



Writing Fine-grained Measurements App with P4 Programmable Switches

Introduction to Packet Parsing and Match-action Tables

Jorge Crichigno
University of South Carolina
<http://ce.sc.edu/cyberinfra>

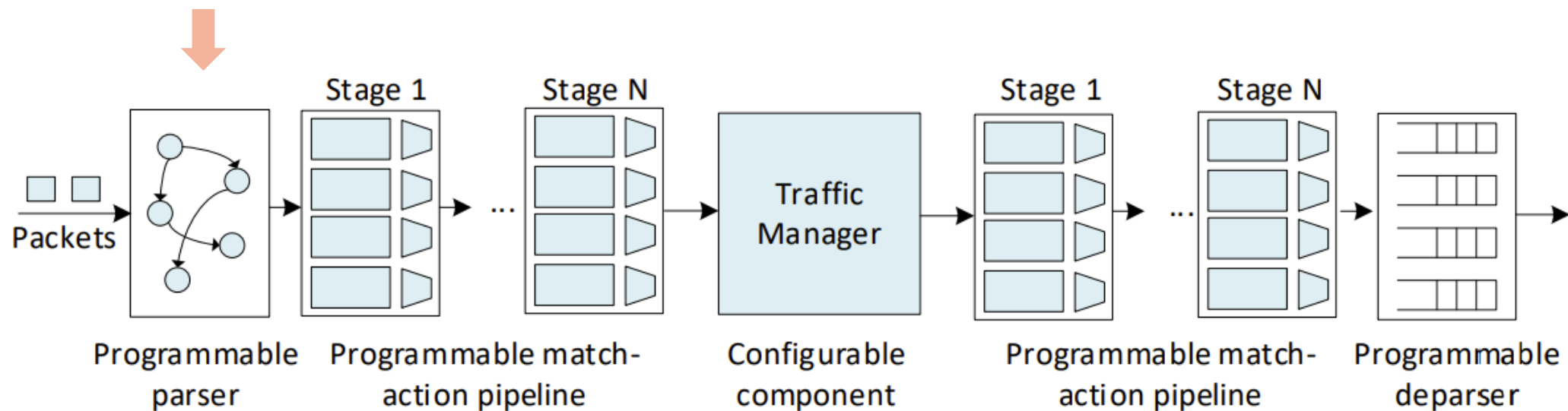
University of South Carolina (USC)
Energy Sciences Network (ESnet)

September 18, 2023

Parser

Programmable Parser

- The parser enables parsing arbitrary headers with a finite state machine
- The state machine follows the order of the headers within the packets
- The packet is split into the defined headers and the remaining is treated as the payload



Packet Headers

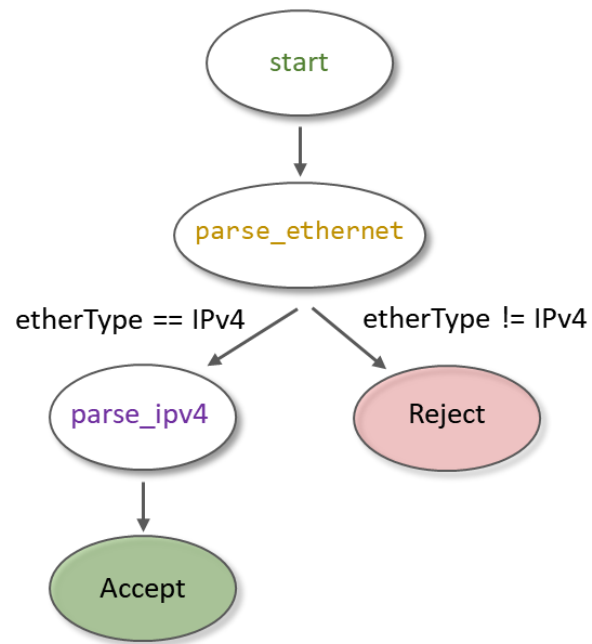
- The packet headers are specified by the programmer
- The programmer has the flexibility of defining custom/non-standardized headers
- Such capability is not available in non-programmable devices

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Version				IHL				DSCP				ECN		Total Length																	
32	Identifier														Flags		Fragment Offset															
64	Time To Live				Protocol				Header Checksum																							
96	Source IP Address																															
128	Destination IP Address																															
160	Options (if IHL > 5)																															

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

Programmable Parser

- Every parser has three predefined states: start, accept, and reject
- Other states may be defined by the programmer
- In each state, the parser executes statements and then transitions to another state

