EBERHARD KARLS UNIVERSITÄT TÜBINGEN

FACULTY OF SCIENCE
Communication Networks

Data Plane Programming With P4

Fabian Ihle <fabian.ihle@uni-tuebingen.de>
University of Tübingen

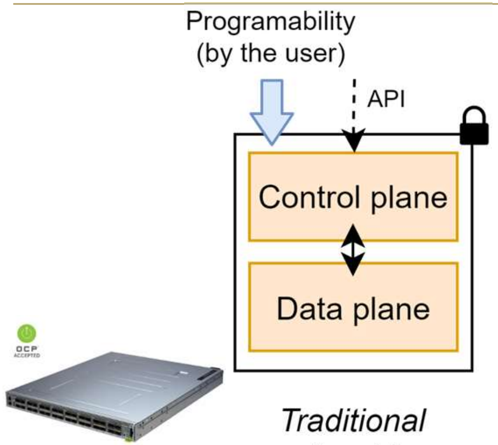
http://kn.inf.uni-tuebingen.de

Agenda

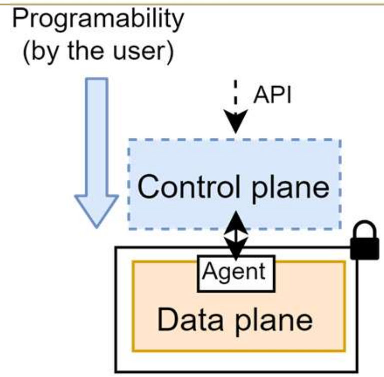
- Introduction
- The P4 Programming Model
- Anatomy of a P4 program
- The Control Plane
- UniCorn-P4
- P4 Tutorial
- Hackathon

INTRODUCTION

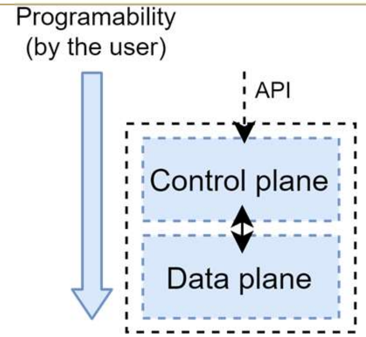
Networking Concepts



Traditional networking



SDN with fixed-function data planes



Full network programability with data plane programming

Source: [1]

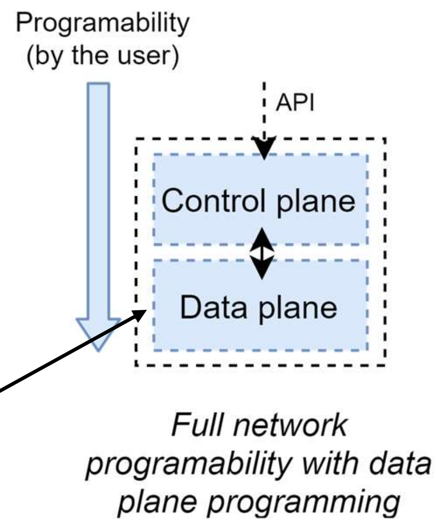
- "Black box" (switch) received from vendor
- Fixed-feature set
- Configure feature set provided by vendor (e.g., via SNMP)
- Feature set not extendable
- "Configure IPv4 Routing for the prefix 10.0.0.0/8"

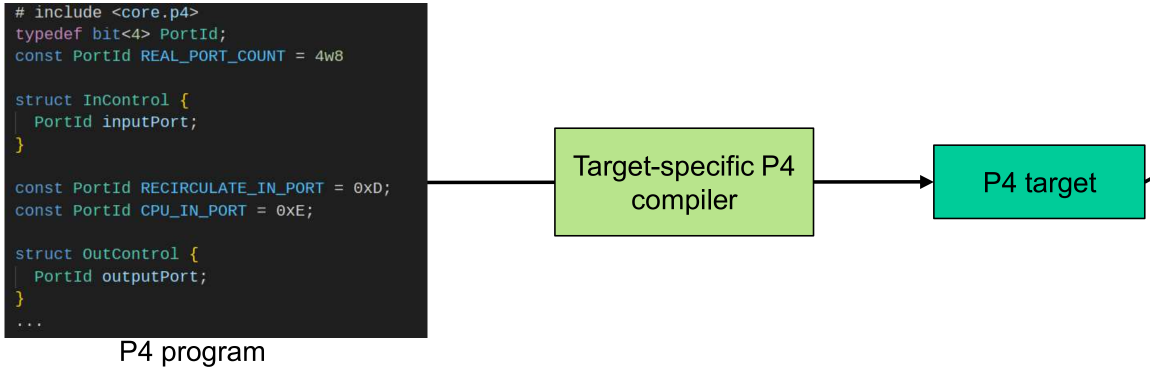
- Switch divided into Control Plane (controller) and Data Plane (switch)
- Data plane provides fixed-functionality, e.g., IPv4 Routing
- Programmable Controller, e.g., "Reroute traffic on a failure by changing the IPv4 routing entries"

- Programmable Data Plane and Control Plane
- Implement full feature set by yourself, e.g., IPv4 routing, IP tunneling, or FRR
- Low-level operations are used to define packet processing

P4: Overview

- ▶ **P4: P**rogramming **p**rotocol-independent **p**acket **p**rocessors [2], [3]
  - High-level programming language to describe data planes
  - Target-specific compiler maps P4 program to target
    - P4 program not tied to a specific vendor or device (target), but can be used on “any” P4 programmable target





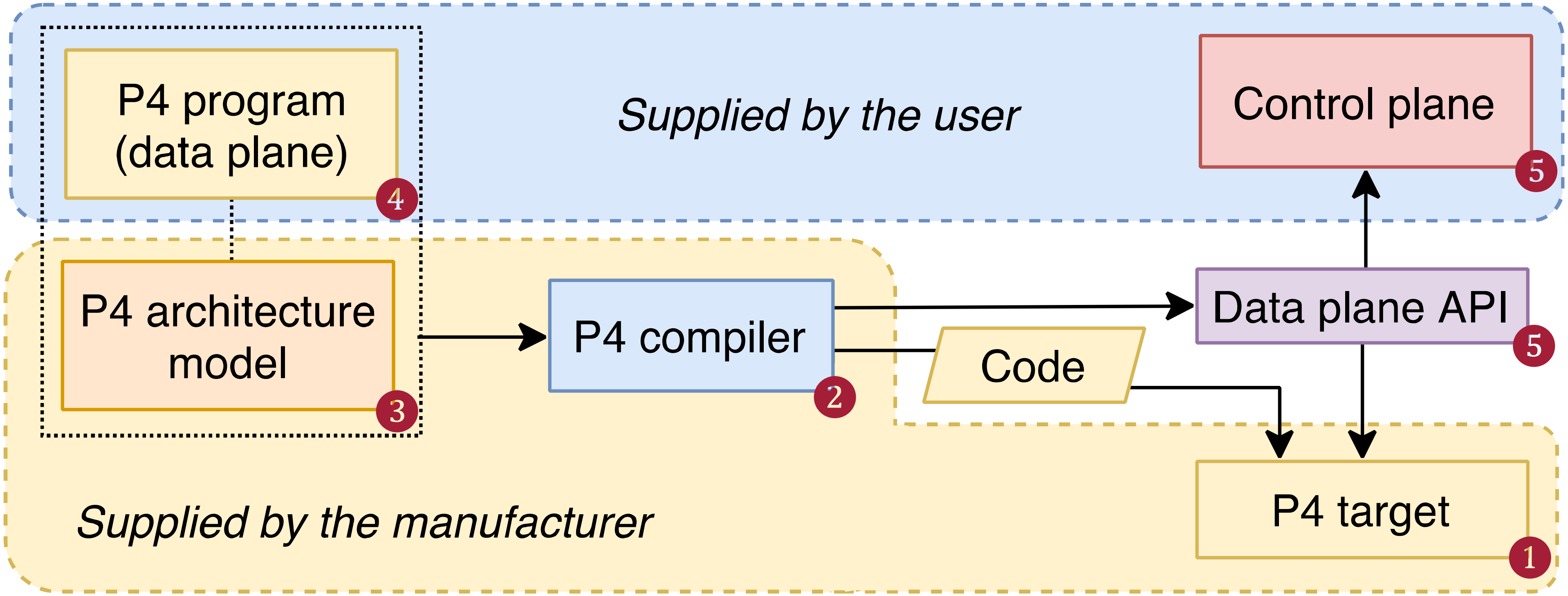
- P4 defines low level (packet processing) operations

⇒ Fully programmable data plane
  - Limited only by expressiveness and features of P4 (and not by vendor)

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

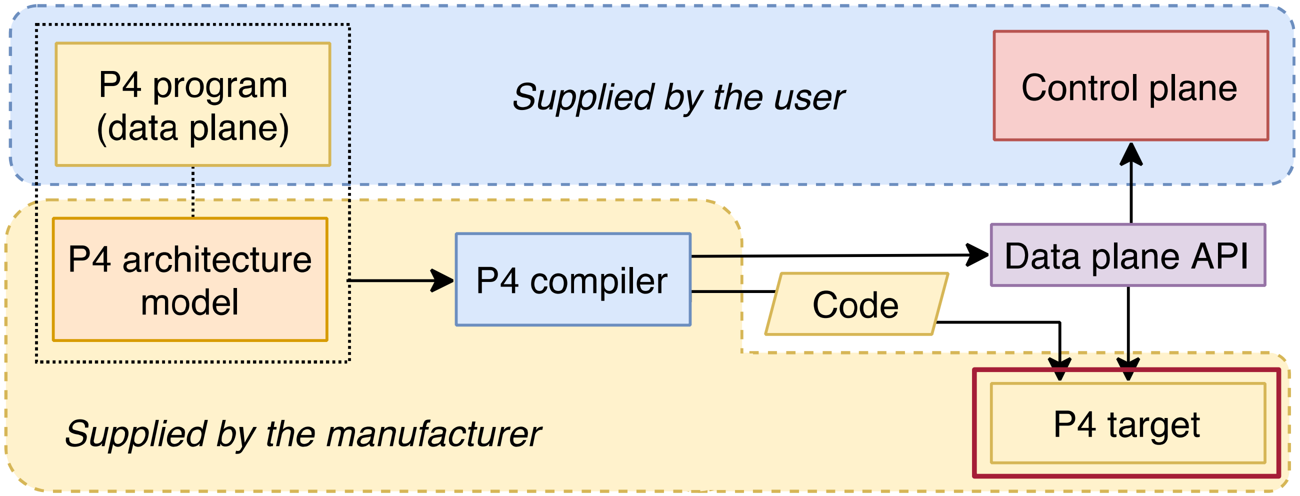
THE P4 PROGRAMMING MODEL

P4 Programming Model



Definition: P4 Target

- **What is a P4 target?**
  - A packet-processing system capable of executing a P4 program
  - P4 targets follow a specific architecture, e.g., PSA, PISA, …



P4 Targets - Categories

▶ Software

- Software-based P4 targets run on a standard CPU
- Not suitable for high performance scenarios
- Good for rapid prototyping

▶ FPGA

- Tool chains translate P4 programs for field programmable gate arrays (FPGAs)
- Includes logic synthesis, verification, validation and placement/routing on the logic circuit for the FPGA

▶ ASIC

- Specialized micro chip for P4
- ASIC = Application-Specific Integrated Circuit

Target	P4 Version	$P4_{16}$ Architecture	Active Development
Software			
p4c-behavioral	$P4_{14}$	n.a.	X
bmv2	$P4_{14}$, $P4_{16}$	v1model, psa	✓
eBPF	$P4_{16}$	ebpf_model.p4	✓
uBPF	$P4_{16}$	ubpf_model.p4	✓
XDP	$P4_{16}$	xdp_model.p4	✓
T4P4S	$P4_{14}$, $P4_{16}$	v1model, psa	✓
Ripple	n.a	n.a	n.a
PISCES	$P4_{14}$	n.a.	X
PVPP	n.a.	n.a.	X
ZodiacFX	$P4_{16}$	zodiacfx_model.p4	n.a.
FPGA			
P4→NetFPGA	$P4_{16}$	SimpleSumeSwitch	✓
Netcope P4	n.a.	n.a.	✓
P4FPGA	$P4_{14}$, $P4_{16}$	n.a.	X
ASIC			
Barefoot Tofino/Tofino 2	$P4_{14}$, $P4_{16}$	v1model, psa, TNA	✓
Pensando Capri	$P4_{16}$	n.a	✓
NPU			
Netronome	$P4_{14}$, $P4_{16}$	v1model	✓

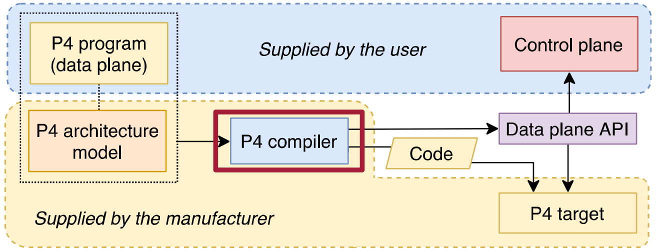
P4 Targets - Categories

▶ NPU

- Network processing units
- Programmable ASICs optimized for networking applications
- Part of standalone network devices or device boards

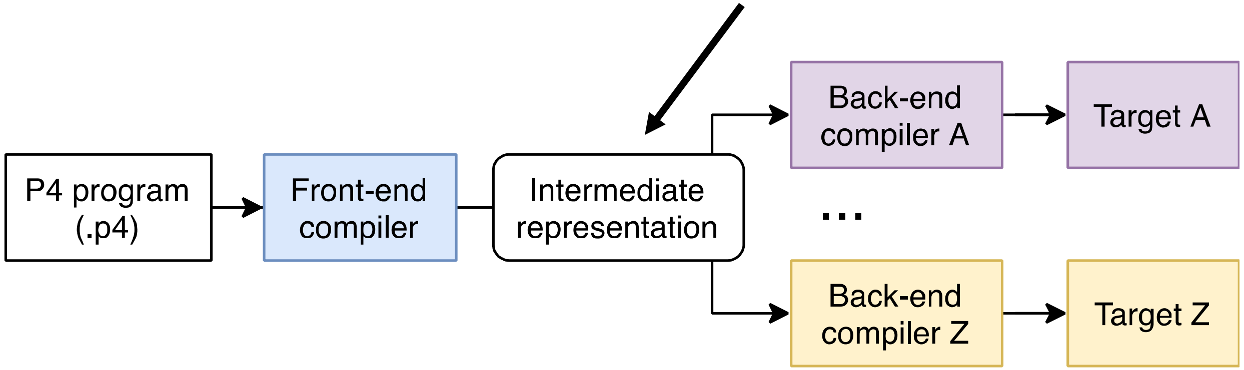
Target	P4 Version	$P4_{16}$ Architecture	Active Development
Software			
p4c-behavioral	$P4_{14}$	n.a.	X
bmv2	$P4_{14}$, $P4_{16}$	v1model, psa	✓
eBPF	$P4_{16}$	ebpf_model.p4	✓
uBPF	$P4_{16}$	ubpf_model.p4	✓
XDP	$P4_{16}$	xdp_model.p4	✓
T4P4S	$P4_{14}$, $P4_{16}$	v1model, psa	✓
Ripple	n.a	n.a	n.a
PISCES	$P4_{14}$	n.a.	X
PVPP	n.a.	n.a.	X
ZodiacFX	$P4_{16}$	zodiacfx_model.p4	n.a.
FPGA			
P4→NetFPGA	$P4_{16}$	SimpleSumeSwitch	✓
Netcope P4	n.a.	n.a.	✓
P4FPGA	$P4_{14}$, $P4_{16}$	n.a.	X
ASIC			
Barefoot Tofino/Tofino 2	$P4_{14}$, $P4_{16}$	v1model, psa, TNA	✓
Pensando Capri	$P4_{16}$	n.a	✓
NPU			
Netronome	$P4_{14}$, $P4_{16}$	v1model	✓

