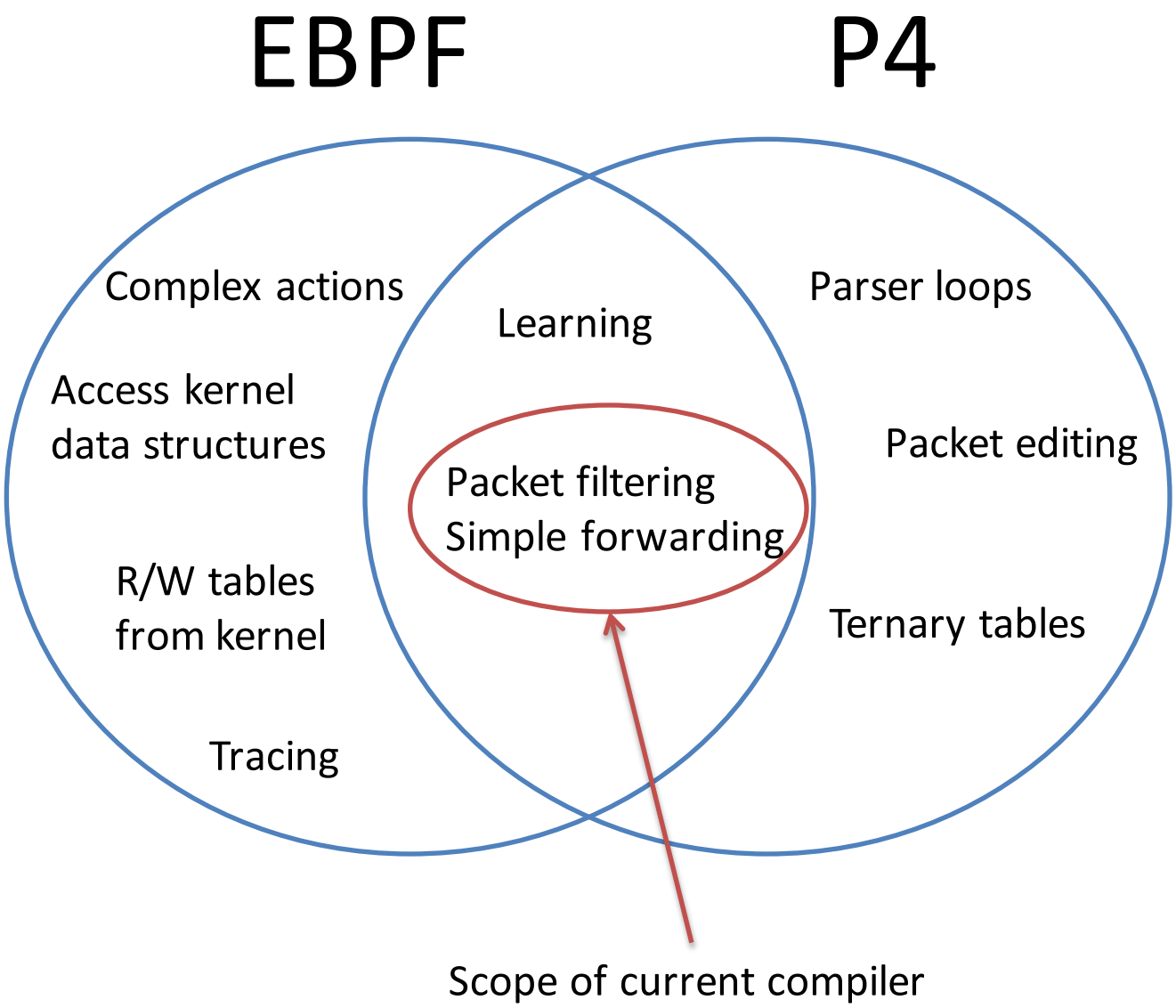
