Crafting the data plane

Carolina Fernández

Bio

- Carolina Fernández
- R&D Engineer at i2cat
- Working on networks, virtualisation, automation
  - SDN, NFV applied to MEC, 5G, security, ...
- More interests: privacy *et al*

CarolinaFernandez
carolinafernandez.github.io

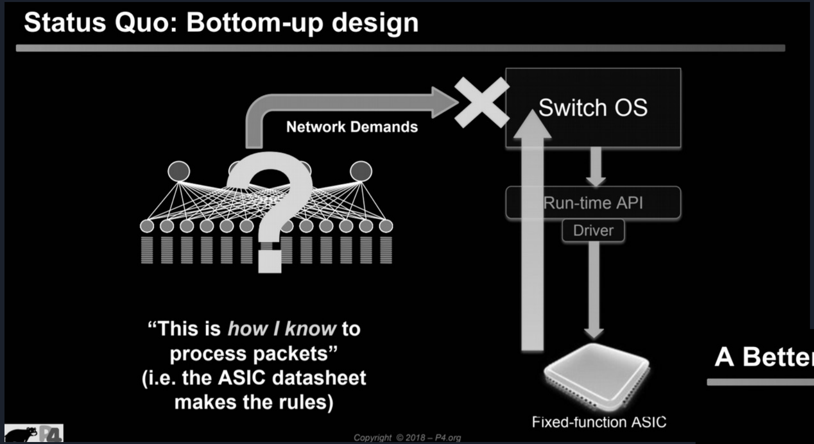
cfermart

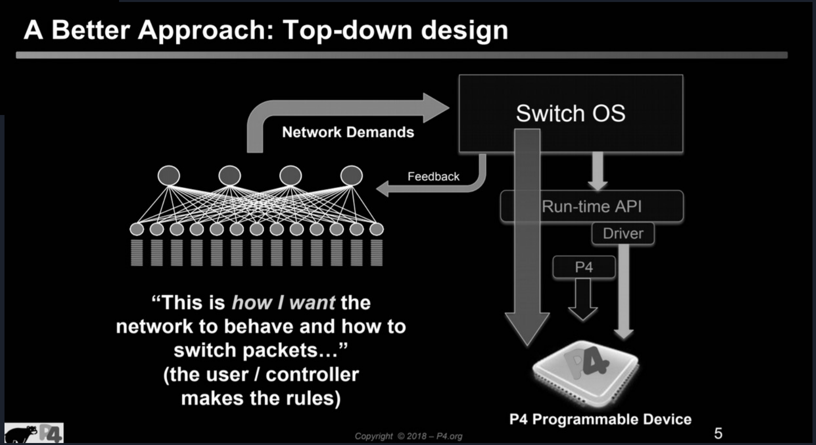
Agenda

1. Considerations
   - Traditional vs flexible data plane interactions
   - Languages
   - Aim & use cases
2. Portability
   - Architectures and targets
3. Language elements
   - Program structure
   - Includes, metadata, headers
   - Parsers
   - Control blocks
3. Language elements
   - Tables, actions and primitives
   - Stateful objects
   - Recursiveness
   - Checksum
4. Running & configuring in P4
   - Compiling and running an app
   - P4Runtime: configuring tables
5. Materials and references
   - Pointers
   - Tools

Considerations

Traditional vs flexible data plane interactions





Source: https://bit.ly/p4d2-2018-spring

Languages

Some examples:

Language	Supporters	First spec	Current version	Comments
POF *(Protocol Oblivious Forwarding)* Protocol-independent instruction set to allow defining the protocol stack & packet processing (enhanced version of OpenFlow/1.3)	Huawei	2013	?	
P4 *(Programming Protocol-Independent Packet Processor)* High-level language to program SDN switch flexibly	Open Networking Foundation	2014/08 *(idea in 2013/05)*	v1.2.0	**Specs** P4_14 / $P4_{14}$ P4_16 / $P4_{16}$
NPL *(Network Programming Language)* Similar to P4	Broadcom	2019/06	v1.3	

Aim & use cases (1)

Used to:

- Implement specific protocols
- Define specific, custom packets
- Maximise efficiency for low-level processing
- Benefit from typical operations at the switch (e.g., mirroring packets) & at the end nodes (e.g., move packet to CPU)

NOT used to:

- Insert rules in the forwarding table (programming the control plane)
- Perform some typical operations at end nodes (e.g., traffic generation, monitoring)

Examples:

- Layer 4 Load Balancer – SilkRoad
- Low Latency Congestion Control – NDP
- In-band Network Telemetry – INT
- In-Network DDoS detection
- In-Network caching and coordination – NetCache / NetChain
- Consensus at network speed – NetPaxos
- Aggregation for MapReduce Applications

Aim & use cases (1)

Used to:

- Implement specific protocols
- Define specific, custom packets
- Maximise efficiency for low-level processing
- Benefit from typical operations at the switch (e.g., mirroring packets) & at the end nodes (e.g., move packet to CPU)

Typical data plane:

- Pipeline hard-coded by the vendor
- Set of default protocols supported

***Virtualised* data plane:**

- Pipeline defined by the user
- Custom set of protocols supported

NOT used to:

- Insert rules in the forwarding table (programming the control plane)
- Perform some typical operations at end nodes (e.g., traffic generation, monitoring)

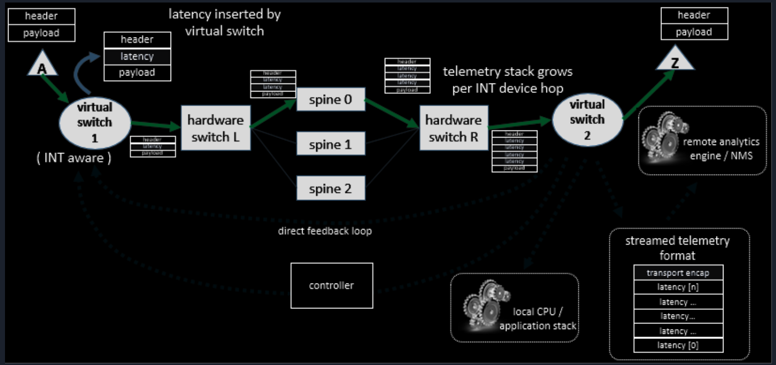
Examples:

- Layer 4 Load Balancer – SilkRoad
- Low Latency Congestion Control – NDP
- In-band Network Telemetry – INT
- In-Network DDoS detection
- In-Network caching and coordination – NetCache / NetChain
- Consensus at network speed – NetPaxos
- Aggregation for MapReduce Applications

Aim & use cases (2): sample use case #1

In-Band Network Telemetry (INT): monitoring network with a reduced footprint (CPU, I/O)

