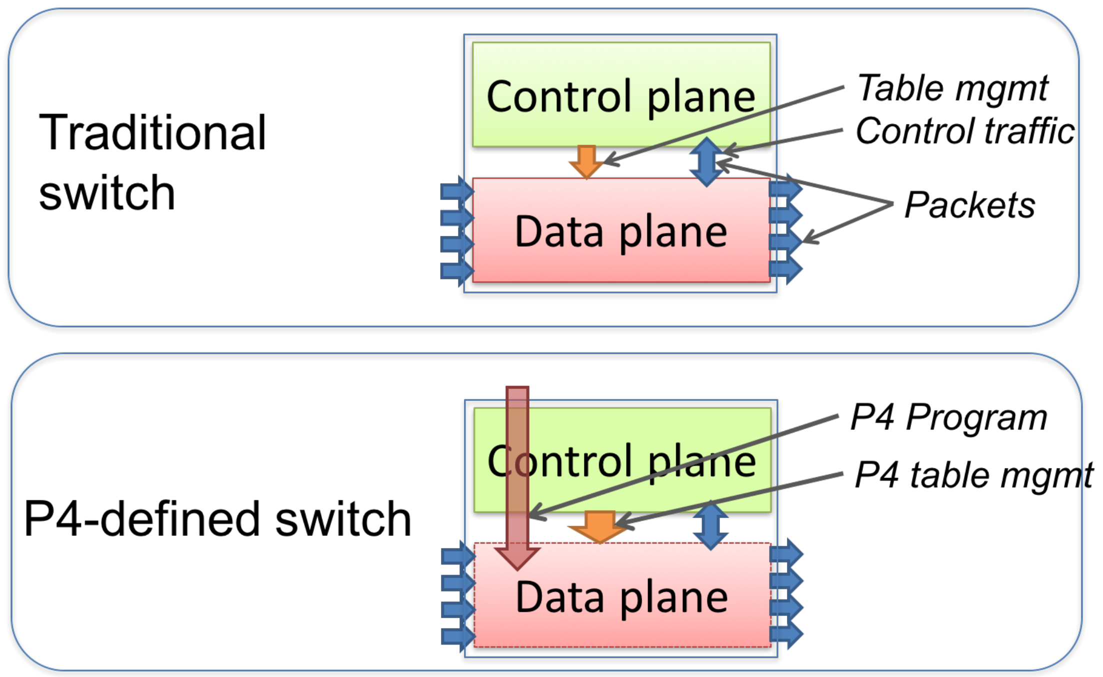
P4 deep dive

## P4 three goals

1. Protocol independence
   - No native support for any protocol
   - The **programmer describes the protocols**
2. Target independence
   - P4 programs are designed to be **implementation-independent**
   - Can be compiled to CPUs, NPUs, FPGAs, ASICs, network processors, etc
3. In-field re-configurability
   - Due to target and protocol independence and the abstract language, P4 **targets (e.g., ASICs) can change their behavior** over their lifetime

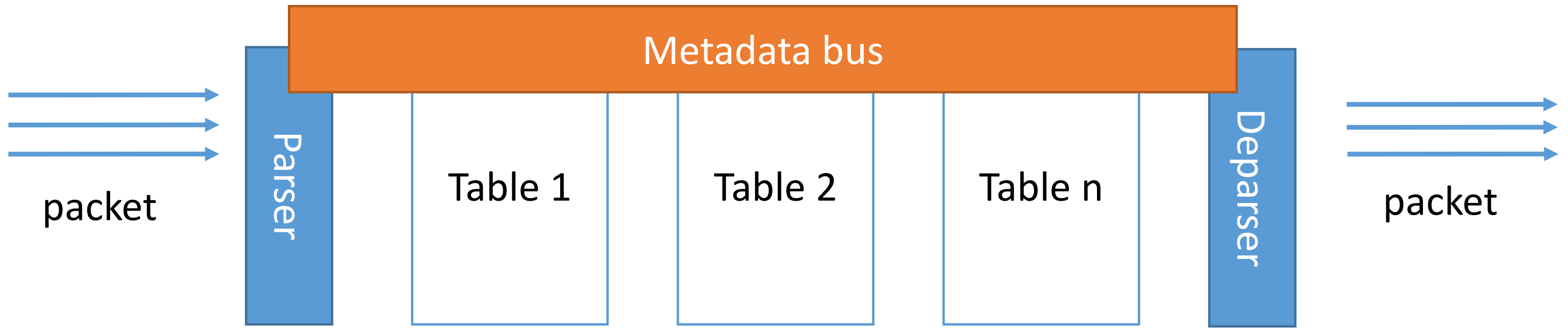
# P4 vs traditional switches

- In a P4 defined switch the **data plane is not fixed**, but defined by a P4 program
- In a P4 defined switch the **API** between the data plane and the control plane is **defined by a P4 program**