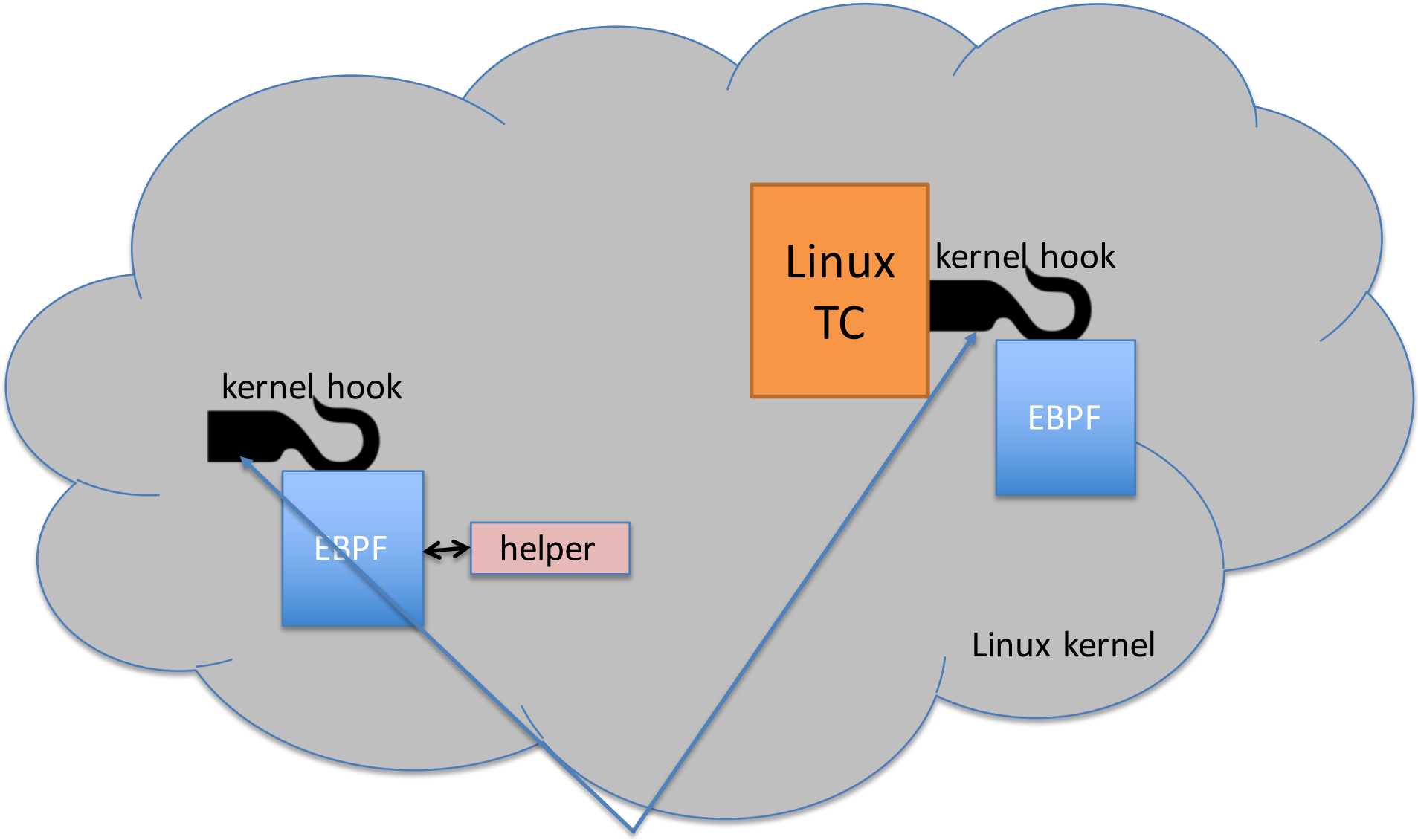
Compiling P4 to EBPF

