P4 tutorial
– *intermediate*

Carolina Fernández

Bio

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

CarolinaFernandez
carolinafernandez.github.io

cfermart

Agenda

1. General considerations *(3m)*
   - History
   - Approaches & aim
2. Architecture *(5m)*
   - Architecture and portability
3. Language components *(23m)*
   - Program sections *(9')*
   - Tables, actions and primitives *(6')*
   - Stateful objects *(2')*
   - Recursiveness *(5')*
   - Checksum *(1')*
4. Lab session *(50m)*
   - Compiling and running a P4 app (5')
   - Labs
     - Basic forwarding *(15')*
     - Basic (encapsulated) forwarding *(10')*
     - Load balancing *(10')*
     - Cloning *(10')*
5. Materials and references *(4m)*
   - Pointers
   - Tools

General considerations

History

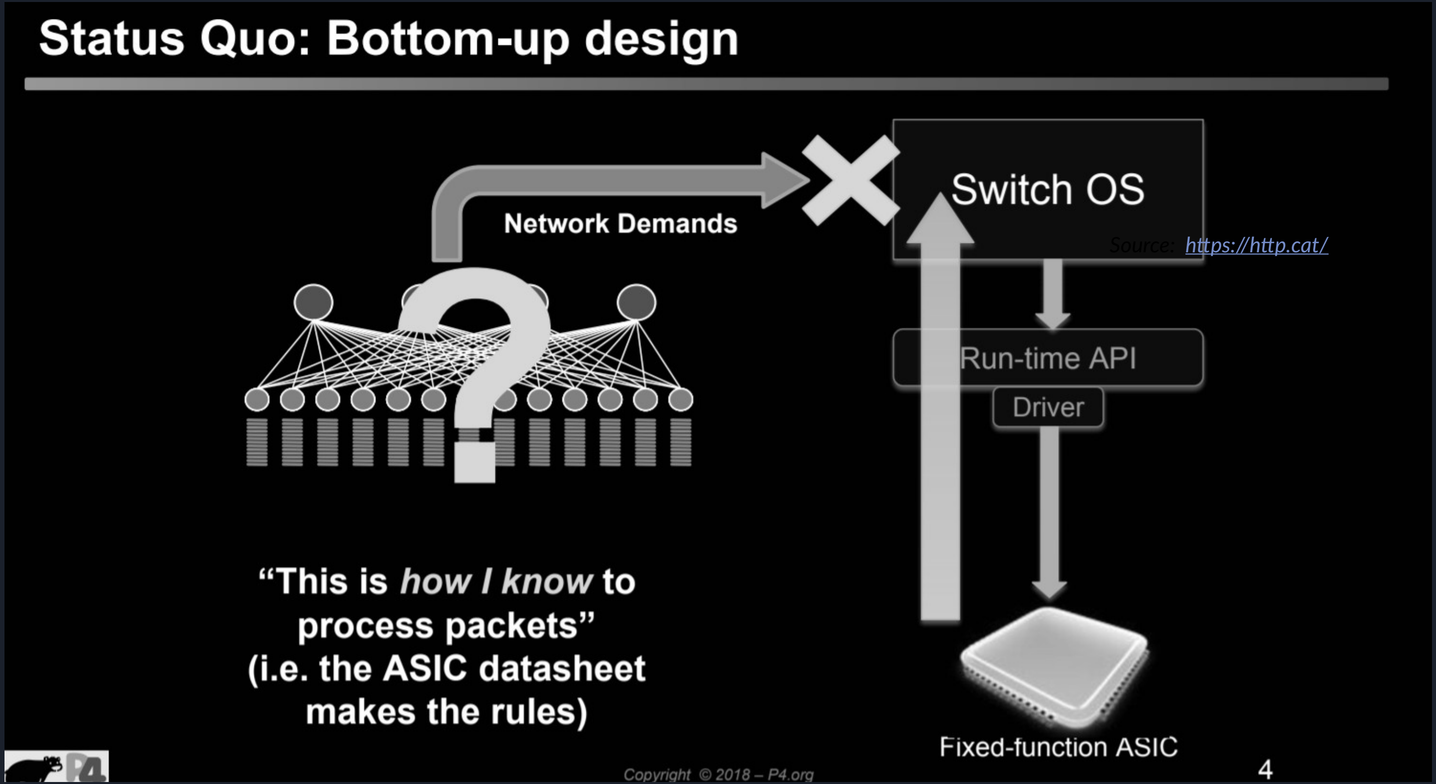
Two specs

- P4_14 / $P4_{14}$
  - Still supported by big vendors, e.g. Barefoot
- P4_16 / $P4_{16}$
  - Mostly used nowadays and supported by open-source compilers

History

- 2013/05: Initial idea and the name “P4”
- 2014/07: First paper (SIGCOMM CCR)
- 2014/08: First P4_14 Draft Specification
- 2014/09: P4_14 Specification released (v1.0.0)
- 2015/01: P4_14 v1.0.1
- …
- 2016/04: P4_16 – first commits
- 2016/12: First P4_16 Draft Specification
- 2017/05: P4_16 Specification released (v1.0.0)
- 2018/11: P4_14 v1.0.5
- 2018/11: P4_16 v1.0.1

