P4 tutorial
– *introductory*

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 *(3m)*
   - Architecture definition
3. Language components *(19m)*
   - Program sections *(9')*
   - Tables and actions *(4')*
   - Stateful objects *(2')*
   - Recursiveness *(3')*
   - Checksum *(1')*
4. Materials and references *(2m)*
   - Pointers

General considerations

History

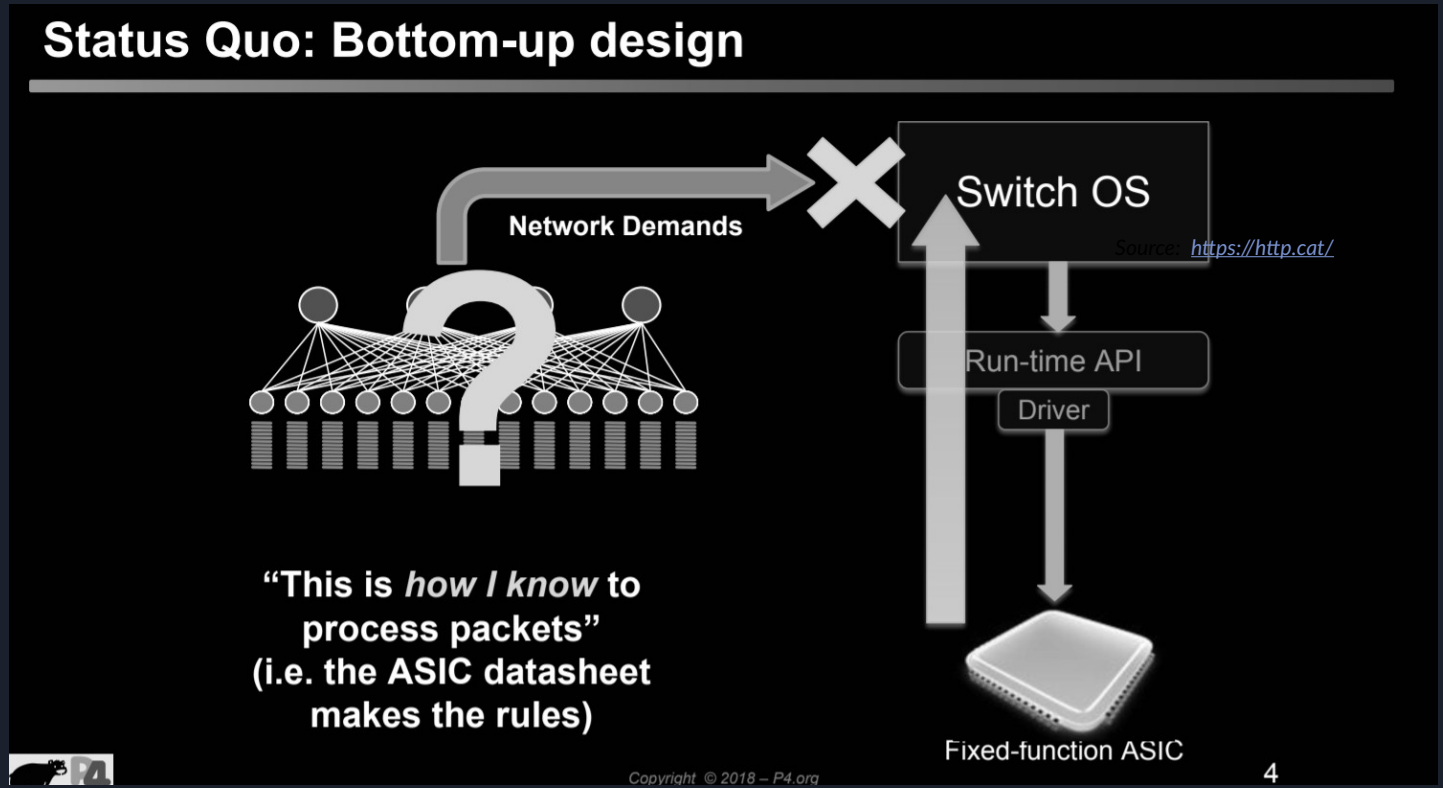
Two specs

