The Power of Programmable Parsing

Dan Talayco

Barefoot Networks

November 2014

P4: Programming Protocol-Independent Packet Processors

ACM CCR. Volume 44, Issue #3 (July 2014)

BAREFOOT NETWORKS

Protocols

- Protocols are what allow networking to work
- There are a lot of protocols but there already

CAPWAP DCTCP L2TP FLIP IPSEC UDP TCP NSH RARP SMTP HTTP IBGP Geneve ARP IP SSL RIP SNAP POP FTP IPv6 ICMP Ethernet BGP IPv4 OSPF NVGRE GRE PPP VLAN POP SCTP MPLS BFD ERSPAN FCOE IMAP

Protocols Evolve

- Protocols change
  - New encapsulations: VXLan, NVGRE, Geneve…

BAREFOOT NETWORKS

Protocols Evolve

- Protocols change
  - New encapsulations: VXLan, NVGRE, Geneve...
- And we want them to be extensible
  - For example, option lists in headers
    - Variable length lists
    - Variable length options
    - New type values over time

Networks are Diverse

- Seeing the growth of MSDCs and other networks with high performance demands
  - Requirements fundamentally different from e.g. WAN
- Network performance has become a competitive advantage

Networks are Diverse

- Seeing the growth of MSDCs and other networks with high performance demands
  - Requirements fundamentally different from e.g. WAN
- Network performance has become a competitive advantage
- Deployments are becoming more specialized.
  - Yes, we can adapt BGP to work to the TOR or even the v-Switch, but is there a better way?

Networks are Diverse

- Seeing the growth of MSDCs and other networks with high performance demands
  - Requirements fundamentally different from e.g. WAN
- Network performance has become a competitive advantage
- Deployments are becoming more specialized.
  - Yes, we can adapt BGP to work to the TOR or even the v-Switch, but is there a better way?
- "One size fits all" is no longer acceptable

Parsing

- Protocols are defined by field semantics

  …and **parsers** identify fields
- Yet parsers have not gotten much attention
  - Too easy? Just a state machine
  - Too hard? Difficult to make programmable and line rate
  - Too important? Controlling the protocols means controlling the feature set

BAREFOOT NETWORKS

Parsers are Stuck in the Past

- Static Parsing Slows Innovation
  - Binds feature changes to slower development cycle (esp in HW, but SW too).
- OpenFlow History
  - Avoided taking on the challenge; just identified fields for match/action
  - Field count, 1.0 to 1.4: 12 => 15 => 36 => 40 => 41