- Device internal state (packet counters, timestamps, etc) exported from data plane
- Use headers on traversing packets to include telemetry data by
  - Re-using existing fields (e.g., custom TCP option) suitable for legacy networks; so that internal transport devices allows the packets transparently
  - Creating a custom packet format, where intermediate devices are able to parse them



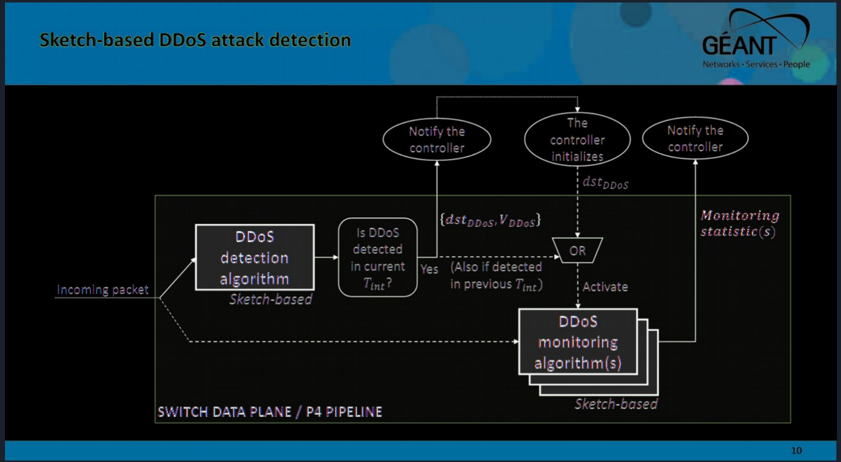
✓ No need for external devices to constantly request information and less end-processing

Source: https://p4.org/p4/inband-network-telemetry/ , https://www.youtube.com/watch?v=FOOL5BeHNVY

Aim & use cases (2): sample use case #2

In-Network DDoS Detection: monitoring network for quick identification and possible reaction

- Detect heavy flows based on data plane statistics and models on network threats
- Direct collaboration with network controller to i) provide statistic data (detection), ii) to configure tables for any remediation action (mitigation)



✓ Less need for scaling (less external detection or even mitigation appliances/applications)

Source: https://p4.org/p4/geant.html , https://wiki.geant.org/display/SIGNGN/2nd+SIG-NGN+Meeting

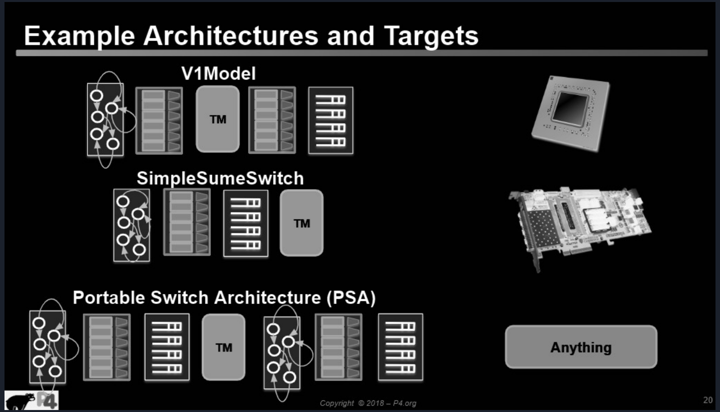
Portability

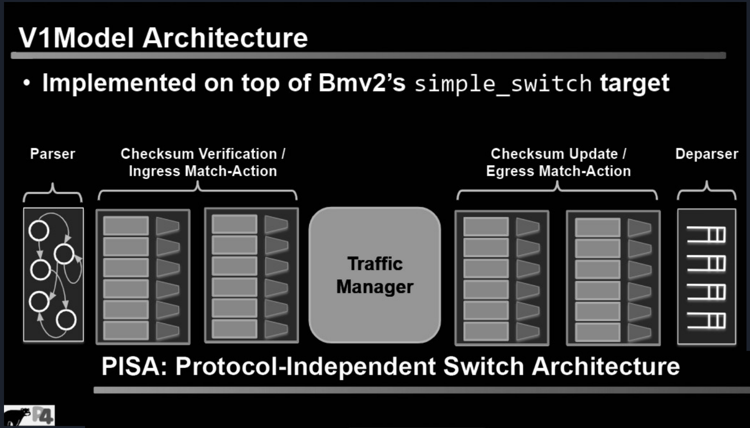
Architectures and targets (1): definition

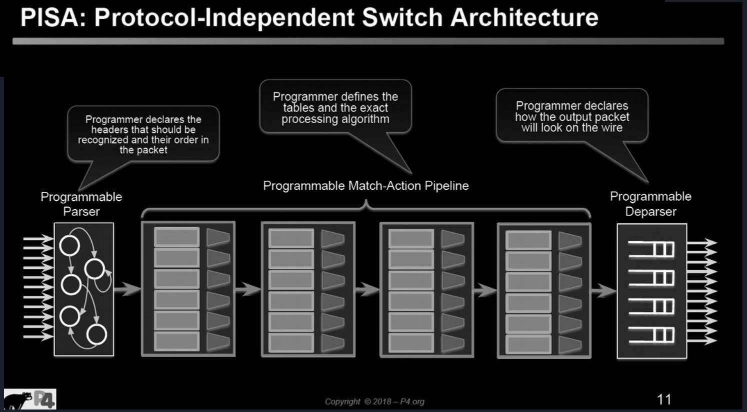
- The architecture is the *programming model*, the *abstraction*
  - Logical view of the pipeline of the virtual/hardware target (device): how the data plane programmer thinks about the underlying platform
  - Enable programming multiple targets *(switches, routers, NICs, OVSs)*
  - Isolate programmer from the target details. Providers define architectures and compiler backends to map architectures to targets

Term	Explanation
Target	Definition of specific HW implementation (e,g,, Tofino)
Architecture	Set of programmable components, externs, fixed components and their interfaces available (e,g,, PISA)
Platform	Architecture implemented on a given target