P4 Compiler (I)



▶ Two-Layer Compiler Model

- Most P4 compilers use the two-layer compiler model
- Consists of common frontend and a target-specific backend
- Front-end compiler
  - syntactic and target-independent semantic analysis
- Back-end compiler
  - Target-specific transformations



P4 Compiler (II)

P4-hlir (high-level intermediate representation)

- First generation P4-compiler for P4 v14 written in Python
- Uses high-level intermediate representation (HLIR)
  - Tree of python objects

P4c

- Current generation P4-compiler for both v14 and v16
- Written in C++
- Uses C++-object-based intermediate representation (IR)
- IR can be represented as JSON file
- Has backends for multiple targets, e.g., bmv2, eBPF, uBPF, …

Vendor specific compilers

- P4 target vendors maintain own compilers based on the common frontend

P4 Model: Benefits

- P4 programming model decouples software and hardware development / evolution
  - P4 architectures as abstraction layer (or interface) between software and hardware
    - Hides low level, target-specific details from high-level processing
  - Software-models of P4 architectures allow software development independently of hardware
  - Interface ensures compatibility

- Resource mapping and management is left to the manufacturer
  - Software developers use only abstract high-level description of resources, e.g., Tables, registers, …
  - Compilers maps software description to hardware resources
    - Manages low-level details, e.g., memory allocation, scheduling, ...

  ⇒ Software developers do not need to worry about efficiency

P4 Language: Benefits

- Packet forwarding expressible as programs
  - Language expressiveness
    - Describe target-independent packet processing with general-purpose operations and table look-ups
    - ⇒ Programs portable across targets
  - Flexibility
    - Easy to adapt
    - Implement novel packet processing
  - Software engineering characteristics
    - Type checking, information hiding (interfaces), software reuse, …
    - Agile development process
  - Component libraries
    - Wrap hardware-specific functions into portable P4 constructs
    - Supplied by manufacturers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

P4 Language Consortium

- Independent non-profit organization (https://p4.org)
- Free membership (in contrast to OpenFlow)
- Partners from industry and academia (https://p4.org/tst/)
  - Technical steering team
    - Nate Foster (Cornell University), Guru Parulkar (ONF), Armin Vahdat (Google)
  - Industry members
    - Cisco, Juniper, Google, Microsoft, Intel, Dell, Xilinx, …
  - Academic members
    - Princeton, Cornell, Stanford, …
- Many working groups (https://p4.org/working-groups/)
  - Language design, API, Architecture, Applications, Education

P4 Architectures

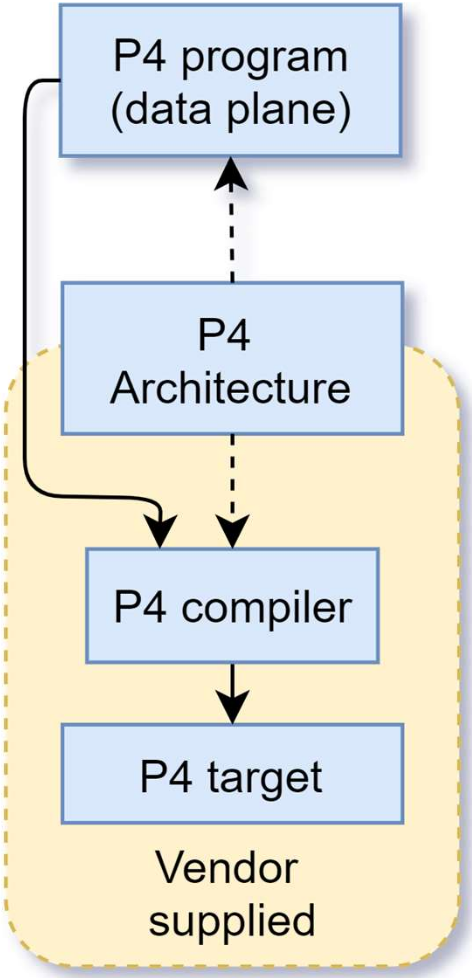
- Diverse targets with different underlying functionalities
  - Software-based, hardware-based, ASICs, FPGAs, ...
  - Challenge: efficient execution of high-level code

  ⇒ Programming models for different types of targets

- P4 architectures
  - Programming models with logical view of the targets
  - Decouples P4 program from targets
    - P4 program is developed for specific P4 architecture

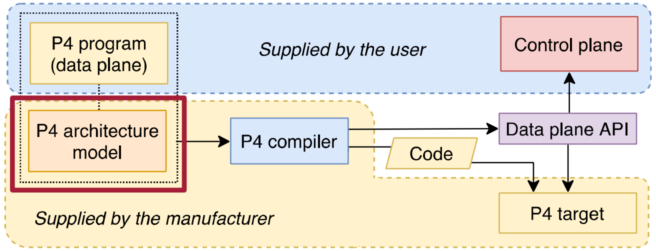
    ⇒ A P4 program can be run on any target following the same architecture
  - Manufacturers
    - "implement" architecture on hardware device
    - provide compiler to map P4 code to device

⇒ P4 is not only a programming language but also a programming model based on architectures



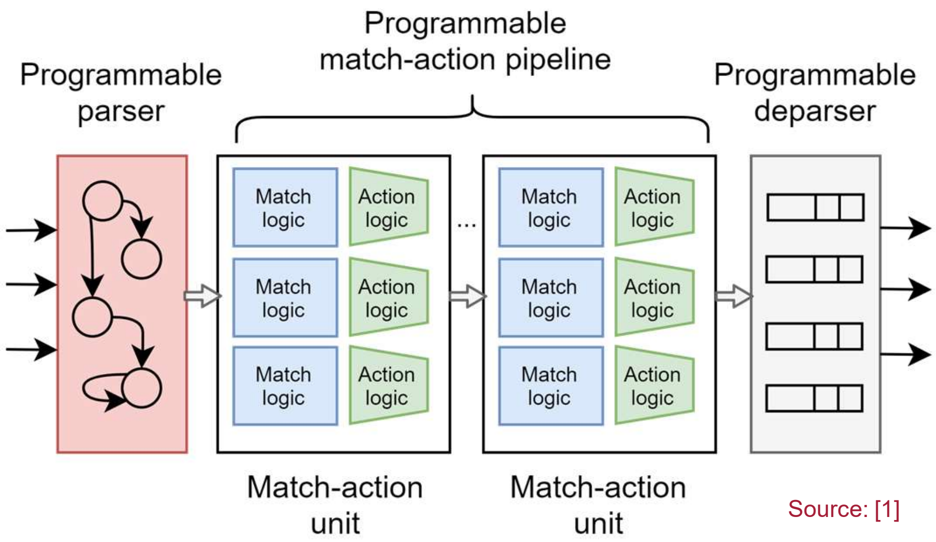
P4 Architecture

- Programming models with logical view of the targets
  - Hardware abstraction layer
- Decouples P4 program from targets
  - → A P4 program can be run on any target following the same architecture
  - Architecture model and corresponding compiler provided by manufacturer



- Network devices have **<u>programmable</u>**
  1. (de)parser: protocol independence
  2. match-action pipeline: custom packet processing

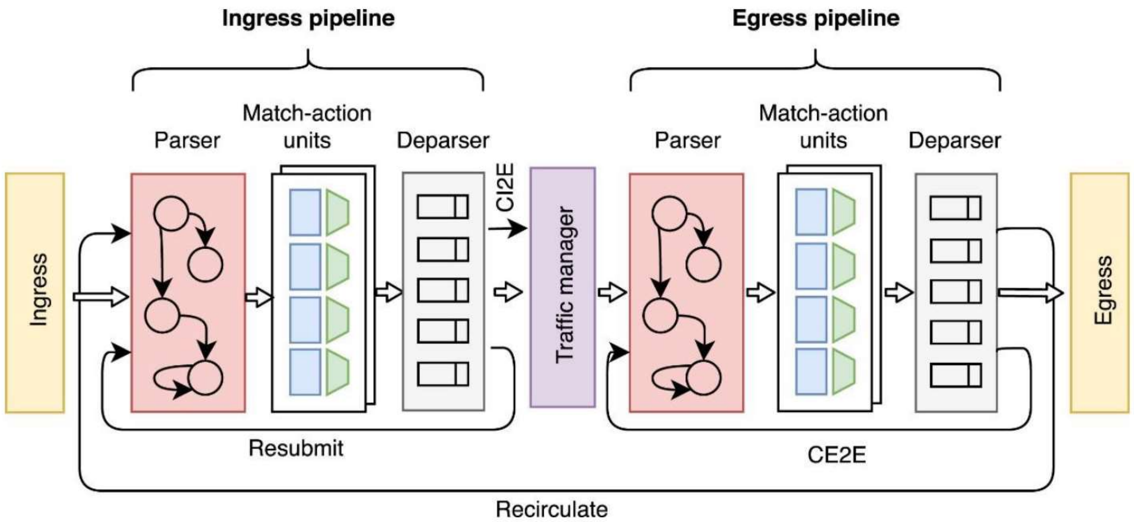
- Protocol-Independent Switch Architecture (PISA)



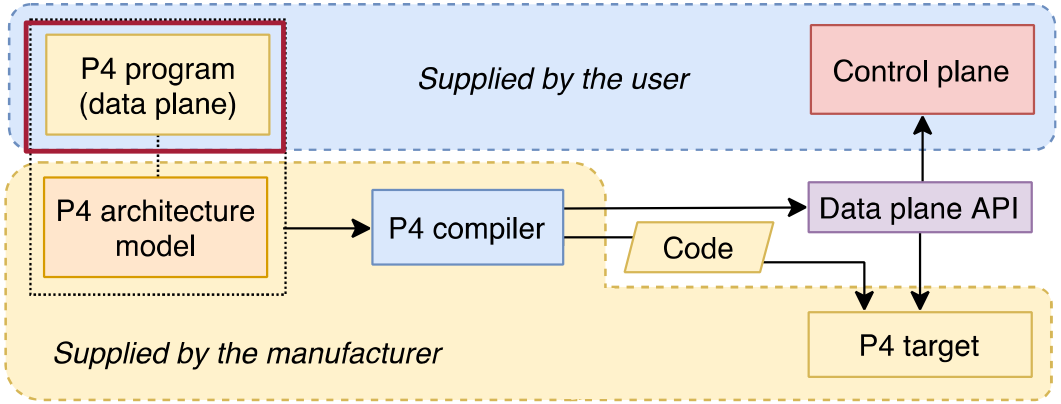
Source: [1]

P4 Architectures: Other

- Portable Switch Architecture (PSA)
  - 2 control blocks with separate (de-)parsers
  - Traffic manager takes care of queueing etc.



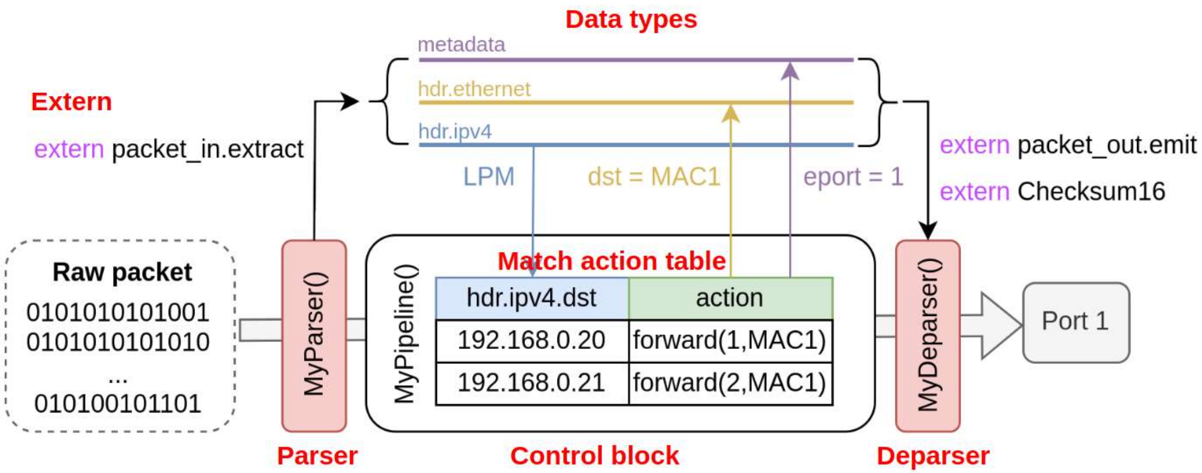
- V1Model Architecture
  - Implemented by BMv2 target
  - Used in the Hackathon
  - More info: https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4



ANATOMY OF A P4 PROGRAM

P4 Components – Overview

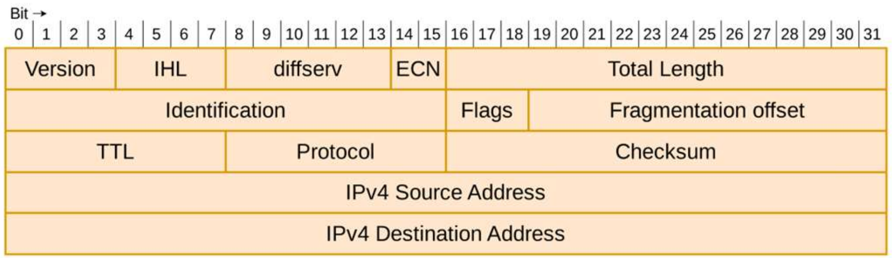
- Data types
  - For header fields and metadata fields
- Parsers
  - Extract information from a packet
- Control Blocks
  - Describe packet processing pipeline
  - Match-action units
- Deparsers
- Externs
  - Architecture/target-specific operations