*From: https://p4lang.github.io/p4-spec/docs/P4-16-v1.0.0-spec.html*

# P4 pipeline



- Parser: Parses raw packets from wire into metadata
- Match + Action tables: operate on metadata
- Deparser: Serializes metadata into a packet
- Metadata bus: available in the entire pipeline

# An example pipeline from Corsa

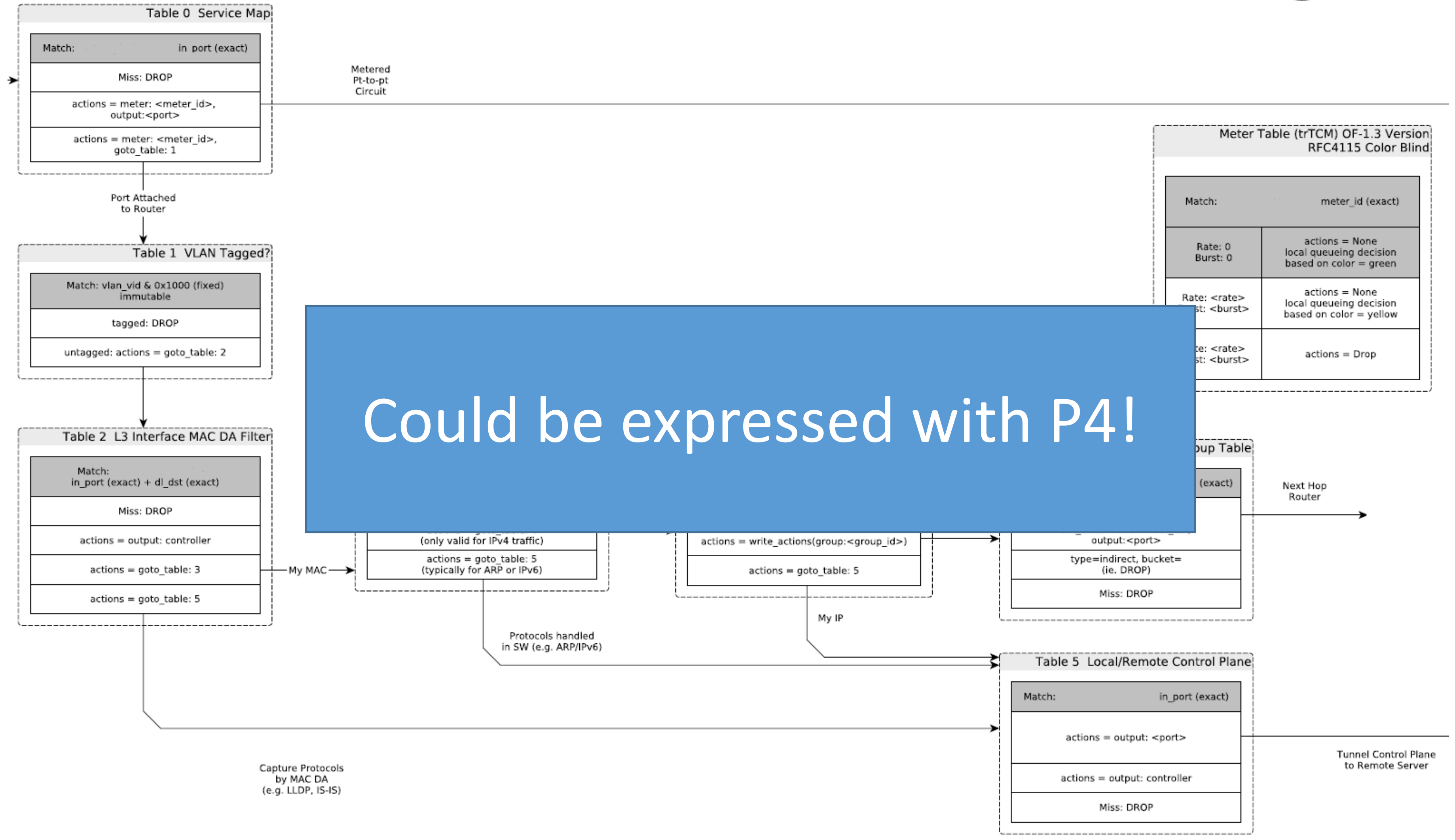


Table 0 Service Map
Match: in_port (exact)
Miss: DROP
actions = meter: <meter_id>, output:<port>
actions = meter: <meter_id>, goto_table: 1
Metered Pt-to-pt Circuit
Port Attached to Router
Table 1 VLAN Tagged?
Match: vlan_vid & 0x1000 (fixed) immutable
tagged: DROP
untagged: actions = goto_table: 2
Table 2 L3 Interface MAC DA Filter
Match: in_port (exact) + dl_dst (exact)
Miss: DROP
actions = output: controller
actions = goto_table: 3
actions = goto_table: 5
My MAC
(only valid for IPv4 traffic)
actions = goto_table: 5 (typically for ARP or IPv6)
actions = write_actions(group:<group_id>)
actions = goto_table: 5
My IP
Protocols handled in SW (e.g. ARP/IPv6)
Capture Protocols by MAC DA (e.g. LLDP, IS-IS)
Meter Table (trTCM) OF-1.3 Version RFC4115 Color Blind
Match: meter_id (exact)
Rate: 0 Burst: 0
actions = None local queueing decision based on color = green
Rate: <rate> st: <burst>
actions = None local queueing decision based on color = yellow
te: <rate> st: <burst>
actions = Drop
oup Table
(exact)
output:<port>
type=indirect, bucket= (ie. DROP)
Miss: DROP
Next Hop Router
Table 5 Local/Remote Control Plane
Match: in_port (exact)
actions = output: <port>
actions = output: controller
Miss: DROP
Tunnel Control Plane to Remote Server

Could be expressed with P4!

ESnet
ENERGY SCIENCES NETWORK

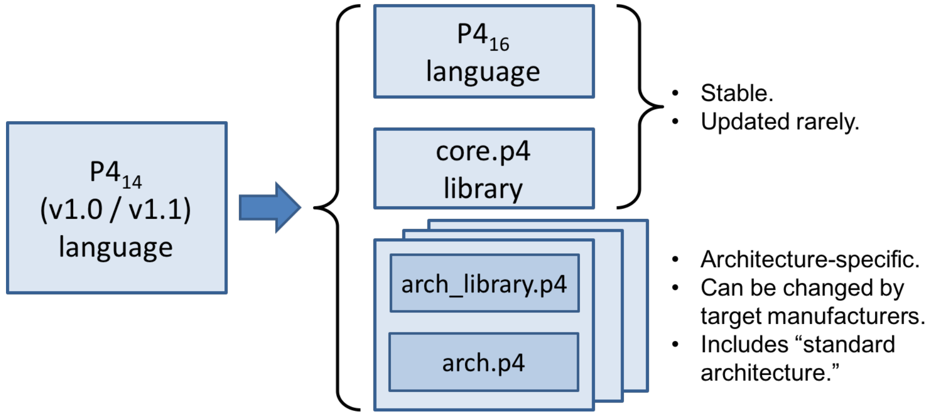