Architectures and targets (2): examples



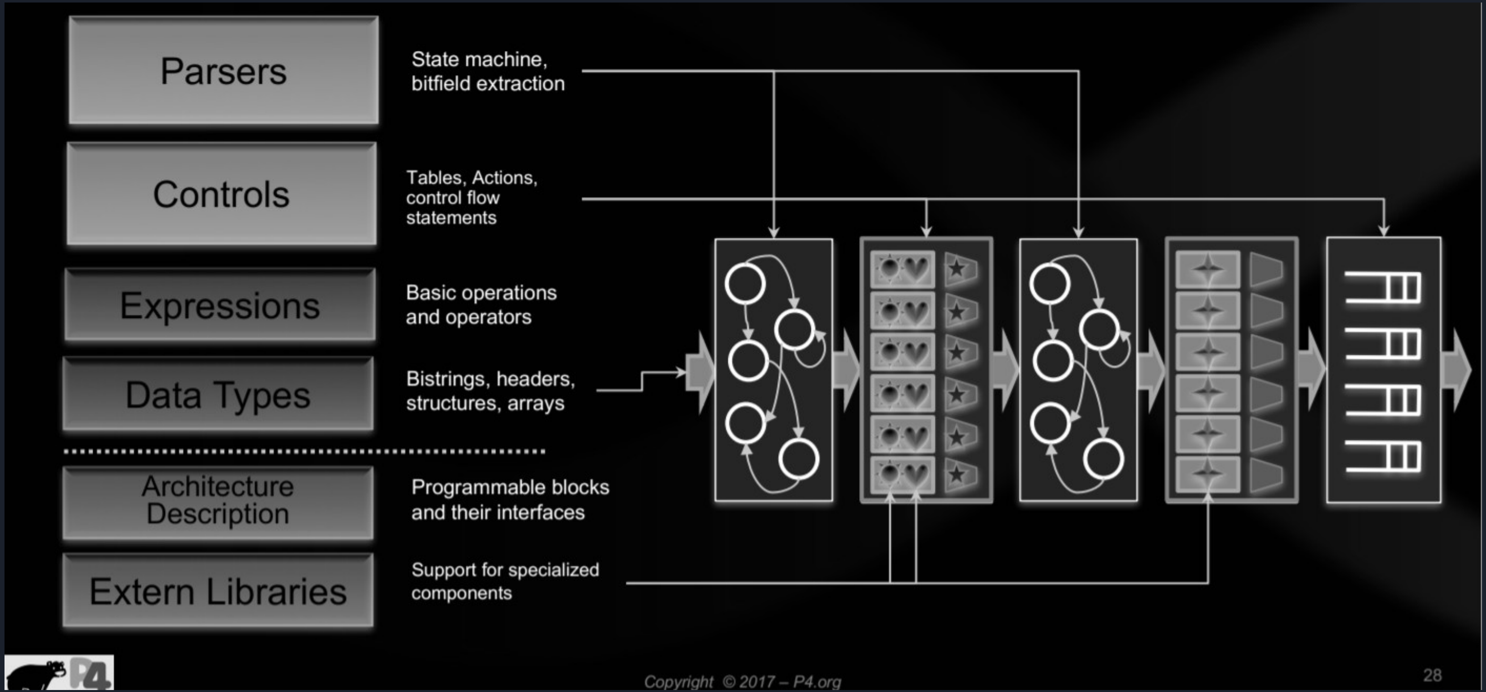




Source: https://bit.ly/p4d2-2018-spring, https://p4.org/assets/p4_d2_2017_p4_16_tutorial.pdf

Language elements

P4$_{16}$'s language elements



Source: *https://p4.org/assets/p4_d2_2017_p4_16_tutorial.pdf*

P4$_{16}$'s program (1)

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
         out headers hdr,
         inout metadata meta,
         inout standard_metadata_t smeta) {
  ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
  ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  ...
}
```

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
  ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
  ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
  ...
}
/* SWITCH */
V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```


21

Source: https://bit.ly/p4d2-2018-spring

P4$_{16}$'s program (2)

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet, out headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    state start { transition accept; }
}

control MyIngress(inout headers hdr, inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    action set_egress_spec(bit<9> port) {
        standard_metadata.egress_spec = port;
    }
    table forward {
        key = { standard_metadata.ingress_port: exact; }
        actions = {
            set_egress_spec;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }
    apply {   forward.apply();   }
}

control MyEgress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    apply {  }
}

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

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

control MyDeparser(packet_out packet, in headers hdr) {
    apply { }
}

V1Switch( MyParser(), MyVerifyChecksum(), MyIngress(),
MyEgress(), MyComputeChecksum(), MyDeparser() ) main;
```

Key	Action Name	Action Data
1	set_egress_spec	2
2	set_egress_spec	1

23

Source: https://bit.ly/p4d2-2018-spring

Program elements (1)

1 **Includes, metadata & headers/structs**

- Import system or custom p4 files
- Define metadata
- Define structs
- Define headers (= *struct* + *validity*)

2 **Parser**

- State machine with 1 start ("accept"), 2 final ("accept", "reject") states
- Extract the packet; move between transitions based on the fields

4 5 **Control: Ingress/Egress**

- Define behaviour of actions
- Define tables and link to actions
- Apply logic of tables based on conditions

Control: Checksum

- 3 Verify checksum
- 6 Compute checksum

7 **Deparser**

- Emits a consolidated packet
- Headers only appended to the packet if these are valid
- Headers are concatenated (*in order of increasing indexes*)

Switch definition

- Sequence of elements (*see numbers*) as sections in the program

NPL Elements not always explicitly defined. *Special functions (IARB, MMU, EDB)* transition to following stage

Program elements (2): 1/includes

- System files or your own programs can be imported
- *(P4)* The import is typically done at the beginning of the file; but can also be imported in other locations
  - For instance; when assigned to a variable

P4

```
// core library needed for packet_in and packet_out definitions
# include <core.p4>
// Include very simple switch architecture declarations
# include "very_simple_switch_model.p4"
```

Program elements (3): 1/metadata

P4 **Metadata** is used as a way to persist intermediate values which are used in the logic of the program, whether for ingress or egress processing. *Life of such data constrained to the life of the packet*

Types:

Standard (intrinsic)
Incorporated in P4's libraries

User-defined
Defined by user through a type / struct

```
action send_to_port(port) {
  standard_meta.egress_port = port;
}
action keep_result(bit<32> res) {
  user_meta.output = res;
}
```

NPL **Buses** communicate results between ingress & egress pipelines. Validity of data not constrained to packet's lifetime

Program elements (4): 1/metadata

Struct **standard_metadata_t** contains the following fields. These can be used to store intermediate data

V1Model Standard Metadata

	`struct standard_metadata_t {`
Ingress/egress movement >	`bit<9> ingress_port;`
Ingress/egress movement >	`bit<9> egress_spec;`
Ingress/egress movement >	`bit<9> egress_port;`
Recursive processing >	`bit<32> clone_spec;`
Recursive processing >	`bit<32> instance_type;`
	`bit<1> drop;`
Recursive processing >	`bit<16> recirculate_port;`
	`bit<32> packet_length;`
Queue management >	`bit<32> enq_timestamp;`
Queue management >	`bit<19> enq_qdepth;`
Queue management >	`bit<32> deq_timedelta;`
Queue management >	`bit<19> deq_qdepth;`
	`bit<48> ingress_global_timestamp;`
	`bit<32> lf_field_list;`
Ingress/egress movement >	`bit<16> mcast_grp;`
Recursive processing >	`bit<1> resubmit_flag;`
Ingress/egress movement >	`bit<16> egress_rid;`
Checksum >	`bit<1> checksum_error;`
	`}`