- 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

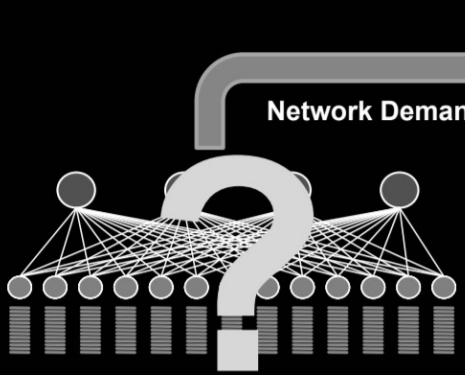
Comparing approaches



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

Comparing approaches

Status Quo: Bottom-up design



Switch OS

Source: https://http.cat/

426
Upgrade Required

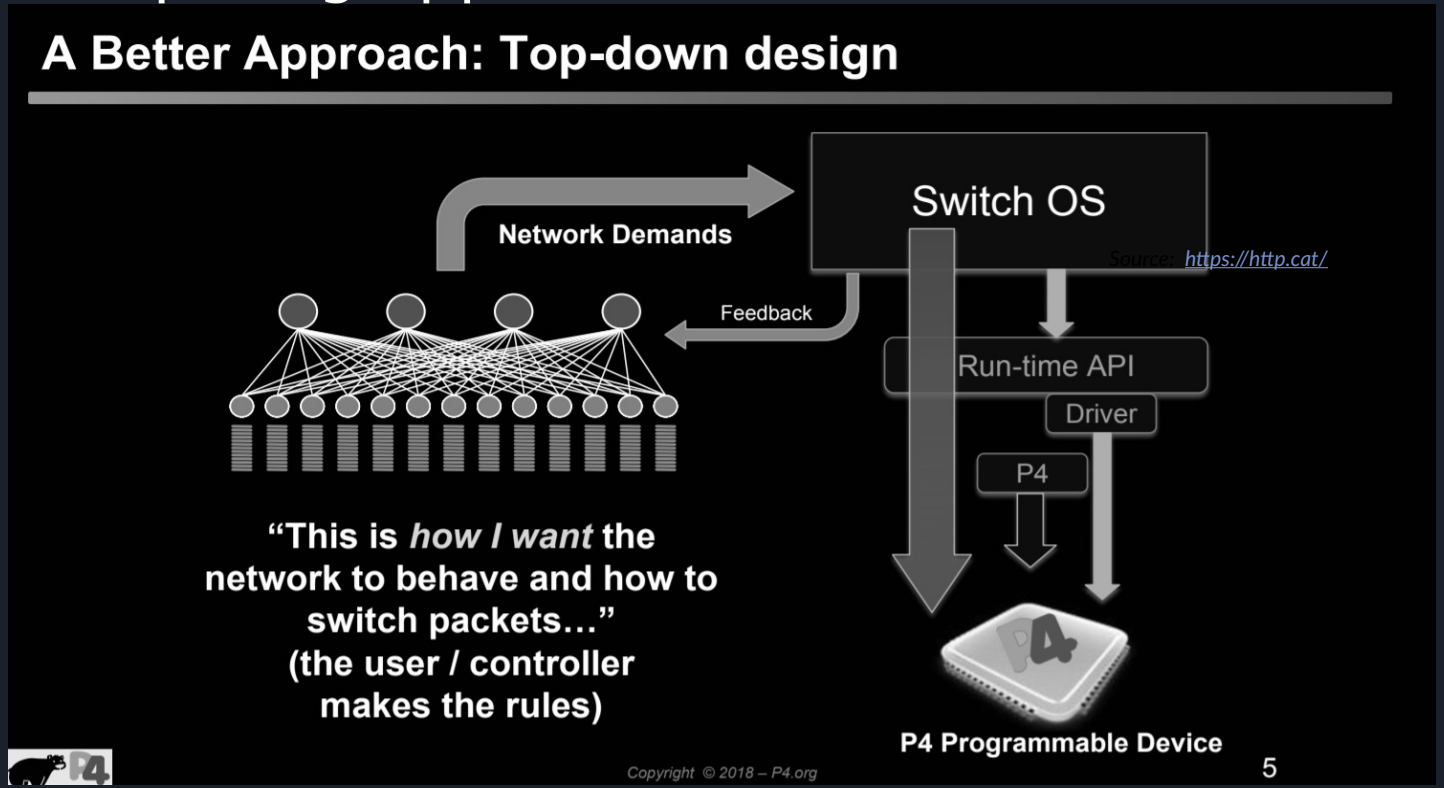
Fixed-function ASIC

"This is *how I know* to process packets" (i.e. the ASIC datasheet makes the rules)

