Programmable Networks – P4 overview

Presented by: Divya Chitimalla

Worlds Fastest Most Programmable Networks Barefoot Networks white paper, 2016.

P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014 Pat Bosshart† , Dan Daly* , Glen Gibb† , Martin Izzard† , Nick McKeown‡ , Jennifer Rexford** , Cole Schlesinger**, Dan Talayco† , Amin Vahdat¶ , George Varghese§ , David Walker** †Barefoot Networks * Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research.

Why programmable networks

- Deploy tests and probes, can reduce time to recover from an outage
- Monitoring networks can be eliminated because network can now monitor itself
- Eliminate redundant equipment. For example, big data-centers today commonly deploy expensive middleboxes – load-balancers, address translators, complex Network Function Virtualization (NFV) cluster of thousands of servers to load-balance incoming packets across web servers.
- Only slow networks are programmable now. NPUs and FPGAs exist and are flexible. But are 1/100th performance of fixed-function ASICs

Worlds Fastest Most Programmable Networks Barefoot Networks white paper, 2016.

Barefoot Networks

- Created first programmable chip that performs like ASIC
- P4 - Programming Protocol-independent Packet Processors – www.p4.org exists now as an independent entity to develop a rich open source ecosystem
- P4 offers a programming abstraction that is familiar to network owners rather than VHDL
- Proposed architecture does for networking what DSP did for signal processing, GPU did for graphics and TPU is doing for machine learning
- Programs are written in a high level domain specific language (P4), compiled down by Barefoot Capilano compiler, and optimized to run at full line-rate on PISA device

Worlds Fastest Most Programmable Networks Barefoot Networks white paper, 2016.

P4

P4 is a high-level language for programming protocol-independent packet processors

OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing complexity of specification

(1) Reconfigurability : Programmers should be able to change way switches process packets once deployed

(2) Protocol independence: Switches should not be tied to any specific network protocols

(3) Target independence: independent of specifics of underlying hardware

Future switches should support mechanisms for parsing packets and matching header allowing controller applications to leverage capabilities of common, open interface (i.e., a new \OpenFlow 2.0" API).

Worlds Fastest Most Programmable Networks Barefoot Networks white paper, 2016.