Source: https://github.com/p4lang/behavioral-model/blob/master/docs/simple_switch.md, https://bit.ly/p4d2-2018-spring

Program elements (5): 1/headers

P4

- Header: struct *(C-like)* + “validity” field *(hidden)*
  - Defines any kind of packet headers:
    - IPv4, IPv6, Ethernet, …
  - Methods: `isValid(), setValid(), setInvalid()`
- Protocol headers recognised & processed by the program
- Ordering
  - Order of fields in declaration ⇔ order of fields in wire
  - Packet has no gaps between fields
  - Packet header length must be multiple of 8 bytes
- Initially, all headers are invalid
  - *Note: accessing invalid header fields leads to undefined behaviours*
  - Successful extract() of header → validity bit = “true”

NPL

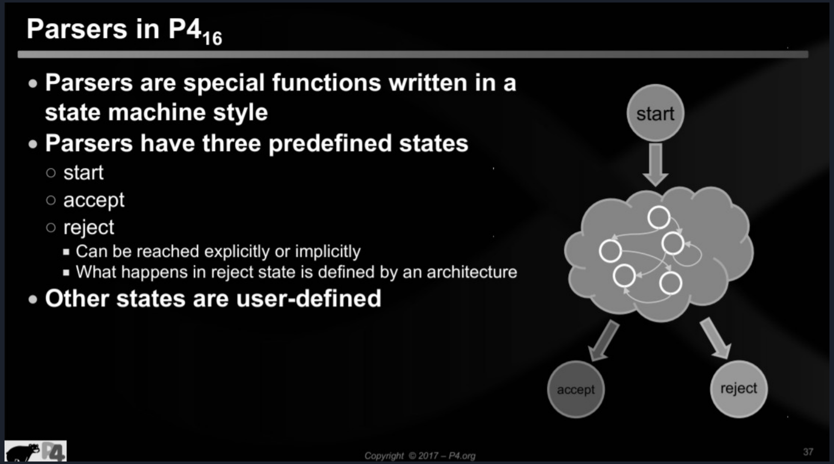
- Headers are structs

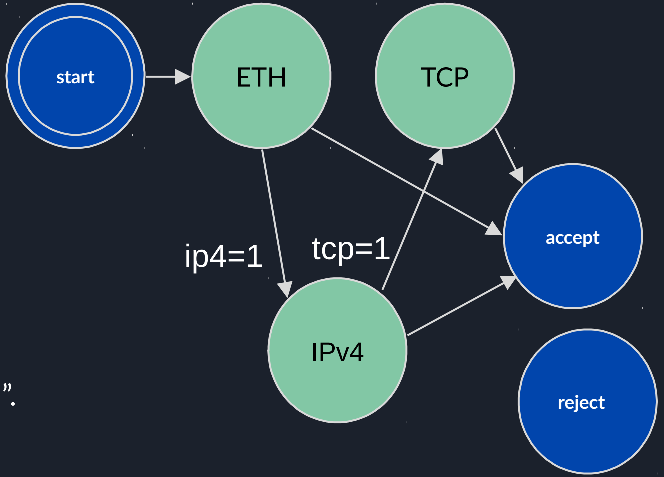
Example: Declaring L2 headers

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

header vlan_tag_t {
    bit<3>      pri;
    bit<1>      cfi;
    bit<12>     vid;
    bit<16>     etherType;
}

struct my_headers_t {
    ethernet_t    ethernet;
    vlan_tag_t[2] vlan_tag;
}
```


Source: https://p4.org/assets/p4_d2_2017_p4_16_tutorial.pdf

Program elements (6): 2&7/parsers





NPL “start” node =~ “root_node: 1”; “accept” node =~ “end_node: 1”.
Re-entrant parser *(invoked from further stages)*

Note: parsing and deparsing are done in a left-to-right fashion (e.g., as the packet would be pictured)

Source: https://p4.org/assets/p4_d2_2017_p4_16_tutorial.pdf

Program elements (7): 2&7/parsers

Implementing Parser State Machine

```
parser MyParser(packet_in             packet,
                out    my_headers_t        hdr,
                inout  my_metadata_t       meta,
                in     standard_metadata_t standard_metadata)
{
    state start {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            0x8100 &&& 0xEFFF : parse_vlan_tag;
            0x0800  : parse_ipv4;
            0x86DD  : parse_ipv6;
            0x0806  : parse_arp;
            default : accept;
        }
    }

    state parse_vlan_tag {
        packet.extract(hdr.vlan_tag.next);
        transition select(hdr.vlan_tag.last.etherType) {
             0x8100 : parse_vlan_tag;
             0x0800 : parse_ipv4;
             0x86DD : parse_ipv6;
             0x0806 : parse_arp;
            default : accept;
        }
    }
```

```
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.ihl) {
            0 .. 4: reject;
                 5: accept;
           default: parse_ipv4_options;
    }

    state parse_ipv4_options {
        packet.extract(hdr.ipv4.options,
                       (hdr.ipv4.ihl - 5) << 2);
        transition accept;
    }