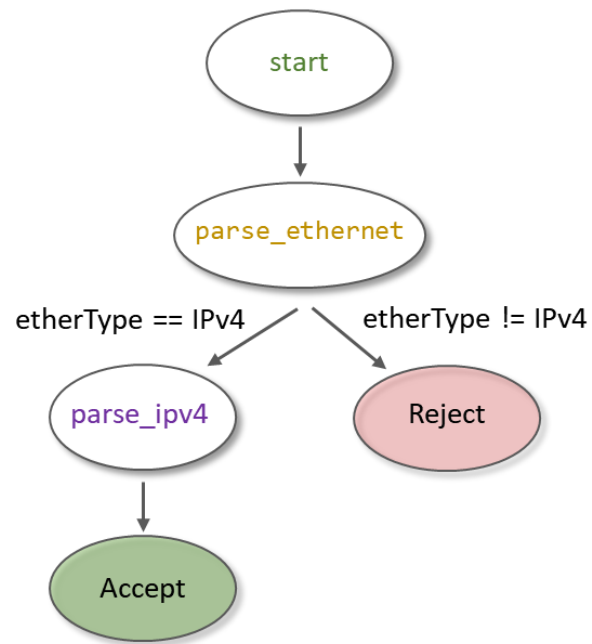


```
state start {
    transition parse_ethernet;
}
state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
        TYPE_IPV4: parse_ipv4;
        default: reject;
    }
}
state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition accept;
}
```

packet is an input parameter; hdr is an output parameter

Programmable Parser

- P4₁₆ has an extract method that can be used to “fill in” the fields of a header from the “raw” packet

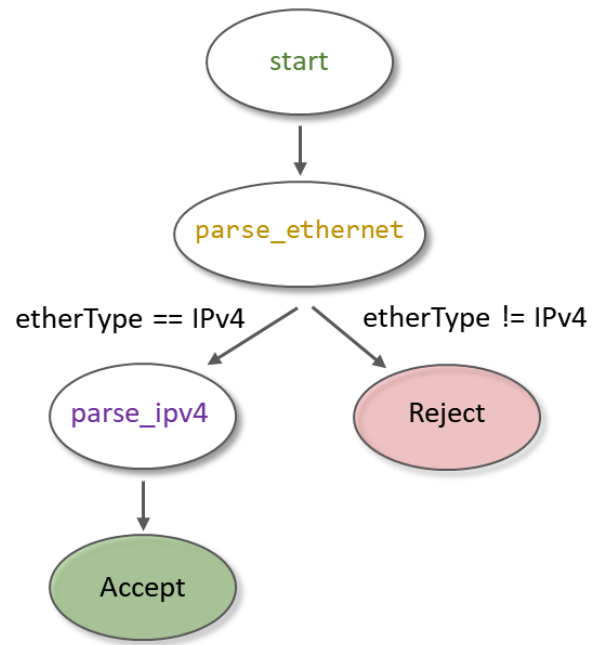


```
state start {
    transition parse_ethernet;
}
state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
        TYPE_IPV4: parse_ipv4;
        default: reject;
    }
}
state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition accept;
}
```

packet is an input parameter; hdr is an output parameter

Programmable Parser

- P4₁₆ has a select statement that can be used to branch in a parser

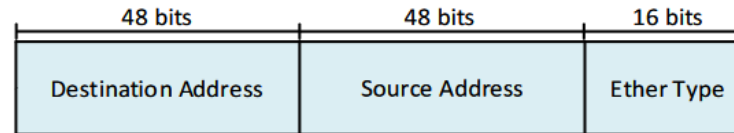


```
state start {
    transition parse_ethernet;
}
state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
        TYPE_IPV4: parse_ipv4;
        default: reject;
    }
}
state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition accept;
}
```

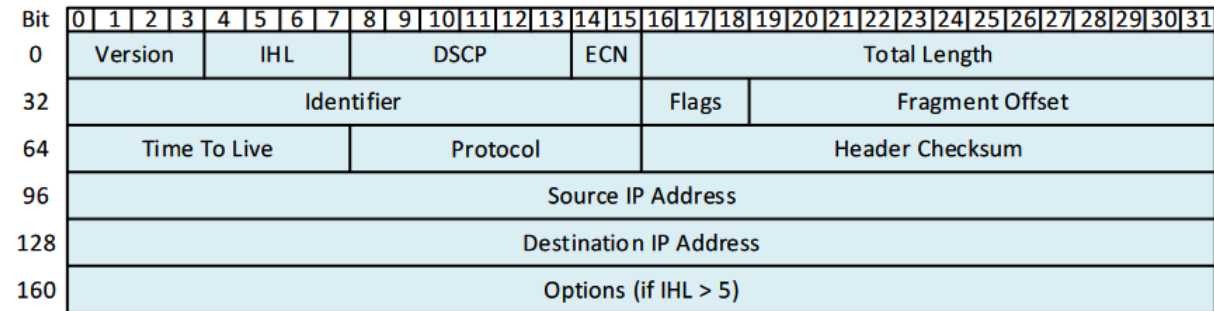
packet is an input parameter; hdr is an output parameter