What does it do

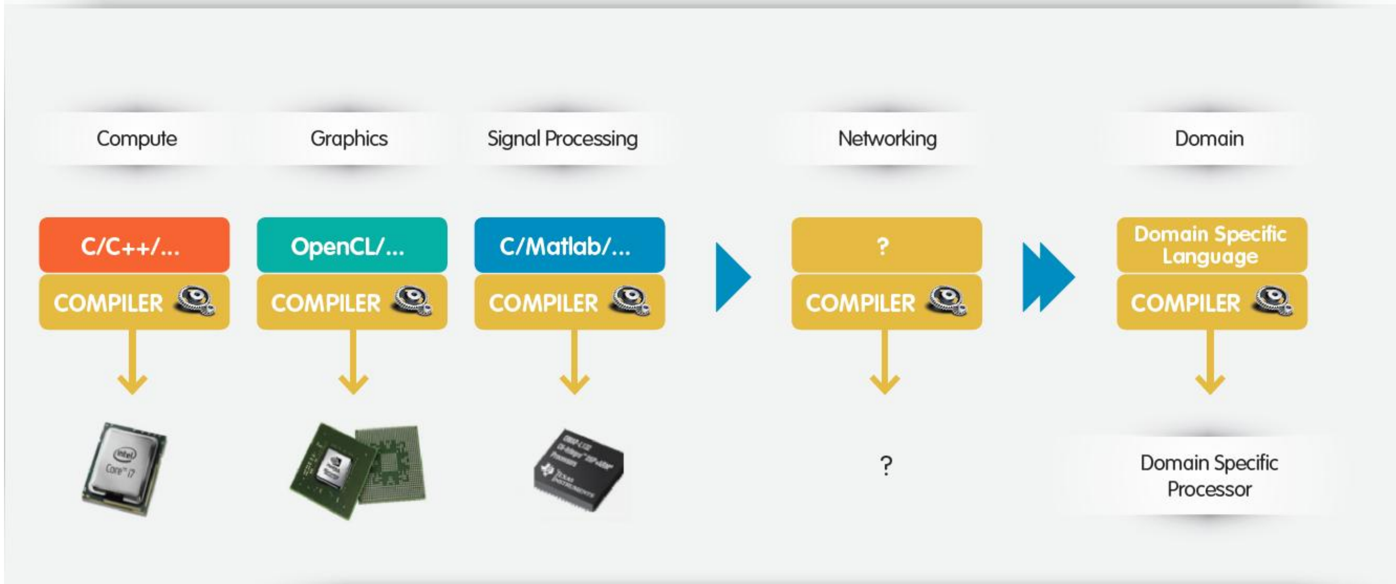
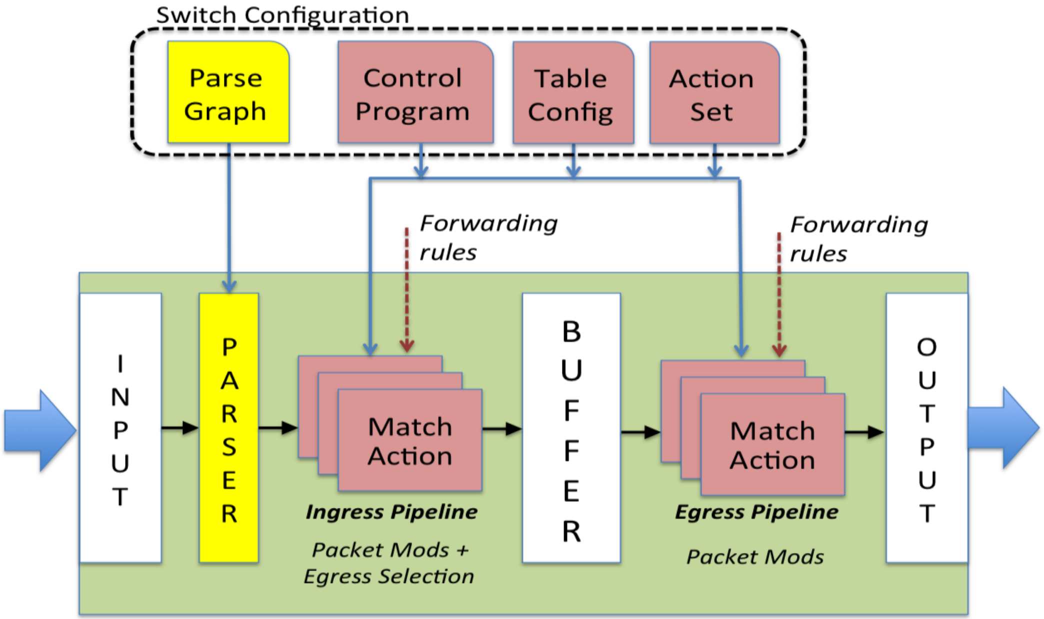


FIGURE 1 Opportunity to "compile" the networks

Worlds Fastest Most Programmable Networks Barefoot Networks white paper, 2016.

Steps in P4



P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014 Pat Bosshart† , Dan Daly* , Glen Gibb† , Martin Izzard† , Nick McKeown‡ , Jennifer Rexford** , Cole Schlesinger**, Dan Talayco† , Amin Vahdat¶ , George Varghese§ , David Walker** †Barefoot Networks * Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research.

Header Formats

```
header ethernet {
fields {
        dst_addr : 48;
//width in bits
        src_addr : 48;
        ethertype : 16;
        }
}
```

```
header vlan {
        fields {
                pcp : 3;
                cfi : 1;
                vid : 12;
                ethertype : 16;
                }
}
```


P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014 Pat Bosshart† , Dan Daly* , Glen Gibb† , Martin Izzard† , Nick McKeown‡ , Jennifer Rexford** , Cole Schlesinger**, Dan Talayco† , Amin Vahdat¶ , George Varghese§ , David Walker** †Barefoot Networks * Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research.