    state parse_ipv6 {
        packet.extract(hdr.ipv6);
        transition accept;
    }
```

- $P4_{16}$ has a `select` statement that can be used to branch in a parser
- Similar to `case` statements in C or Java, but without “fall-through behavior”—i.e., `break` statements are not needed
- In parsers it is often necessary to branch based on some of the bits just parsed
- For example, etherType determines the format of the rest of the packet
- Match patterns can either be literals or simple computations such as masks

 38 – P4.org 35

Source: https://bit.ly/p4d2-2018-spring

Program elements (8): 4&5/control blocks

- Must follow a Direct Acyclic Graph (DAG) processing *(no loops)*

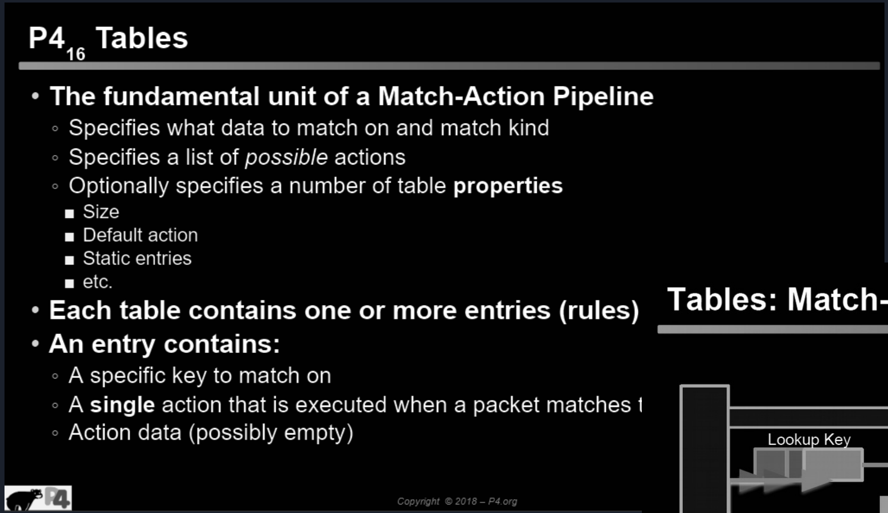
- apply() performs match-action in a table
- apply() { ... } uses match results to determine further processing
  - hit/miss clause
  - selected action clause
- Conditional statements
  - Comparison operations: (==, !=, >, <, >=, <=)
  - Logical operations (not, and, or)
  - Header validity checks (*unknown results otherwise*)

- During the the “apply” method evaluation, the “hit” field is set to true if a match is found in the lookup-table. That can be used to drive the execution of the control-flow in the control block that invoked the table

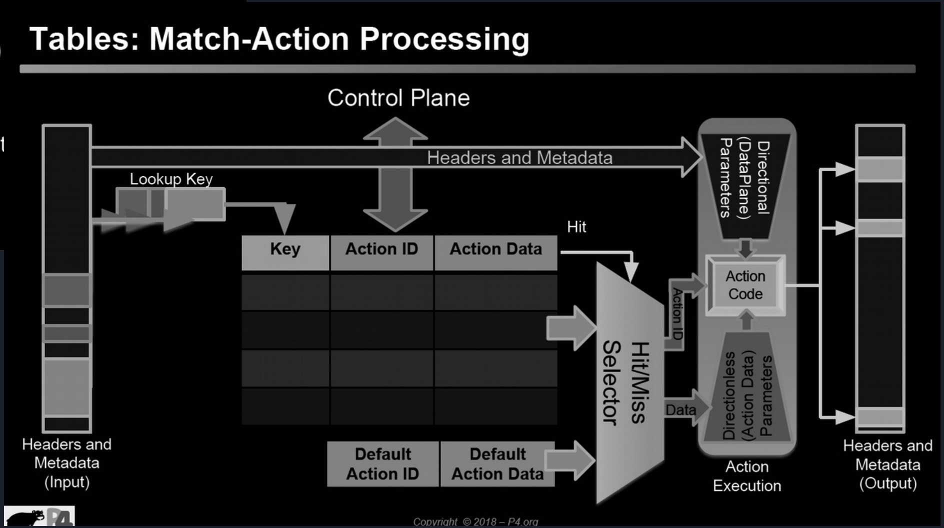
```
  apply {
    if (hdr.ipv4.isValid() &&
hdr.ipv4.ttl > 0) {
        ecmp_group.apply();
            ecmp_nhop.apply();
        }
  }
```