# Language overview (P4_16)

P4 deep dive

## Language versions

- There are multiple versions of the language
  - Unfortunately the versions are backwards-incompatible
  - And some things only work in former versions just now

- I am introducing the latest: **P4_16**
  - Smaller core language than previous versions (only 40 keywords)
  - This is supposed to be a stable language definition



P4$_{14}$ (v1.0 / v1.1) language
P4$_{16}$ language
core.p4 library
arch_library.p4
arch.p4
• Stable.
• Updated rarely.
• Architecture-specific.
• Can be changed by target manufacturers.
• Includes "standard architecture."

*From: https://p4lang.github.io/p4-spec/docs/P4-16-v1.0.0-spec.html*

## Core abstractions

- *Header types*: define how your protocol headers look like
- *Parsers*: tell them how to parse headers
- *Tables*: contain state associating user-defined keys with actions
- *Actions*: describing how a packet should be manipulated
- *Match-Action units*: create lookup key, perform table lookup, execute action
- *Control flow:* specifies the order of applying match-action units
- *Extern objects*: registers, counters, meters, etc
- *Metadata*: data structures associated with each packet

## Header definitions

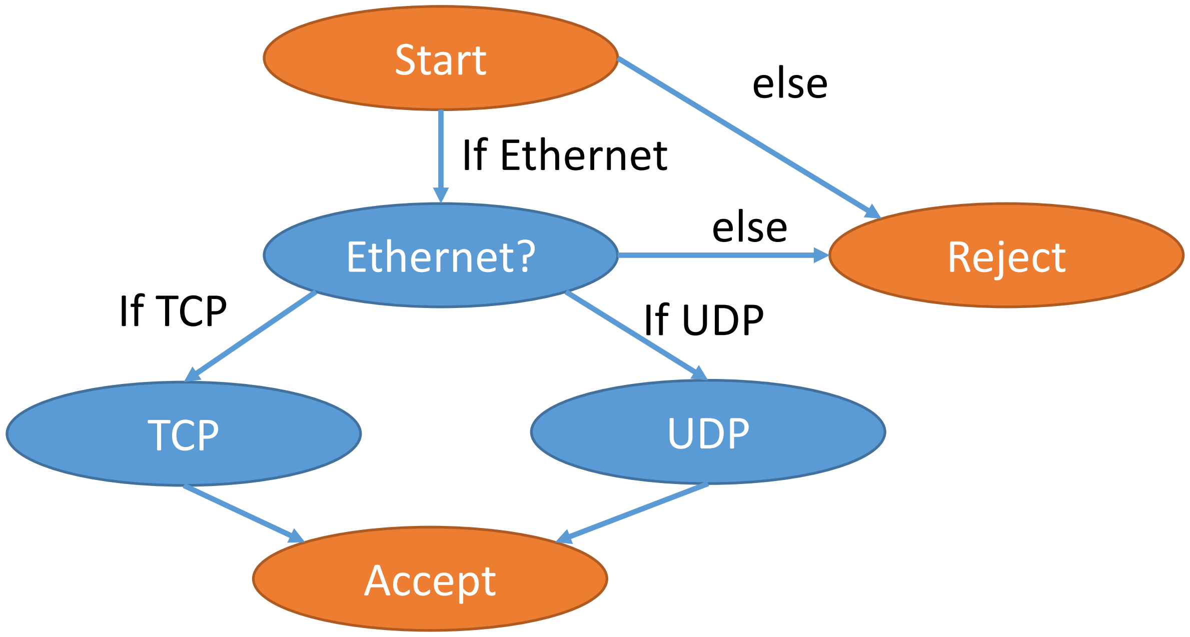
- Allows you, the developer to describe packet formats and name header fields
- Fixed vs variable length fields are both permitted

```
header Ethernet_h {
        bit<48> dstAddr;
        bit<48> srcAddr;
        bit<16> etherType;
}
```

```
header IPv4_h {
        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;
        bit<32> srcAddr;
        bit<32> dstAddr;
        varbit<320> options;
}
```