Headers Format

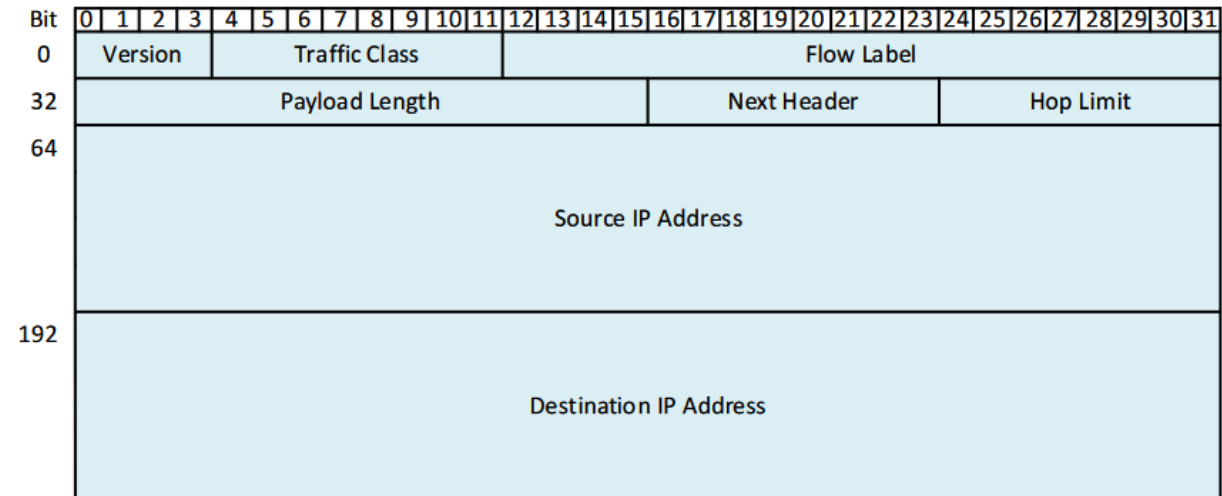
- Ethernet header:



- IPv4 header:

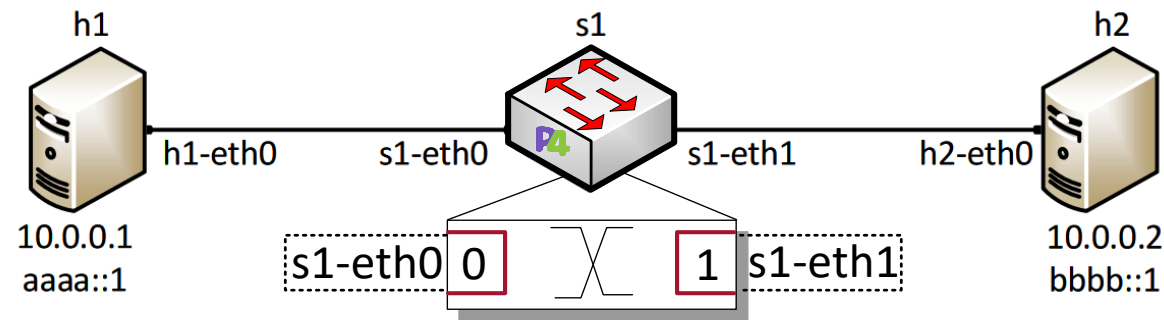


- IPv6 header:



Lab 4 Topology and Objectives

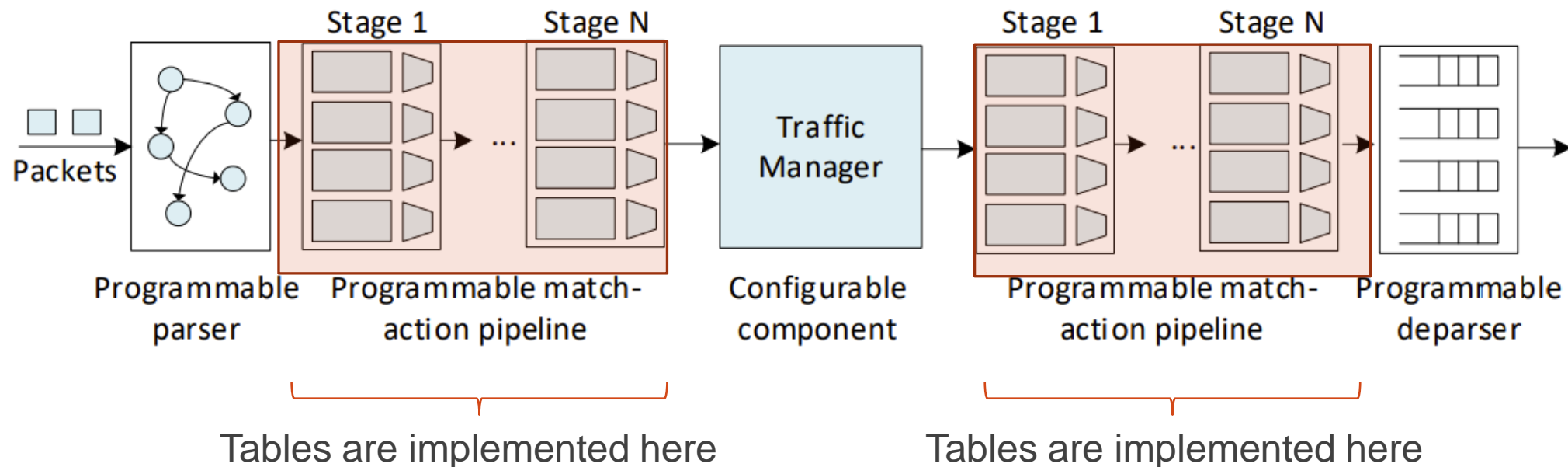
- The topology consists of two hosts: h1 and h2; one P4 switch: s1
- The objectives are:
 - Defining the headers for Ethernet, IPv4 and IPv6
 - Implementing the parser
 - Testing and verifying the switch behavior when IPv4 and IPv6 packets are received