```
# Internal evaluation
  if (ipv4_match.apply().hit) {
      // There was a hit
  } else {
      // There was a miss
  }
```

Program elements (9): 4&5/tables



Architecture	Match kinds
Core	exact, ternary *(bitmask)* , lpm *(longest-prefix)*
V1Model	range, selector



NPL (Logical) tables have different matches (“*index*”, “*hash*”, ...). “Fields” assigned during table “*lookup*” (instead of “*apply*”).

Source: [*https://p4.org/assets/p4-ws-2017-p4-architectures.pdf*](https://p4.org/assets/p4-ws-2017-p4-architectures.pdf)

Program elements (9): 4&5/tables

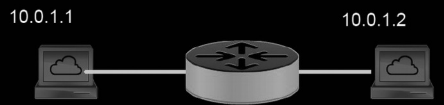
$P4_{16}$ Tables

- **The fundamental unit of a Match-Action Pipeline**
  - Specifies what data to match on and match kind
  - Specifies a list of *possible* actions
  - Optionally specifies a number of table **properties**
    - Size
    - Default action
    - Static entries
    - etc.
- **Each table contains one or more entries (rules)**
- **An entry contains:**
  - A specific key to match on
  - A **single** action that is executed when a packet matches t
  - Action data (possibly empty)

Architecture	Match kinds
Core	exact, ternary *(bitmask)* , lpm *(longest-prefix)*
V1Model	range, selector

Example: IPv4_LPM Ta

Action data to be filled by control plane



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 the lookup
  - Executes the chosen action
- **Control Plane (IP stack, Routing protocols)**
  - Populates table entries with specific information
    - Based on the configuration
    - Based on automatic discovery
    - Based on protocol calculations

37

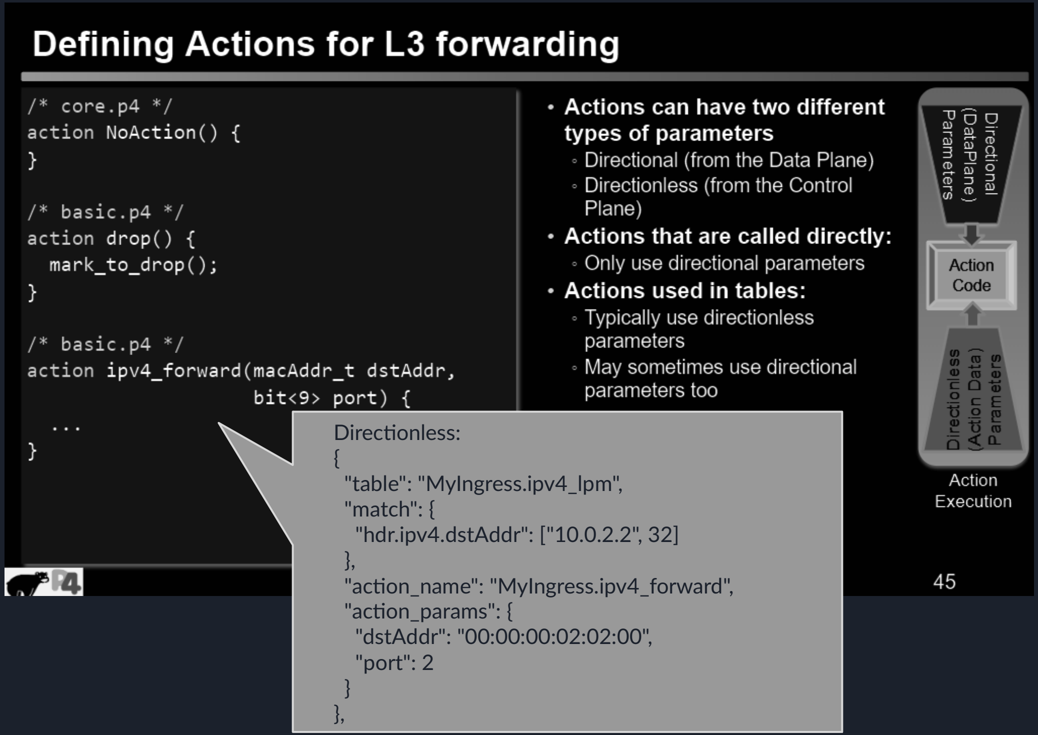
NPL (Logical) tables have different matches ("*index*", "*hash*", ...). "Fields" assigned during table "*lookup*" (instead of "*apply*").

Source: https://p4.org/assets/p4-ws-2017-p4-architectures.pdf

Program elements (9): 4&5/actions

Action:

- May contain data values (written via control plane, read by data plane) -- *the control plane can influence dynamically the behaviour of the data plane*
- Primitives and other actions called inside
- Operate on headers, metadata, constants, action data
- Linked to 1..N tables
- Sequential execution
- By default: NoAction



Source: [*https://bit.ly/p4d2-2018-spring*](https://bit.ly/p4d2-2018-spring)

Program elements (10): 4&5/primitives

Note: used inside actions, may affect metadata

Types:

- Basic: no operation, drop, emit,…
- Moving data: modify fields, shift, …
- Calculations: boolean, bitwise, hash-based, random number generators, min, max, …
- Headers: add, copy, remove, …
- Stateful objects: count, execute meter, read/write register, …
- Recursive processing: clone packet {in ingress to reappear at egress, in egress to reappear at egress}, resubmit (re-send after crossing ingress pipeline), recirculate (re-send after crossing both pipelines)
- Interaction: copy packet to CPU, …
- …

NPL Very different set of primitives

Program elements (11): 4&5/stateful objects

- P4 objects can be classified by their lifespan
  - Stateless (transient): state is not preserved upon processing *(life within packet)*
    - Metadata
    - Packet headers
  - Stateful (persistent): state is preserved upon processing *(outlives the packet)*
    - Counters *(associate data to entries in table; i.e., count #{packets, bytes, both})*
    - Meters *(colour & measure data rate: packets/second, bytes/second)*
    - Registers *(sort of counters that can be operated from actions in a general way)*

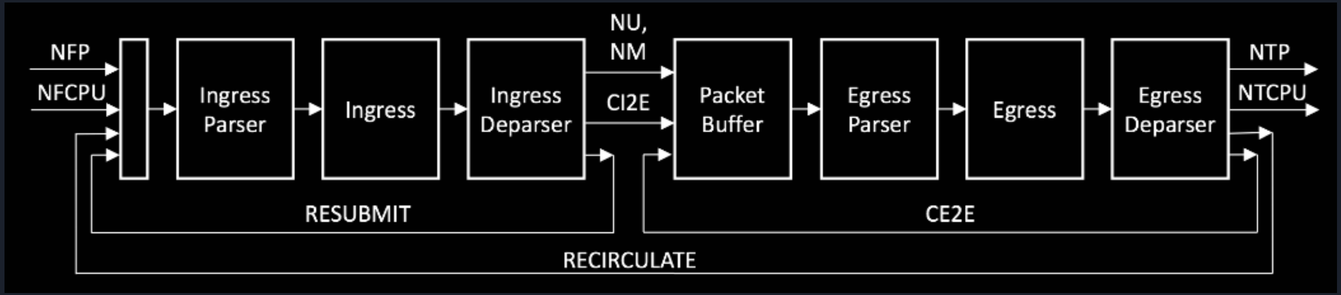
- Aim: persist state for longer than one packet (stateful memories)
- Allow complex, interesting processing over data
- These require resources on the target and hence are managed by a compiler

Program elements (12): 4&5/recursiveness

Complex parsing may require a packet to be processed recursively by being:

- duplicated (**clone**d) – packet appears at egress (*from ingress*: **CloneType.I2E,** *from egress*: **CloneType.E2E**)
- re-sent from ingress to ingress (**resubmit**ted) – e.g., further processing in ingress pipeline (ex., since P4 does now allow applying a table multiple times, this is the way to go);
- re-sent from egress to ingress (**recirculate**d) – e.g., reuse original packet upon modifications in egress pipeline

Note: *implementation of such features depends on the architecture – e.g., in the "simple_switch", the metadata is only copied at the end of the current pipeline where the packet is cloned*



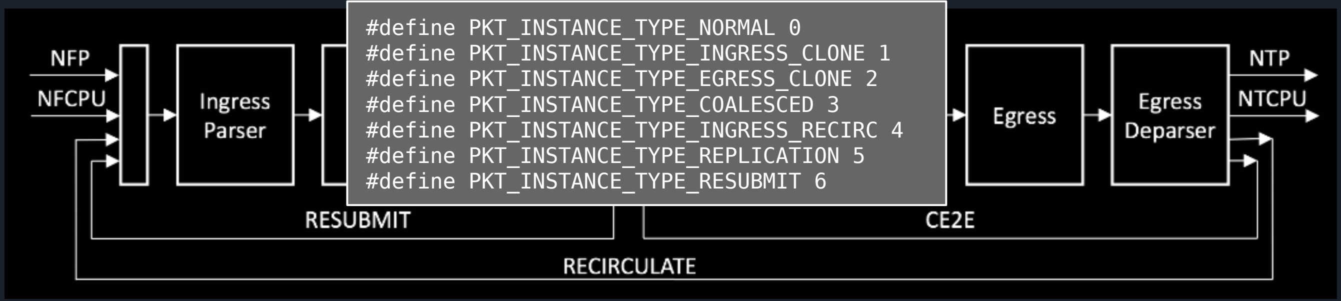
Source: https://p4.org/p4-spec/docs/PSA-v1.1.0.html

Program elements (12): 4&5/recursiveness

Complex parsing may require a packet to be processed recursively by being:

- duplicated (**clone**d) – packet appears at egress (*from ingress*: **CloneType.I2E,** *from egress*: **CloneType.E2E**)
- re-sent from ingress to ingress (**resubmit**ted) – e.g., further processing in ingress pipeline (ex., since P4 does now allow applying a table multiple times, this is the way to go);
- re-sent from egress to ingress (**recirculate**d) – e.g., reuse original packet upon modifications in egress pipeline

Note: *implementation of such features depends on the architecture – e.g., in the "simple_switch", the metadata is only copied at the end of the current pipeline where the packet is cloned*