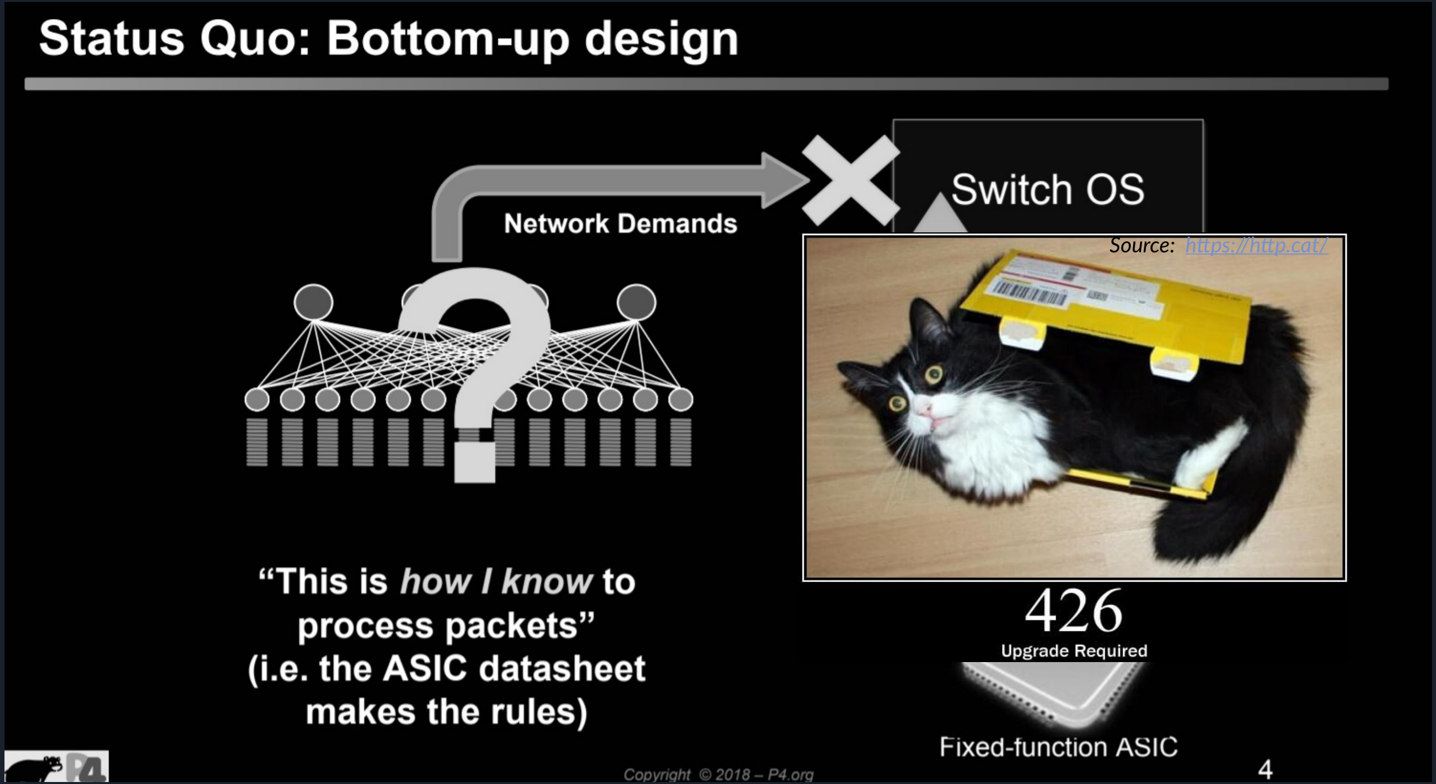
Comparing approaches



Source: https://http.cat/

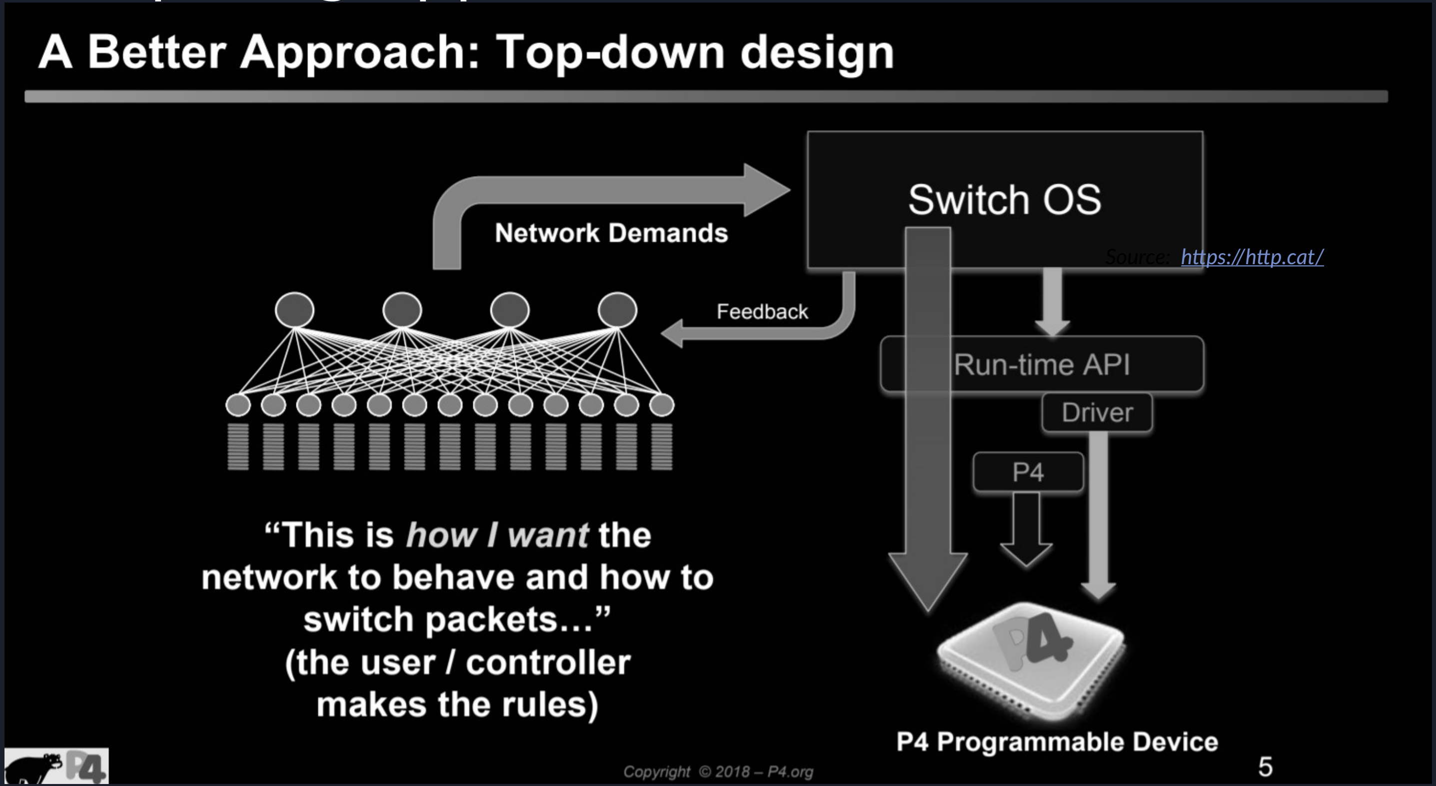
Source: https://bit.ly/p4d2-2018-spring

Comparing approaches



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

Comparing approaches



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

Aim of P4

Used to:

- Define protocols in the data plane
- Use specific, custom packets
- Maximise efficiency for low-level processing
- Benefit from typical operations at the core switches (e.g., mirroring packets)
- Benefit from some typical operations at end nodes (e.g., move packet to CPU)

NOT used to:

- Inserting rules in the forwarding table (programming the control plane)
- Perform some typical operations at end nodes (e.g., traffic generation, packet modification, 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
- Burn-after-read transmissions

Architecture

Architecture (1): definition

What is a P4 Architecture

- Architectures are the *programming model*:
  - The view of the pipeline targeted by the P4 program
  - How the P4 programmer thinks about the underlying platform (data plane)
  - May be different from the hardware target

BAREFOOT NETWORKS

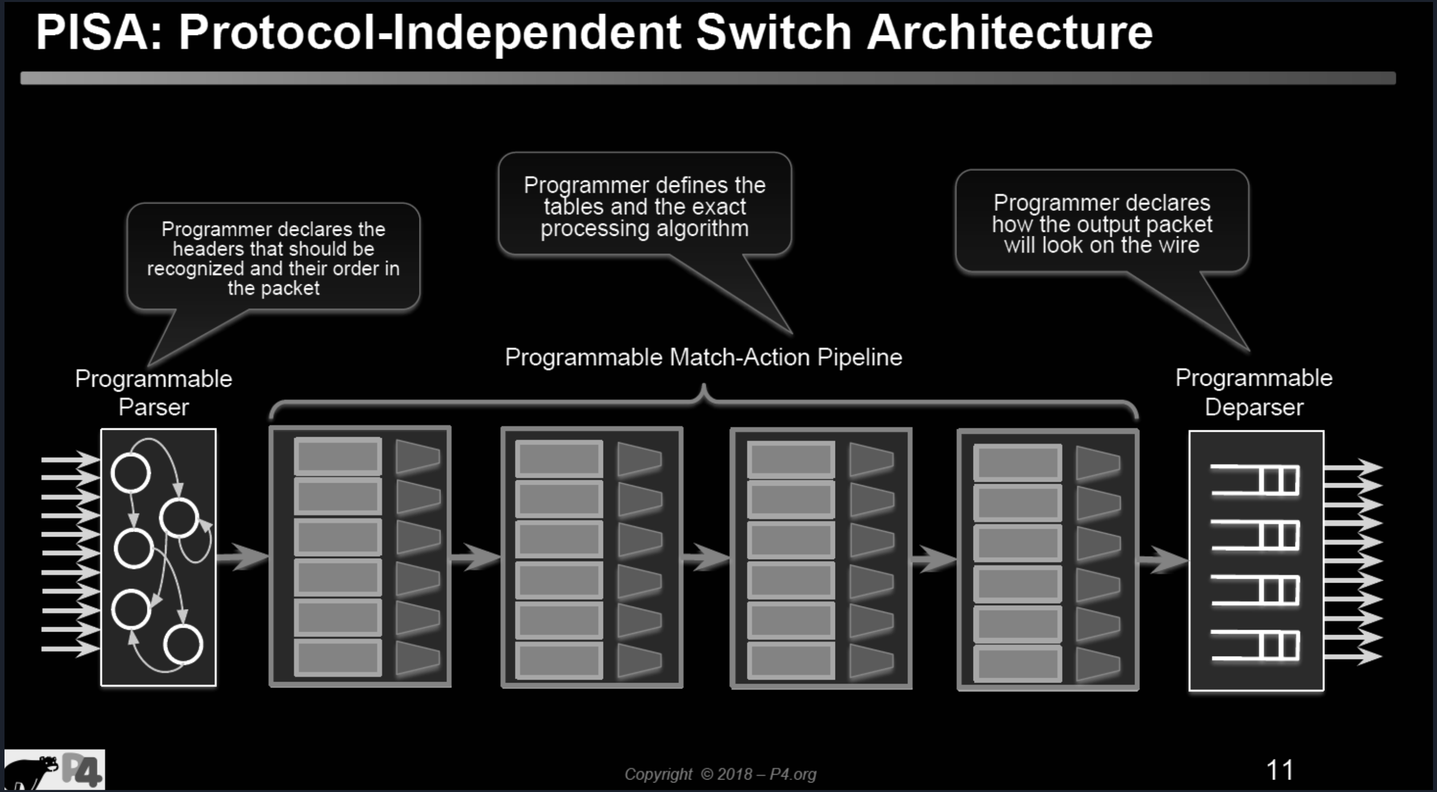
Architectures in $P4_{16}$

- Architectures are a new capability in $P4_{16}$ to enable P4 on a diversity of devices:
  - Hardware: switches, routers, NICs
  - Software: OVS
- In general provide a logical view of the processing
- Architectures insulate programmers from the hardware details
  - Providers define architectures and implement compiler backends to map architectures to targets