4

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

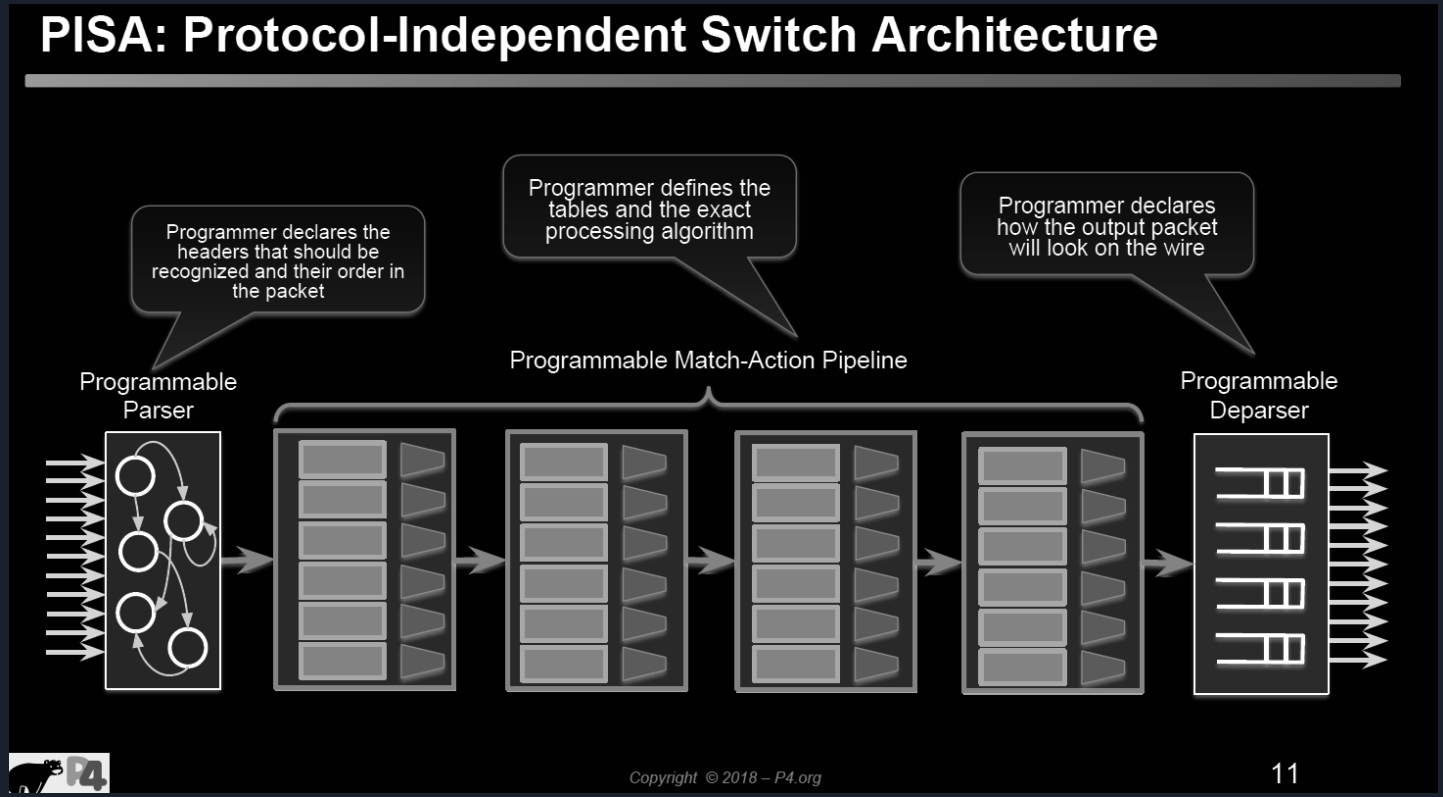
Architectures in $P4_{16}$

- 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

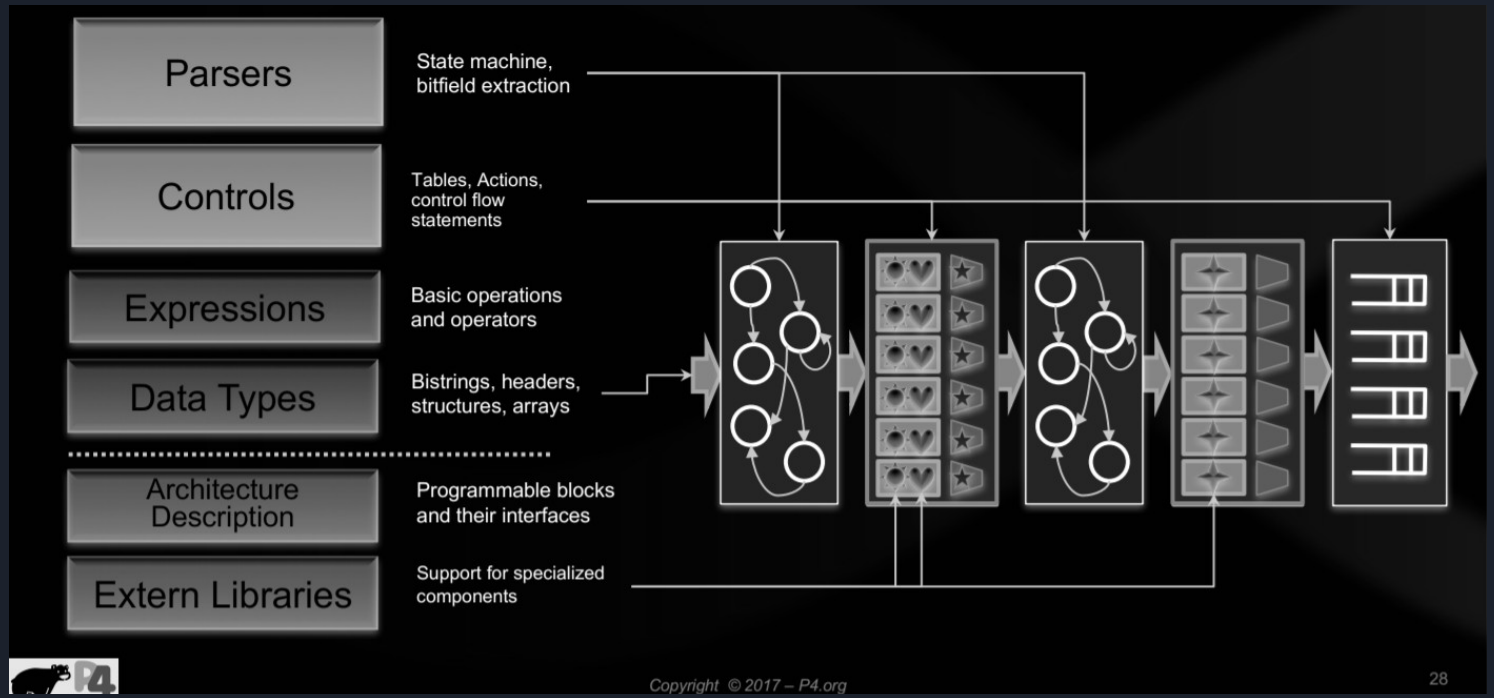
Architecture (2): PISA



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

Language components

$P4_{16}$'s language elements



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

$P4_{16}$'s program

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

3 6 **Control: Checksum**
- Verify checksum
- 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, headers, etc