Match-action Tables

Match-action Pipeline

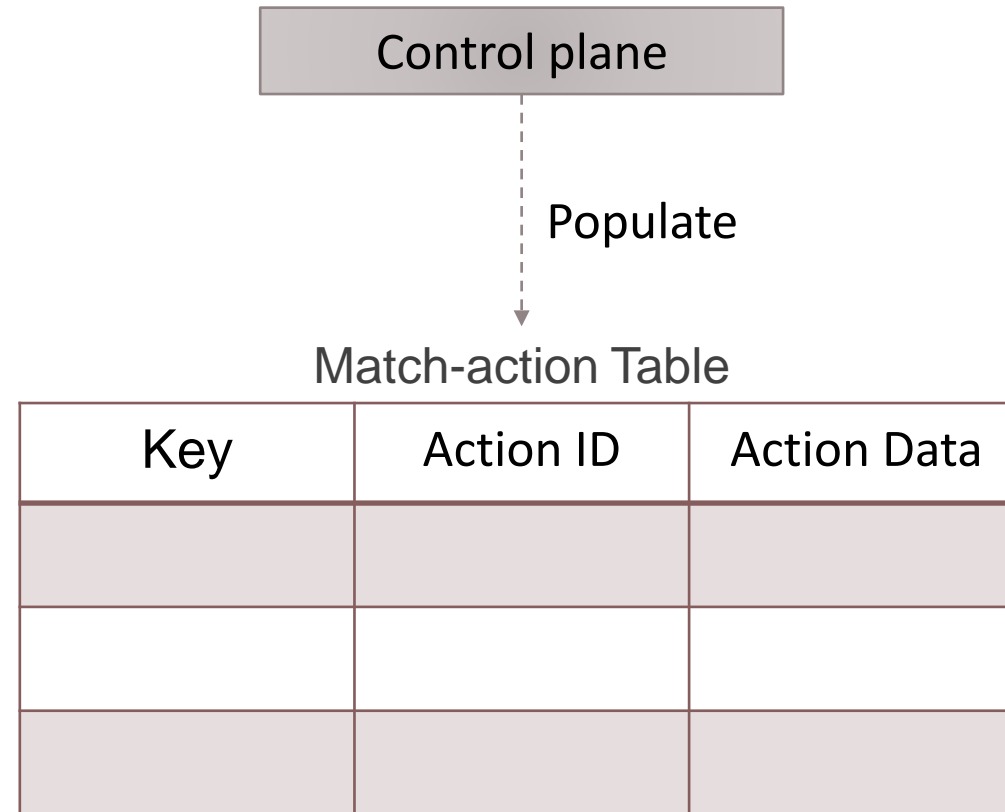
- Tables are the fundamental unit of a Match-Action Pipeline; they define the processing logic inside the match-action pipeline
- They can be used to implement traditional switch tables (e.g., routing, flow lookup, access-control lists)
- They can implement custom user-defined complex logic