Programmable Parsers

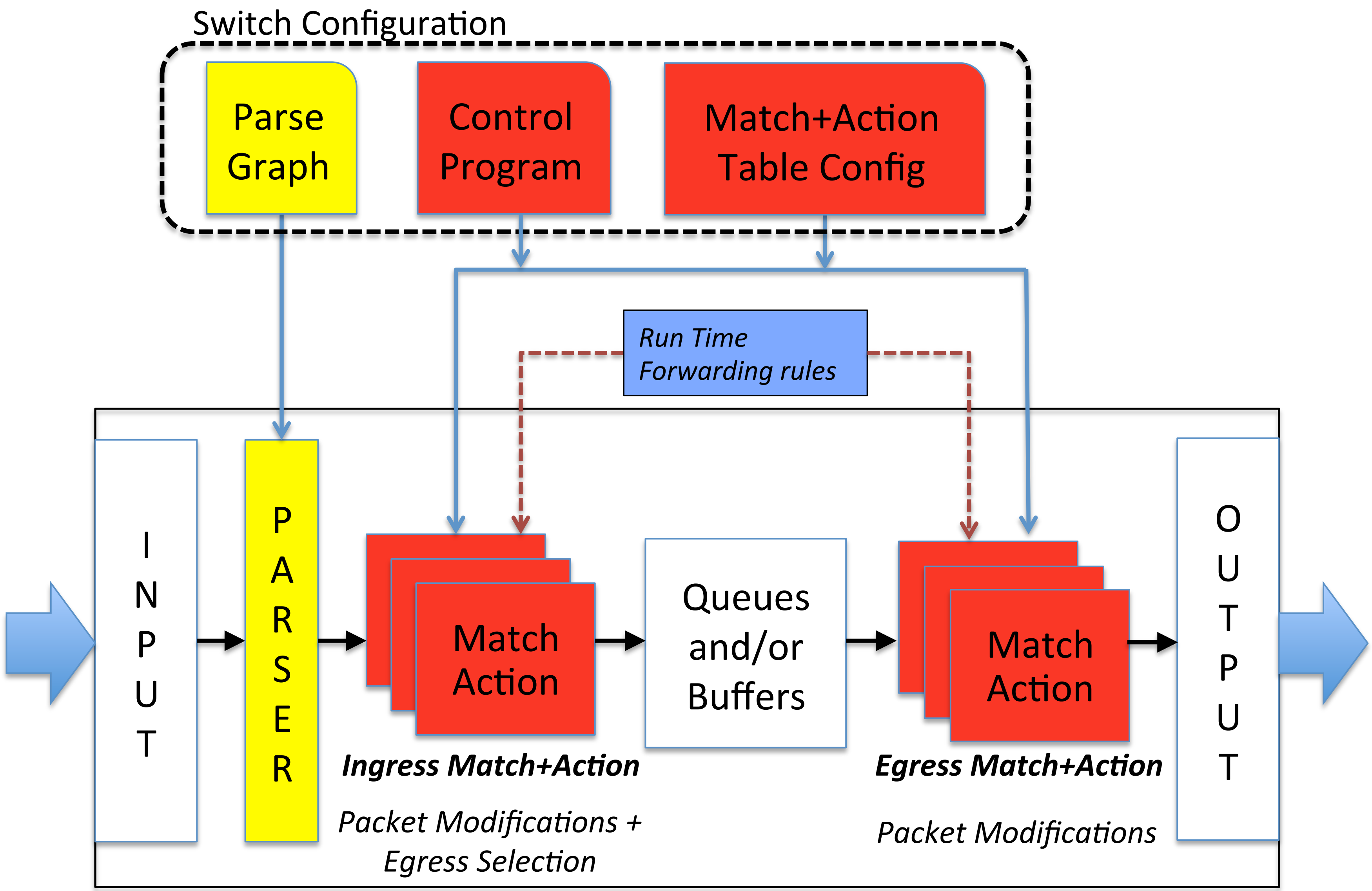
- Programmable parsing is the key to unlocking OpenFlow's Match+Action processing model
- Obviously, possible in SW
  - But not (yet) in OVS
- Also possible in HW
  - Gibb, et al, Design Principles for Packet Parsers

High Level Language Support

- The P4 language
  - Provides a single means of configuring forwarding
  - Based on an abstract forwarding model
  - With a programmable parser
  - Allows the definition of arbitrary headers and fields
  - Provides a context for match+action definitions

BAREFOOT NETWORKS

High Level P4 Abstraction



BAREFOOT NETWORKS

Headers and Fields

- Fields have a width and other attributes
- Headers are collections of fields
- These are types which are used to declare instances

Metadata is a header instance

```
header_type ethernet_t {
  fields {
    dstAddr    : 48;
    srcAddr    : 48;
    etherType  : 16;
  }
}

/* Instance of eth header */
header ethernet_t inner_ethernet;
```

```
header_type egress_metadata_t {
  fields {
    nhop_type    : 8;  /* 0: L2, 1: L3, 2: tunnel */
    encap_type   : 8;  /* L2 Untagged; L2ST; L2DT */
    vnid         : 24; /* gnve/vxlan vnid/gre key */
    tun_type     : 8;  /* vxlan; gre; nvgre; gnve*/
    tun_idx      : 8;  /* tunnel index */
  }
}

metadata egress_metadata_t egress_metadata;
```

BAREFOOT NETWORKS

Headers and Fields

- Fields have a width and other attributes
- Headers are collections of fields
- These are types which are used to declare instances