BAREFOOT NETWORKS

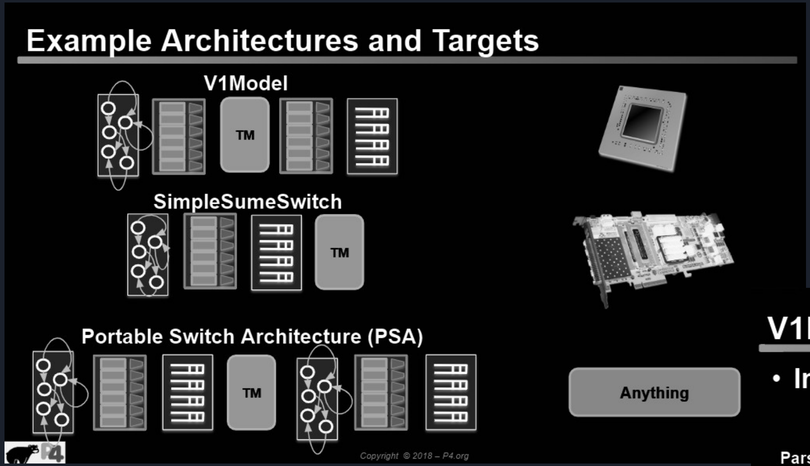
Source: *https://p4.org/assets/p4-ws-2017-p4-architectures.pdf*

Architecture (2): PISA

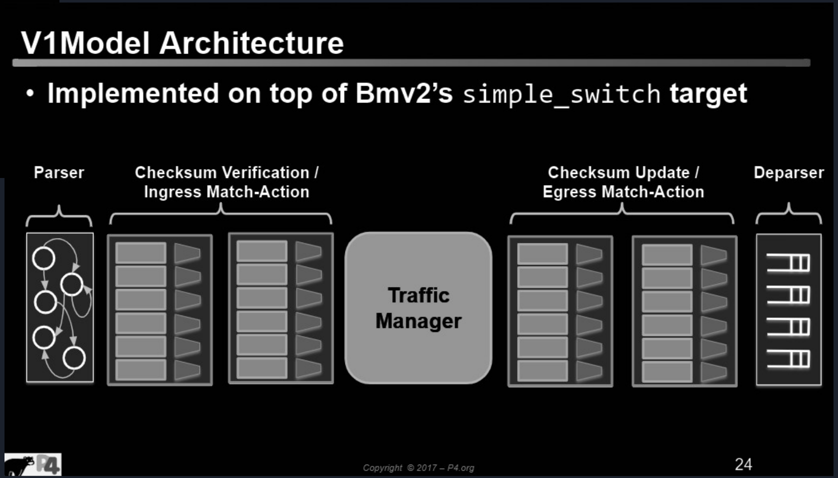


Source: https://bit.ly/p4d2-2018-spring

Architecture (3): portability



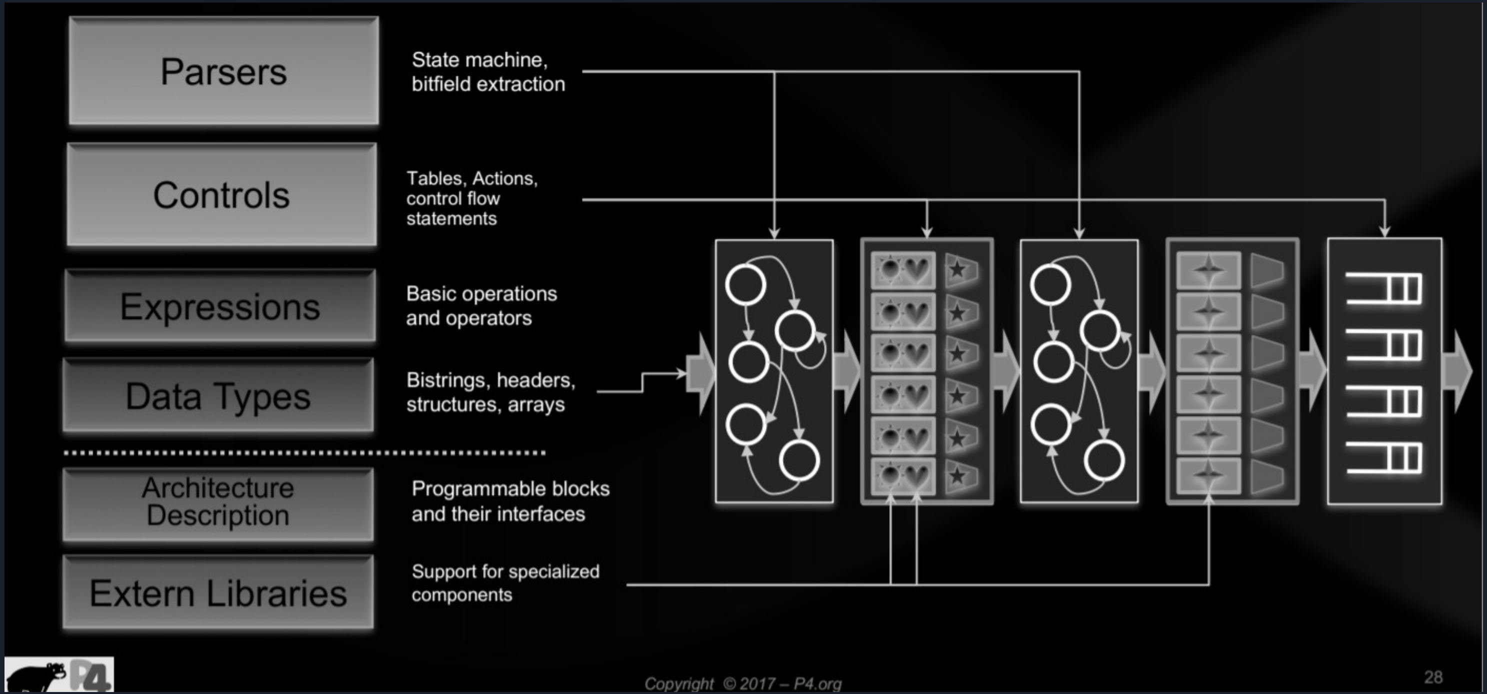
Term	Explanation
Target	Definition of specific HW implementation
Architecture	Set of programmable components, externs, fixed components and their interfaces available
Platform	Architecture implemented on a given Target



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

Language components

$P4_{16}$'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

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

Program sections (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 sections (*see numbers*) to be interpreted

Program sections (2): 1/includes

- System P4 files or your own P4 programs can be imported
- 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

```
// 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 sections (3): 1/metadata

Metadata is used to persist intermediate results associated to packets or structures during their lifetime

- Types: standard (intrinsic) ; user-defined

Standard (intrinsic)

Data associated to each packet. Incorporated in P4's libraries

This data is always valid. It defaults to "0"

Can be related to processing during ingress or egress pipelines

User-defined

Metadata associated to types/structs.

Defined by user, can follow any format

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

Program sections (4): 1/metadata

Struct **standard_metadata_t** contains the following fields that can be used to store intermediate data:

Recursive processing:
- bit<32> instance_type
- bit<32> clone_spec
- bit<16> recirculate_port
- bit<1> resubmit_flag

Queue management:
- bit<32> enq_timestamp
- bit<19> enq_qdepth
- bit<32> deq_timedelta
- bit<19> deq_qdepth

Ingress/egress movement:
- bit<9> ingress_port
- bit<9> egress_spec
- bit<9> egress_port
- bit<16> egress_rid
- bit<16> mcast_grp

Checksum:
- bit<1> checksum_error

Others:
- bit<48> ingress_global_timestamp
- bit<32> lf_field_list
- bit<32> packet_length
- bit<1> drop

Program sections (4): 1/metadata

V1Model Standard Metadata