P4 Data Types (I)

- Base types
  - `bool`: Boolean
  - `bit<n>`: Unsigned integer (bitstring) of size `n` (`bit` → `bit<1>`)
  - `int<n>`: Signed integer of size `n` (>=2)
  - `varbit<n>`: Variable-length bitstring (fixed maximum length `n`)
- `typedef`
  - Alternative name for a type
  - „Syntactic sugar“
- header
  - Ordered collection of base types
  - Describes a packet header, e.g., an IPv4 header

```
typedef bit<32> ipv4_addr_t;

header ipv4_t {
    bit<4> version;
    bit<4> ihl;
    bit<6> diffserv;
    bit<2> ecn;
    bit<16> total_len;
    bit<16> identification;
    bit<3> flags;
    bit<13> frag_offset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdr_checksum;
    ipv4_addr_t srcAddr;
    ipv4_addr_t dstAddr;
}
```



P4 Data Types (II)

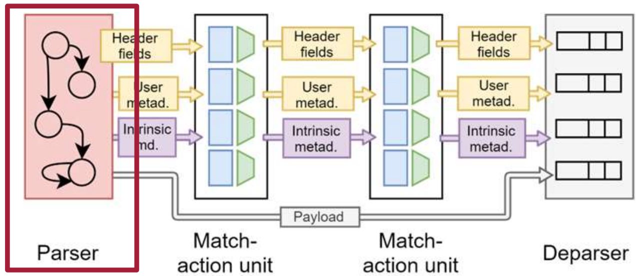
- struct
  - Unordered collection of members
- Two types of metadata structs
  - Intrinsic metadata
    - Architectural metadata associated with each packet
    - Example: input port, timestamp, …
  - User-defined metadata
    - User-defined data structures associated with each packet
    - Comparable to variables
  - is discarded when the packet leaves the switch
  - can be used to exchange information between control blocks
    - No other variables than metadata between control blocks!

```
struct standard_metadata_t {
    PortId_t    ingress_port;
    PortId_t    egress_spec;
    PortId_t    egress_port;
    bit<32> enq_timestamp;
    bit<32>     instance_type;
    bit<32>     packet_length;
    ...
}

struct metadata {
    bit<16> register_index;
    bit<32> self_defined_fields;
    ...
}

 struct headers {
     ethernet_t   ethernet;
     mpls_t[16]   mpls;
     ipv4_t       ipv4;
 }
```

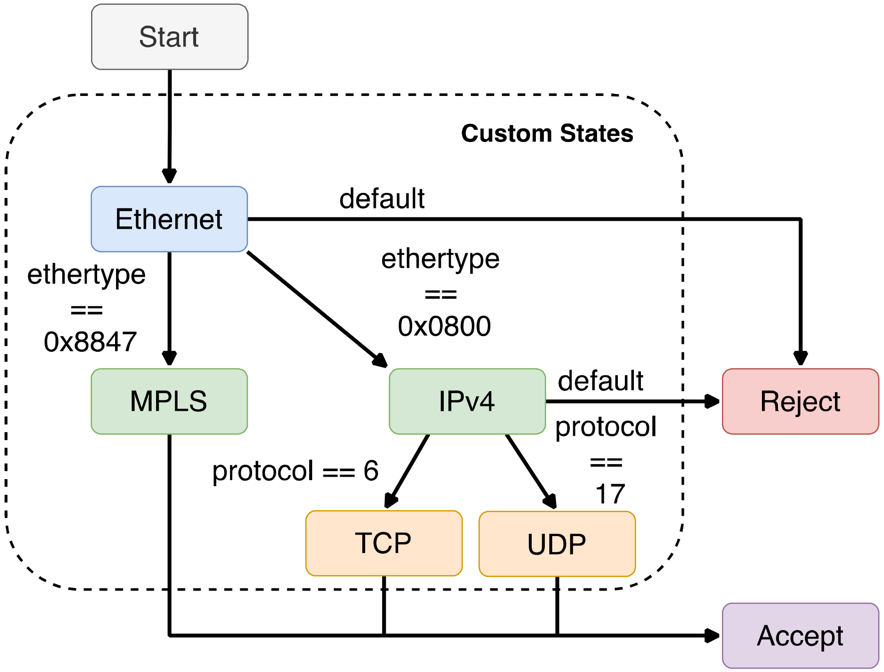
- headers struct
  - Describes the complete packet header

Parsers (I)

- Parser maps serialized packets to header fields and metadata fields for later use
  - 1010110101 → Ethernet header | IP header …
  - Packets consist of headers and payload
  - Non-extracted headers (= payload) cannot be accessed



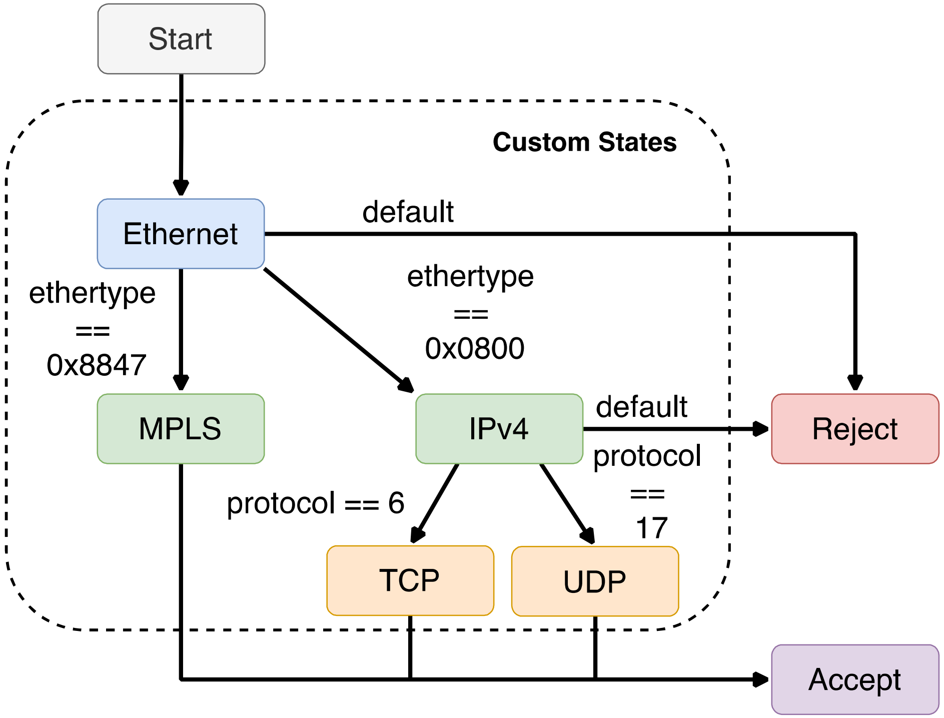
- Parser described as state machine
  - Three predefined states
    - Start, Accept, Reject
  - Other states may be defined by the developer
    - Extract information from packets
    - Mark extracted header as `valid`
    - Transition to another state (loops are OK)



Parser (II)

- Packets consist of headers and payload
- Parser extracts headers for later use (e.g., MATs)
- Non-extracted headers (= payload) cannot be accessed
- Requires: Header definition

```
typedef bit<32> ipv4_addr_t;

header ipv4_t {
    bit<4> version;
    bit<4> ihl;
    bit<6> diffserv;
    bit<2> ecn;
    bit<16> total_len;
    bit<16> identification;
    bit<3> flags;
    bit<13> frag_offset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdr_checksum;
    ipv4_addr_t srcAddr;
    ipv4_addr_t dstAddr;
}
```



Parser (II)

```
parser packetParser(packet_in packet,
                    out headers hdr,
                    inout metadata meta,
                    inout standard_metadata_t standard_metadata) {

    // entry point
    state start {
        transition parse_ethernet;
    }

    // parse ethernet packet
    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4 :         parse_ipv4;
            TYPE_ARP:           parse_arp;
            default :           reject;
        }
    }

    ...

}
```

1. Definition of parser
2. Extract header with given name
3. Select next header to parse based on header field
4. Go to next state

```
// Ethernet header
header ethernet_t {
    macAddr_t dst_addr;
    macAddr_t src_addr;
    bit<16> etherType;
}

// header naming
struct headers {
    ethernet_t          ethernet;
    arp_t               arp;
    ipv4_t              ipv4;
}
```

Control Blocks (I)

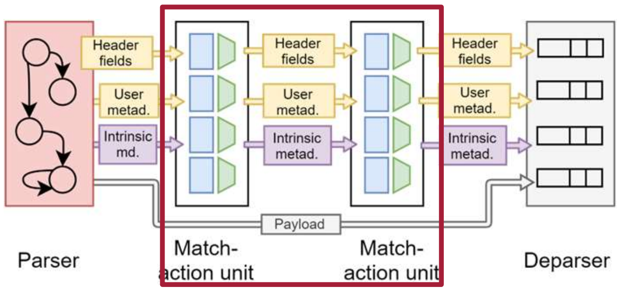
- Control Blocks…
  - encapsulate functionality
    - Some similarities with classes in other languages
  - define packet processing operations

```
control ingress(inout headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {
  // do something
  // MATs etc

}
```

- Two required control blocks
  - Ingress and egress

- Data (e.g., variables) is carried in *user-defined metadata* to other control blocks

- Control blocks can…
  - Use branching (`if`, `select`)
  - Use logical and simple arithmetic operations (`&&, ||, +, -, …`)
  - NOT use loops
  - Use match+action tables (MATs)



EBERHARD KARLS UNIVERSITÄT TÜBINGEN

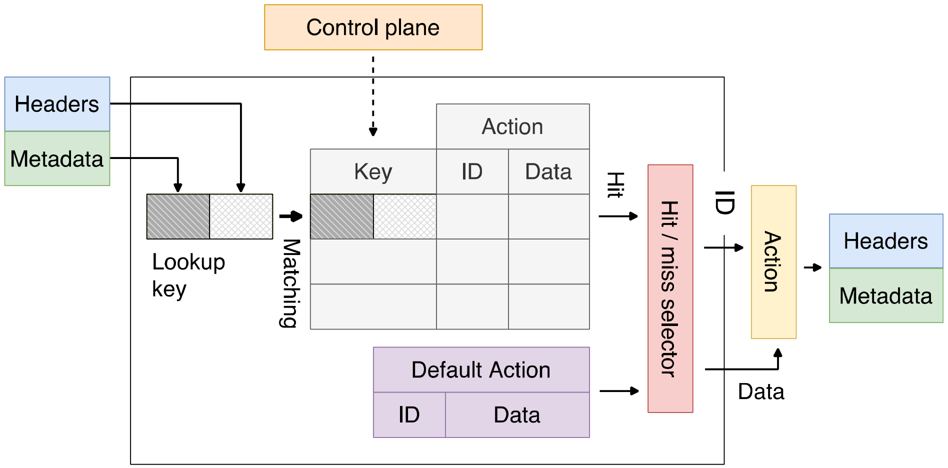
Match-action tables (MATs) (I)

- *Match* on selected key fields, execute an *action* accordingly

- Structure of MAT entries, i.e., table columns
  - (Match) key(s)
    - header / metadata field for comparison with table entries
    - Match types, i.e., longest-prefix match (lpm), exact, wildcard, …
  - Possible action(s)
    - Actions are defined outside of the MAT in the P4 program
  - Define most of the program logic

- Packet is matched with selected header or metadata fields to the defined key

```
action forward(egressSpec_t port) {
    standard_metadata.egress_spec = port;

    // decrement time to live (ttl)
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}

table ipv4 {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        forward;
    }
}
```



Match-action table

MATs (II)

- Data plane only defines format
- Requires control plane to populate entries
  - Specify key value
  - Specify action and parameter(s)

Key	Action
10.0.1.1/32	forward(1)
10.0.1.2/32	drop

```
standard_metadata.egress_spec = 1;
hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
```

- „Matching a packet onto a MAT“
  - Specified fields of the packet are compared with key(s) of table entries
  - If a matching entry is found, corresponding action is executed
- A MAT can be applied only once per packet!

Reminder:

```
action forward(egressSpec_t port) {
    standard_metadata.egress_spec = port;

    // decrement time to live (ttl)
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}

table ipv4 {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        forward;
    }
}
```

Control Blocks (II)

- Control Blocks contain program logic, e.g.,
  - Match-action tables
  - Conditions
  - …
- Control Blocks can be encapsulated
  - Call with `.apply(..)`

```
#include "IPv4.p4";

control ingress(inout headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {
  IPv4() ipv4_c; // ipv4 control unit

  apply {
    // its a ipv4 packet
    if(hdr.ethernet.etherType == TYPE_IPV4) {
      // apply ipv4 control
      ipv4_c.apply(hdr, meta, standard_metadata);
    }
  }
}
```

Resource definitions

Apply block

```
control IPv4(inout headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata){

  action forward(egressSpec_t port) {
      // send packet out of specified port
      standard_metadata.egress_spec = port;

      // decrement time to live (ttl)
      hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
  }

  // table for ipv4 unicast match
  table ipv4 {
      key = {
          hdr.ipv4.dstAddr: lpm;
      }
      actions = {
          forward;
      }
  }

  apply {
    ipv4.apply();
  }
}
```

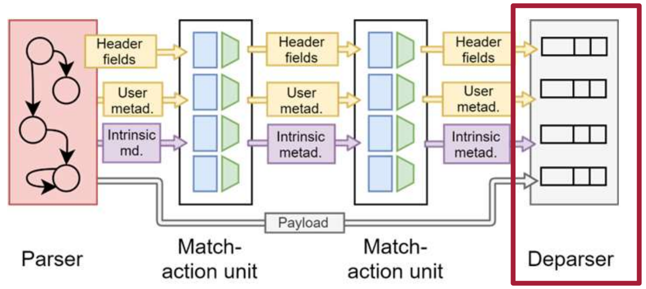
P4 Actions

- P4 `actions`
  - Similar to functions in other programming languages
  - Not only tied to MATs
  - Available programming constructs
    - Variables (only visible within the `action`)
    - Many standard arithmetic and logical operations
      - +, -, *, ~, &, |, ^, >>, <<, ==, !=, >, >=, <=
    - Non-standard operations: bit-slicing and bit concatenation
- An action can be applied only once per packet!

```
action swap_mac(mac_addr_t src, mac_addr_t dst) {
  mac_addr_t tmp = src;
  src = dst;
  dst = tmp;
}

apply {
  …
  swap_mac(hdr.ethernet.srcAddr, hdr.ethernet.dstAddr);
  …
}
```