## Parser

- A finite state machine with one start state and two final states



```
parser MyParser(){
 state start {      transition parse_ethernet;    }
 state parse_ethernet {
             packet.extract(hdr.ethernet);
             transition select(hdr.ethernet.etherType) {
                          TYPE_IPV4: parse_ipv4;
                          default: accept;
             }
 }...
}
```

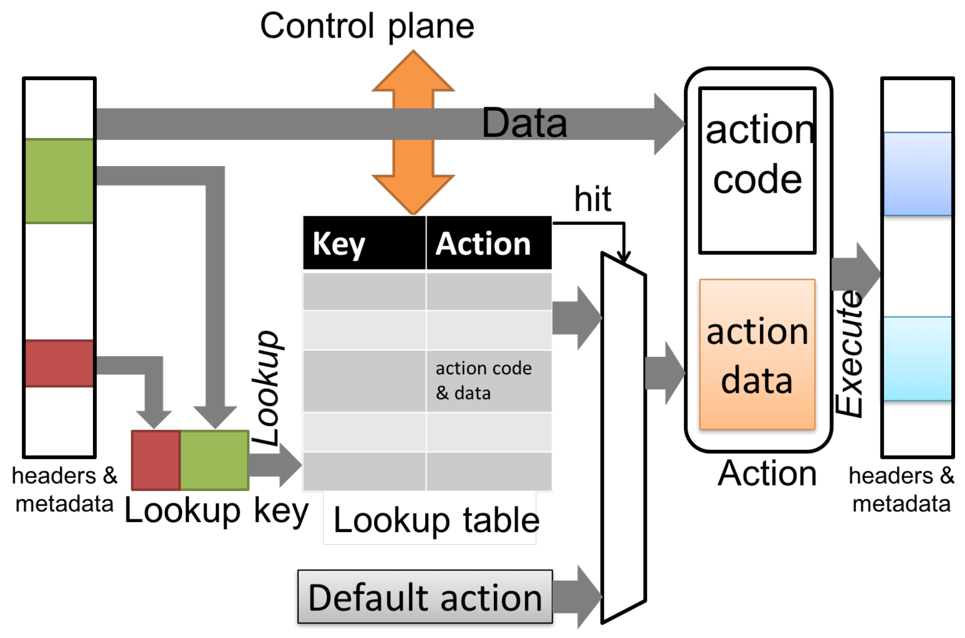
## Tables

- Tables contain state (pushed by the control plane) to forward packets
- Tables describe a *match-action unit*
- Packet matching can be done either:
  - Exact matching
  - Longest Prefix Match (LPM)
  - Ternary matching (masking)

```
table ipv4_lpm {
        reads {
            ipv4.dstAddr: lpm;
        }
        actions {
            forward()
        }
}
```

- All possible actions have to be defined in advance

# Match-Action units



*From: https://p4lang.github.io/p4-spec/docs/P4-16-v1.0.0-spec.html*

# Actions

- Actions consist of code and take action data
    - Action data comes from the control plane (e.g., IP addresses / port numbers)
- Specific, loop-free primitives can be executed in an action
- Number of instructions must be predictable, so no loops or conditional statements allowed here

```
action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
        standard_metadata.egress_spec = port;
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

## Externs

- Externs are architecture specific constructs with well defined APIs
- Examples:
  - Checksum calculation
  - Registers
  - Counters
  - Meters

```
extern register<T> {
        register(bit<32> size);
        void read(out T result, in bit<32> index);
        void write(in bit<32> index, in T value);
}
```