```
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<9>  egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1>  drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    bit<19> deq_qdepth;
    bit<48> ingress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<1>  resubmit_flag;
    bit<16> egress_rid;
    bit<1>  checksum_error;
}
```

- **ingress_port** - the port on which the packet arrived
- **egress_spec** - the port to which the packet should be sent to
- **egress_port** - the port that the packet will be sent out of (read only in egress pipeline)

25

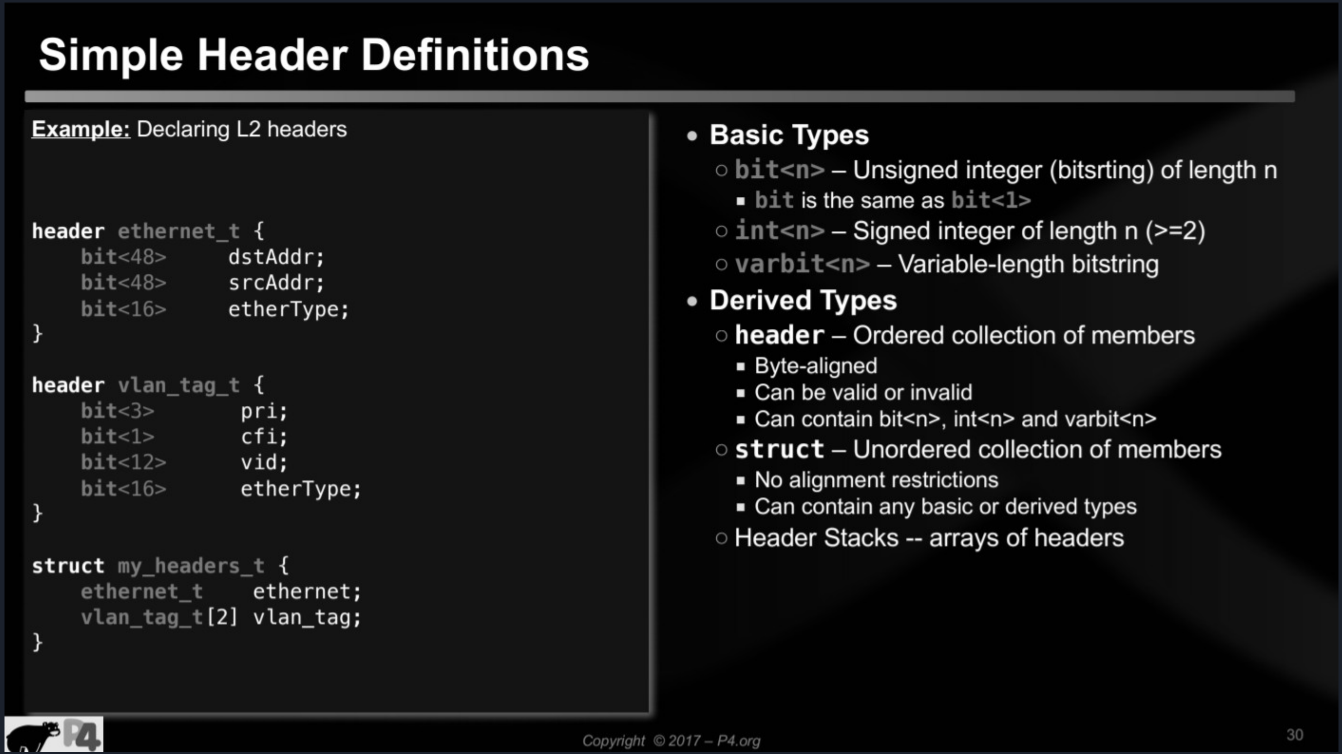
Source: https://github.com/p4lang/behavioral-model/blob/master/docs/simple_switch.md, https://bit.ly/p4d2-2018-spring

Program sections (5): 1/headers

- Header: struct *(C-like)* + “validity” field *(hidden)*
  - Methods: `isValid(), setValid(), setInvalid()`
  - Note: successful extract() of a header sets its validity bit to “true”
- Network protocol headers to be recognised and processed by the program
- Ordering
  - Order of fields in the declaration ⇔ order of fields in the wire
  - Packet has no gaps between fields
  - Packet header length must be multiple of 8 bytes
- Initially, all headers are invalid
  - *Note: accessing header fields of invalid headers leads to undefined behaviours*

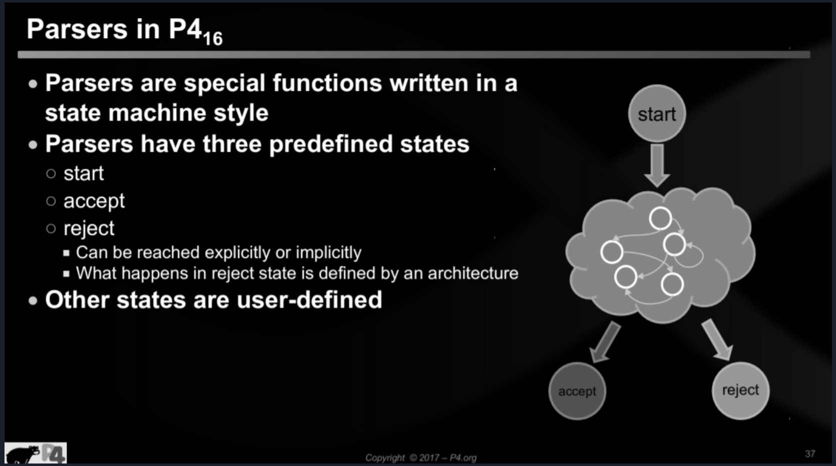
```
header H {
  bit<32> x;
  bit<32> y;
}

Struct InControl {
  PortId input_port;
}
```

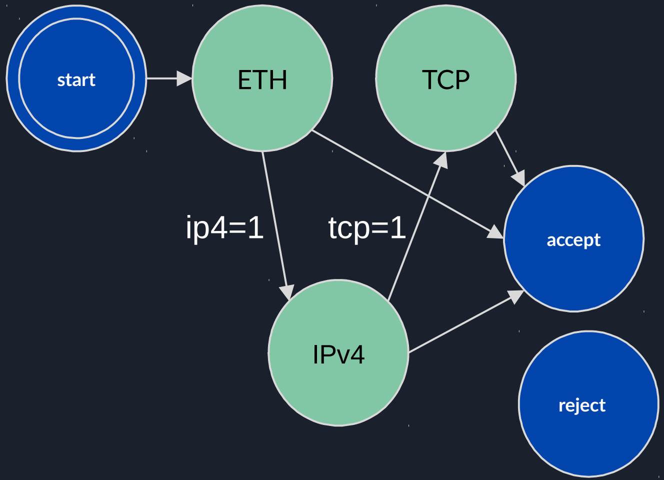
Program sections (5): 1/headers



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

Program sections (6): 2&7/parsers





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*](https://p4.org/assets/p4_d2_2017_p4_16_tutorial.pdf)

Program sections (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 sections (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();
      }
  }
```

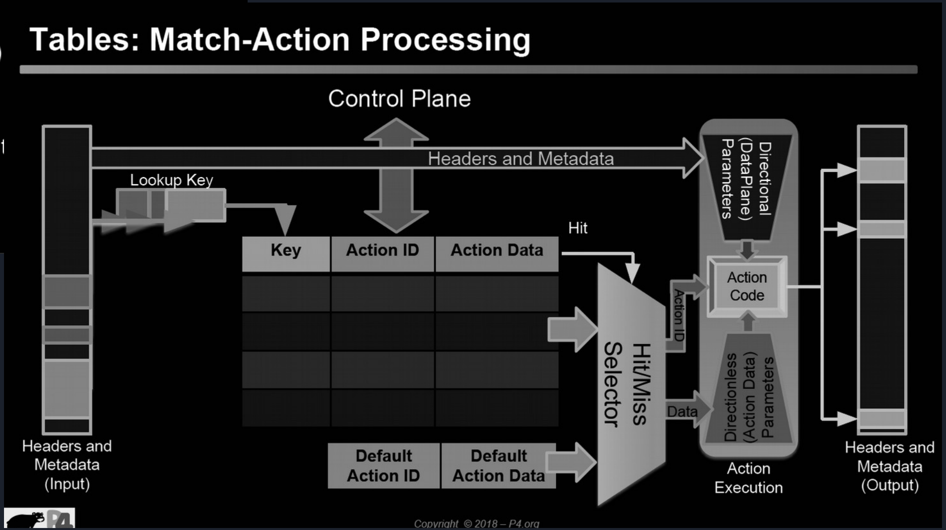
```
# Internal evaluation
  if (ipv4_match.apply().hit) {
      // There was a hit
  } else {
      // There was a miss
  }
```