Deparser

- Serializes headers back into a well-formed network packet
  - `Emit` packet headers
  - Order is relevant
  - Only valid headers are added
    - During processing, headers may be added with `.setValid()` or removed with `.setInvalid()`
    - `.isValid()` to check if header is valid
    - Extracted headers in the parser are automatically marked as `valid`




```
control deparser(packet_out packet, in headers hdr) {
  apply {
    packet.emit(hdr.ethernet);
    packet.emit(hdr.ipv4);
    packet.emit(hdr.igmp);
  }
}
```

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

P4 Extern Objects

- Externs extend core P4 functionality
  - P4 specification defines certain mandatory externs, e.g., registers, parsing, cloning, counters, …
  - Other externs defined by target
    - E.g., traffic generator in Intel Tofino switching ASIC

- `extern` describes set of methods but not the implementation!
  - Similarity: abstract class in an object-oriented language
  - Example: incremental checksum unit

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

The Register Extern

- Metadata is per-packet and discarded after processing

- How to implement stateful algorithms?
  - → Register extern
  - A packet can trigger reading from / writing a value into a register

- Extern: Implementation is target-specific!
  - The v1model architecture provides a `read` and `write` function
  - Other targets allow custom register actions

Data type of stored values

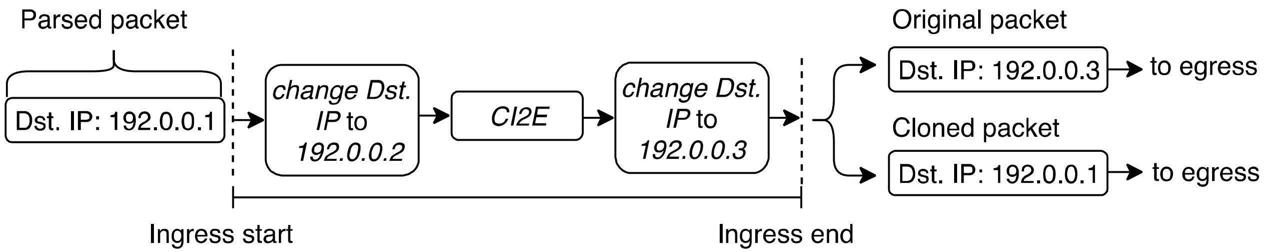
Register size

Index in register

```
register<bit<16>>(16) stateful_register;

...
apply {
    stateful_register.read(value_stored_here, 0);
    stateful_register.write(0, value_to_write);
}
```

Packet Cloning

- What to do if we need a copy of a packet, e.g., for 1+1 protection?
  - Clone-Ingress-to-Egress (CI2E)
    - Cloned packet does not contain modifications from ingress
  - or Clone-Egress-to-Egress (CE2E)
    - Cloned packet contains modifications from ingress



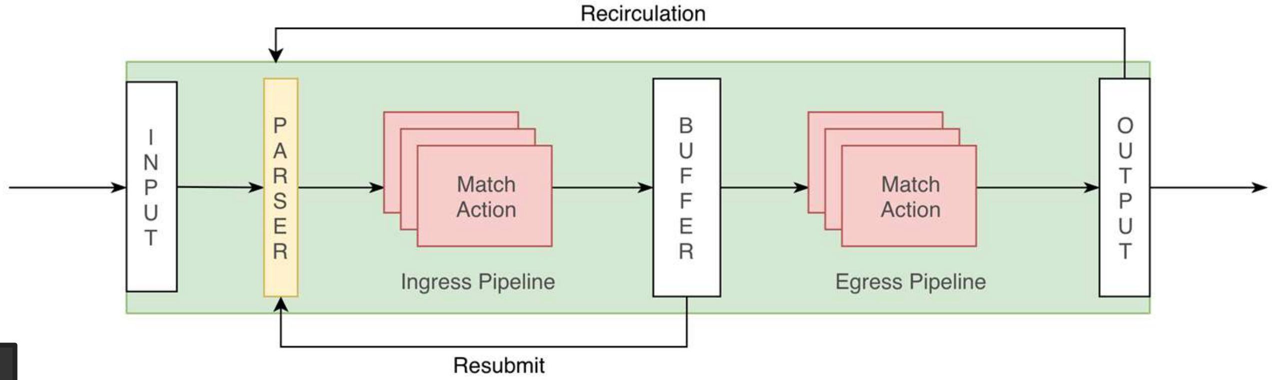

```
apply {
    if (meta.clone == 1) {
        clone(CloneType.E2E, meta.sessionId);
    }
}
```

Clone type, here Egress-to-Egress

Clone session, needs configuration from control plane

Recirculation

- There are no loops in P4!
  - How to implement iterative algorithm? → Recirculation / Resubmit



Need to configure a recirculation port

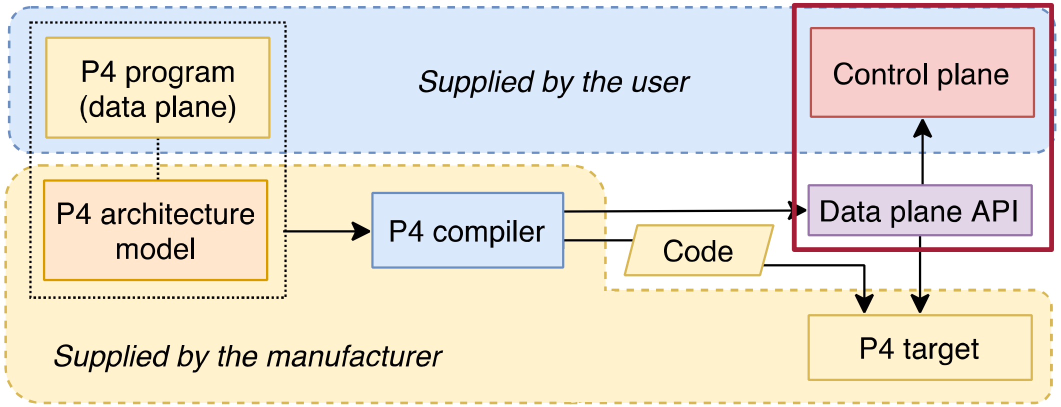
```
// if its an IPv4 packet, recirculate
if(hdr.ethernet.etherType == TYPE_IPV4) {
  recirculate<metadata>(meta);
}
```

P4 Switch

- Switch.p4
  - Connects all components

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

#include "src/headers.p4"
#include "src/parser.p4"
#include "src/ingress.p4"
#include "src/egress.p4"

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

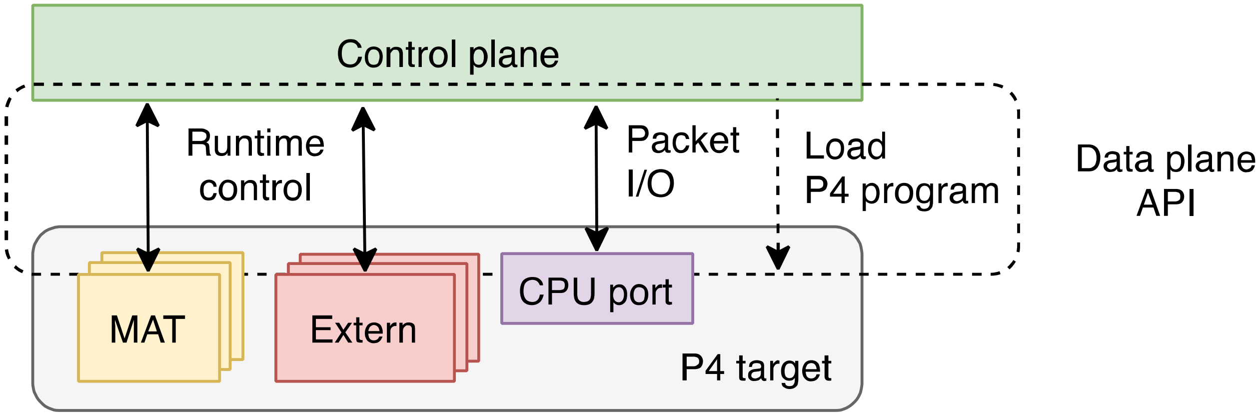
- Import of the switch architecture
- <v1model.p4>: bmv2 (your target switch for hackathon)



THE CONTROL PLANE

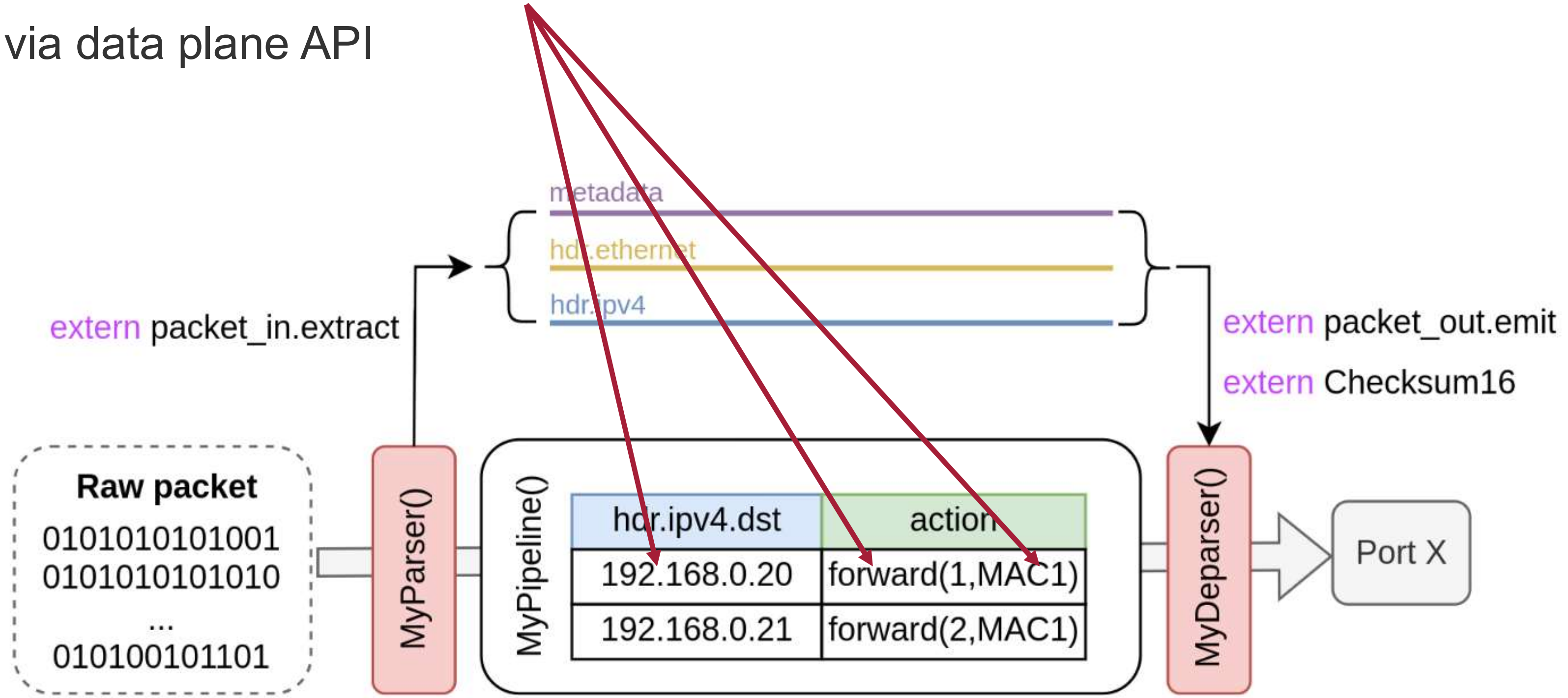
P4 Control Plane (I)

- Control plane manages the runtime behavior of P4 targets via data plane APIs
- Data plane API is provided by a device driver or software component
  - Exposes data plane features in a well-defined way
  - If data plane feature is not exposed, it cannot be used by the control plane
- P4 targets may be used without a control plane with static MAT entries



P4 Control Plane (II) – Recap

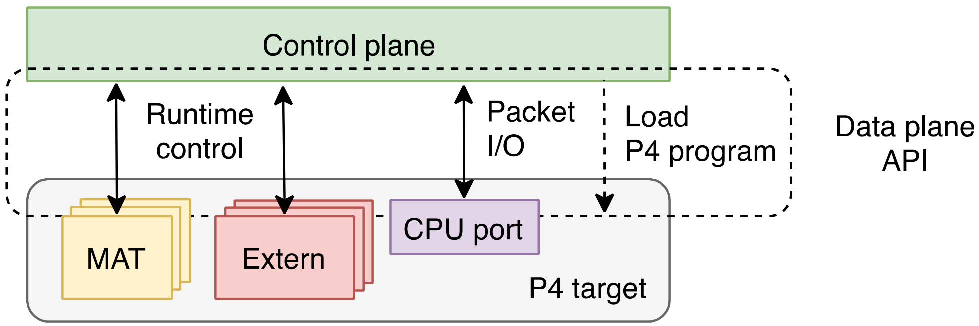
- Example
  - Data plane defines the table structure
  - Control plane fills those tables with entries
- Communication via data plane API



P4 Control Plane

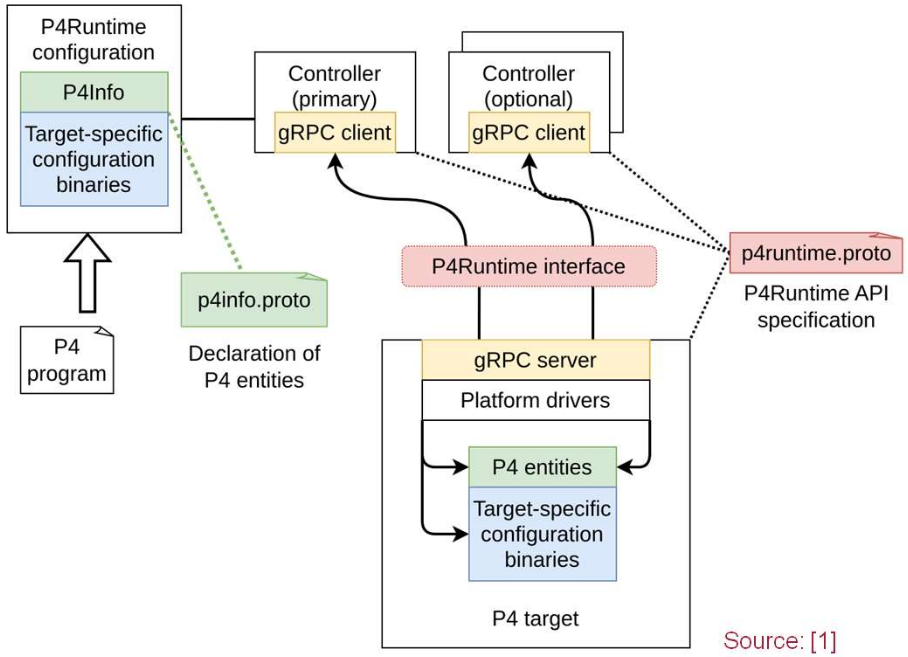
- Different approaches and control planes for runtime control of P4 switches
  - SDN controller, CLI, …
- Crucial: API between control plane and data plane required
  - P4 compiler auto-generated runtime APIs
    - Program-dependent
  - BMv2 CLI
    - Program-independent, but target-specific
    - Control plane not portable!