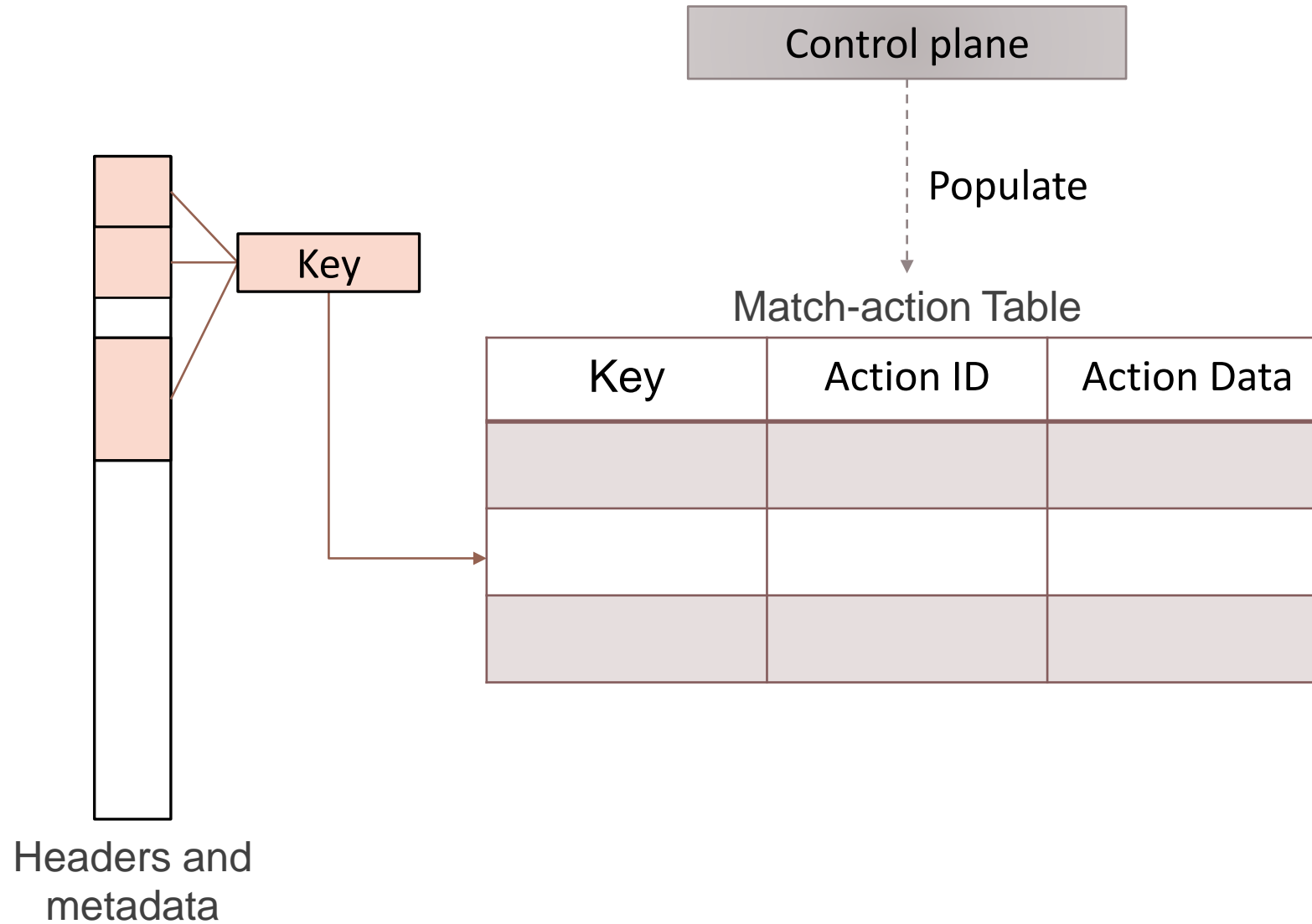
Match-action Table

- Specifies what data to match on
- Specifies a list of possible actions
- Optionally specifies a number of table properties; e.g.,
 - Size
 - Default action
 - Static entries
- An entry contains
 - A specific key to match on
 - An action that is executed when a packet matches the entry
 - Action data (possibly empty)

