http://tiny.cc/aiotwin1709

Hands-On Tutorial Of P4

– The Data Plane Programming Language

Xuanchi Guo @ TU Berlin @ AIoTwin Summer School

Software-Defined Networking (SDN)

Software-Defined Networking (SDN)

A. Emerged in the early 2000s to address limitations of conventional networks.
B. **Separation of control and data planes**
   a. **Data plane:** consists of network devices (switches/routers) responsible for storing, forwarding, and processing data packets.
   b. **Control plane:** protocols used for defining the matching and processing rules of the data plane elements.

Why Do We Need P4?

A. **Fixed Protocol Design**: Specific packet headers and fixed-function switches.
B. **Lack of Flexibility**: Difficult to add new protocols or modify packet formats.

Three Goals of P4 Language

Protocol Independence

- P4 stands for **Programming Protocol-Independent Packet Processing**
- Configure a packet parser
- Define a set of typed match+action tables

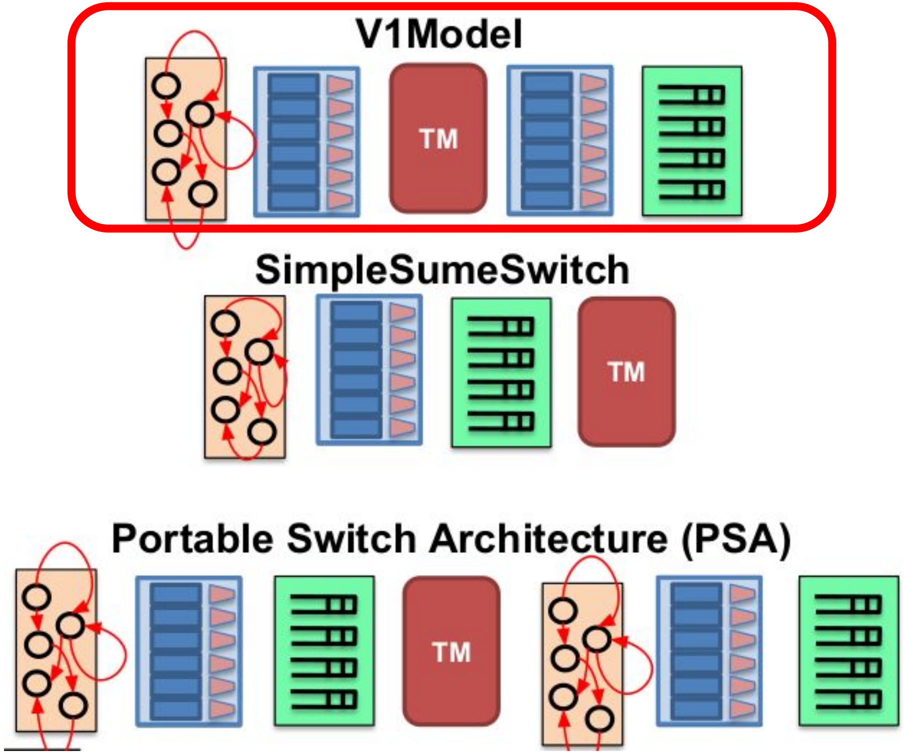
Target Independence

- Program without knowledge of switch details
- Rely on compiler to configure the target switch

