Netdev Conference 0x17

P4 Compiler Backend for P4TC

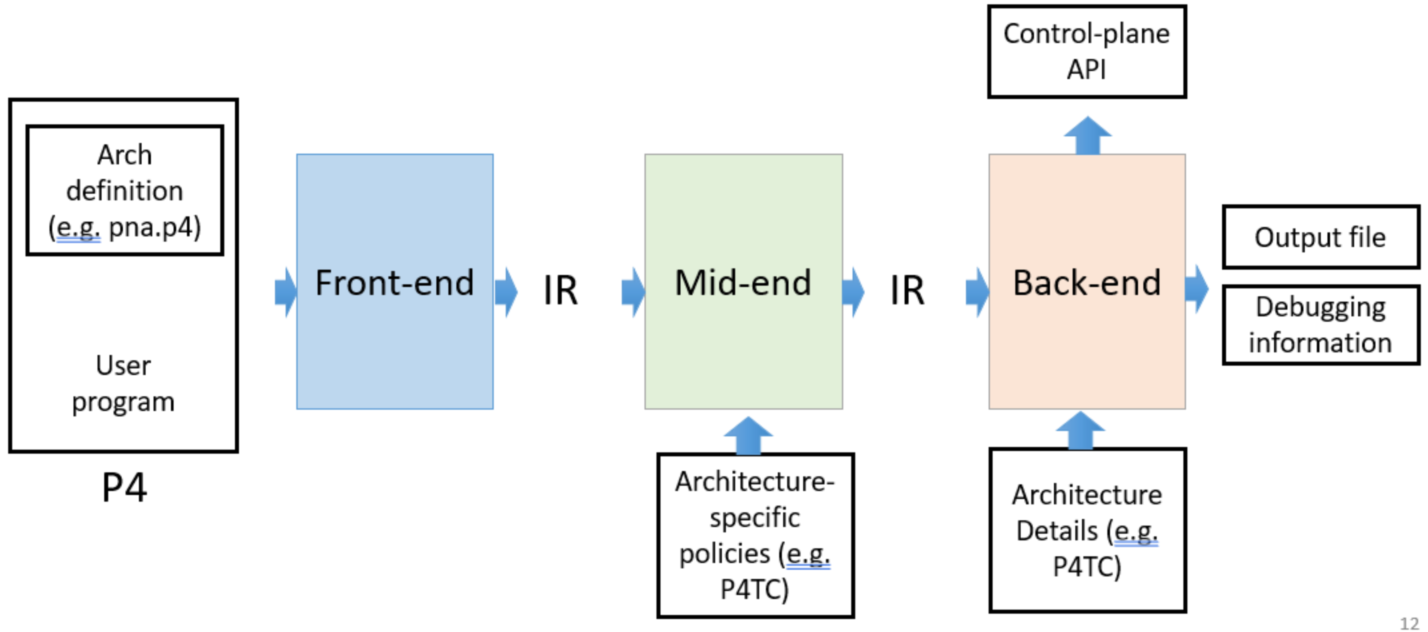
Balachandher Sambasivam, Komal Jain

Contents

P4 Language

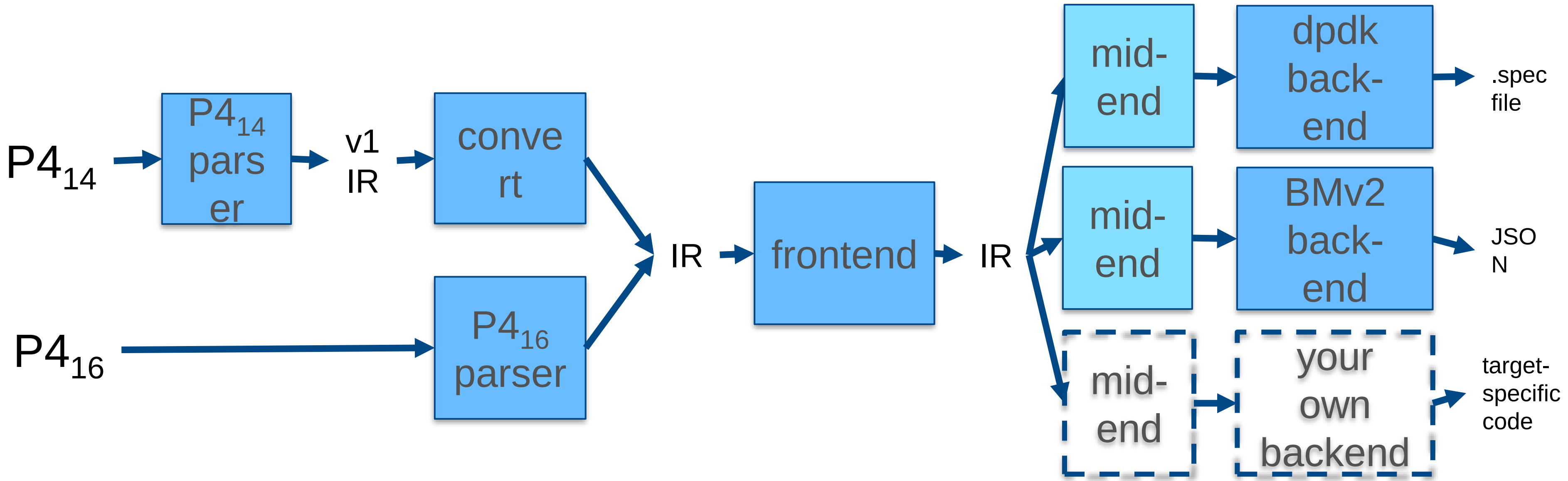
- P4 is a high-level Domain specific Language (DSL) for programming the data plane of network devices. It is used to describe the data-plane functionality and the way control-plane and data-plane communicate.
- P4 language can be viewed as:
  - Core language: types, variables, scope, declaration, statements, expressions, etc.
  - A sub language for describing parsers
  - A sub language for describing computations using match-action units, based on traditional imperative control-flow
  - A sub language for describing architecture