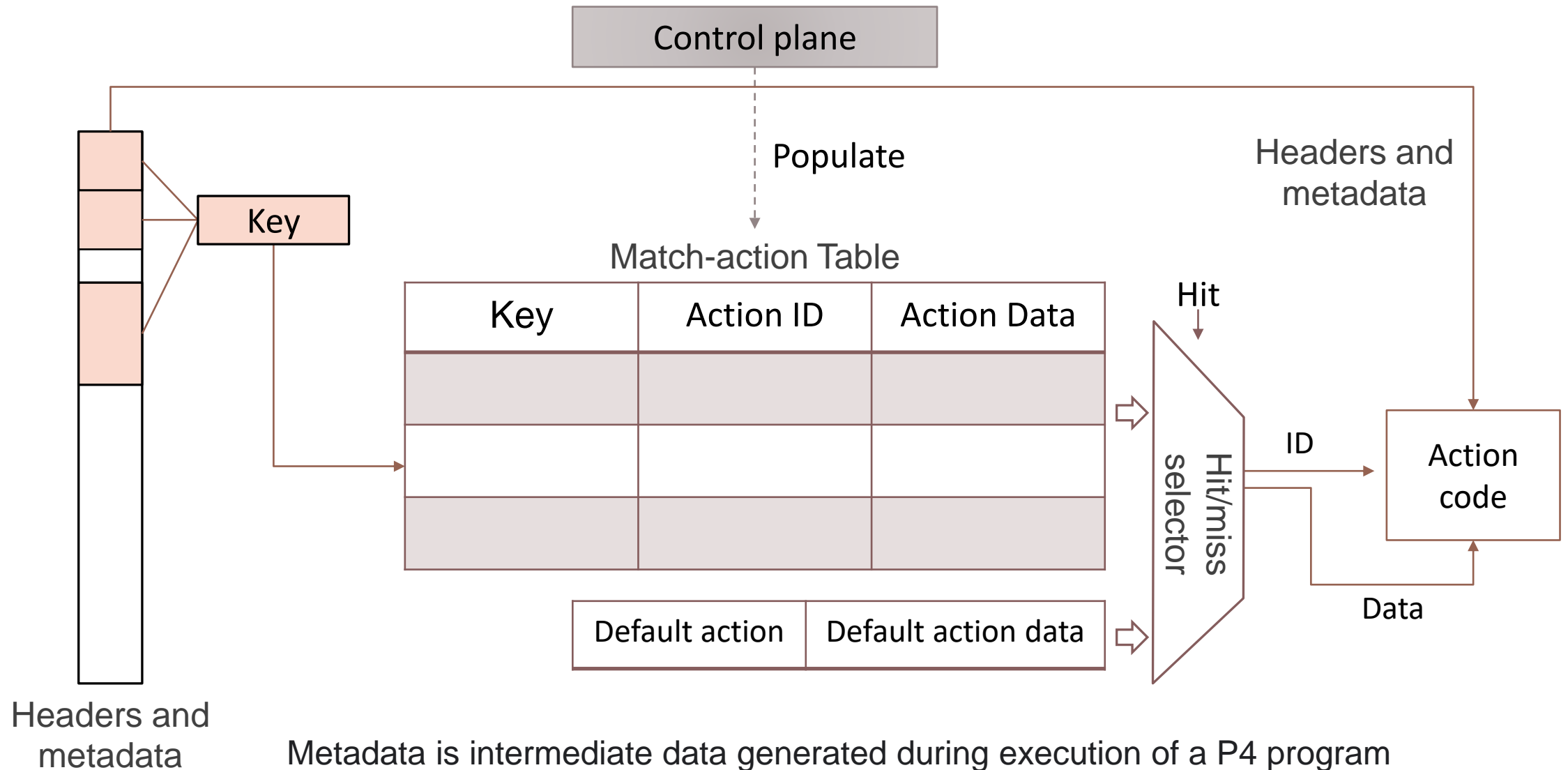
Match-action Table



Match-action Table



Match-action Table



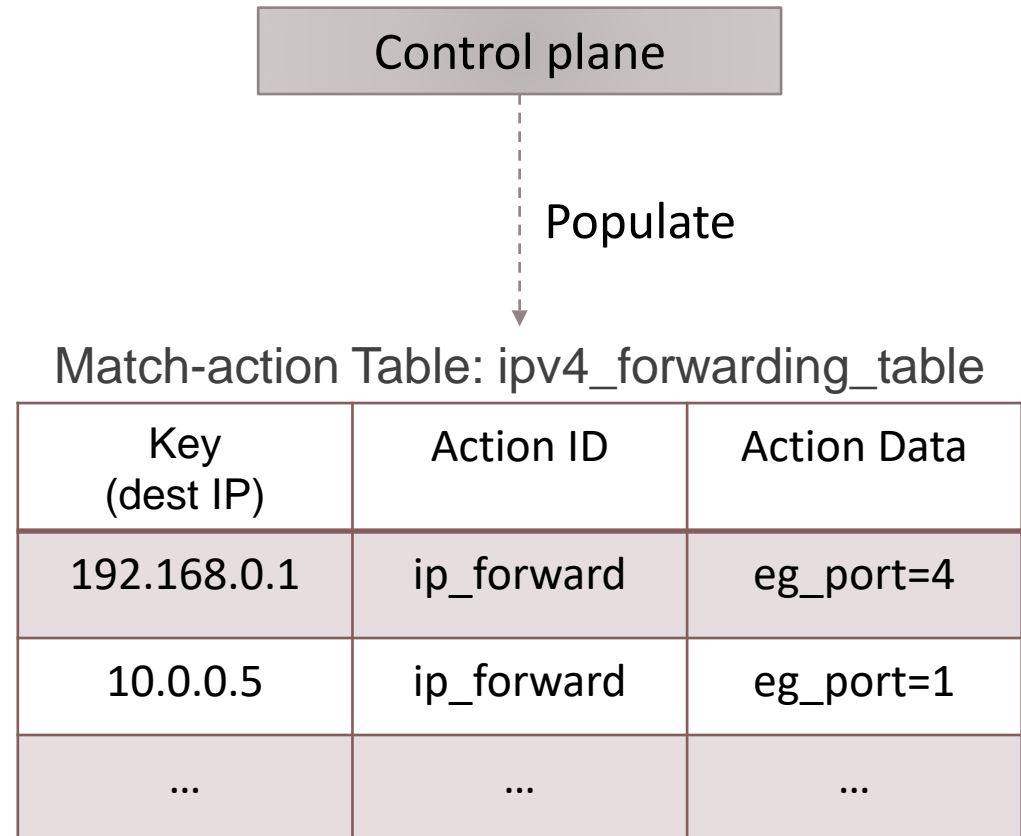
Match-action Table

- Metadata is intermediate data generated during execution of a P4 program
- Standard metadata - data that must be provided by targets
 - `ingress_port`: port on which the packet arrived
 - `egress_spec`: port to which the packet should be sent to
 - `egress_port`: port on which the packet is departing from (read only in egress pipeline; useful value on ingress pipeline only)