⇒P4 Runtime: *-independent API



P4 Runtime (I)

- Framework for runtime control of P4 targets
  - Standardized gRPC communication
    - `p4runtime.proto` defines messages and semantics (part of P4 runtime standard)
    - P4 targets include a gRPC server
    - Controller implements gRPC client
- P4 compiler generates `p4info.proto` file from P4 program
  - Contains all accessible P4 entities (MATs, Externs, …)
- P4 *-independent
  - Not restricted to specific data plane protocols
  - Target manufacturer ensures compatibility
  - API doesn’t change with the P4 program



Source: [1]

P4Runtime (III)

- In Scope
  - Runtime control of P4 built-in objects and PSA externs
    - MATs, registers, …
  - In-the-field device-reconfiguration with a new P4 data plane
    - Dynamically load a new P4 program on a switch during runtime
- Not in Scope
  - Runtime control of elements outside the P4 language
    - e.g., ports, traffic management, etc.
  - Protobuf message definition for non-PSA externs

Alternative Data Plane APIs

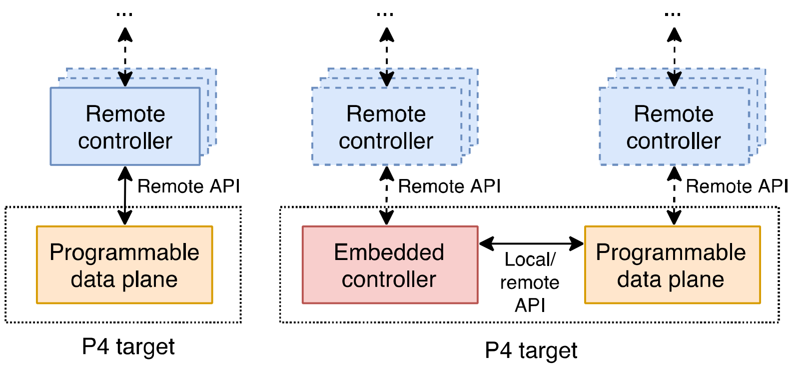
- Barefoot Runtime Interface (BRI)
  - BRI consists of two independent APIs available for Tofino-based P4 hardware targets
  - BfRt API: local control including C, C++ and Python bindings
  - BF Runtime: based on gRPC framework and protobuf (similar to P4Runtime)
- BM Runtime API
  - Program independent data plane API for bmv2
  - Based on Thrift RPC

API	Program independence	Control plane location
P4Runtime	✓	Remote (gRPC)
BF Runtime	✓	Remote (gRPC)
BfRt API	✓	Local (C, C++ and Python bindings)
BM Runtime	✓	Remote (Thrift RPC)

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Controller Use Case Patterns

- ▶ Embedded/Local Controller
  - P4 hardware targets comprises / are attached to a computing platform
  - Running controller directly on the P4 target
  - ➔Fast interaction and updates



(a) Remote controllers

(b) Local/embedded controller + remote controllers

- ▶ Remote Controllers
  - Typical SDN setup
  - Hybrid control planes might be used
    - Local tasks, e.g., MAC learning, port monitoring, done by embedded controller
    - Global tasks, e.g., routing, done by remote controller

A Universal Control Plane and GUI for P4

UNICORN-P4

P4 Development Workflow

The developer …

❶ … writes a P4 program

❷ … compiles it for the target switch

❸ … implements the corresponding control plane

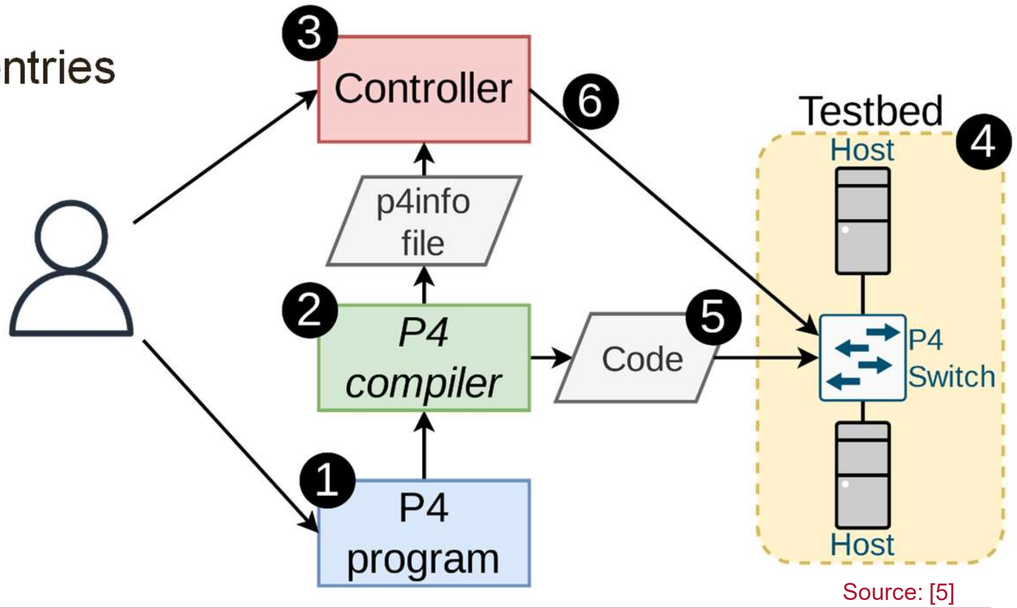
❹ … sets up a (virtual) testbed for validation

❺ … loads the P4 program onto the switches

❻ … configures the control plane to write MAT entries in the data plane

→Focus should be on P4 developing!

→Not on control plane or testbed emulation

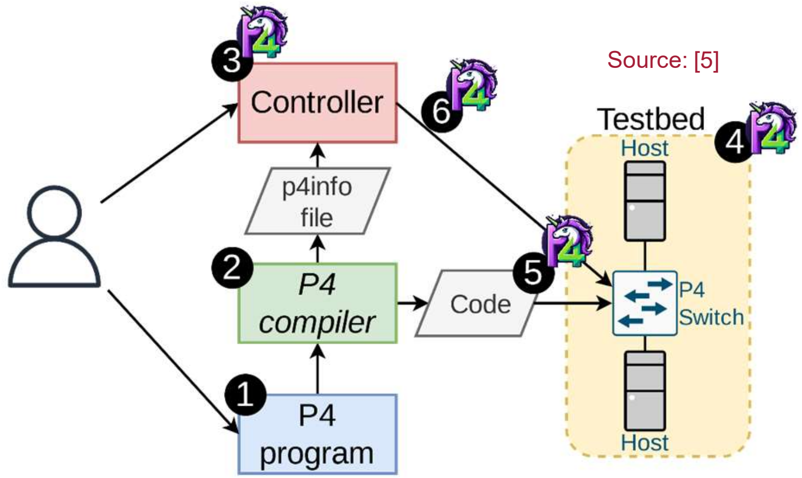


Source: [5]

UniCorn-P4

- ► UniCorn-P4 simplifies the development process
  - Universal control plane using the P4 runtime data plane API
  - Web GUI to configure MAT entries
- ► P4info file (compilation artifact) can be loaded in the web GUI
  - UniCorn-P4 automatically detects available MATs and actions from the P4 program
  - MAT entries can be added, modified, and deleted in the frontend
  - P4 program can be loaded onto multiple switches in the network
    - Communication via P4 Runtime + gRPC
- ► Network topologies can be specified as .json file
  - Starts up a virtualized Mininet network testbed
  - Each switch can be programmed individually
- ► UniCorn-P4 keeps a history of previous configurations
  - Load P4 programs and MAT entries from history

Source: [5]



EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

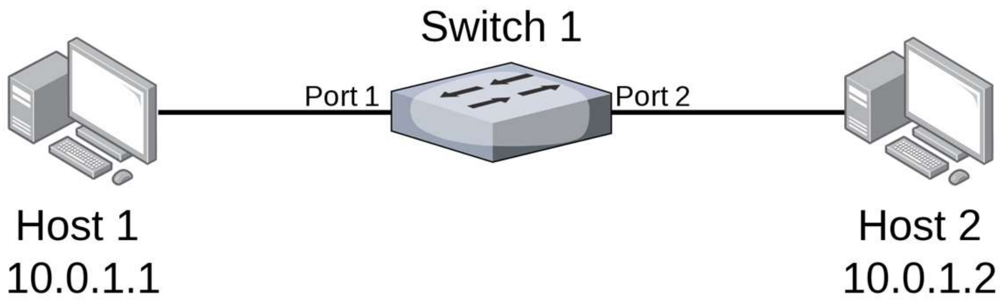
P4 TUTORIAL

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Example Setting

- Host 1 must reach Host 2 via ICMP ping

- Given data plane: P4 program
  - Header
  - Parser
  - Ingress: Table + Action
  - Egress
  - Deparser



- Given control plane: UniCorn-P4
  - Writes table entries for packet forwarding

- In the tutorial, you will learn …
  - how to operate with the data plane and the control plane
  - how to debug
  - how to implement a new protocol

Getting started

1. Start the UniCorn-P4 control plane
   - Navigate to `/home/p4/UniCorn-P4/docker`
   - Run `sudo docker compose up`
   - Navigate to `http://localhost:3000` in the web browser

2. Write your P4 program in VSCode
   - Place the project in a subfolder in `/home/p4/UniCorn-P4/p4-files`
     - Already done for you

3. Compile your P4 program
   - P4c is bundled in the UniCorn-P4 backend container

   i. Enter a shell in the container: `sudo docker exec -it backend bash`

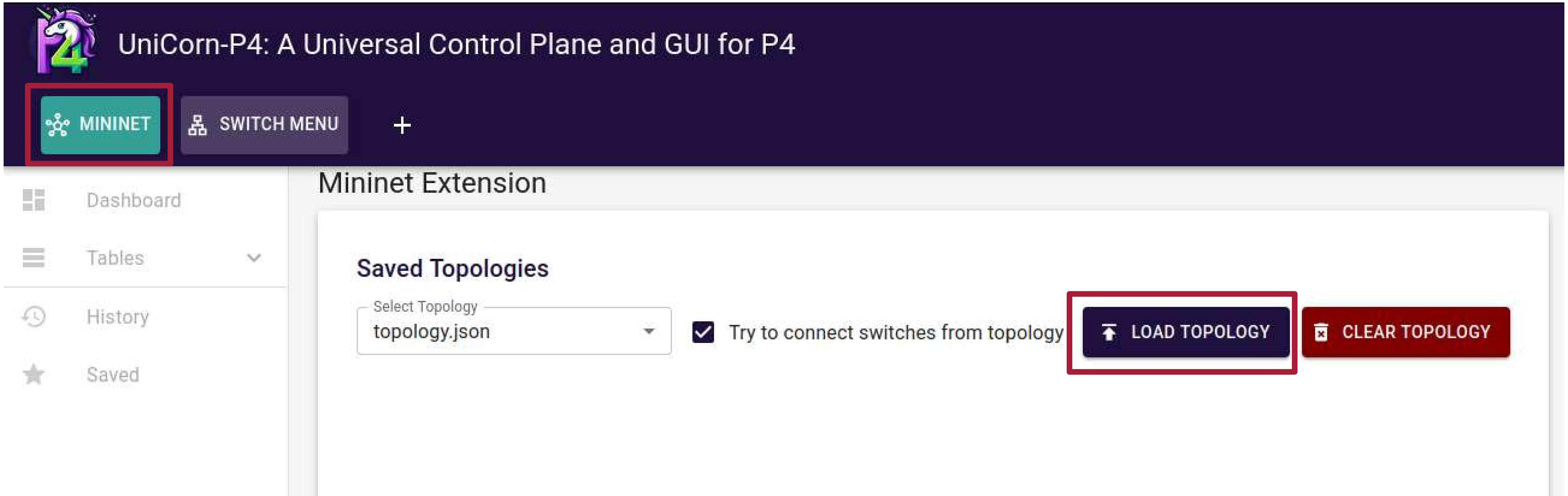
   ii. Enter your project folder: `cd /p4/basic/<project>`

   iii. Compile the P4 program: `p4c p4_tutorial.p4 --target bmv2 --arch v1model --p4runtime-files p4_tutorial.p4info.txt -o .`

Getting started – Working with UniCorn-P4

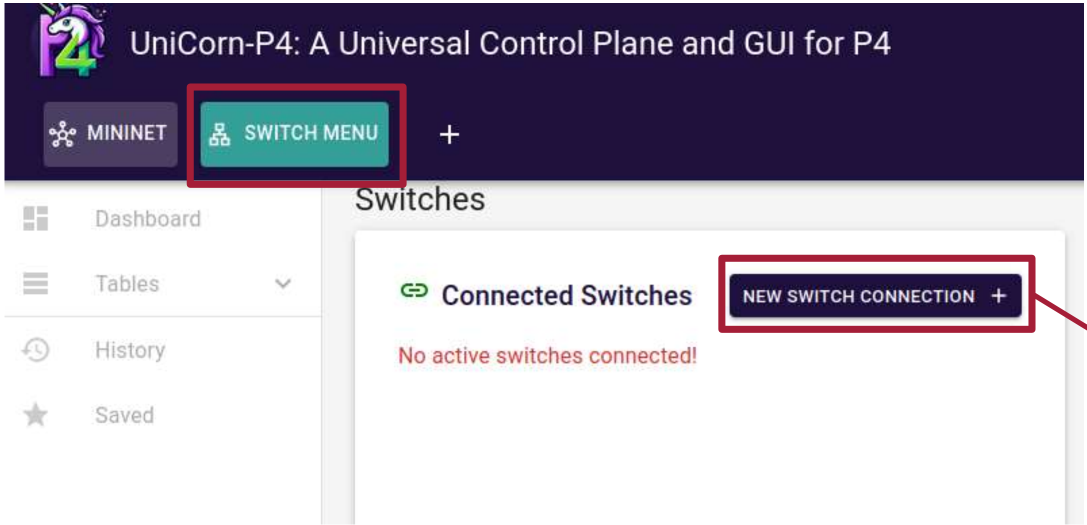
4. Adapt topology.json to your needs.
   - Located in `/home/p4/UniCorn-P4/netsim/topology*.json`
   - Mininet automatically assigns IP adresses