```
extern Checksum16 {
        Checksum16(); // constructor
        void clear(); // prepare unit for computation
        void update<T>(in T data); // add data to checksum
        void remove<T>(in T data); // remove data from existing checksum
        bit<16> get(); // get the checksum for the data added since last clear
}
```

## Metadata

1. User-defined metadata (empty *struct* at first for each packet)
   - You can put here anything you want
   - Accessible everywhere in the pipeline
   - Useful e.g., for storing the packet hash
2. Intrinsic metadata - **this is provided by the architecture**
   - Ingress port, egress port is defined here
   - Timestamp when packet was queued, queue depth
   - Multicast hash / multicast queue
   - Packet priority, packet color
   - Egress port specification (e.g., output queue)

## Control flow

- Stitching everything together
- Imperative program expressing the high-level logic and the sequence for match-action unit invocations

```
V1Switch(
ParserImpl(),
verifyChecksum(),
ingress(),
egress(),
computeChecksum(),
DeparserImpl()) main;
```

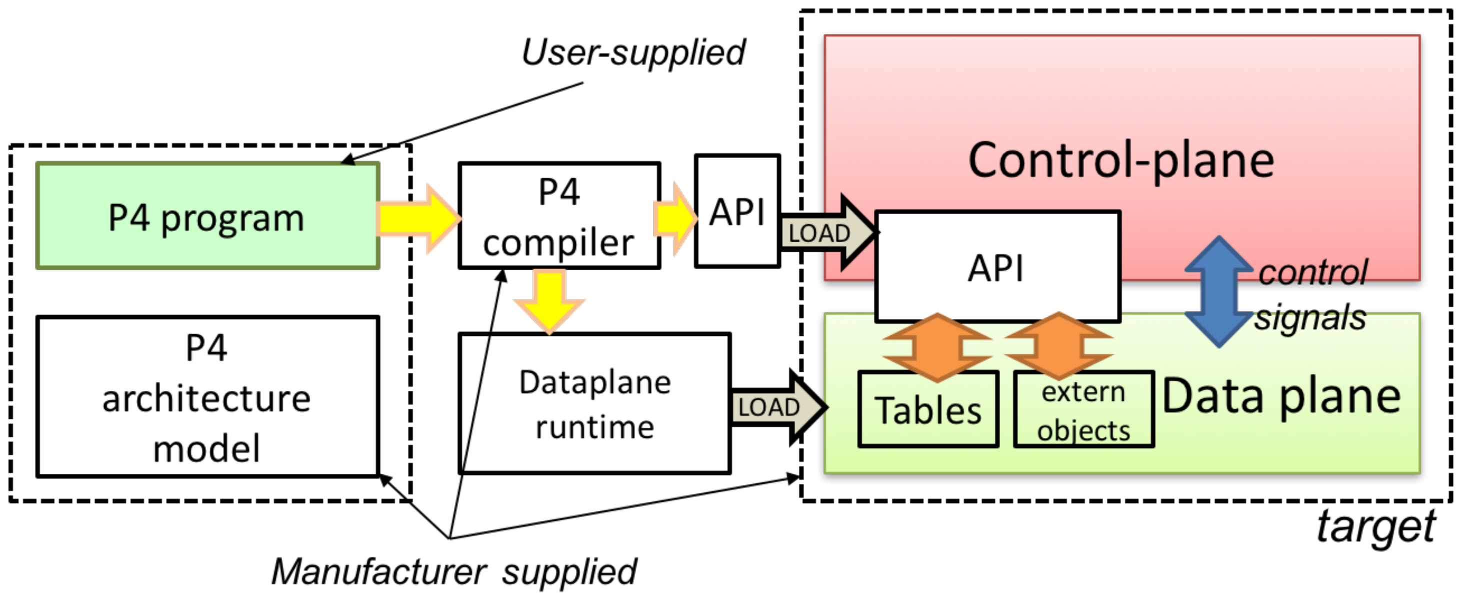
```
control ingress(){
              [table definitions here]
              [action definitions here]
              apply  {
                 if  (hdr.ipv4.isValid()) {
                    ipv4_lpm.apply();
                 }
               }
}
```

# P4 compiler, tools

P4 deep dive

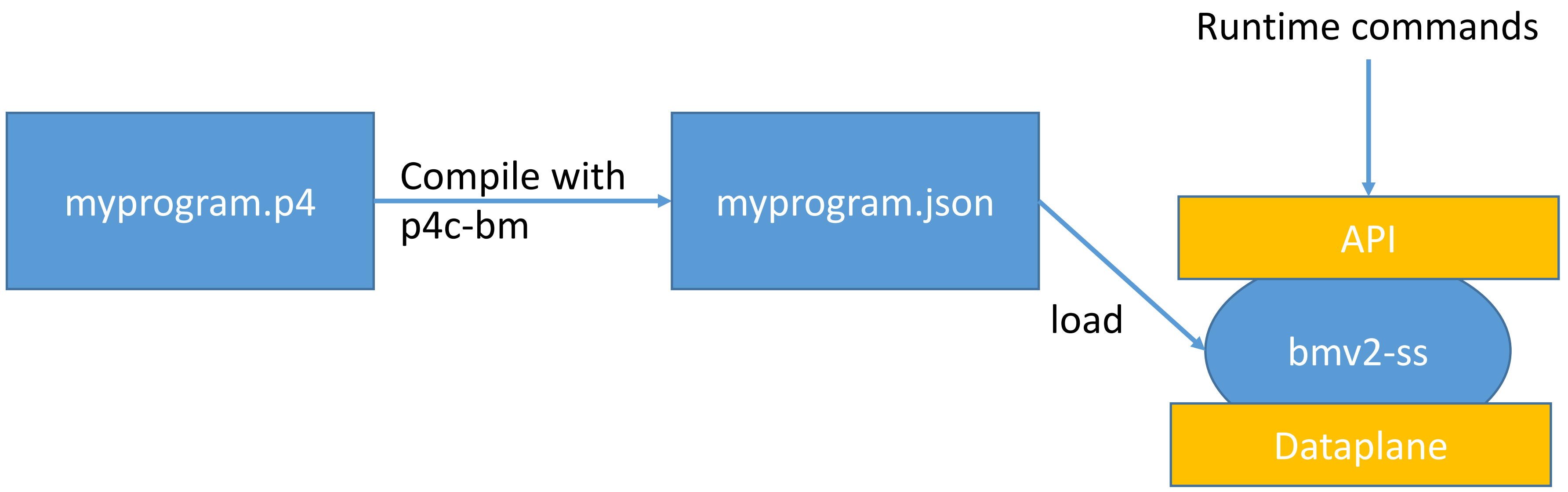
# P4 compiler

- The P4 compiler (p4c) produces:
  1. Data plane runtime
  2. API for managing the state in the data plane



# P4 software switch

- p4lang/p4c-bm: generates JSON configuration for bmv2
- p4lang/bmv2: software switch that understands the JSON config
  - bmv2: behavior model version 2



## Runtime API for software switch

- Allows you to manipulate tables, read registers, counters…