P4 Open-Source Compiler Framework

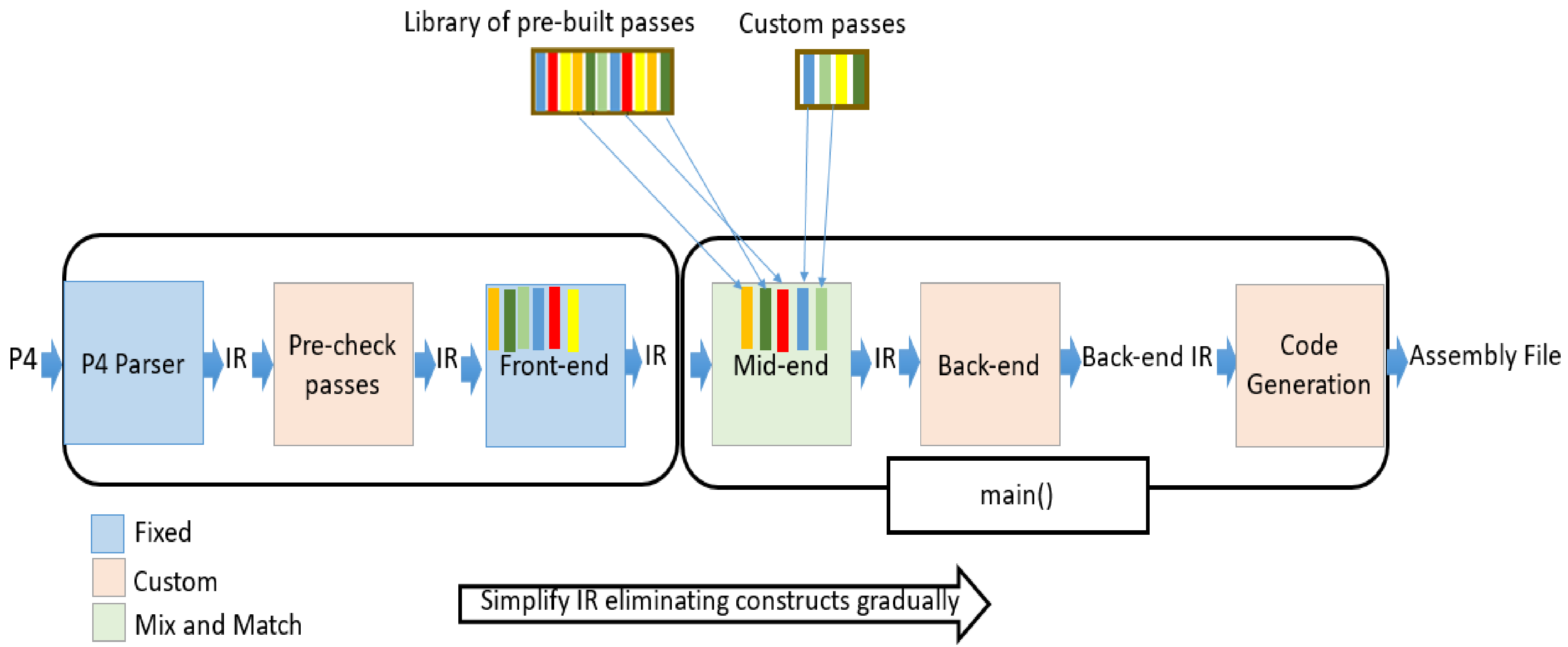

Arch definition (e.g. pna.p4)
User program
P4
Front-end
IR
Mid-end
IR
Back-end
Control-plane API
Output file
Debugging information
Architecture-specific policies (e.g. P4TC)
Architecture Details (e.g. P4TC)