Packet Parser

Parsing starts in start state and proceeds until an explicit stop state is reached

Extracted headers are forwarded to match+action processing

```
parser start {
        ethernet;
}
parser ethernet {
        switch(ethertype) {
        case 0x8100: vlan;
        case 0x9100: vlan;
        case 0x800: ipv4;
        // Or cases
        }
}
parser vlan {
        switch(ethertype) {
        case 0xaaaa: mTag;
        case 0x800: ipv4;
        // Or cases
        }
}
parser mTag {
        switch(ethertype) {
        case 0x800: ipv4;
        // Or cases
}
}
```


P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014 Pat Bosshart† , Dan Daly* , Glen Gibb† , Martin Izzard† , Nick McKeown‡ , Jennifer Rexford** , Cole Schlesinger**, Dan Talayco† , Amin Vahdat¶ , George Varghese§ , David Walker** †Barefoot Networks * Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research.

Table specification

- Programmer describes how header fields are to be matched in match+action stages (e.g., should they be exact matches, ranges, or wildcards?) and what actions should be performed when a match occurs
- Reads attribute declares which fields to match, qualified by match type (exact, ternary, etc)
- Actions attribute lists possible actions which may be applied to a packet by table

P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014 Pat Bosshart† , Dan Daly* , Glen Gibb† , Martin Izzard† , Nick McKeown‡ , Jennifer Rexford** , Cole Schlesinger**, Dan Talayco† , Amin Vahdat¶ , George Varghese§ , David Walker** †Barefoot Networks * Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research.

Table action

P4's primitive actions include:
set field: Set a header to a value. Masked sets are supported.
copy field: Copy one field to another.
add header: Set a specific header instance (and all its fields) as valid.
remove header: Delete (\pop") a header (and all its fields) from a packet.
increment: Increment or decrement value in a field.
checksum: Calculate a checksum over some set of header fields (e.g., an IPv4 checksum).

```
action add_mTag(up1, up2, down1, down2,
egr_spec) {
        add_header(mTag);
        // Copy VLAN ethertype to mTag
        copy_field(mTag.ethertype, vlan.ethertype);
        // Set VLAN's ethertype to signal mTag
        set_field(vlan.ethertype, 0xaaaa);
        set_field(mTag.up1, up1);
        set_field(mTag.up2, up2);
        set_field(mTag.down1, down1);
        set_field(mTag.down2, down2);
        // Set  destination egress port as well
        set_field(metadata.egress_spec, egr_spec);
}
```


P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014 Pat Bosshart† , Dan Daly* , Glen Gibb† , Martin Izzard† , Nick McKeown‡ , Jennifer Rexford** , Cole Schlesinger**, Dan Talayco† , Amin Vahdat¶ , George Varghese§ , David Walker** †Barefoot Networks * Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research.

Control Program

Once tables and actions are defined, only remaining task is to specify flow of control from one table to next

Control flow is specified as a program via a collection of functions, conditionals, and table references

```
control main() {
	// Verify mTag state and port are
consistent
table(source_check);
	// If no error from source_check,
continue
	if (!defined(metadata.ingress_error))
{
	// Attempt to switch to end hosts
table(local_switching);
	if (!defined(metadata.egress_spec))
{
	// Not a known local host; try
mtagging
```


P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014 Pat Bosshart† , Dan Daly* , Glen Gibb† , Martin Izzard† , Nick McKeown‡ , Jennifer Rexford** , Cole Schlesinger**, Dan Talayco† , Amin Vahdat¶ , George Varghese§ , David Walker** †Barefoot Networks * Intel ‡Stanford University **Princeton University ¶Google §Microsoft Research.

Conclusion

- Proposed a step towards more flexible switches whose functionality is specified and may be changed once deployed
- Programmer decides how forwarding plane processes packets without worrying about implementation details
- A compiler transforms an imperative program into a table dependency graph that can be mapped to many target switches, including optimized hardware implementations