```
- table_set_default <table name> <action name> <action parameters>
- table_add <table name> <action name> <match fields> => <action parameters> [priority]
- table_delete <table name> <entry handle>
```

- API is easily usable with the provided **simple_switch_CLI** program

```
p4@p4d2:~$ simple_switch_CLI --thrift-port 9090
Obtaining JSON from switch...Done
Control utility for runtime P4 table manipulation
RuntimeCmd: counter_read counter 0
this is the direct counter for table
ipv4_lpmcounter[0]=  BmCounterValue(packets=4, bytes=88)
```

# Switch logs + debugger

- Simple switch logs are useful enough:

```
p4@p4d2:~$ tail -f /home/p4/p4_tutorial/P4D2_2017/exercises/mri/build/logs/s1.log
[20:58:45.689] [bmv2] [D] [thread 1572] [94.0] [cxt 0] Pipeline 'ingress': start
[20:58:45.689] [bmv2] [D] [thread 1572] [94.0] [cxt 0] Pipeline 'ingress': end
[20:58:45.689] [bmv2] [D] [thread 1572] [94.0] [cxt 0] Egress port is 0
….
[22:28:07.263] [bmv2] [T] [thread 1572] [166.0] [cxt 0] Applying table 'ipv4_lpm’
[22:28:07.263] [bmv2] [D] [thread 1572] [166.0] [cxt 0] Looking up key:
            * ipv4.dstAddr      : 0a00020a
[22:28:07.263] [bmv2] [D] [thread 1572] [166.0] [cxt 0] Table 'ipv4_lpm': hit with handle 1
```

- There is also a gdb like debugger:
  - Breakpoint, watching a header field, prints the value of a header field, etc