Program sections (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



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

Program sections (9): 4&5/tables

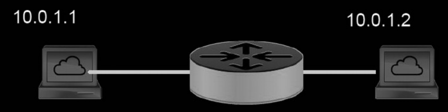
$P4_{16}$ Tables

- **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

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

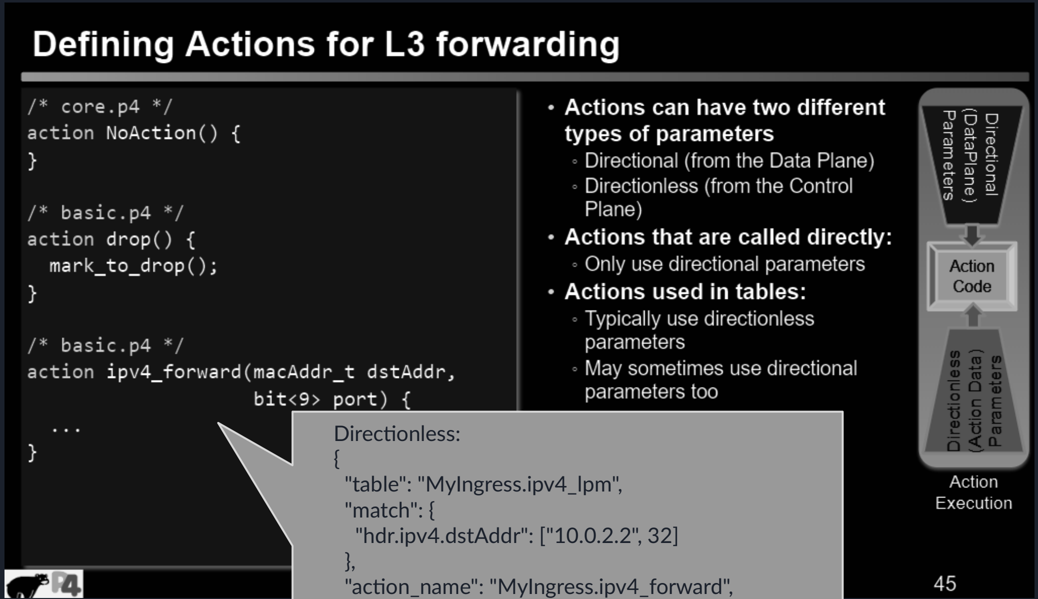
37

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

Program sections (9): 4&5/actions

Action:

- May contain data values (written by control plane, read by data plane) -- *the control-plane can influence dynamically the behavior 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



Directionless:


```
{
  "table": "MyIngress.ipv4_lpm",
  "match": {
    "hdr.ipv4.dstAddr": ["10.0.2.2", 32]
  },
  "action_name": "MyIngress.ipv4_forward",
  "action_params": {
    "dstAddr": "00:00:00:02:02:00",
    "port": 2
  }
},
```


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

Program sections (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, ...
- ...

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

- P4 objects can be classified by their lifespan
  - Stateless (transient): state is not preserved upon processing (lifespan ≤1 packet)
    - Metadata
    - Packet headers
  - Stateful (persistent): state is preserved upon processing (lifespan ≥ 1 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 sections (12): 4&5/recursiveness

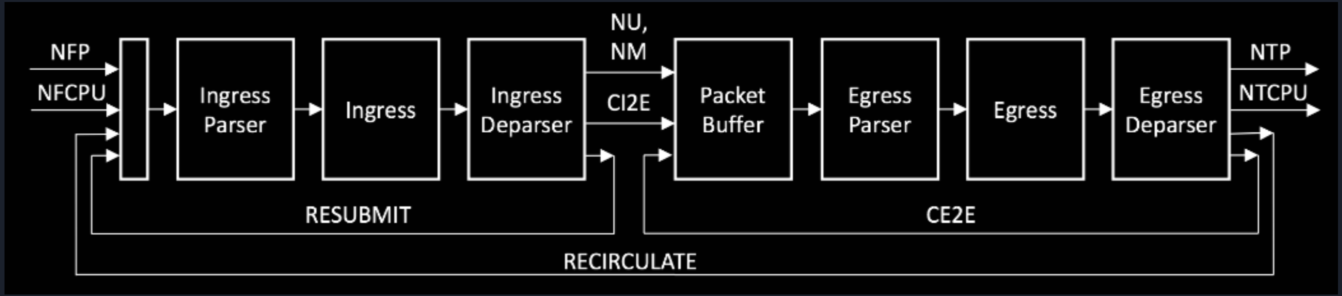
Complex parsing may require a packet to be processed recursively by being:

- duplicated (**clone**d) – e.g., to monitor how the packet looks like in the wire;
- sent again to pipelines (**recirculate**d) – e.g., for reusing the original packet upon modifications in the egress pipeline;
- sent again to pipelines (**resubmit**ted) – e.g., for further processing in the ingress pipeline (for instance, to apply a table multiple times)

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*

Program sections (12): 4&5/recursiveness

```
#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*](https://p4.org/p4-spec/docs/PSA-v1.1.0.html)

Program sections (13): 4&5/recursiveness

- **Cloning**: copy a packet. The cloned packet appears at the egress pipeline in both cases. Types:
    - Packet cloned in the ingress pipeline – *Ingress to egress*: **CloneType.I2E**
    - Packet cloned in the egress pipeline – *Egress to egress*: **CloneType.E2E**
  - *Use case*: monitor how the packet looks in the wire
  - *Note*: mirror session_id used to tag and to identify the cloned packets

- **Resubmit**: send the packet to the pipelines after crossing the ingress pipeline
  - *Use case*: perform packet processing that cannot be completed in a single pass

- **Recirculate**: send the packet to the pipelines after crossing the ingress & egress pipelines
  - *Use case*: for reusing the original packet upon modifications in the egress pipeline

Program sections (14): 3&6/checksum

- Checksum can be **verified** and **computed**
  - Depends on switch architecture (some may be missing)
  - Verified (for error correction):
    - If checksum does not match, pkt is discarded
    - If checksum matches, removed from pkt payload
- No built-in constructs in P4_16 — expressed as externs (provided by specific libraries)
  - E.g., the “Checksum16” extern, available from the VSS architecture
- “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);
```

Lab session

Compiling and running a P4 app (1)

```
P4C_ARGS = --p4runtime-file $(basename $@).p4info
           --p4runtime-format text
RUN_SCRIPT = ../../utils/run_exercise.py
TOPO = topology.json
```

dirs:

```
    mkdir -p build pcaps logs
```

build: for each P4 program, generate BMv2 json file

```
    p4c-bm2-ss --p4v 16 $(P4C_ARGS) -o $@ $<
```

run: build, then ***[default target]***

```
    sudo python $(RUN_SCRIPT) -t $(TOPO)
```

stop: `sudo mn -c`

clean: stop, then

```
    rm -f *.pcap
    rm -rf build pcaps logs
```

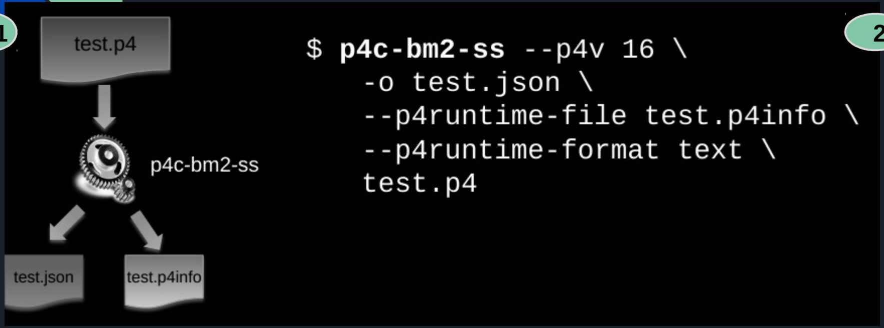
```
make run
make stop; make clean
```


61

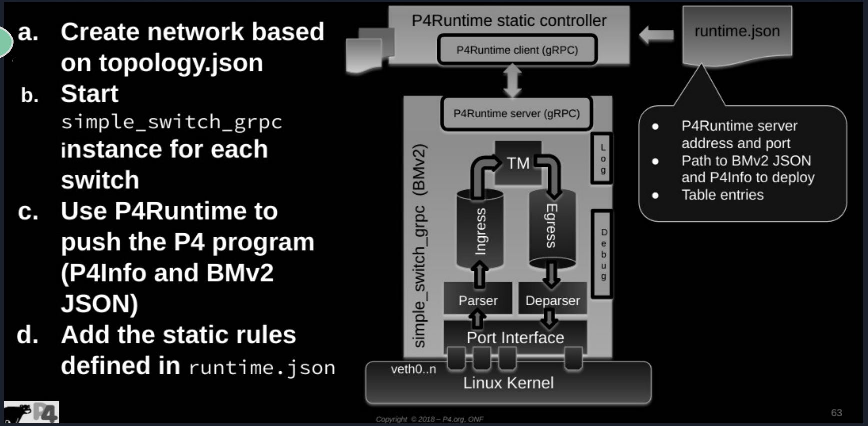
Source: *https://bit.ly/p4d2-2018-spring*