```
#define PKT_INSTANCE_TYPE_NORMAL 0
#define PKT_INSTANCE_TYPE_INGRESS_CLONE 1
#define PKT_INSTANCE_TYPE_EGRESS_CLONE 2
#define PKT_INSTANCE_TYPE_COALESCED 3
#define PKT_INSTANCE_TYPE_INGRESS_RECIRC 4
#define PKT_INSTANCE_TYPE_REPLICATION 5
#define PKT_INSTANCE_TYPE_RESUBMIT 6
```

Source: https://p4.org/p4-spec/docs/PSA-v1.1.0.html

Program elements (13): 3&6/checksum

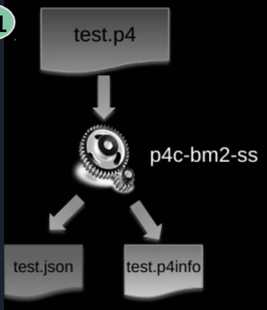
- Checksum can be **verified** and **computed**
  - Depends on switch architecture (e.g., in the VSS arch., the "Checksum16" extern is available)
  - Verified (for error correction):
    - If checksum does not match, pkt is discarded
    - If checksum matches, removed from pkt payload
- "hdr.ipv4.hdrChecksum" is a calculated field — ensures the egress packet has a correct IPv4 header checksum
  - Creates a list of fields that participate in checksum calculation, and the calculation parameters

```
update_checksum(
  hdr.ipv4.isValid(),
  {
    hdr.ipv4.version,
    hdr.ipv4.ihl,
    hdr.ipv4.diffserv,
    hdr.ipv4.totalLen,
    hdr.ipv4.identification,
    hdr.ipv4.fragOffset,
    hdr.ipv4.ttl,
    hdr.ipv4.protocol,
    hdr.ipv4.srcAddr,
    hdr.ipv4.dstAddr
  },
  hdr.ipv4.hdrChecksum,
  HashAlgorithm.csum16);
```

Running & configuring in P4

Compiling and running an app (1)

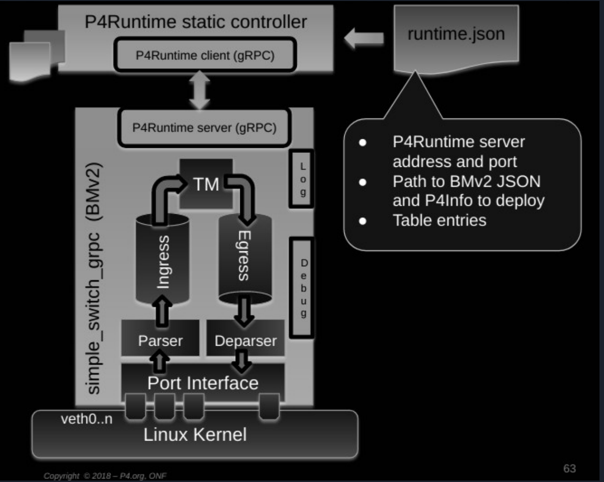
1




```
$ p4c-bm2-ss --p4v 16 \
    -o test.json \
    --p4runtime-file test.p4info \
    --p4runtime-format text \
    test.p4
```

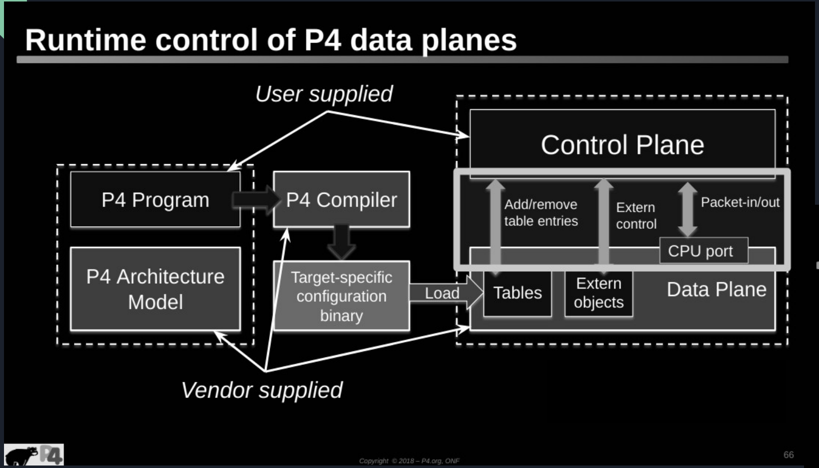
2

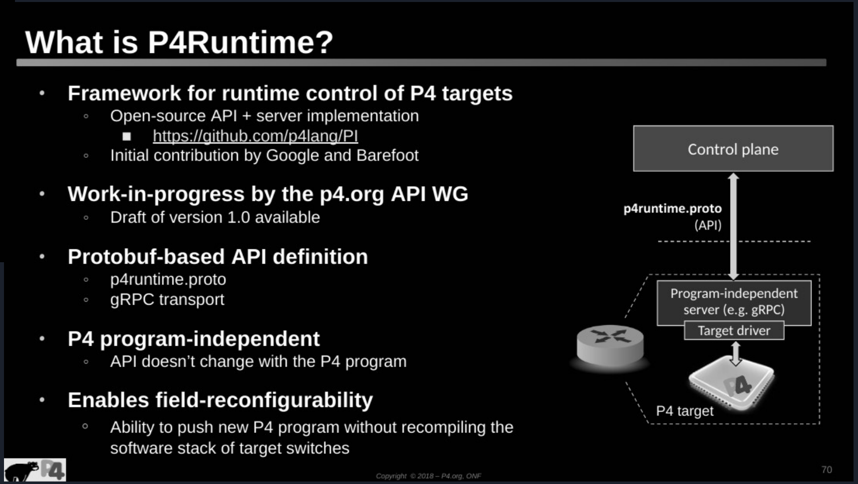
a. **Create network based on topology.json**
b. **Start** `simple_switch_grpc` **instance for each switch**
c. **Use P4Runtime to push the P4 program (P4Info and BMv2 JSON)**
d. **Add the static rules defined in** `runtime.json`



Source: https://bit.ly/p4d2-2018-spring

Compiling and running an app (2)





Source: https://bit.ly/p4d2-2018-spring

P4Runtime: configuring tables (1)

P4Runtime provides Target & Protocol independent API to control the data plane (fills it with commands and flows)

```
sX-commands.txt (send flows as commands)

table_set_default switchp_nhop drop
table_set_default switchp_tag add_switchp_tag 1
table_add switchp_nhop set_nhop 10.1.1.2/32 => 2 0
table_add switchp_nhop set_nhop 10.1.1.1/32 => 1 1
```

table name action name match action arguments