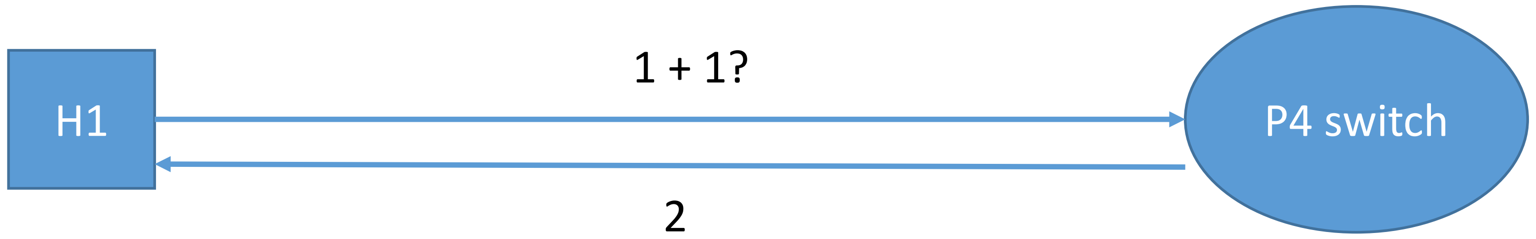
ESnet
ENERGY SCIENCES NETWORK

# P4 examples

P4 deep dive

## Calculator

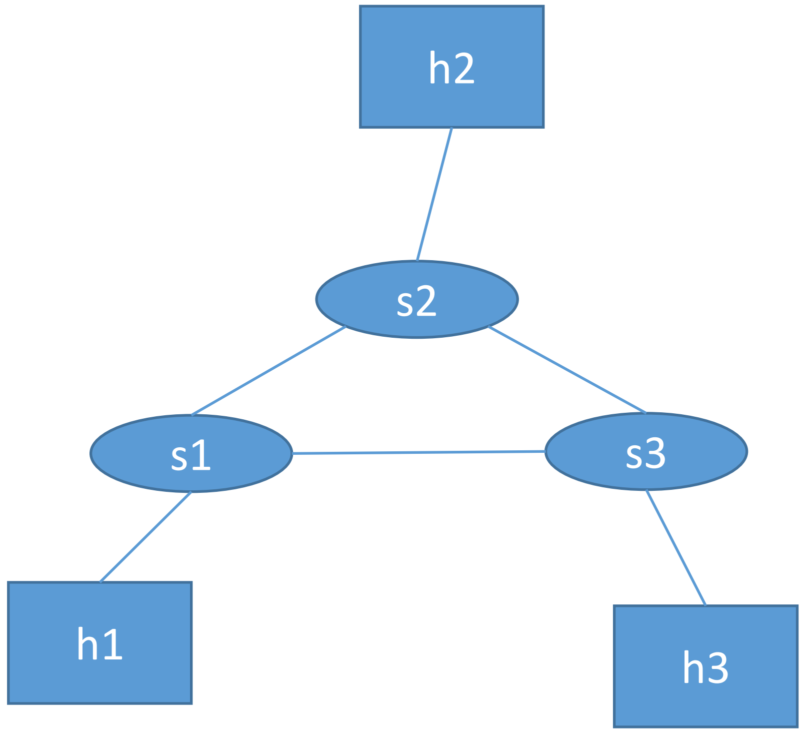
- Who said a network device can not be used as a **calculator**?
- Idea:
  - we send packets describing an operation (e.g., 1 + 1)
  - the device performs the operation and writes the result to the packet
  - the device sends the packet back to the sender that parses the result (2)



## A simple switch

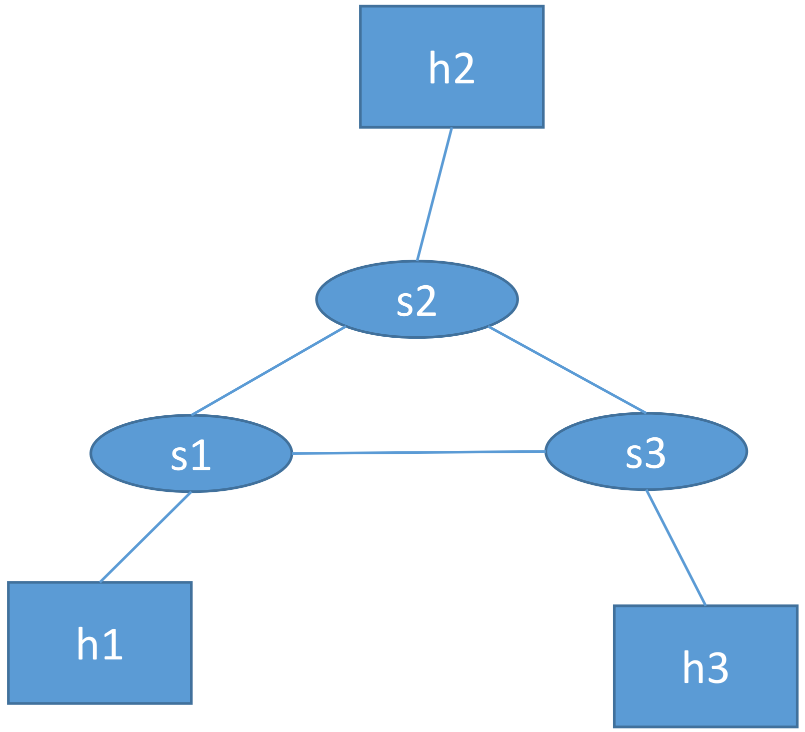
ESnet
ENERGY SCIENCES NETWORK

- Basic IPv4 forwarding switch
- Step by step:
  1. Packet is parsed (Ethernet, IPv4 only)
  2. Packet is matched using LPM on IPv4 destination addresses
  3. Egress port is set
  4. Packet's Ethernet source and destination is set
  5. TTL = TTL -1
  6. Checksum recalculation
- Additional: **counters**



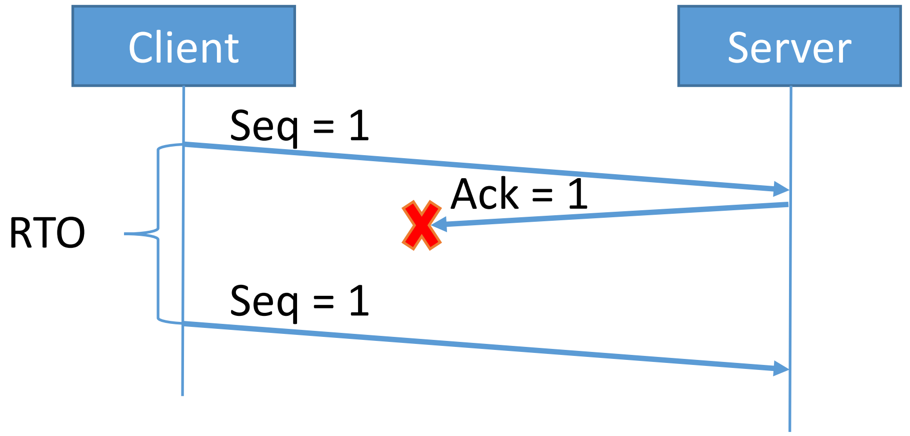
## Multi-Hop Route Inspection (MRI)

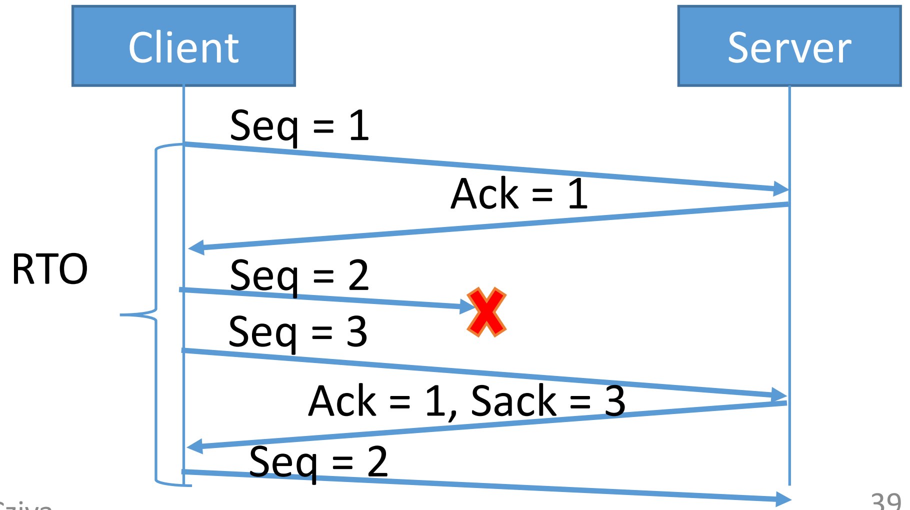
- Let’s find out where has our been in the network
- We will use TCP’s options field for the data (parsing a stack of headers)
- Every switch will write it’s own ID (*swid*) to the packet
- Packet from H1 --> H2 will collect *swids* [1, 2]



# TCP retransmission tracker

- Goal: Identify all flows that are experiencing TCP retransmission
  1. Track if we have seen the same TCP packet before (SYN number)