Compiling and running a P4 app (2)

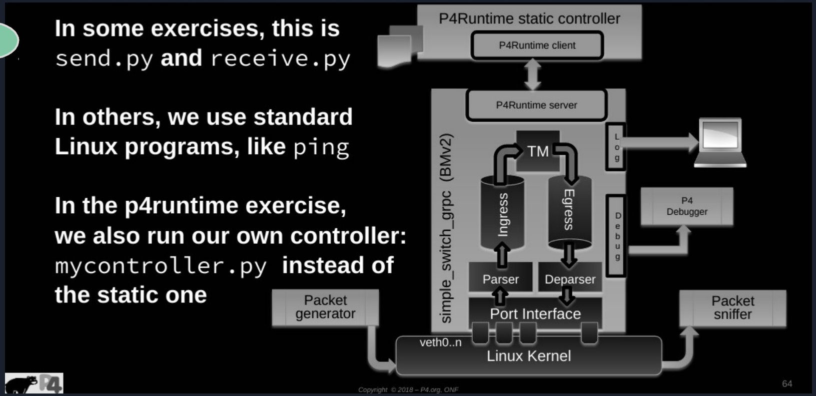
1



2

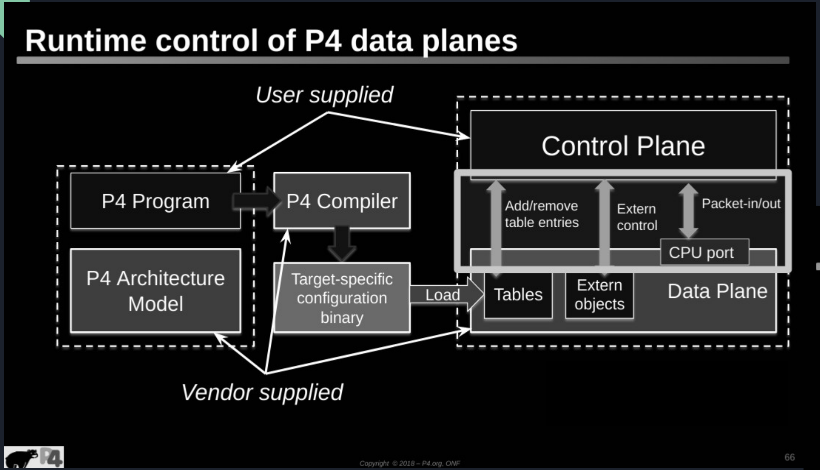


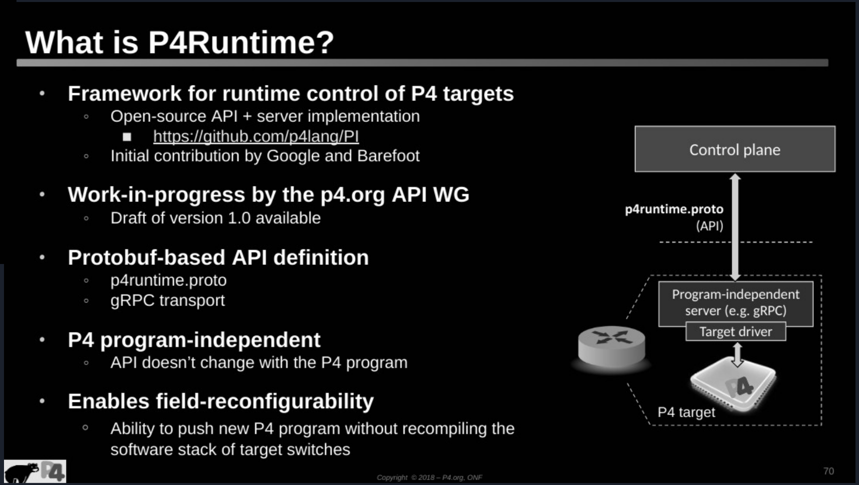
3



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

Compiling and running a P4 app (3)





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

Compiling and running a P4 app (4)

P4Runtime provides Target & Protocol independent API to control the dataplane (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

Frequent questions

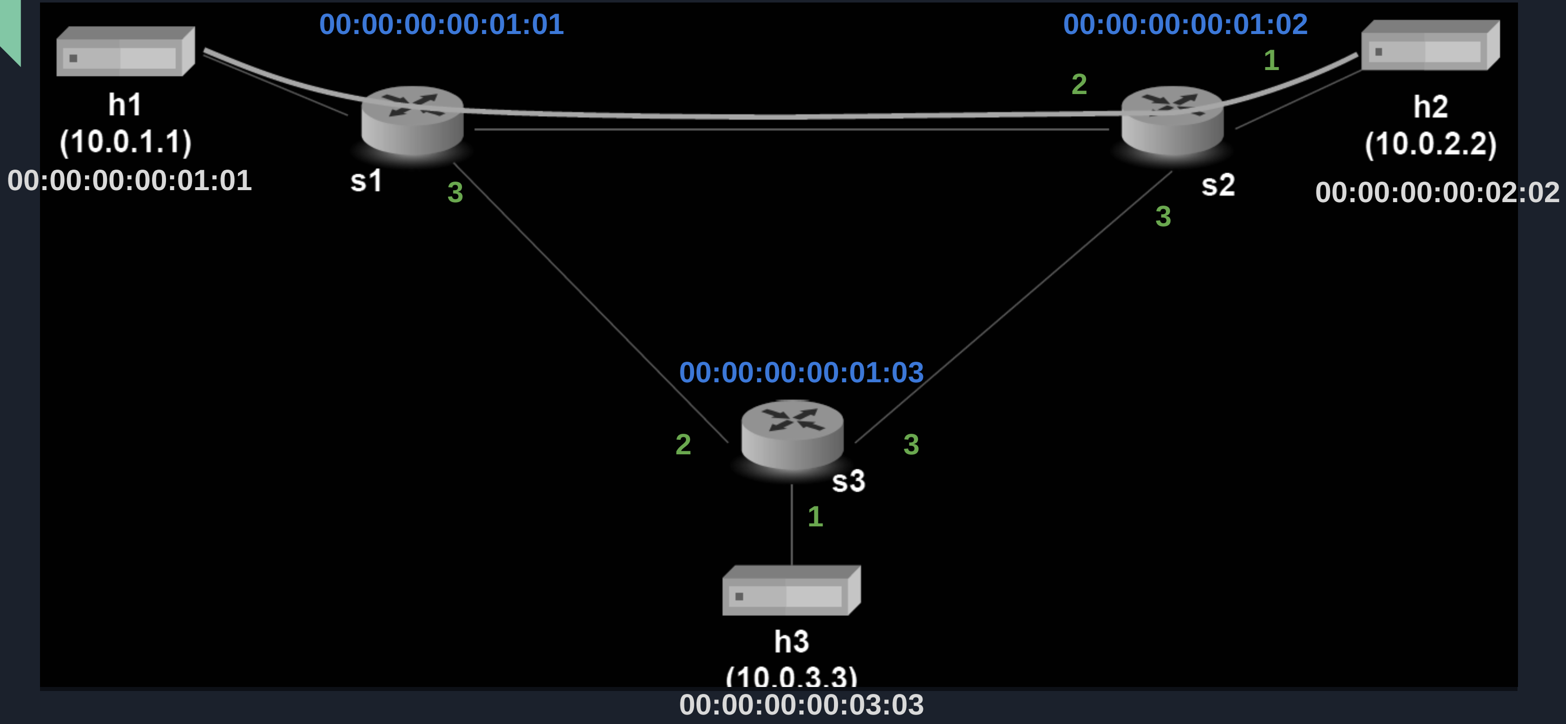
FAQs

- **Can I apply a table multiple times in my P4 Program?**
  - No (except via resubmit / recirculate)
- **Can I modify table entries from my P4 Program?**
  - No (except for direct counters)
- **What happens upon reaching the `reject` state of the parser?**
  - Architecture dependent
- **How much of the packet can I parse?**
  - Architecture dependent

53

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

Labs: common topology



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

Lab1: basic forwarding (1)