Includes

- System/your own P4 files can be imported
- Import typically done, yet not restricted to, at the beginning of the file

Headers

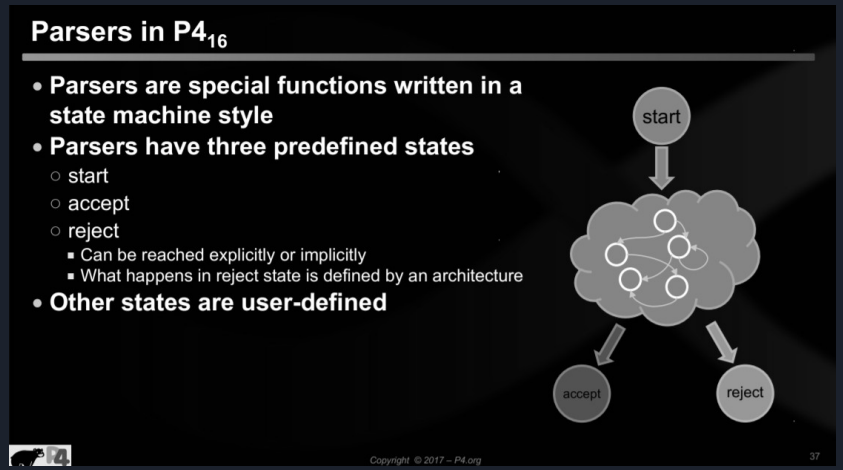
- Struct *(C-like)* + “validity” field *(hidden)*
  - Methods to check/set validity
  - *Note: Initially, headers are invalid. Successful extract() of a header sets its validity bit to “true”. Must not access fields of invalid headers*
- Headers recognised and processed by program
- Order of fields in declaration ⇔ order of fields in the wire (multiple of 8 bytes)

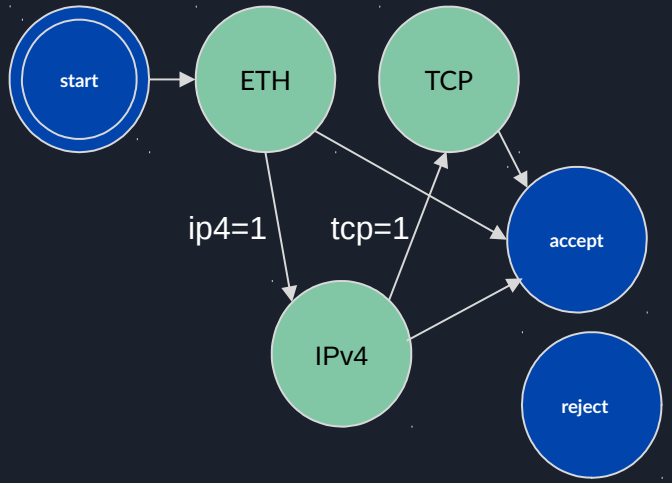
Metadata

Persists intermediate results associated to packets or structures during their lifetime

- Standard (intrinsic)
  - Data associated to each packet. Incorporated in P4’s libraries
  - Always valid. It defaults to “0”
  - Can be related to processing during ingress or egress pipelines
- User-defined
  - Associated to types/structs
  - Defined by user, can follow any format

Program sections (3): 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

Program sections (4): 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 (5): 4&5/tables, actions