     1. Based on RTO (min 1s), exponential backoff
  2. Track SACK flags
     - Better with bursts of packets in one RTO
- ”Controller”: reads the counters from the data-plane





# PISA / PSA

- PSA = *Portable Switch Architecture* – WG in P4 consortium
- PISA = *Protocol Independent Switch Architecture* – reference chip design
- Reference architecture for a P4 programmable switch
  - Configurable parser, tables, etc.
  - Barefoot Tofino is a fully programmable PISA
  - “World’s fastest and most programmable switch series up to 6.5Tbps”
- Actual device: Edgecore Wedge100BF-65X: 2RU 65x100GE, 6.5 Tbit/s

## Takeaway

- P4 advocates a top-down approach for network programming
  - Instead of leaving the devices tell us how they process packets
    => we would like to tell them how to do it
- P4 is just one standard, many more to come, however:

In the next few years **data-plane programmability**
will become commonplace and P4 plays an important role in it

## References

- Figures and some demos have been selected:
  - P4 tutorial slides at SIGCOMM: https://github.com/p4lang/tutorials/blob/master/SIGCOMM_2016/p4-tutorial-slides.pdf
  - P4 language docs: https://p4lang.github.io/p4-spec/docs/P4-16-v1.0.0-spec.html
  - Examples from Github: https://github.com/p4lang/tutorials
- P4 paper

## Thank you

Thank you for your attention!