12

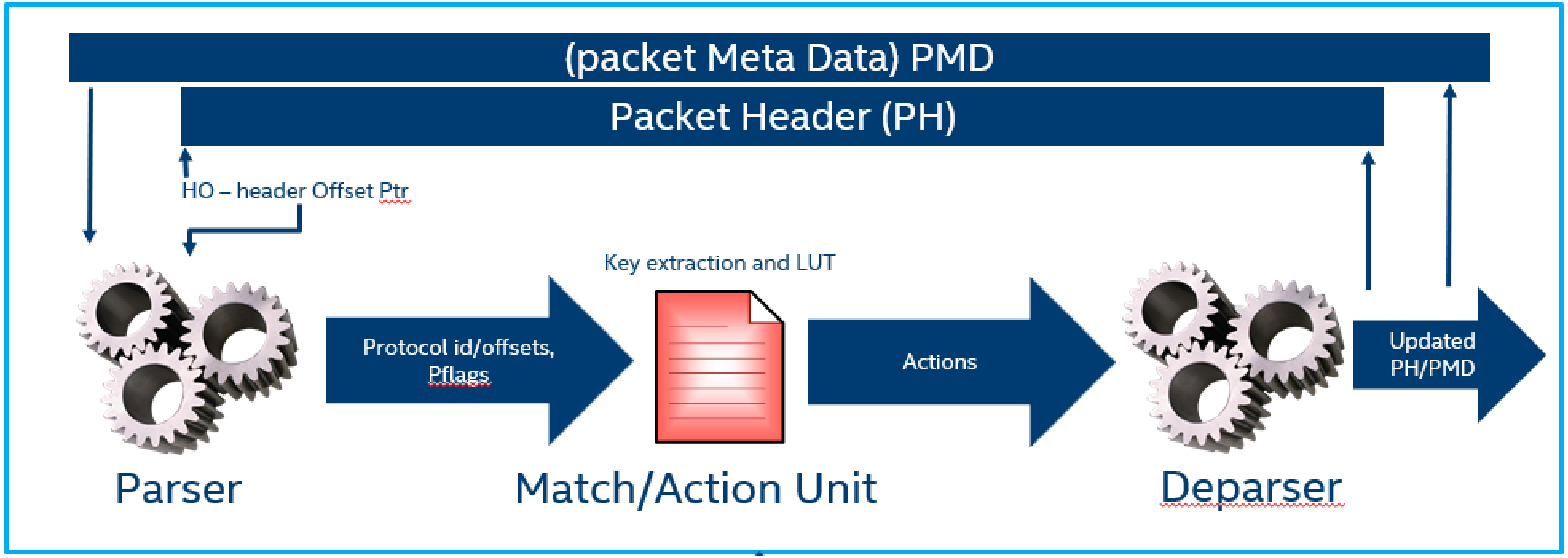
Compiler Data Flow



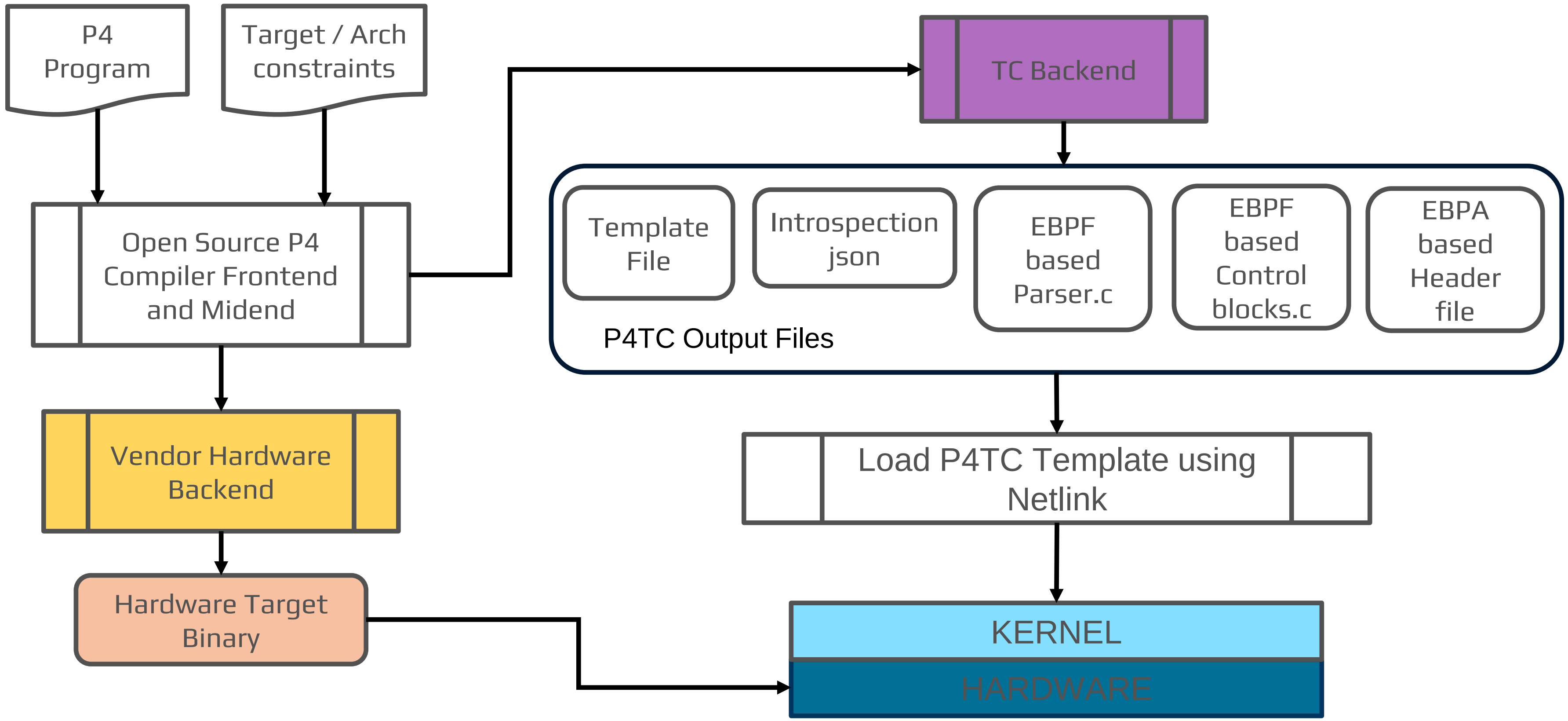
Compiler Structure



Typical Network data plane mapped to P4



Compiler Workflow for P4TC Environment



P4TC Output Files

- **Template File** – This file is a shell script that forms template definitions for various P4 objects (e.g. tables, actions).