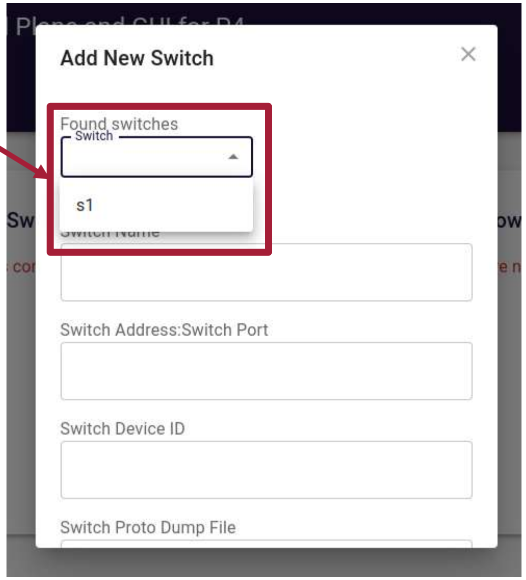
5. Load the topology in the UniCorn-P4 GUI



Getting started – Working with UniCorn-P4

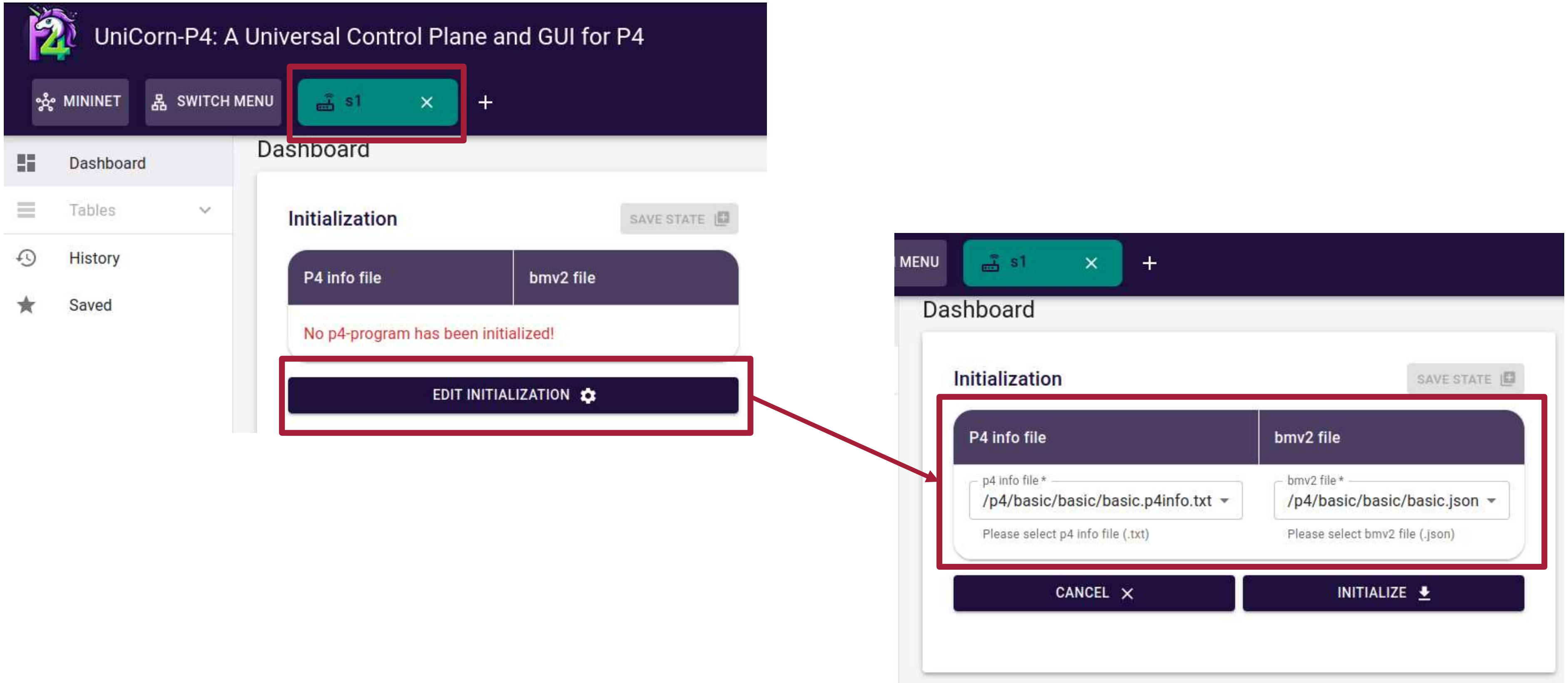
6. Connect to the switches in the UniCorn-P4 GUI





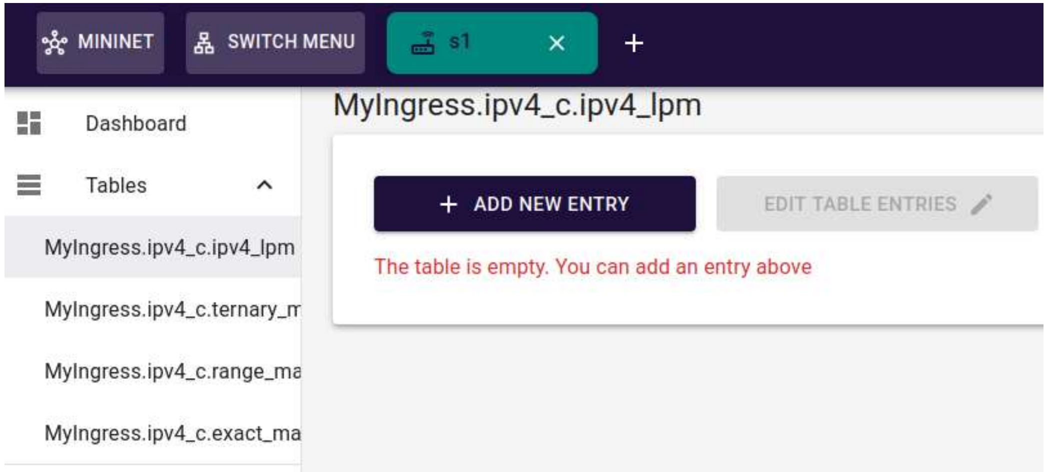
Getting started – Working with UniCorn-P4

7. Load the P4 program onto each switch



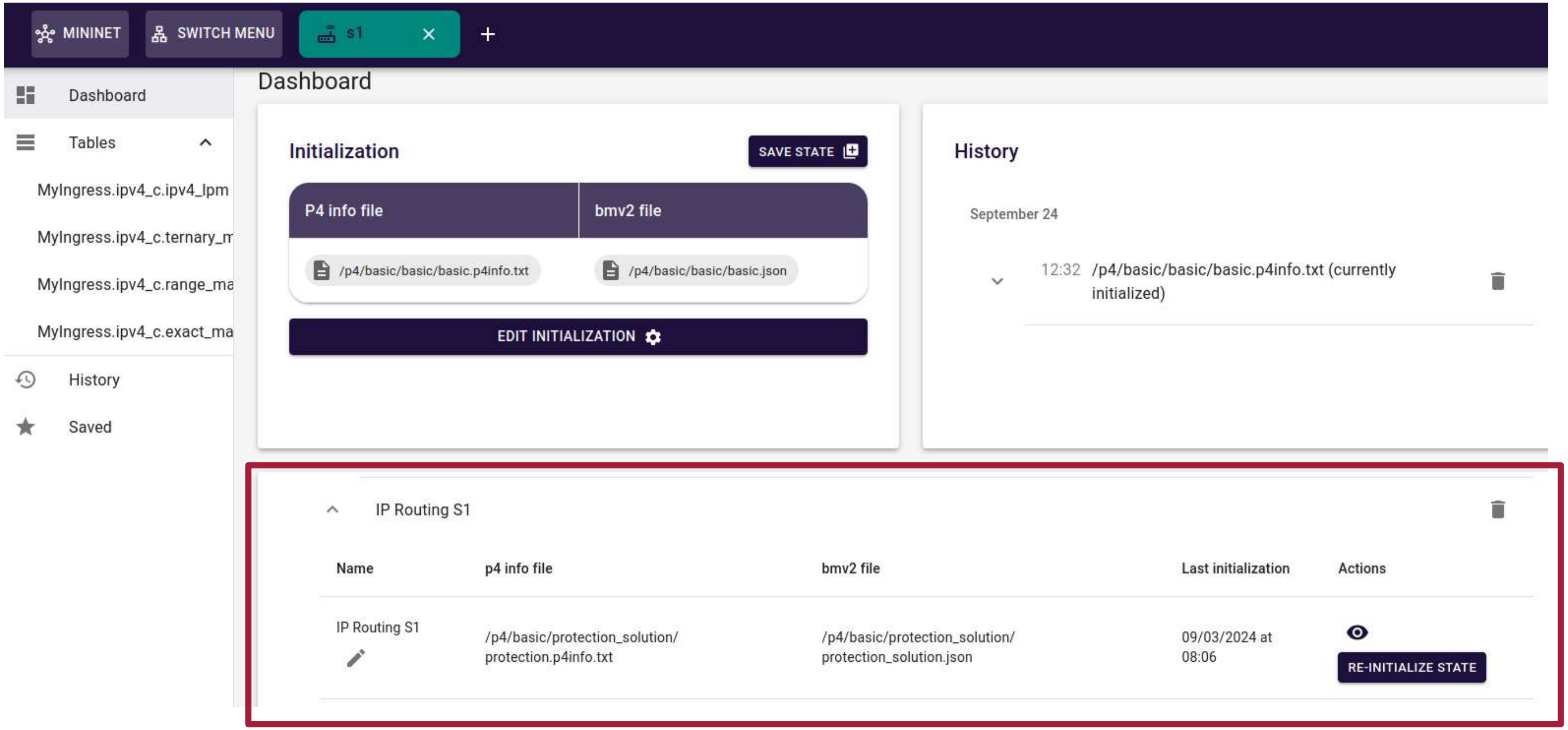
Getting started – Working with UniCorn-P4

8. Write table entries in the UniCorn-P4 GUI



Getting started – Working with UniCorn-P4

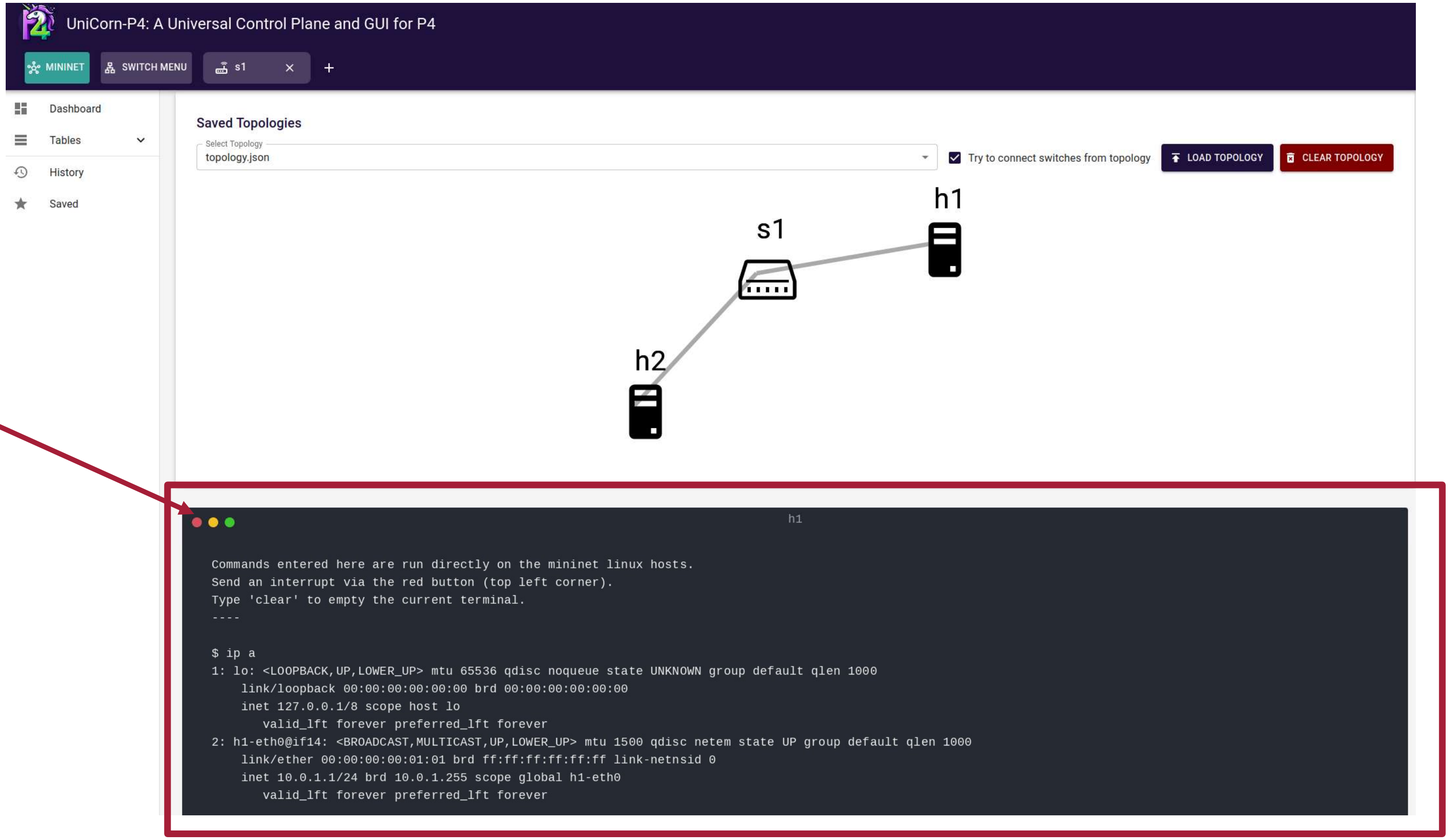
9. Alternative to step 7 and 8: Load a state including a P4 program and table entries onto a switch from „Saved“



Getting started – Working with UniCorn-P4

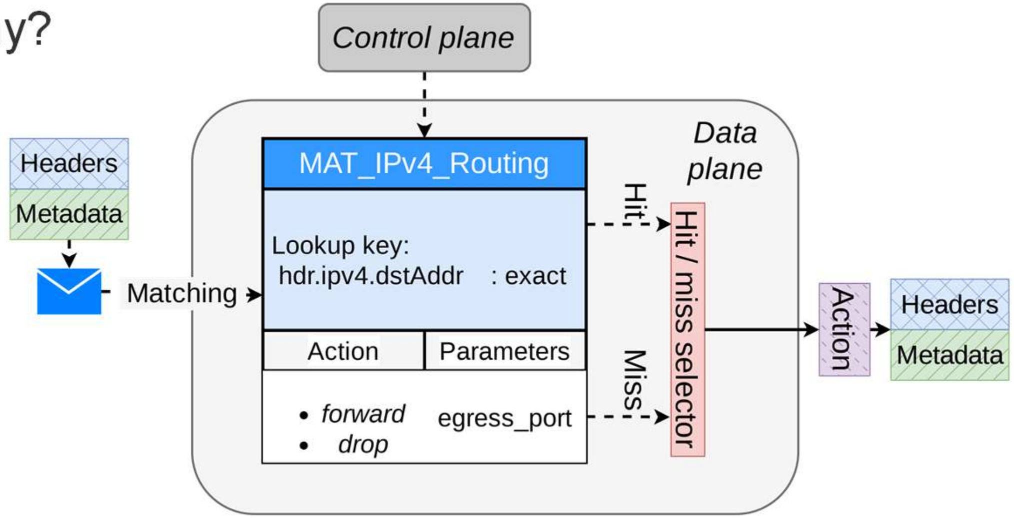
10. Access the host terminals

Interrupt



Debugging

- `ping 10.0.1.2` from Host 1 does not work. Why?