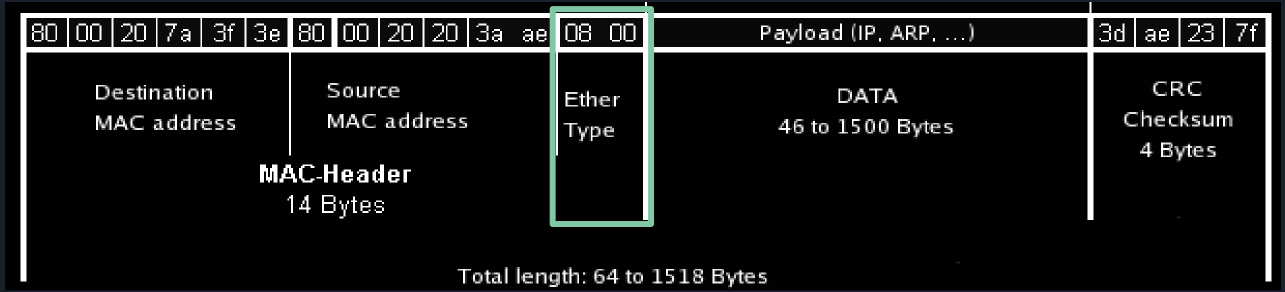
Running Example: Basic Forwarding

- We'll use a simple application as a running example—a basic router—to illustrate the main features of $P4_{16}$
- Basic router functionality:
  - Parse Ethernet and IPv4 headers from packet
  - Find destination in IPv4 routing table
  - Update source / destination MAC addresses
  - Decrement time-to-live (TTL) field
  - Set the egress port
  - Deparse headers back into a packet
- We've written some starter code for you (`basic.p4`) and implemented a static control plane

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

Lab1: basic forwarding (2)

1. Access the example in your VM:
   - cd p4-tutorials/exercises/basic
2. Define how packets are parsed
   - Ethernet frame arrives



   - Packet is parsed (from outer to inner headers / left to right order)
   - etherType field is matched (possible values: 0x0800 for IPv4, 0x8847 for MPLS unicast, ...)
   - Based on the result above:
     - If IPv4, parse it as an IPv4 datagram
     - Otherwise, continue to the “accept” transition
3. Define control sequences (N/A for this example)

Source: http://www.helldragon.eu/marcello/galli_lezioni/D_internet/tcpip.html

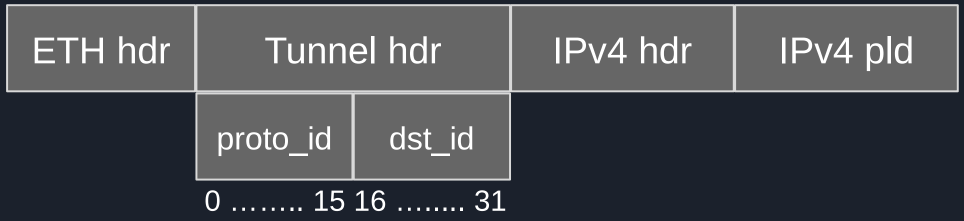
Lab1: basic forwarding (3)

4. Define checksum verification process
    - Not used
5. Define I/O sequences
    - Ingress:
        - Define tables: which information from the packet should be matched
            - LPM match on the “dstAddr” field, define forwarding action
        - Define actions: what to do based on specific data
            - Output to port; update src, dst fields; decrement TTL
        - Apply tables based on conditions (e.g., validity of header whose fields are matches on the table)
    - Egress: not used
6. Define checksum computation
    - Not used
7. Define how packets are deparsed
    - Reconstruct packet from headers: from outer to inner headers / left to right order (Ethernet; IPv4)

Lab2: basic forwarding & encapsulation (1)

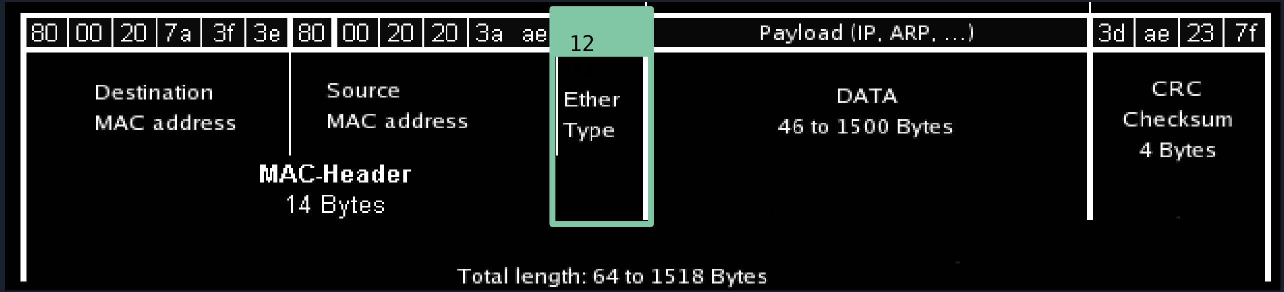
Based on the previous basic forwarding (⇒ keep support for IPv4 routing):

- Add support for a basic tunneling protocol
- Such tunneling protocol will forward to the destination port based on the new tunnel header
- The new header type will contain a protocol ID (type of packet) and the destination ID in use for routing



Lab2: basic forwarding & encapsulation (2)

1. Access the example in your VM:
    - cd p4-tutorials/exercises/basic_tunnel
2. Define how packets are parsed
    - Ethernet frame arrives



    - Packet is parsed (from outer to inner headers / left to right order)
    - etherType field is matched (possible values: 0x0800 for IPv4, 0x1212 for myTunnel, ...)
    - Based on the result above:
        - If IPv4, parse it as an IPv4 datagram
        - If myTunnel, parse its headers. Within this transition, parse IPv4 if it is inside
        - Otherwise, continue to the “accept” transition

Source: http://www.helldragon.eu/marcello/galli_lezioni/D_internet/tcpip.html

Lab2: basic forwarding & encapsulation (3)

3. Define control sequences (N/A for this example)
4. Define checksum verification process
    - Not used
5. Define I/O sequences
    - Ingress:
        - Define tables: which information from the packet should be matched
            - Exact match on the “dst_id” field, define forwarding action
        - Define actions: what to do based on specific data
            - Output to port (based on the “dst_id” field)
        - Apply tables based on conditions (e.g., validity of header whose fields are matches on the table)
            - Check first for encapsulating header -- otherwise process inside packet
    - Egress: not used
6. Define checksum computation
    - Not used
7. Define how packets are deparsed
    - Reconstruct packet from headers: from outer to inner headers / left to right order (Ethernet; myTunnel; IPv4)

Lab2: basic forwarding & encapsulation (4)

Considerations:

- The parser must take into account that the “myTunnel” header may not be present
- The EtherType value for the “myTunnel” protocol is “0x1212”
- The parser and deparser blocks process the fields in a left-to-right fashion; as you would depict the packet
- The “myTunnel” forward will simply output the packet on the same port as stated by the node id (check topology)

Test:

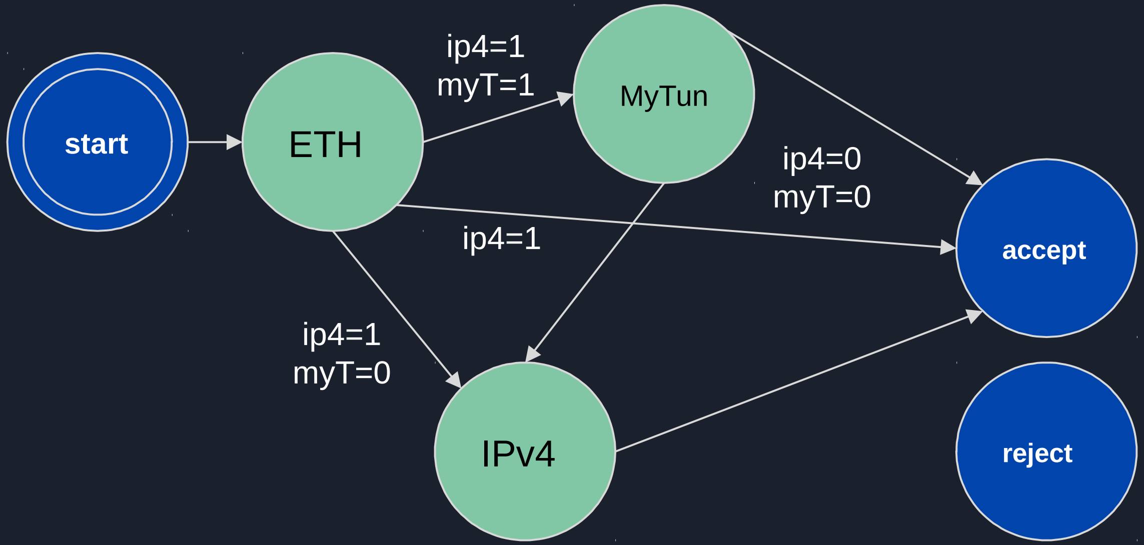
- From h1, run the following and check for output in h2:
    - `./send.py 10.0.2.2 "P4 is cool" --dst_id 2`
    - `./send.py 10.0.3.3 "P4 is cool" --dst_id 2`