Mihai Budiu

Sept 30, 2015


EBPF
P4
Complex actions
Access kernel
data structures
R/W tables
from kernel
Tracing
Learning
Packet filtering
Simple forwarding
Parser loops
Packet editing
Ternary tables
Scope of current compiler

EBPF's world

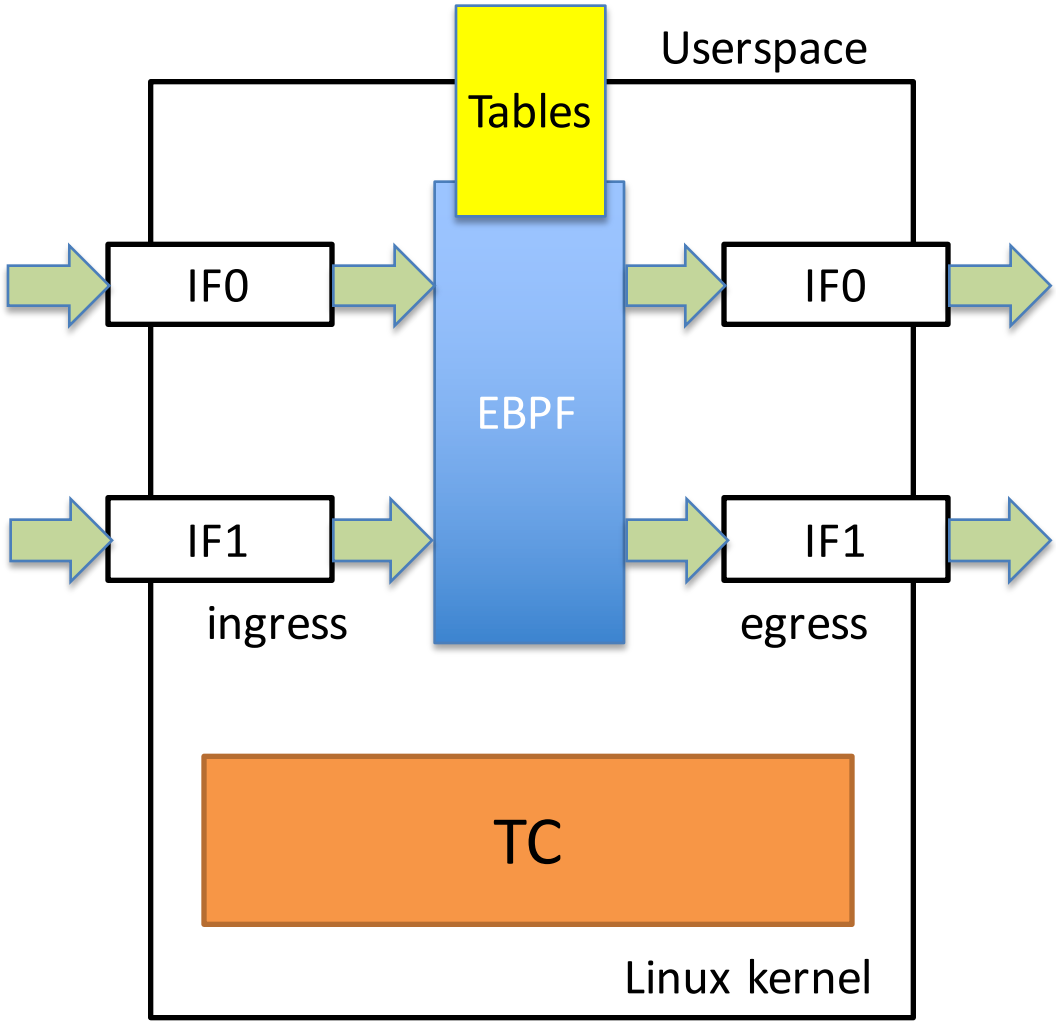


Each hook provides different capabilities for the EBPF programs.

P4 v1.0 -> C -> EBPF

- Prototype back-end for P4 compiler
  - Only a subset of P4 v1.0
- Generate stylized C
  (no loops, only helper method calls)
- Assume EBPF program hooked using TC
- Use BCC to generate EBPF
- https://github.com/iovisor/bcc/tree/master/src/cc/frontends/p4

Packet Processing Model



Simple.p4

```
header_type ethernet_t { … }
header_type ipv4_t { … }

parser start {
   return parse_ethernet;
}

header ethernet_t ethernet;
header ipv4_t ipv4;

parser parse_ethernet {
   extract(ethernet);
   return select(latest.etherType) {
      0x800 : parse_ipv4;
      default: ingress;
   }
}

parser parse_ipv4 { … }
```

```
action nop() {}

action forward(port)
{ modify_field(
  standard_metadata.egress_port, port);
}

table routing {
  reads {
    ipv4.dstAddr: exact;
  }
  actions { nop; forward; }
  size : 512;
}

control ingress
{
   apply(routing);
}
```

Parser translation

- header_type -> struct
- Header -> struct instance + valid field
- Metadata -> struct instance
- Parser state -> code block
- State transitions -> goto
- Header extract -> load/shift/mask from skb
- Most work: handling unaligned header fields

Smple generated parser code

```
if (skb->len < BYTES(ebpf_currentOffsetInBits + 16)) {
        error = ebpf_PacketTooShort;
        goto reject;
}
headers.ethernet.etherType =
            (u16)((load_half(skb, BYTES(ebpf_currentOffsetInBits))));
ebpf_currentOffsetInBits += 16;

headers.ethernet.valid = 1;
switch (headers.ethernet.etherType) {
     case 2048: goto ip;
     default: goto reject;
}
```

Tables => Tables

- 1 P4 table => 2 EBPF tables
  - 1 table for data
  - 1 table for default_action
- P4 counters => 1 EBPF table

```
BPF_TABLE("hash", struct routing_key_1, struct routing_value_2, routing, 512);
BPF_TABLE("array", u32, struct routing_value_2, routing_miss, 1);
```

Reads => Table keys

```
struct routing_key_1 { u32 key_field_0; };

struct routing_key_1 key;
key.key_field_0 = (ebpf_headers.ipv4.dstAddr);
```

Actions => Table values

Tagged union with data for each action.

```
enum routing_actions {
   routing_nop,
   routing_forward
};
struct routing_value_2 {
    u32 action;
    union {
        struct {} nop;
        struct { u16 port; } forward;
    } u;
};
```

Table Lookup => 2 EBPF Table lookups

```
value = routing.lookup(&key);
if (value == NULL) {
    ebpf_hit = 0;
    /* miss; find default action */
    value = ebpf_routing_miss.lookup(&ebpf_zero);
}
else {  /* Update counters cnt */  }
if (value != NULL) {
     /* run action */ …
}
```

Actions => Switch statement

inlined action body

```
switch (value->action) {
   case routing_noop:
   {
   }
   break;
   case routing_forward:
   {
        ebpf_ metadata.standard_metadata.egress_port = value->u.forward.port;
   }
   break;
}
```

Packet forwarding

```
if (!ebpf_drop)
        bpf_clone_redirect(ebpf_packet,
            ebpf_metadata.
            standard_metadata.egress_port, 0);
return TC_ACT_SHOT
/* drop packet; clone is forwarded */;
```

Limitations

- Current implementation is incomplete
  - No support for parser loops
  - No deparser is synthesized:
    - Cannot modify packet, just route it
  - Many P4 constructs unsupported
    - Learning, checksums, meters, registers, cloning, recirculation
    - Arbitrary precision arithmetic
  - P4 v1.0 is under-specified
- Some limitations are fundamental, others can be fixed