- Check switch log file `netsim/logs/s1.log`
  - `IPv4.isValid()` is false. Why?
- Use wireshark or tcpdump
  - ARP packets are not forwarded!
- ARP handling must be implemented
  - Simplified, hardcoded entries, no „real“ ARP handling
    - ARP request from Port 1 will be sent to Port 2 and vice versa

Implementing a new protocol (ARP)

- Steps to implement a new protocol
  - Data Plane
    1. Define EtherType ARP
    2. Define ARP header
    3. Add ARP to header stack
    4. Adapt the parser state machine
    5. Add parser state
    6. Add Match-Action logic
    7. Call Match-Action logic
    8. Deparse ARP header
  - Recompile the program and load it onto the switch
  - Control plane
    1. Write Match-Action table entries

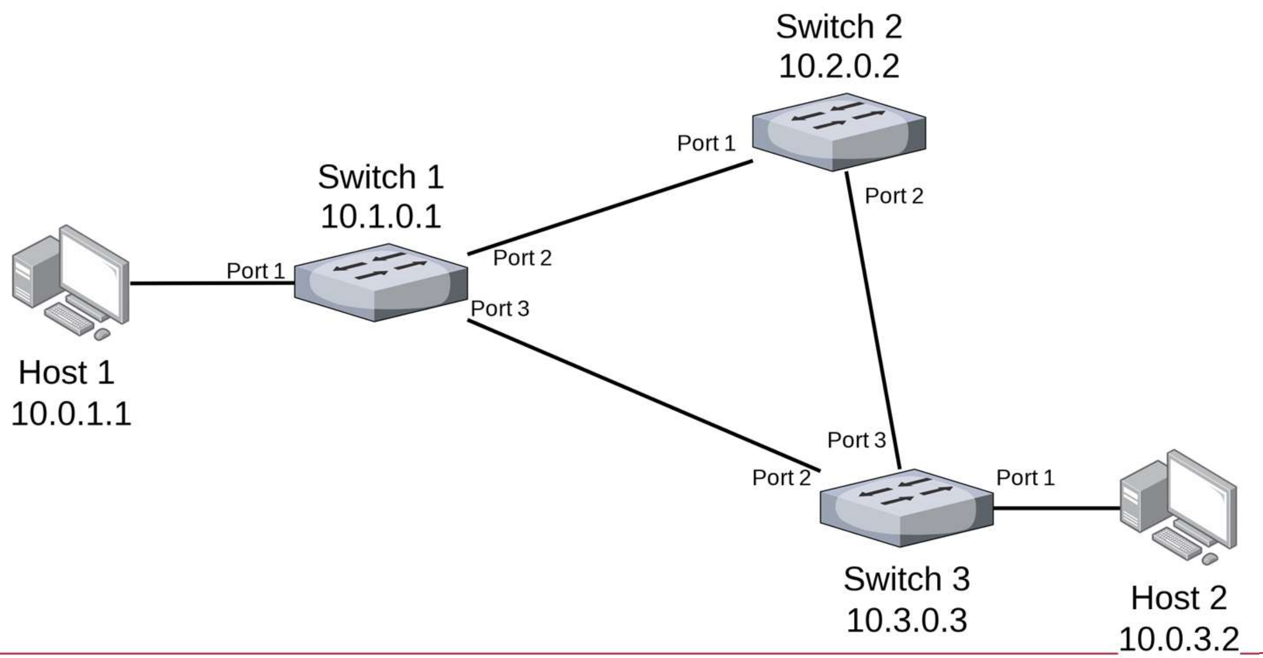
EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Simplified, wired 1+1 protection in P4

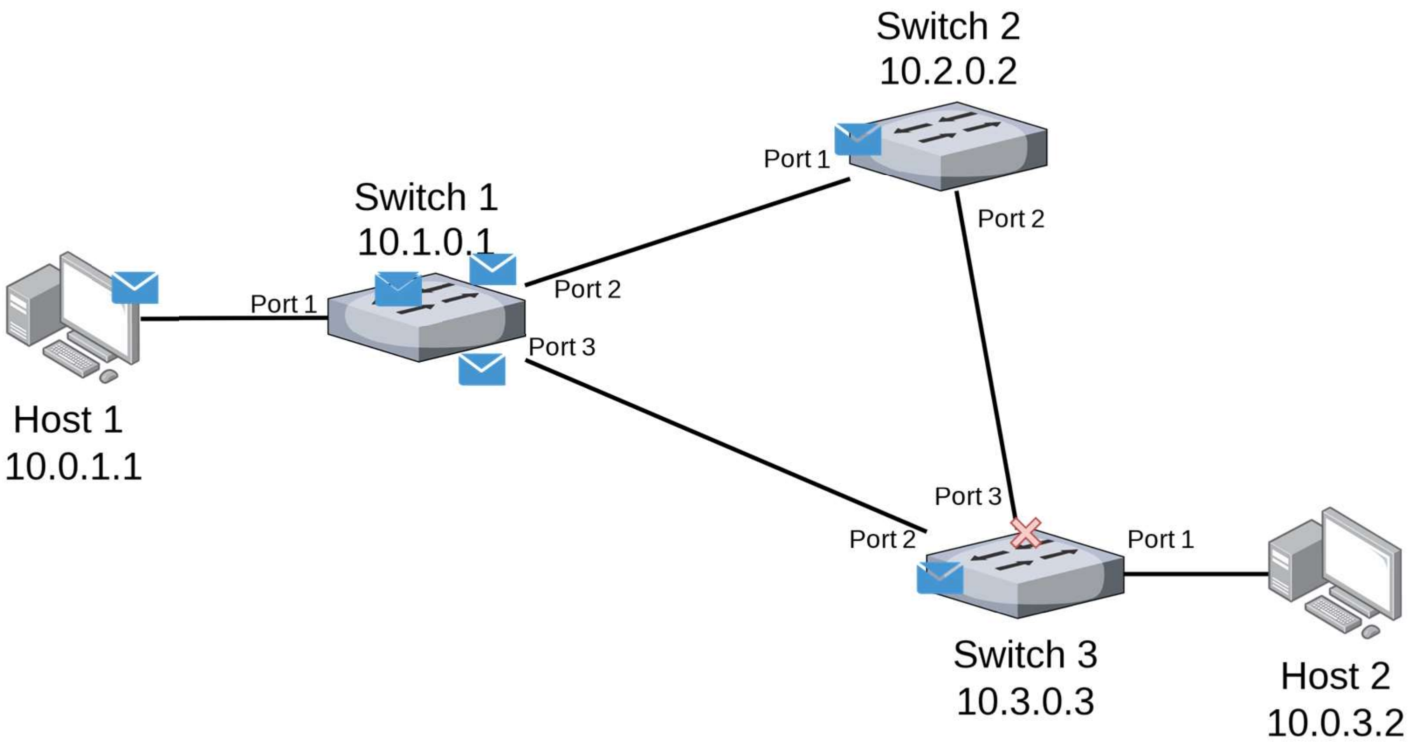
HACKATHON

1+1 Protection – Concept

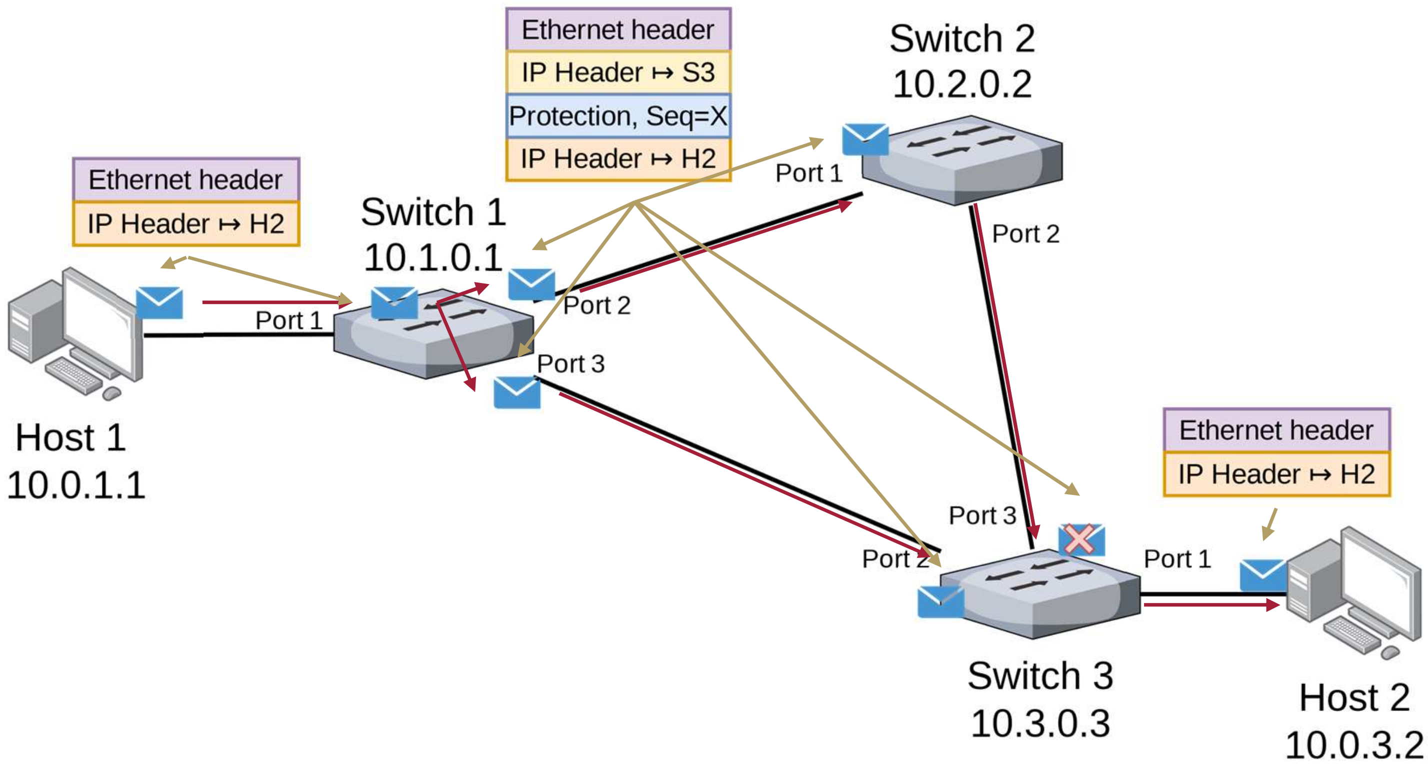
- Sender (duplication node) duplicates traffic and forwards it over disjoint paths
- Receiver (deduplication node) forwards only the first copy received and drops the other
- → On a failure of one link, no interruption in forwarding!



1+1 Protection – Concept (animated)



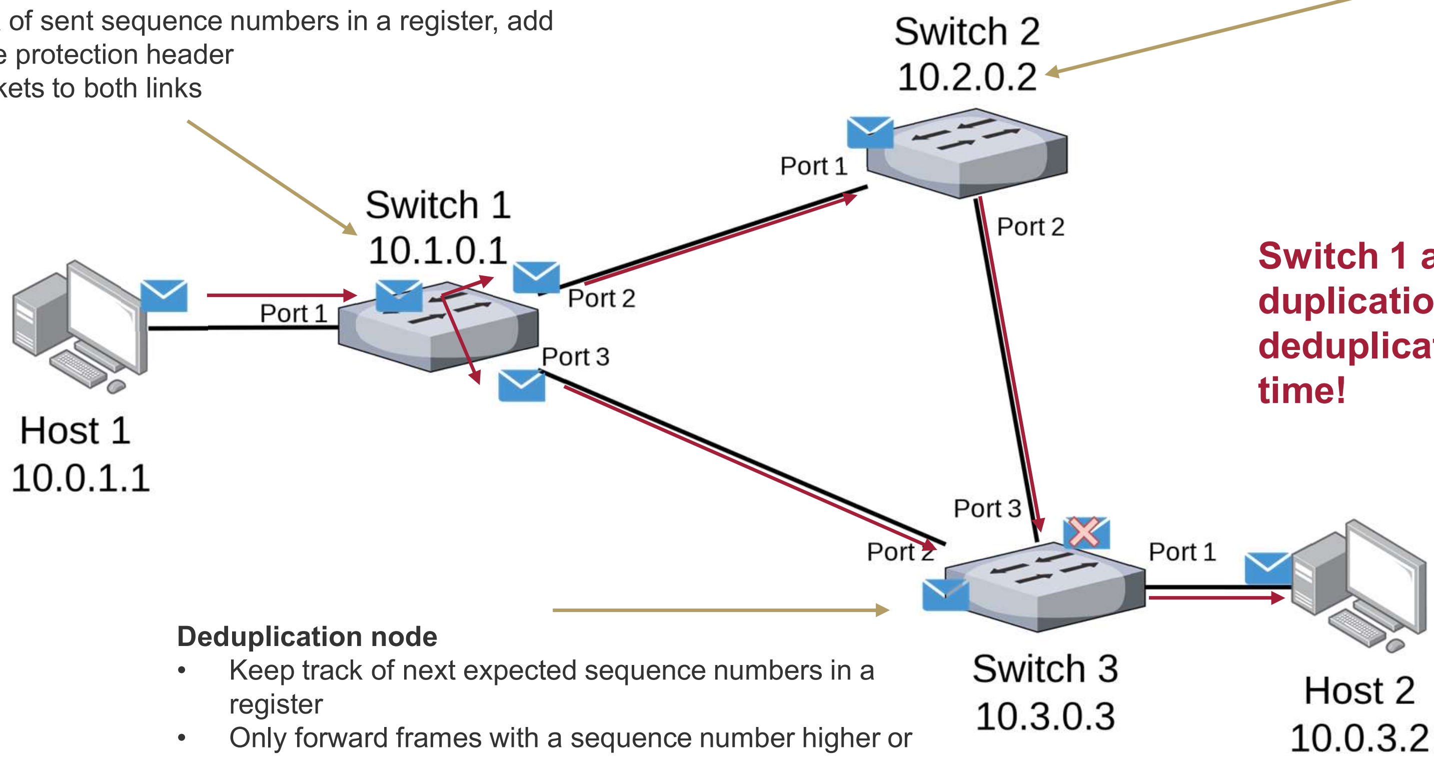
1+1 Protection – Concept



1+1 Protection – Concept

Duplication node
- Encapsulate them with an IP tunnel header and a protection header
- Keep track of sent sequence numbers in a register, add them to the protection header
- Clone packets to both links

Forwarding node
- Does normal IPv4 LPM forwarding



Switch 1 and Switch 3 are duplication nodes and deduplication nodes at the same time!