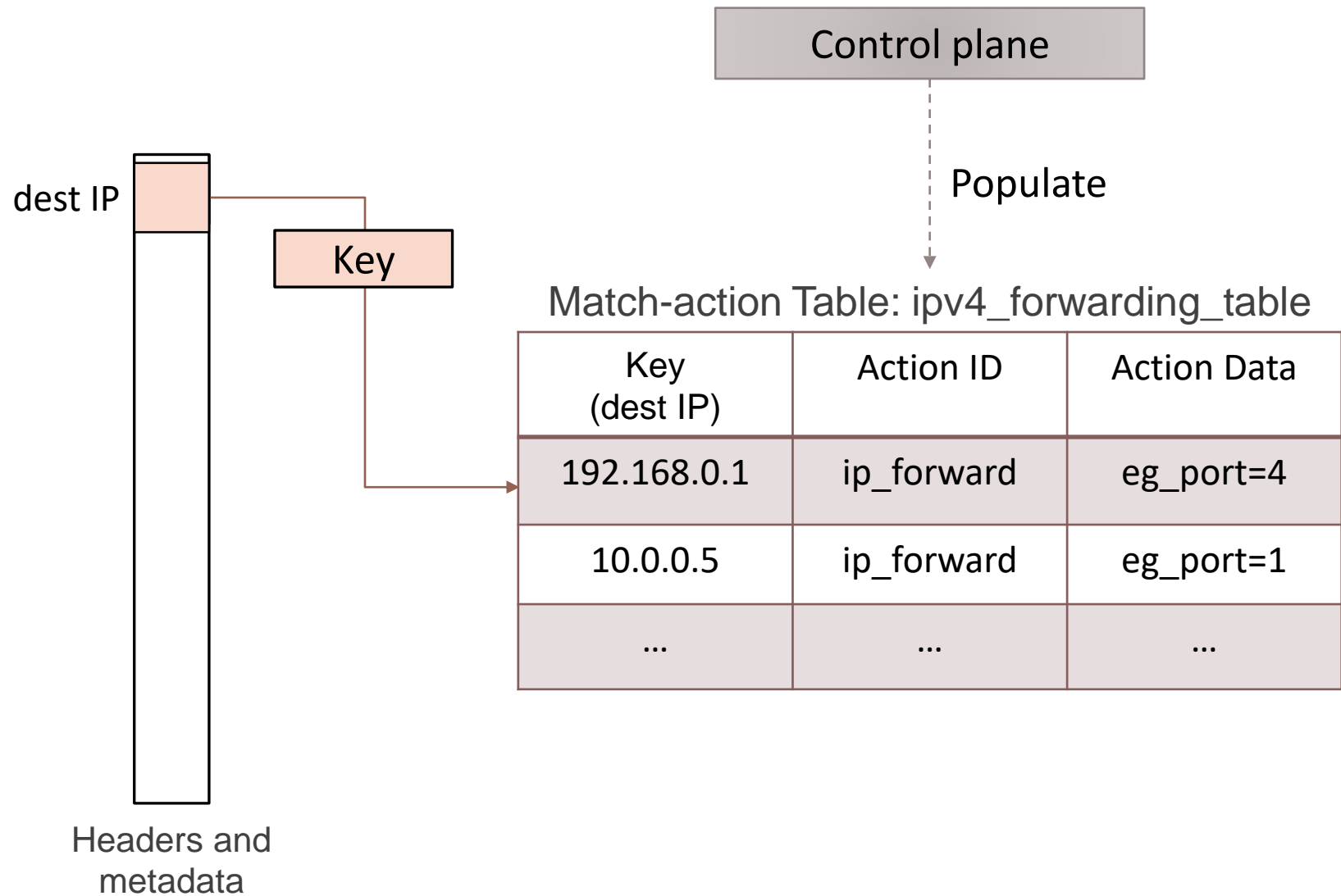
```
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<9>  egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1>  drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    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;
}
```