Extra:

- Change order (priority!) of the “apply” to different tables (under *MyIngress*) to be 1) ipv4, 2) myTunnel:
    - Then, from h1 run the following and check for output in h2:
        - `./send.py 10.0.2.2 "P4 is cool" --dst_id 2`
        - `./send.py 10.0.2.2 "P4 is cool" --dst_id 3`
- Craft your own Scapy packets (you may check the sample of send.py)

Lab2: basic forwarding & encapsulation (5)

State machine for the parser process:



Lab3: load balancing (1)

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
- “send_frame” forwards the packet and rewrite 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

Tables filled via P4Runtime (“PI”), BFRuntime, etc

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=00:00:00:00:00:00:01:02, nhop_ipv4=10.0.2.2, port=2
1	set_nhp	ndop_dmac=00:00:00:00:00:00:01:03, nhop_ipv4=10.0.3.3, port=3

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

Lab3: load balancing (2)

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

Lab3: load balancing (3)

Considerations:

- The load balancing is performed based on the field "meta.ecmp_select"
  - If ecmp_select == 0 → packet is forwarded to h2
  - If ecmp_select == 1 → packet is forwarded to h3

Test:

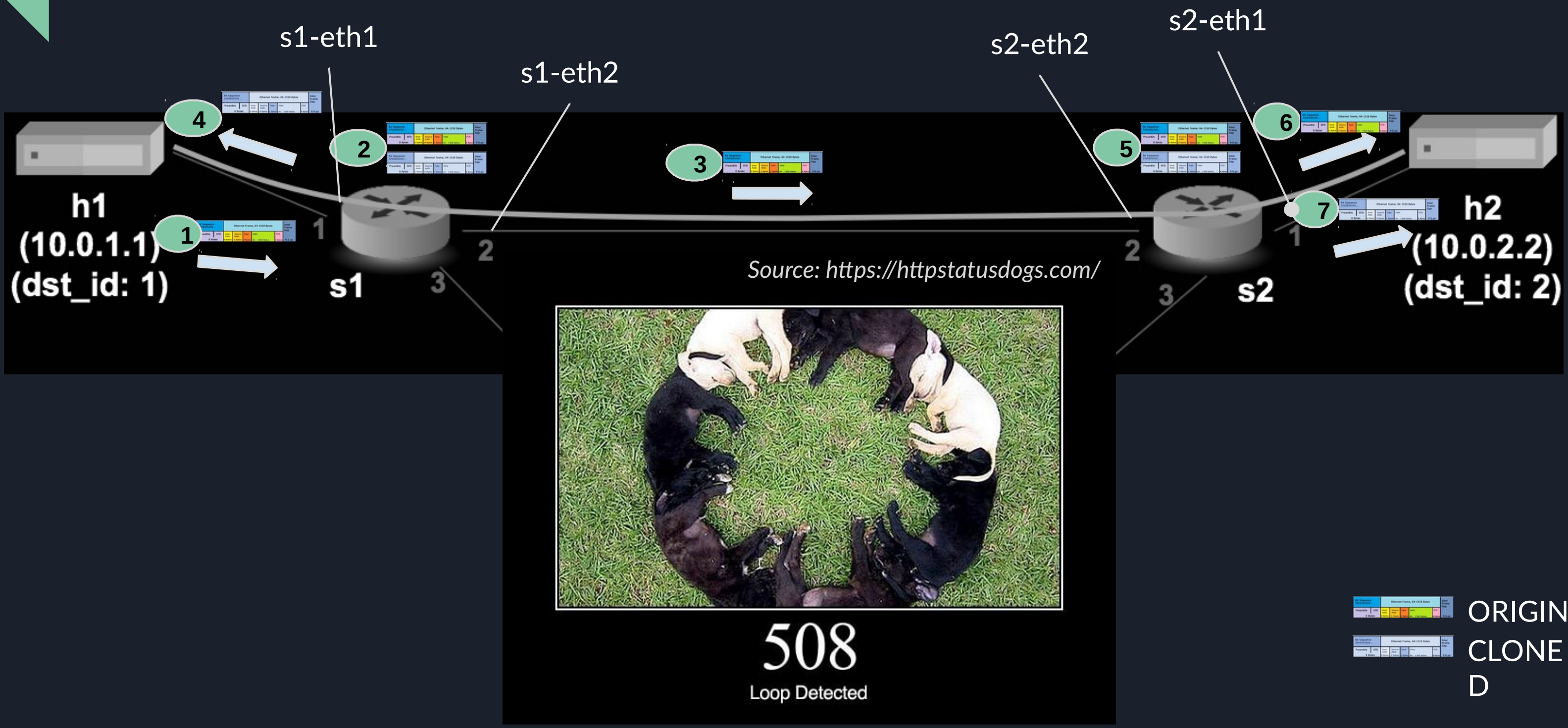
- From h1, run the following and check for output in h3:
  - `./send.py 10.0.0.1 "P4 is cool"`

Extra:

- Change the entry in "ecmp_nhop" in s1-runtime.json where "meta.ecmp_select : 1" to the following. Packet should arrive to h2 instead:

```
{
  "table": "MyIngress.ecmp_nhop",
  "match": { "meta.ecmp_select": 1 },
  "action_name": "MyIngress.set_nhop",
  "action_params": { "nhop_dmac": "00:00:00:00:01:02", "nhop_ipv4": "10.0.2.2",
"port" : 2 }
},
```

Lab4: packet cloning (1)



Source: https://httpstatusdogs.com/

ORIGINAL
CLONED

Lab4: packet cloning (2)

Ignore RESET packets

Traffic outputted by h1 and shared with s1 (switch 1, port 1)

```
p4@p4-vm:~/tutorials/exercises/clone$ clear; sudo tcpdump "tcp[tcpflags] & (tcp-syn) != 0" -i s1-eth1 -vv
01:23:37.805322 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6), length 50)
    10.0.1.1.60784 > 10.0.2.2.1234: Flags [S], cksum 0xcff9 (correct), seq 0:10, win 8192, length 10
01:23:37.807092 IP (tos 0x0, id 1, offset 0, flags [none], proto TCP (6), length 50)
    10.0.3.3.60784 > 10.0.2.2.1234: Flags [S], cksum 0xcff9 (incorrect -> 0xcdf7), seq 0:10, win 8192, length 10
```

Original packet: TTL=64

Cloned packet: TTL=0, IPv4src=fake

Traffic outputted by s1 and shared with s2 (switch 1, port 2)

```
p4@p4-vm:~/tutorials/exercises/clone$ clear; sudo tcpdump "tcp[tcpflags] & (tcp-syn) != 0" -i s1-eth2 -vv
01:23:37.807138 IP (tos 0x0, ttl 63, id 1, offset 0, flags [none], proto TCP (6), length 50)
    10.0.1.1.60784 > 10.0.2.2.1234: Flags [S], cksum 0xcff9 (correct), seq 0:10, win 8192, length 10
```

Original packet: TTL=63

Cloned packet: TTL=0, IPv4src=fake

Traffic outputted by s2 and shared with h2 (switch 2, port 1)

```
p4@p4-vm:~/tutorials/exercises/clone$ clear; sudo tcpdump "tcp[tcpflags] & (tcp-syn) != 0" -i s2-eth1 -vvv
01:23:37.808094 IP (tos 0x0, id 1, offset 0, flags [none], proto TCP (6), length 50)
    10.0.3.3.60784 > 10.0.2.2.1234: Flags [S], cksum 0xcff9 (incorrect -> 0xcdf7), seq 0:10, win 8192, length 10
01:23:37.808141 IP (tos 0x0, ttl 62, id 1, offset 0, flags [none], proto TCP (6), length 50)
    10.0.1.1.60784 > 10.0.2.2.1234: Flags [S], cksum 0xcff9 (correct), seq 0:10, win 8192, length 10
```

Original packet: TTL=62

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)
  - Sample:

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

```
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)
```