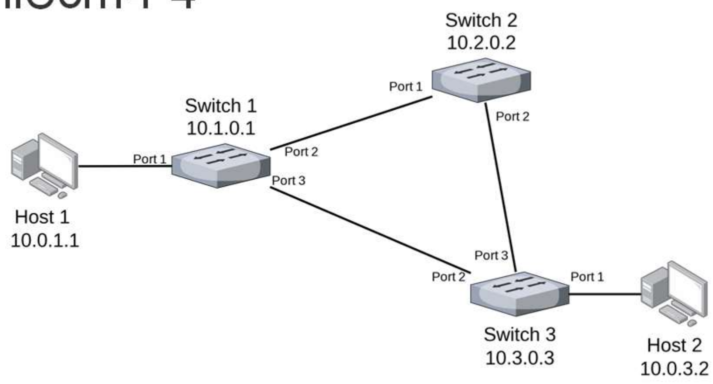
Deduplication node
- Keep track of next expected sequence numbers in a register
- Only forward frames with a sequence number higher or equal to the expected sequence number
- Drop other frames (duplicates)
- Remove protection and IP tunnel header

Hackathon VM – Getting started

- Install a virtualization environment
  - VirtualBox 7: https://www.virtualbox.org/wiki/Downloads (platform independent)
  - Mac users may use the .qcow2 file and UTM
- Download the .ova file shared in the Nextcloud with you and import it
  - In VirtualBox: „File“ → „Import Appliance“
    - 2 CPUs and 4096 MiB RAM
    - Start the VM
      - username p4, password resilience
- Open VSCode in the virtual machine and navigate to `/home/p4/UniCorn-P4/docker` in a terminal
- Start UniCorn-P4 with `sudo docker compose up`
- Navigate to `http://localhost:3000` in a web browser

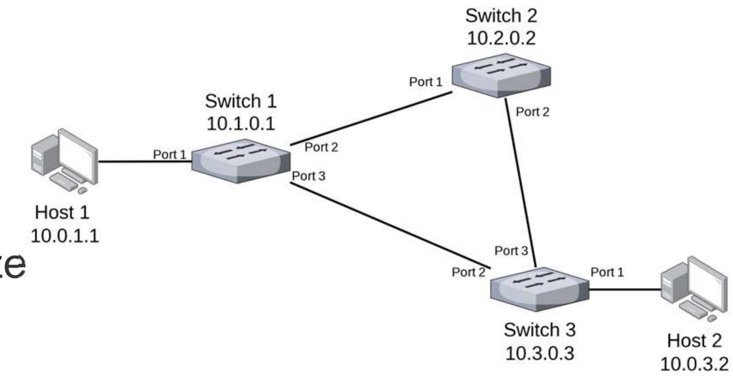
1+1 Protection – Getting started

1. Load the given topology file `topology_protection.json` in UniCorn-P4
   - Navigate to http://localhost:3000 after starting UniCorn-P4
   - Click on the „Mininet“ tab
   - Select the topology file in the dropdown menu and load it
   - IP addresses and ARP handling are already configured



1+1 Protection – Getting started

2. Compile and load the template code from the protection folder in `UniCorn-P4/p4-files` in UniCorn-P4 onto each switch
   - Compile your code as described on slide 54
   - Click on the switch tab „s1“, „s2“, etc. in UniCorn-P4
   - Click on „Edit Initialization“
   - Select the .p4info.txt and the protection.json file and click on initialize
   - Do this for every switch
   - You have to repeat this every time you recompile your P4 program

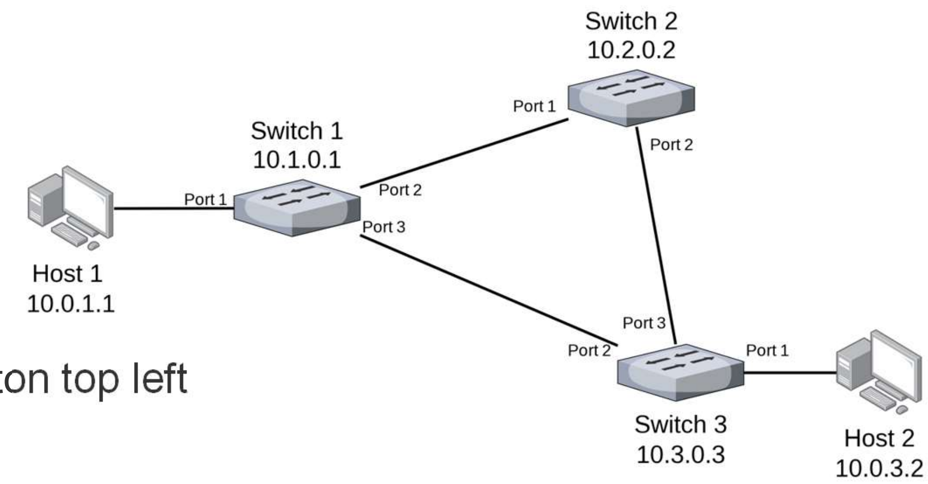


   - The given code implements IPv4 LPM routing
     - Lookup the IPv4 destination in a MAT and execute the forward action
       - Set the egress port
       - Set the ethernet source address to the destination address
       - Set the destination ethernet address to the next hop
       - Decrement TTL
     - The egress port, and ethernet addresses are provided by the control plane (UniCorn-P4)
     - ARP handling is not necessary

1+1 Protection – Getting started

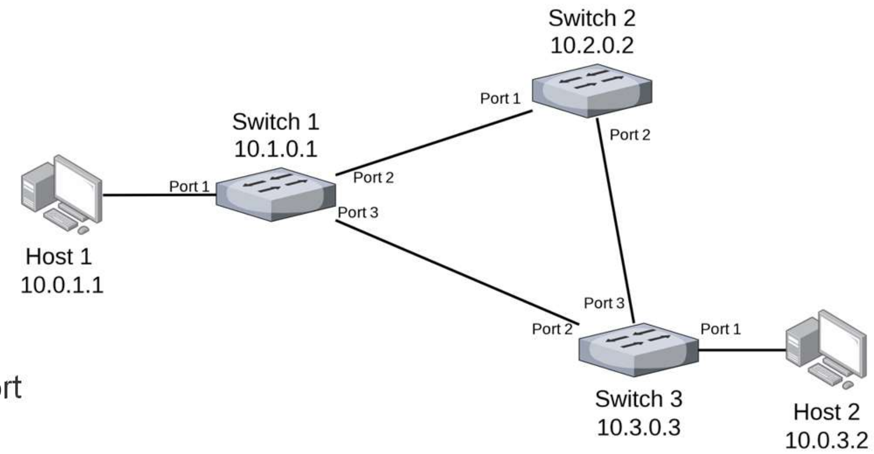
3. Set up IPv4 LPM forwarding entries so that H1 can reach H2 via the `ping` command
   - Without 1+1 protection
   - You can load the table states for each switch in UniCorn-P4 under „Saved“
     - Click on a switch tab
     - Scroll down below the initialization
     - Select the corresponding protection state from the list of saved states and load it with „Re-initialize state“
     - Make sure to select the correct switch state for the current switch
     - This also reloads the P4 program onto the switch as done in step 2

► Ensure that H1 can reach H2 before you continue!
- Click on the Mininet tab
- Scroll down to see the terminals
- Start pinging from the host1 terminal: `ping 10.0.3.2`
- Send an interrupt to the terminal by clicking the red button top left



1+1 Protection – Implementation

- Implement 1+1 protection between s1 and s3
  - Set up packet mirroring (cloning)
    - Run the following in the terminal in VSCode
    - `sudo docker exec -it netsim bash`
      - s1
        ```
        simple_switch_CLI --thrift-port 9090
        mirroring_add 1 2
        ```
        Mirror session ID → 1; Traffic mirrored to this port → 2
      - s3
        ```
        simple_switch_CLI --thrift-port 9092
        mirroring_add 3 3
        ```
    - In your P4 program, use this session ID with the clone extern to clone packets to the configured port



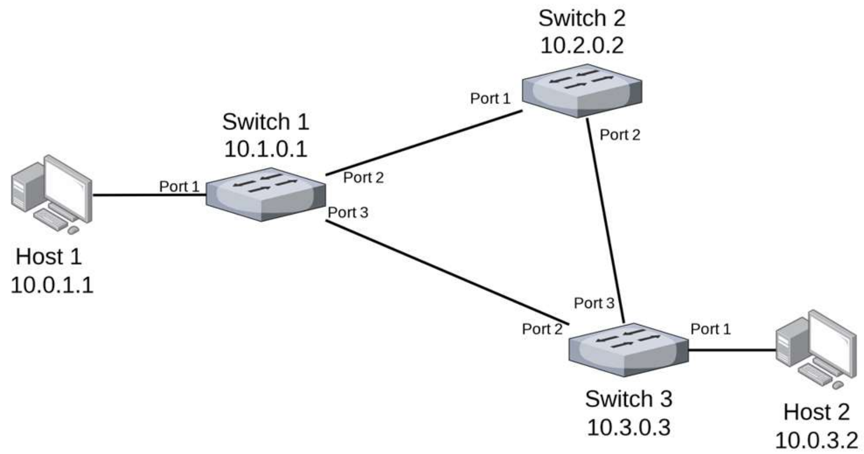
1+1 Protection – Implementation

- Every packet between s1 and s3 destined for H1 and H2 should be forwarded between s1 and s3 using both paths
  - s1 (duplication node) duplicates packets for H2 and sends the packets to s2 and s3 (de-duplication node)
  - Use IPv4 tunnels for the disjoint paths to address the de-duplication node
  - Build your own protection header with sequence numbers.
    - It might contain further fields, e.g., a protocol field to enable flexible parsing

- Both directions H1<->H2 and H2<->H1 should be protected
  - Switch 1 and Switch 3 are duplication nodes and deduplication nodes at the same time!
    - You still implement only one P4 program that runs on all switches
    - If a packet should be encapsulated or decapsulated can be determined from the IP destination address

1+1 Protection – Verification

- Verify if your protection works
  - Ping between H1 and H2 from the terminal in UniCorn-P4
  - No duplicates allowed between H1<->s1 and H2<->s3
    - Verify with wireshark
  - Kill the `s1<->s3` connection
    - In the VSCode terminal
      - `sudo docker exec –it netsim bash`
      - `simple_switch_CLI --thrift-port 9090`
      - `port_remove 3`
  - → No interruption in pinging



Hints

- Define a protection header and a second IP header in `headers.p4`, add them to the packet header
  - e.g., Ethernet – Outer IP – Protection – Inner IP
  - Adapt `parser.p4` to parse your new header structure based on the `ether_type`
    - Define a protection `ether_type` in `headers.p4`
    - The outer IP header is always parsed, the inner only if the protection type is set
  - Adapt `parser.p4` to emit the new headers in the deparser

Hints

- Define a protection header and a second IP header in `headers.p4`, add them to the packet header
  - e.g., Ethernet – Outer IP – Protection – Inner IP
  - Adapt `parser.p4` to parse your new header structure based on the `ether_type`
    - Define a protection `ether_type` in `headers.p4`
    - The outer IP header is always parsed, the inner only if the protection type is set
  - Adapt `parser.p4` to emit the new headers in the deparser

- Use two registers for sequence numbers in `ingress.p4`
  a. Next sequence number to push (duplication node)
  b. Next sequence number expected (de-duplication node)
  - Both switches need both registers!

Hints

- Implement two MATs to determine if a packet needs to be protected or decapsulated
  - Protection needed: Destination of original packet (outer IP) is a host
  - Deduplication needed: Destination of tunnel header (inner IP) is the deduplication node
  - Fill those tables from the Control Plane, i.e., from UniCorn-P4!
    - Save your table state so you don’t have to enter the new entries on every reload

Hints

- Implement two actions to encapsulate and decapsulate
  - Decapsulate
    - Verify sequence number: `if hdr.protection.seq >= expected_seq forward else drop`
    - Increment expected sequence number in register
    - Copy inner IP header to outer IP header
    - Remove the protection and the inner IP header
  - Encapsulate
    - Parameters of the action, filled in by control plane
      - IP address of tunnel endpoint (s1 or s3)
      - Source IP address of this switch
      - Session ID for packet mirroring
    - Create the protection header and set it valid
      - Fill it with the next sequence number read from the register
      - Increment the register value
    - Copy the original IP header (outer) to the inner IP header and set it valid
    - Rewrite the outer IPv4 header to address the tunnel endpoint

Hints

- Add packet cloning
  - Set the session ID in metadata during the encapsulation action
    - Session ID is configured during set up
    - Session ID is given to the action from the control plane
  - Use the Egress-to-Egress clone extern in `egress.p4` to clone a packet
    - Session ID as parameter

Sources

[1] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth: A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research, (preprint), in Journal of Network and Computer Applications (JNCA), vol. 212, March 2023, Elsevier

[2] https://p4.org/

[3] https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf

[4] P4 16 Language Specification (v.1.2.1," https://p4.org/p4-spec/docs/P4-16-v1.2.1.html, accessed 04-19-2021.

[5] F. Ihle, M. Flüchter, S. Lindner, and M. Menth: UniCorn-P4: A Universal Control Plane and GUI for P4, in KuVS Fachgespräch - Workshop on Modeling, Analysis and Simulation of Next-Generation Communication Networks, Sept. 2024, Würzburg, Germany