V1 model standard metadata

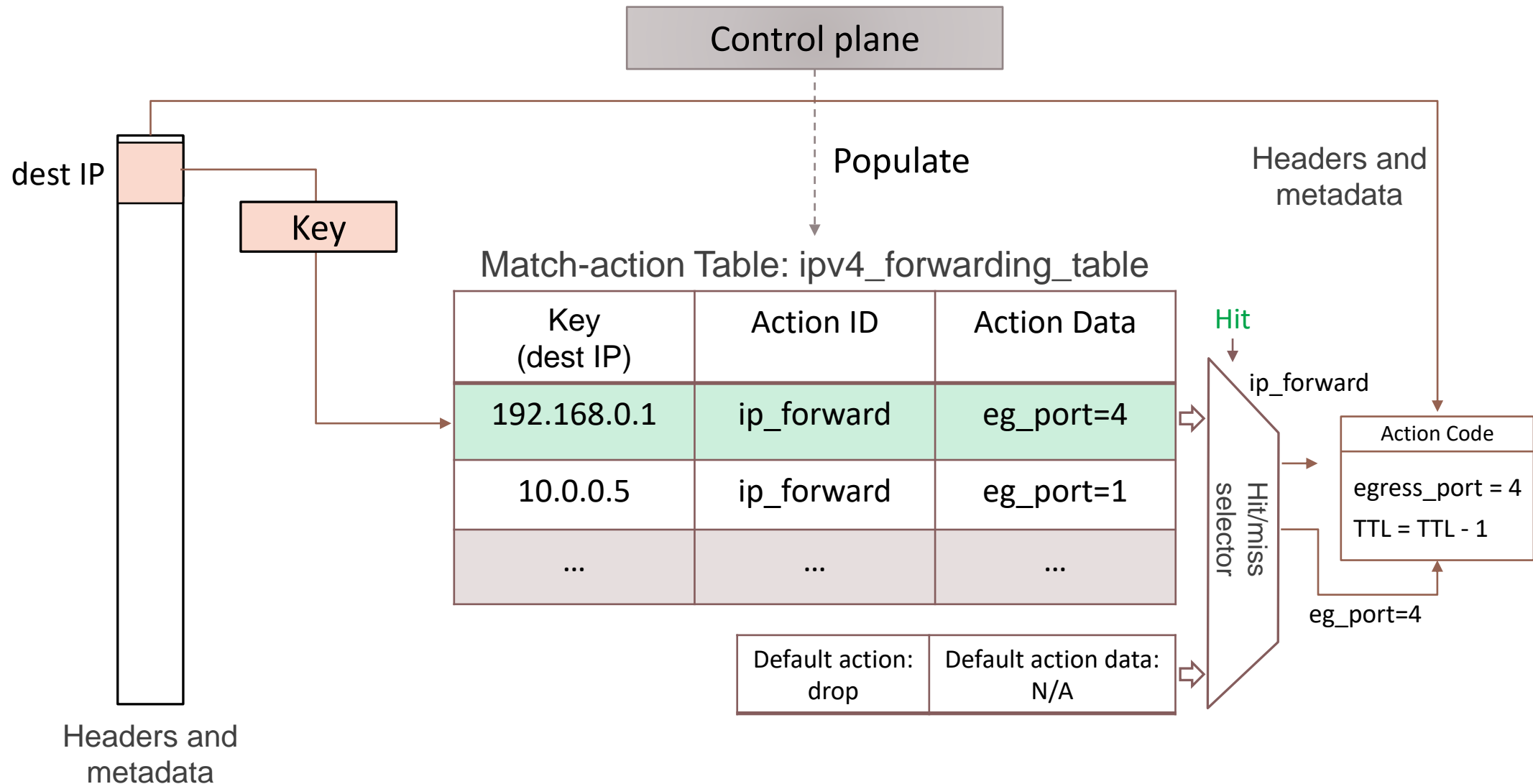
Example: IPv4 Forwarding



Example: IPv4 Forwarding



Example: IPv4 Forwarding



Controls

- Similar to C functions (without loops)
- Can declare tables, variables
- Functionality specified by code in `apply` statement

```
control MyIngress(inout headers hdr,  
                 inout metadata meta,  
                 inout standard_metadata_t std_meta) {  
    bit<48> tmp;  
    apply {  
        tmp = hdr.ethernet.dstAddr;  
        hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;  
        hdr.ethernet.srcAddr = tmp;  
        std_meta.egress_spec = std_meta.ingress_port;  
    }  
}
```

Swap source and destination
MAC addresses

Bounce the packet back out on
the physical port that it came
into the switch on

Actions

- Similar to C functions
- Can be declared inside a control or globally
- Parameters have type and direction

```
control MyIngress(inout headers hdr,  
                 inout metadata meta,  
                 inout standard_metadata_t std_meta) {  
  
    action swap_mac(inout bit<48> src,  
                   inout bit<48> dst) {  
        bit<48> tmp = src;  
        src = dst;  
        dst = tmp;  
    }  
  
    apply {  
        swap_mac(hdr.ethernet.srcAddr,  
                hdr.ethernet.dstAddr);  
        std_meta.egress_spec = std_meta.ingress_port;  
    }  
}
```

Swap source and destination
MAC addresses

Bounce the packet back out on
the physical port that it came
into the switch on