- **Json File -** 'json' introspection file used for control plane.

- The backend for TC reuses code from the p4c-ebpf library for generating the C files. p4c-ebpf is an existing opensource p4 backend compiler. The 3 files generated by tc backend using ebpf are –

    - **Parser C file** – This C file represents the parser definition.

    - **Control_blocks C file** – This C files defines rest of the software datapath.

    - **Header file** – The Header File defines all the structure definitions, include files.

```
/******* M A T C H - A C T I O N ****************/
  control ingress(
   inout my_ingress_headers_t  hdr,
   inout my_ingress_metadata_t meta,
   in    pna_main_input_metadata_t  istd,
  inout pna_main_output_metadata_t ostd)
  {
   action send_nh(@tc_type("dev") PortId_t port_id,
                  @tc_type("macaddr") bit<48> dmac,
                  @tc_type("macaddr") bit<48> smac) {
       hdr.ethernet.srcAddr = smac;
       hdr.ethernet.dstAddr = dmac;
       send_to_port(port_id);
   }

  table nh_table {
     key = {
        hdr.ipv4.srcAddr : exact;
     }
     actions = {
        send_nh;
     }
     size = 262144;
  }

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

Template File -

```
#Create Pipeline
tc p4template create pipeline/simple_exact_example pipeid 1 numtables 1

#Create Action
tc p4template create action/simple_exact_example/ingress/send_nh actid 1 \
   param port_id type dev \
   param dmac type macaddr \
   param smac type macaddr
tc p4template update action/simple_exact_example/ingress/send_nh state
active

#Create Table
tc p4template create table/simple_exact_example/ingress/nh_table \
   tblid 1 \
   type exact \
   keysz 32 tentries 262144 \
   table_acts act name simple_exact_example/ingress/send_nh

tc p4template update pipeline/simple_exact_example state ready
```

```
/********** P A R S E R **********/
parser Ingress_Parser(
      packet_in pkt,
      out   my_ingress_headers_t  hdr,
      inout my_ingress_metadata_t meta,
      in    pna_main_parser_input_metadata_t istd)
{
   const bit<16> ETHERTYPE_IPV4 = 0x0800;
   state start {
      transition parse_ethernet;
   }
   state parse_ethernet {
      pkt.extract(hdr.ethernet);
      transition select(hdr.ethernet.etherType) {
         ETHERTYPE_IPV4 : accept;
         default       : reject;
      }
   }
}
```

EBPF based C code –

```
start: {
      /* extract(hdr->ethernet) */
     if (ebpf_packetEnd < pkt + BYTES(ebpf_packetOffsetInBits + 112 + 0)) {
         ebpf_errorCode = PacketTooShort;
         goto reject;
     }

      __builtin_memcpy(&hdr->ethernet.dstAddr, pkt + BYTES(ebpf_ packetOffsetInBits), 6);
      ebpf_packetOffsetInBits += 48;

      __builtin_memcpy(&hdr->ethernet.srcAddr, pkt + BYTES(ebpf_packetOffsetInBits), 6);
      ebpf_packetOffsetInBits += 48;

      hdr->ethernet.etherType = (u16)((load_half_ne(pkt, BYTES(ebpf_packetOffsetInBits))));
      ebpf_packetOffsetInBits += 16;