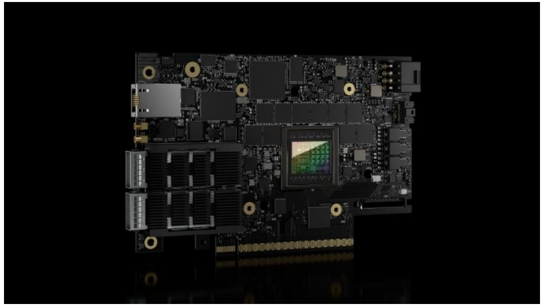
Reconfigurability

- Change parsing and processing in the field

Example Architectures and Targets



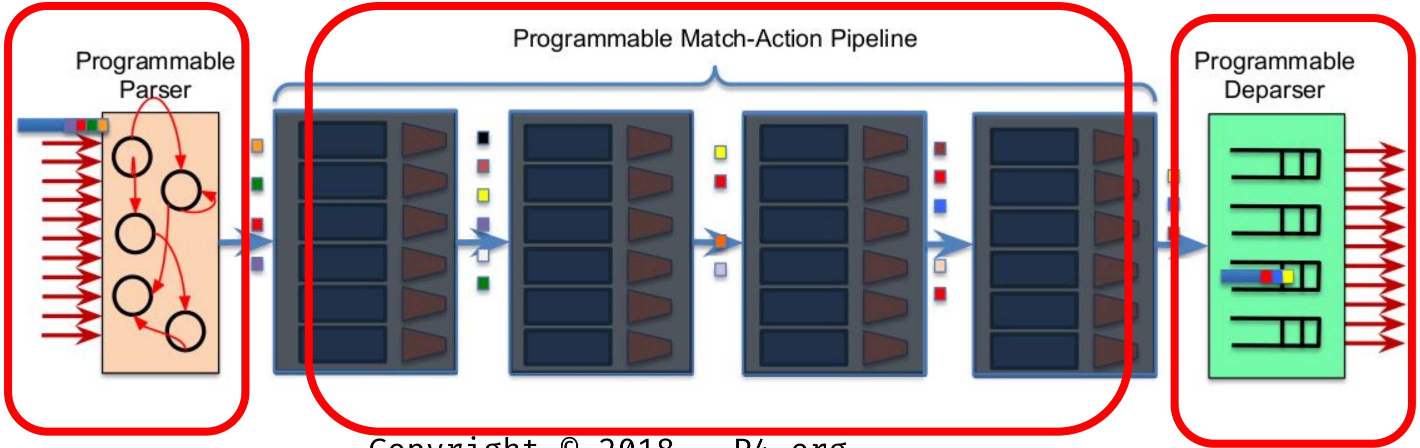
P4-programmable 100Gb bare metal switch with 6.4 Tbps Intel Tofino ASIC



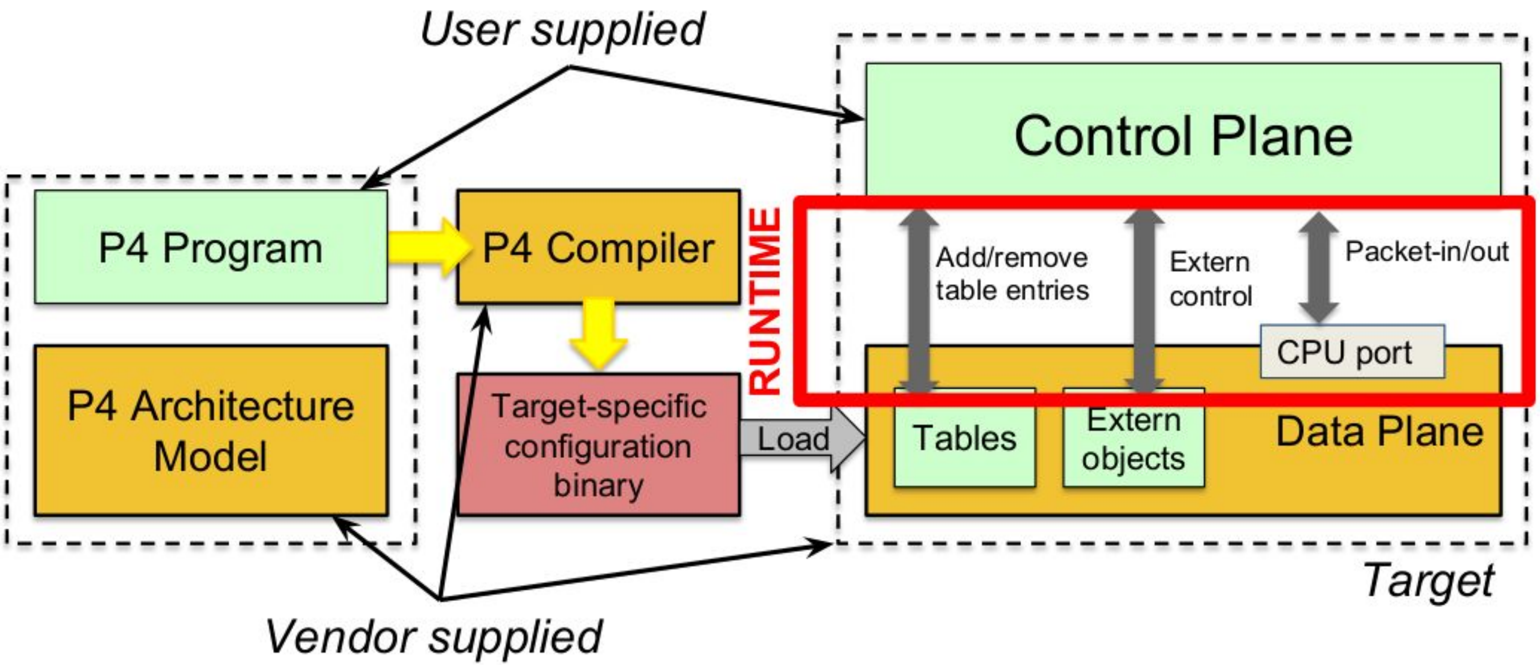
NVIDIA BlueField-3 DPU

P4 Programmable Pipeline

- **Parsers** – Reads incoming packets and extracts headers.
- **Match-Action Tables** – Makes forwarding or processing decisions.
- **Deparsers** – Reassembles and modifies packets before sending them out.