P4$_{16}$ Tables

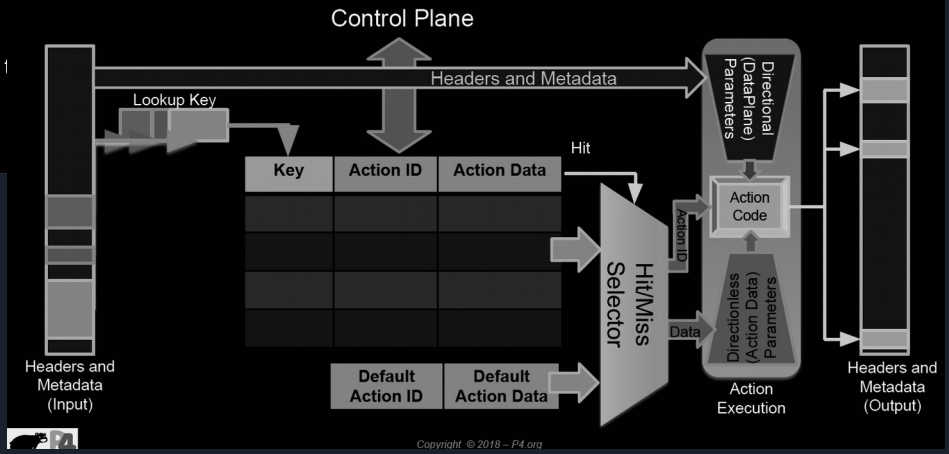
- **The fundamental unit of a Match-Action Pipeline**
  - Specifies what data to match on and match kind
  - Specifies a list of *possible* actions
  - Optionally specifies a number of table **properties**
    - Size
    - Default action
    - Static entries
    - etc.
- **Each table contains one or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

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

Action:

- **Primitives** and other actions called inside (add logic to processing: arithmetic ops, etc)
- Operate on headers, metadata, constants, action data
- Linked to 1..N tables

Tables: Match-Action Processing



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

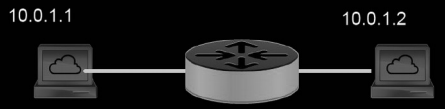
Program sections (5): 4&5/tables, actions

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
  - Action data (possibly empty)

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

Example: IPv4_LPM



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	

Action data to be filled by control plane

- **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

Action:

- **Primitives** and other actions called inside (add logic to processing: arithmetic ops, etc)
- Operate on headers, metadata, constants, action data
- Linked to 1..N tables

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

Program sections (6): 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 *(measure data rate: packets/second, bytes/second)*
    - Registers *(sort of counters that can be operated from actions in a general way)*

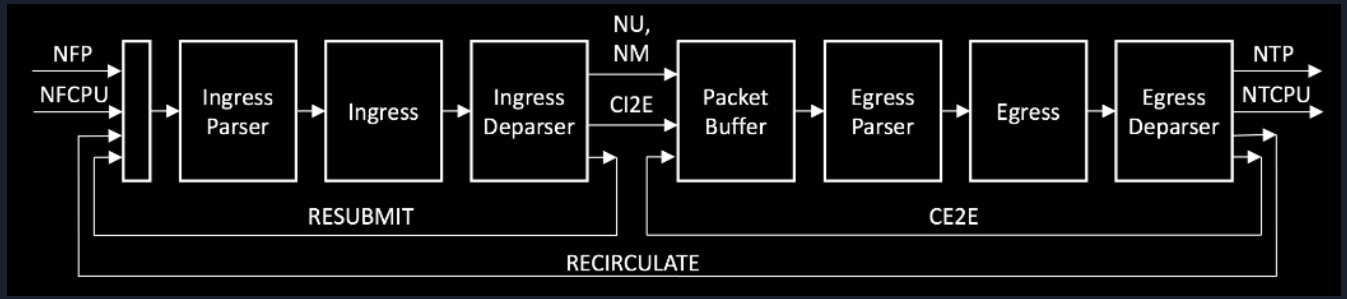
- Aim: persist state for longer than one packet (stateful memories)
- Allow complex, interesting processing over data
- These require resources on the target and hence are managed by a compiler

Program sections (7): 4&5/recursiveness

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

- duplicated (**cloned**) – e.g., to monitor how the packet looks like in the wire;
- sent again to pipelines (**recirculated**) – e.g., to reuse original packet after modifications in egress pipeline;
- sent again to pipelines (**resubmitted**) – e.g., to apply a table multiple times in the ingress 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 sections (8): 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
  - “hdr.ipv4.hdrChecksum” is a calculated field — ensures the egress packet has a correct IPv4 header checksum
- No built-in constructs in P4_16 — provided by specific libraries

```
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);
```

Materials

Materials: 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