      hdr->ethernet.ebpf_valid = 1;
      u16 select_0;
      select_0 = hdr->ethernet.etherType;
      if (select_0 == 0x800) goto parse_ipv4;
      if ((select_0 & 0x0) == (0x0 & 0x0))goto reject;
      else goto reject;
   }
reject: {
   if (ebpf_errorCode == 0) {
      return TC_ACT_SHOT;
   }
   goto accept;
}
```

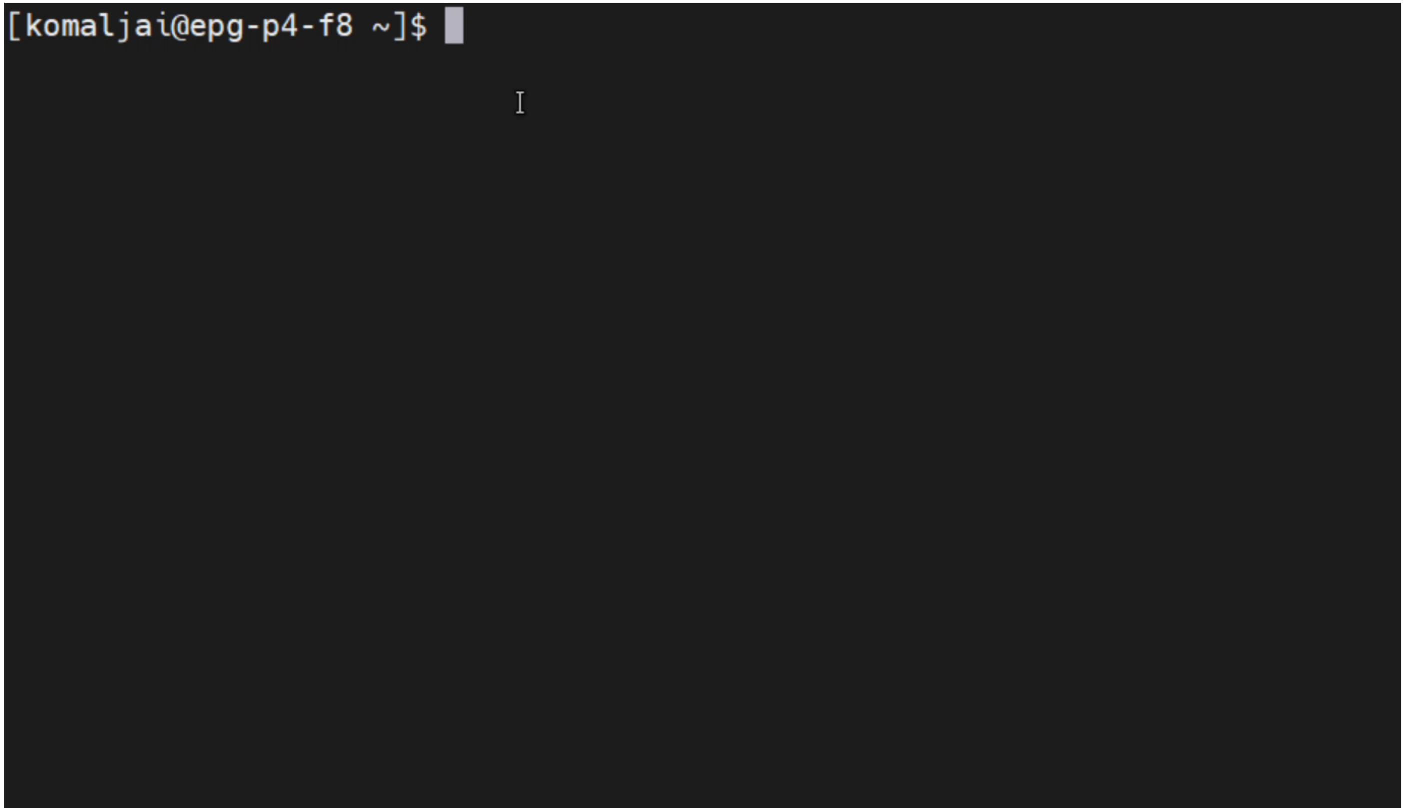
```
/**********  CONTROL BLOCKS **********/
action send_nh(@tc_type("dev") PortId_t port_id,
@tc_type("macaddr") bit<48> dmac,
@tc_type("macaddr") bit<48> smac) {
      hdr.ethernet.srcAddr = smac;
      hdr.ethernet.dstAddr = dmac;
      send_to_port(port_id);
   }

   table nh_table {
      key = {
         hdr.ipv4.srcAddr : exact;
      }
      actions = {
         send_nh;
      }
      size = PORT_TABLE_SIZE;
   }