Programming a P4 Target



What you can do with P4

In-band Network Telemetry – INT[1]

Fast In-Network cache for key-value stores – NetCache[2]

Aggregation for MapReduce Applications[3]

In-Network Machine Learning Inference[4](e.g. SmartEdge Usecase)

… and much more

[1]Kim, Changhoon, et al. "In-band network telemetry via programmable dataplanes." ACM SIGCOMM. Vol. 15. 2015.
[2]Jin, Xin, et al. "Netcache: Balancing key-value stores with fast in-network caching." Proceedings of the 26th Symposium on Operating Systems Principles. 2017.
[3]Sapio, Amedeo, et al. "In-network computation is a dumb idea whose time has come." Proceedings of the 16th ACM Workshop on Hot Topics in Networks. 2017.
[4]Zheng, Changgang, et al. "Automating in-network machine learning." arXiv preprint arXiv:2205.08824 (2022).

P4 Primitive Types

P4 is statically-typed; ill-typed programs will be reject by compiler!

P4 p… ibing various kinds of packet data:

- bit<n>: Unsigned integer (bitstring) of size n
- bit is the same as bit<1>
- int<n>: Signed integer of size n (>=2)
- varbit<n>: Variable-length bitstring

P4 Header Formats

- Packet - Headers - *Fields*
- E.g. an Ethernet packet has the following structure

```
+------------+------------+------+-----------
| Destination |    Source    | Type |    Payload
+------------+------------+------+-----------
-
```

P4 Header Formats

- P4 provides a built-in type for headers; syntax resembling *C struct*
- Ordered list of fields
- A field has a name and width
- Can contain bit<n>, int<n>, and varbit<n>
- “dot” notation for a field, e.g: ethernet.dstAddr

```
header IPv4_h {
    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;
    bit<32>       srcAddr;
    bit<32>       dstAddr;
    varbit<320>   options;
}
```

Parsers

- Maps the bits in the actual packet into typed representations
- Behaves like a **state machine**
- Every parser has three predefined states: **start, accept, reject**
- Other states may be defined by the programmer

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

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            0x800: parse_ipv4;  // 0x800 = IPv4
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }
}
```

Match-Action Tables

- In P4, the primary building block for packet processing is the match-action table
- **Action**: Procedure with a sequence of commands.
- Parameters can come from the control plane or within the program.
- Example: Modifying the packet's output port.

```
action set_output_port(bit<9> output_port) {
    std_meta.egress_spec = output_port;
}
```

Match-Action Tables

- Compare packet headers to table entries (matching).
- Execute actions like forwarding, dropping, or modifying packets.
- Example: *next_hop* for ipv4 destination-based forwarding.
- **Key**: hdr.ipv4.dstAddr : *lpm (Longest Prefix Match).*
- **Actions**: *set_output_port, drop.*
- **Default Action**: *drop* for unmatched packets.

```
table next_hop {
  key = {
      hdr.ipv4.dstAddr : lpm;
  }
  actions = {
      set_output_port;
      drop;
  }
  default = drop;
}
```

P4 Controls

```
control C(inout headers_t headers, inout meta_t meta,
            inout standard_metadata_t std_meta) {
  ...
  apply {
     ...
  }
}
```

- **Control**: Similar to C functions (without loops).
- Can declare variables, create tables, etc.
- Functionality specified by code in apply statement.

Example: Simple Actions

```
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;
    }
}
```

Deparsers

```
/* User Program */
control DeparserImpl(packet_out
packet,                   in headers hdr) {
    apply {
        ...
        packet.emit(hdr.ethernet);
        ...
    }
}
```

Assembles the headers back into a well-formed packet

P4 Program Template(V1Model)