Type used in Instance Declaration

```
header_type ethernet_t {
  fields {
    dstAddr     : 48;
    srcAddr     : 48;
    etherType   : 16;
  }
}

/* Instance of eth header */
header ethernet_t inner_ethernet;
```

```
header_type egress_metadata_t {
  fields {
    nhop_type    : 8;   /* 0: L2, 1: L3, 2: tunnel */
    encap_type   : 8;   /* L2 Untagged; L2ST; L2DT */
    vnid         : 24;  /* gnve/vxlan vnid/gre key */
    tun_type     : 8;   /* vxlan; gre; nvgre; gnve*/
    tun_idx      : 8;   /* tunnel index */
}
}

metadata egress_metadata_t egress_metadata;
```

BAREFOOT NETWORKS

The Parser

- Imperative functions for “states”
- Extract header instances
- Select a next “state” by returning a parser function

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_CPU  : parse_cpu_header;
    ETHERTYPE_VLAN : parse_vlan;
    ETHERTYPE_MPLS : parse_mpls;
    ETHERTYPE_IPV4 : parse_ipv4;
    ETHERTYPE_IPV6 : parse_ipv6;
    ETHERTYPE_ARP  : parse_arp_rarp;
    ETHERTYPE_RARP : parse_arp_rarp;
    ETHERTYPE_NSH  : parse_nsh;
  }
}
```

- Produces a *Parsed Representation* of the packet

The Parser

- Imperative functions for “states”
- Extract header instances
- Select a next “state” by returning a parser function

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_CPU  : parse_cpu_header;
    ETHERTYPE_VLAN : parse_vlan;
    ETHERTYPE_MPLS : parse_mpls;
    ETHERTYPE_IPV4 : parse_ipv4;
    ETHERTYPE_IPV6 : parse_ipv6;
    ETHERTYPE_ARP  : parse_arp_rarp;
    ETHERTYPE_RARP : parse_arp_rarp;
    ETHERTYPE_NSH  : parse_nsh;
  }
}
```

- Produces a *Parsed Representation* of the packet

The Parser

- Imperative functions for “states”
- Extract header instances
- Select a next “state” by returning a parser function

```
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    ETHERTYPE_CPU  : parse_cpu_header;
    ETHERTYPE_VLAN : parse_vlan;
    ETHERTYPE_MPLS : parse_mpls;
    ETHERTYPE_IPV4 : parse_ipv4;
    ETHERTYPE_IPV6 : parse_ipv6;
    ETHERTYPE_ARP  : parse_arp_rarp;
    ETHERTYPE_RARP : parse_arp_rarp;
    ETHERTYPE_NSH  : parse_nsh;
  }
}
```

- Produces a *Parsed Representation* of the packet

Parsing and Headers Give Context for Match + Action

```
table acl {
    reads {
        ipv4.dstAddr : ternary;
        ipv4.srcAddr : ternary;
        ipv4.protocol : ternary;
        udp.srcPort : ternary;
        udp.dstPort : ternary;
        ethernet.dstAddr : exact;
        ethernet.srcAddr : exact;
        ethernet.etherType : ternary;
    }
    actions {
        no_op;    /* permit */
        acl_drop; /* reject */
        nhop_set; /* policy-based routing */
    }
}
```

Programmable Parsing in OVS

- Supporting a programmable parser in OVS is tractable
- OVS is critical for providing deployment specific features in the future
  - Selective protocol engagement
  - Agile response to protocol evolution
  - Overlay/underlay architectures

The P4 Language Consortium

- **Consortium:** Independent, open-source, CA non-profit.
- **Original authors** from Google, Microsoft, Intel, Princeton, Stanford, Barefoot
- **Sign up for the P4 Announcement mailing list:** **www.p4.org**
    - Currently on the site: **P4 Language Spec**, the original paper, reference links

P4

Home | About | Spec | Resources | Membership

Welcome to the P4 website.

In 2013 a group of us* came together to define P4, a high-level language for programming future flexible network switches. P4 has three goals:

1. **Protocol independence:** Switches should not be tied to any specific network protocols.
2. **Target independence:** Programmers should be able to describe packet processing functionality independently of the specifics of the underlying hardware.
3. **Reconfigurability in the field:** Programmers should be able to change the way switches process packets once they are deployed.