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

```
/* structure definition */
struct __attribute__((__packed__)) ingress_nh_table_key {
   u32 keysz;
   u32 maskid;
   u32 field0; /* hdr.ipv4.srcAddr */
} __attribute__((aligned(4)));
#define INGRESS_NH_TABLE_ACT_INGRESS_SEND_NH 1
struct __attribute__((__packed__)) ingress_nh_table_value {
   unsigned int action;
   union {
      struct {
      } _NoAction;
      struct __attribute__((__packed__)) {
         u32 port_id;
         u64 dmac;
         u64 smac;
      } ingress_send_nh;
   } u;
};
#define INGRESS_NH_TABLE_ACT_INGRESS_SEND_NH 1

/* construct key */
 struct p4tc_table_entry_act_bpf_params__local params = {
     .pipeid = 1,
     .tblid = 1
 };
 struct ingress_nh_table_key key = {};
 key.keysz = 32;
 key.field0 = hdr->ipv4.srcAddr;
 struct p4tc_table_entry_act_bpf *act_bpf;
```

```
/**********  CONTROL BLOCKS **********/
action send_nh(@tc_type("dev") PortId_t port_id,
@tc_type("macaddr") bit<48> dmac,
@tc_type("macaddr") bit<48> smac) {
      hdr.ethernet.srcAddr = smac;
      hdr.ethernet.dstAddr = dmac;
      send_to_port(port_id);
   }

   table nh_table {
      key = {
         hdr.ipv4.srcAddr : exact;
      }
      actions = {
         send_nh;
      }
      size = PORT_TABLE_SIZE;
   }

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

```
/* value */
struct ingress_nh_table_value *value = NULL;
/* perform lookup */
act_bpf = bpf_p4tc_tbl_read(skb, &params, &key, sizeof(key));
value = (struct ingress_nh_table_value *)act_bpf;
if (value == NULL) {
   hit = 0; /* miss; find default action */
} else {
   hit = 1;
}

 if (value != NULL) {
     /* run action */
     switch (value->action) {
           case INGRESS_NH_TABLE_ACT_INGRESS_SEND_NH:
               {
                    hdr->ethernet.srcAddr = value->u.ingress_send_nh.smac;
                    hdr->ethernet.dstAddr = value->u.ingress_send_nh.dmac;
                    /* send_to_port(value->u.ingress_send_nh.port_id) */
                    compiler_meta__->drop = false;
                    send_to_port(value->u.ingress_send_nh.port_id);
               }
               break;
          default:
                 return TC_ACT_SHOT;
      }
 } else {
                return TC_ACT_SHOT;
 }
```

Demo



```
[komaljai@epg-p4-f8 ~]$
```

Future Work

- P4 Externs - P4 programs can also interact with objects and functions provided by the architecture. Such objects are described using the extern construct, which describes the interfaces that such objects expose to the data-plane.

  There are two types of supported externs :

  - Architecture specific externs (e.g PNA, PSA, etc)

  - Custom externs

- Same P4 program to compile with both P4-TC and vendor specific compiler

- P4 Testgen for P4TC - P4Testgen tool automatically generate input-output packet tests for P4 programs.

References

- P4 language Specification -
  - https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html
  - https://github.com/p4lang/p4c

- PNA (Portable NIC Architecture - https://p4.org/p4-spec/docs/PNA-v0.5.0.html
- P4TC Git Repo - https://github.com/p4lang/p4c/tree/main/backends/tc
- P4TC TestGen - https://github.com/p4lang/p4c/tree/main/backends/p4tools/modules/testgen

Thank You