```
#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;
```

“Tell me and I forget,
teach me and I remember,
involve me and I learn.”

—Benjamin Franklin & Confucius

Hands-On Session

Good To know Before P4 Coding

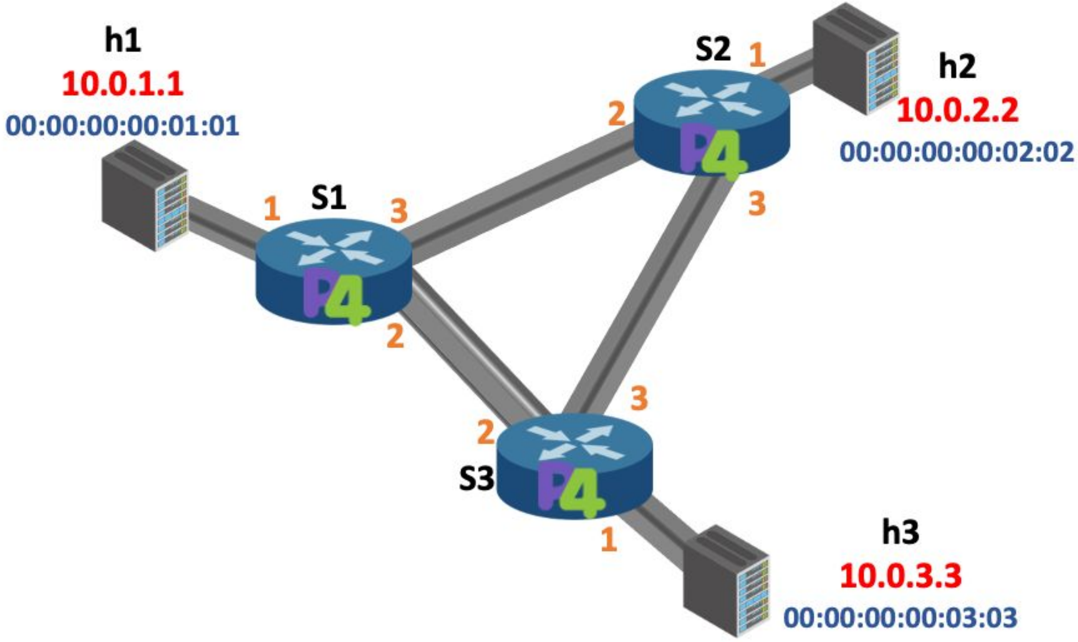
We'll be using several software tools for our experiments:

- **Bmv2**: a P4 software switch
- **p4c**: the reference P4 compiler
- **Mininet**: a lightweight network emulation environment

P4APP

- a *docker-based tool* that can be used to develop, run, debug, and test P4 programs.
- It is easy to install and simple to use.
- ***p4app*** uses a ***software switch*** to run the developed P4 program, and ***mininet*** can be used for testing in an emulation environment.

Topology



- The network topology used in our tutorial is triangular
- Each host connects to a switch.
- The details of hosts (i.e., h1, h2, and h3) and switches (i.e., S1, S2, and S3) are shown in the figure.

Access these slides
http://tiny.cc/aiotwin1709

Run P4 in P4app

Following instructions are also available in the repository's README.

1. Install **docker**

2. Clone repository via this command:

```
git clone https://git.tu-berlin.de/xuanchiguo97/aiotwinp4.git
```

3. Open the directory

```
cd aiotwinp4/examples
```

4. Execute p4app run command at the first time, which will take a while to download the docker image containing the P4 compiler and tools:

```
./p4app run [example_name.p4app]
```

5. [Optional] move p4app to system path $path so that you can run p4app from any location, e.g:

```
cp p4app /usr/local/bin
```

Demo: Simple L3 forwarding

Step 1: Run the starter code

- **./p4app run basic.p4app**
- After this step you'll see the terminal of mininet
- Try to ping between hosts in the topology:
- **mininet> pingall** or
- **mininet> h1 ping h2**
- Quit mininet: **mininet > exit**

Step 2: Implement the forwarding logic

1) Header type definitions for Ethernet (ethernet_t) and IPv4 (ipv4_t).
2) **TODO:** Parsers for Ethernet and IPv4 that populate *ethernet_t* and *ipv4_t* fields.
3) An action to drop a packet, using *mark_to_drop()*.
4) **TODO:** An action (called *ipv4_forward*) that:
   a) Sets the egress port for the next hop.
   b) Updates the ethernet source address with the address of the switch.
   c) Updates the ethernet destination address with the address of the next hop.
   d) Decrements the TTL.
5) ~~**TODO:** Fix ingress control logic that:~~
   a) ~~ipv4_lpm table should be applied only when IPv4 header is valid~~
6) **TODO:** A deparser that selects the order in which fields inserted into the outgoing packet.

Step 3: Populate flow rules

1) **TODO** control plane logic: you need to *define different flow rules in each switch so that they know how to forward the traffic to the destination.*

2) commands1.txt, commands2.txt, commands3.txt represent the rules for the tables in the switch S1, S2, and S3, respectively.

3) The format of adding flow rules in commands[1-3].txt should be like:
table_add [table name] [action name] [table key] ⇒ [action parameter] [action parameter 2] [...]

4) An example using ipv4_lpm table in basic.p4:
table_add ipv4_lpm ipv4_forward 10.0.1.1/32 ⇒ 00:00:00:00:01:01 1

Step 4: Run your solution

If the P4 program and the defined flow rules are correct, it is possible to reach all hosts by using pingall in mininet:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
```

Demo: Calculator

A super simple P4 program showing you the in-computing function of P4!

01 Computer graphics

You can describe the topic of the section here

02 Software and media apps

You can describe the topic of the section here

03 Data modeling/warehousing

You can describe the topic of the section here

04 Modeling, virtual environments

You can describe the topic of the section here

05 Digital & inf. resources design

You can describe the topic of the section here