```
sX-runtime.json (send flows as structures)

{
  "target": "bmv2",
  "p4info": "build/clone.p4.p4info.txt",
  "bmv2_json": "build/clone.json",
  "table_entries": [
    …
    ]
}
```

```
sX-runtime.json (send flows as structures)

  {
     "table": "MyIngress.switchp_nhop",
     "default_action": true,
     "action_name": "MyIngress.drop",
     "action_params": { }
   },
   {
     "table": "MyIngress.switchp_tag",
     "default_action": true,
     "action_name": "MyIngress.add_switchp_tag",
     "action_params": { }
   },
   {
     "table": "MyIngress.switchp_nhop",
     "match": {
       "hdr.ipv4.dstAddr": ["10.1.1.2", 32]
     },
     "action_name": "MyIngress.set_nhop",
     "action_params": {
       "port": 2,
       "remove_tags": 0
     }
   },
   {
     "table": "MyIngress.switchp_nhop",
     "match": {
       "hdr.ipv4.dstAddr": ["10.1.1.1", 32]
     },
     "action_name": "MyIngress.set_nhop",
     "action_params": {
       "port": 1,
       "remove_tags": 1
     }
   }
```

Source: https://github.com/PathDump/SwitchPointer/blob/master/implementation/p4/apps/ping/s2-commands.txt

P4Runtime: configuring tables (2)

Implementation of load balancing to random host, based on a simple version of Equal-Cost Multipath Forwarding:

- "**ecmp_group**" uses a hash function (applied to a 5-tuple) to select one of two hosts
- "**ecmp_nhop**" defines (based on the hash) to which host the packet will be forwarded
  - ecmp_select == 0 → packet to h2 *(port==2)*; ecmp_select == 1 → packet to h3 *(port == 3)*
- "**send_frame**" forwards the packet and rewrites the MAC address

table: ecmp_group (s1)		
Match fields	**Action**	**Action data**
hdr.ipv4.dstAddr	{drop, set_ecmp_select}	bit<16> ecmp_base, bit<32> ecmp_count
10.0.0.1/32	set_ecmp_select	ecmp_base=0, ecmp_count=2

table: ecmp_nhop (s1)		
Match fields	**Action**	**Action data**
meta.ecmp_select	{drop, set_nhopt}	bit<48> nhop_dmac, bit<32> nhop_ipv4, bit<9> port
0	set_nhop	ndop_dmac=00:00:00:00:00:01:02, nhop_ipv4=10.0.2.2, port=2
1	set_nhp	ndop_dmac=00:00:00:00:00:01:03, nhop_ipv4=10.0.3.3, port=3

Tables filled via P4Runtime ("PI"), BFRuntime, etc

Note: 5-tuple: (Source IP, Destination IP, Protocol, L4 Source Port, L4 Destination Port)

P4Runtime: configuring tables (3)

table: send_frame (s1)		
Match fields	**Action**	**Action data**
egress_port	{drop, rewrite_mac}	bit<48> smac
2	rewrite_mac	smac=00:00:00:01:02:00
3	rewrite_mac	smac=00:00:00:01:03:00

Ingress pipeline

- Generate hash for packet (based on 5-tuple)
- Table that matches on hash and forwards the packet (changes ethernet.dstAddr, ipv4.dstAddr, egress_port)

Egress pipeline

- Define table that matches on egress_port and rewrites ethernet.srcAddr to that of the nearby switch

Materials

Materials (1): docs, sources and projects

Documentation

- P4 guide: https://github.com/jafingerhut/p4-guide/tree/master/docs
- P4 official tutorials: https://github.com/p4lang/tutorials
- P4 tutorial (2018): https://bit.ly/p4d2-2018-spring
- P4_16 v1.2.0 spec: https://p4.org/p4-spec/docs/P4-16-v1.2.0.pdf
- P4 cheat sheet: https://github.com/p4lang/tutorials/blob/master/p4-cheat-sheet.pdf

Implementation sources

- P4 compiler: https://github.com/p4lang/p4c
- P4_16 commented application

Projects

- STRATUM project (switch OS for SDN): https://stratumproject.org
- GÉANT: R&E NOS; DDoS detection, FPGA compiling, etc: https://github.com/frederic-loui/RARE ; https://wiki.geant.org/display/SIGNGN/2nd+SIG-NGN+Meeting
- ONOS controller with P4 support: https://wiki.onosproject.org/display/ONOS/P4+brigade

Materials (2): open-source tools

- **p4c-bm2-ss**: compiles a P4 program (*must be used with other steps to load the output in the switch/model*)
    - Can compile on P4_14 and P4_16, based on target device, architecture, ...
    - --p4-runtime allows writing the control plane API description (i.e., rules to be installed on the devices)

```
p4c-bm2-ss --p4v 16 --p4runtime-files basic_tunnel.p4.p4info.txt basic_tunnel.p4

p4c-bm2-ss --arch v1model -o p4src/build/bmv2.json --Wdisable=unsupported \
--p4runtime-files p4src/build/p4info.txt p4src/some_proto.p4
```

- **simple_switch_grpc:** P4 software switch (codenamed “behavioural model v2 / bmv2”)
- **PI**: P4 Runtime -- API run-time update (w/o restarting control plane), extending schema to describe new features
- **ptf**: Packet Test Framework. Define Python unit tests to verify the behaviour of the dataplane
- **scapy**: generate packets for testing

```
from scapy.all import sendp, get_if_hwaddr, send, Ether, IP, TCP
import random
pkt =  Ether(src=get_if_hwaddr("ens3"), dst="ff:ff:ff:ff:ff:ff")
pkt = pkt / IP(dst="10.102.10.56") / TCP(dport=1234,
sport=random.randint(49152,65535)) / "Payload data"
pkt.show2()
sendp(pkt, iface="ens3", verbose=False)
```

Materials (3): open-source tools

- **Stratum**: OS for SDN-enabled switches. Based on ONLPv2 and supporting Tofino and Broadcom Tomahawk devices. A “**stratum_${target}**” binary (previously compiled per target/device) communicates Stratum with the device.

```
./bazel-bin/stratum/hal/bin/bmv2/stratum_bmv2 \
    --external_stratum_urls=0.0.0.0:28000 \
    --persistent_config_dir=${cfg_path}/stratum_cfg \
    --forwarding_pipeline_configs_file=${cfg_path}/p4_pipeline_config.pb.txt \
    --chassis_config_file=${cfg_path}/chassis_config.proto.txt \
    --bmv2_log_level=debug
```

pkt.emit()