

Programming The Network Data Plane in P4

Changhoon Kim (chang@barefootnetworks.com)

P4 Language Consortium – www.p4.org

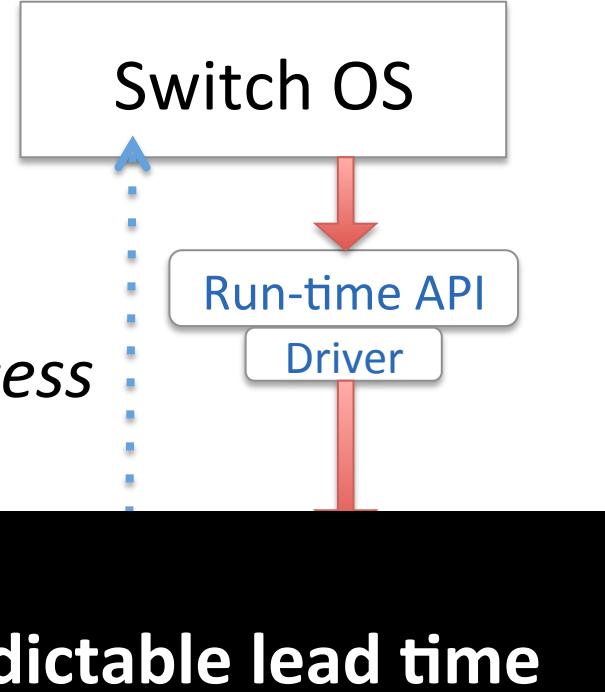


Status quo



*"This is roughly how I process
packets ..."*

- Prone to bugs
- Very long and unpredictable lead time



Fixed-function ASIC

Protocols evolve ... more rapidly than we wish they would

- Encapsulation protocols for network virtualization
 - 1st gen (~2010): VXLAN, NVGRE
 - 2nd gen (~2012): STT, VXLAN-GPE
 - 3rd gen (~2014): NSH, Geneve
- OpenFlow
 - OF 1.0 (Dec 2009): 12 fields (Ethernet, TCP, IPv4)
 - OF 1.1 (Feb 2011): 15 fields (MPLS, inter-table metadata)
 - OF 1.2 (Dec 2011): 36 fields (ARP, ICMP, IPv6)

Everybody has opinions!

Can't we just let them build their own protocols?

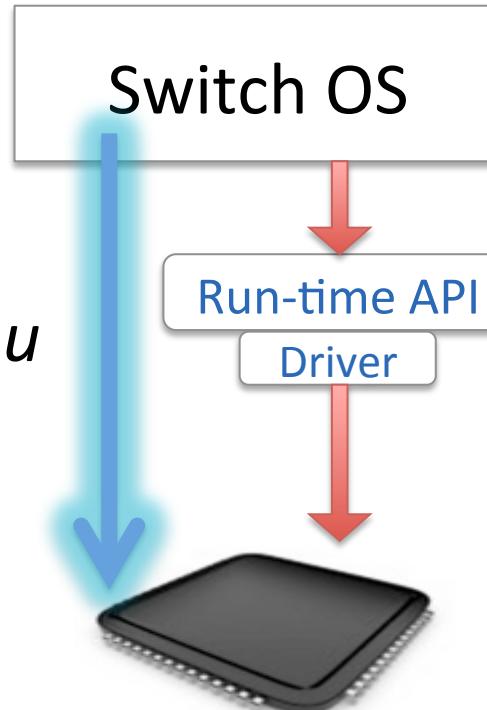
Programmable network devices

- Some devices are or will be more programmable than fixed-function ASICs
- CPUs: 10s of Gb/s
- FPGAs, NPUs: 100s of Gb/s
- Protocol-independent switch ASICs: a few Tb/s
 - RMT [SIGCOMM'13] and a few emerging solutions
 - Merchant silicon with fully programmable parser and generic match-action logic
 - In next few years this kind of silicon will dominate

Turning the tables

in P4

*"This is precisely how you
must process packets"*



PISA device

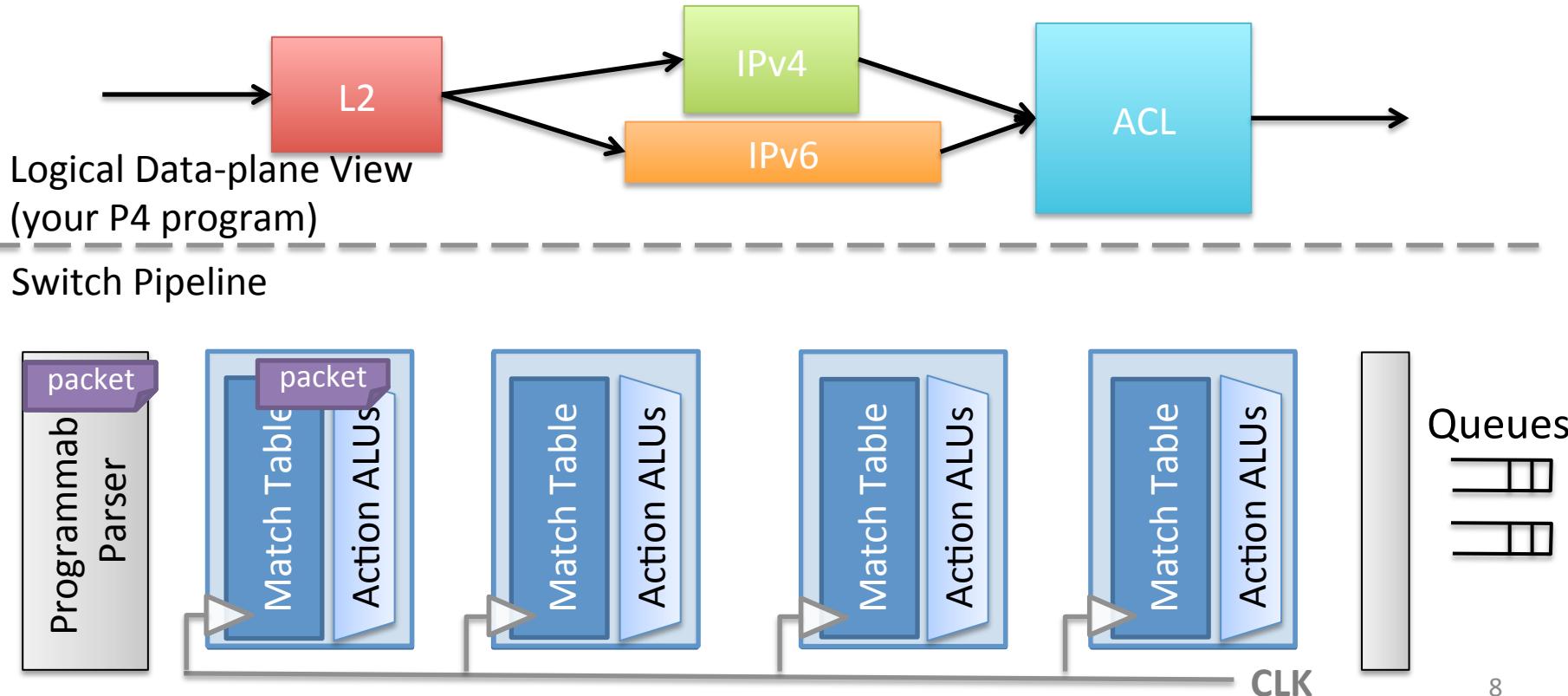
(Protocol-Independent Switch Architecture)

What does this mean?

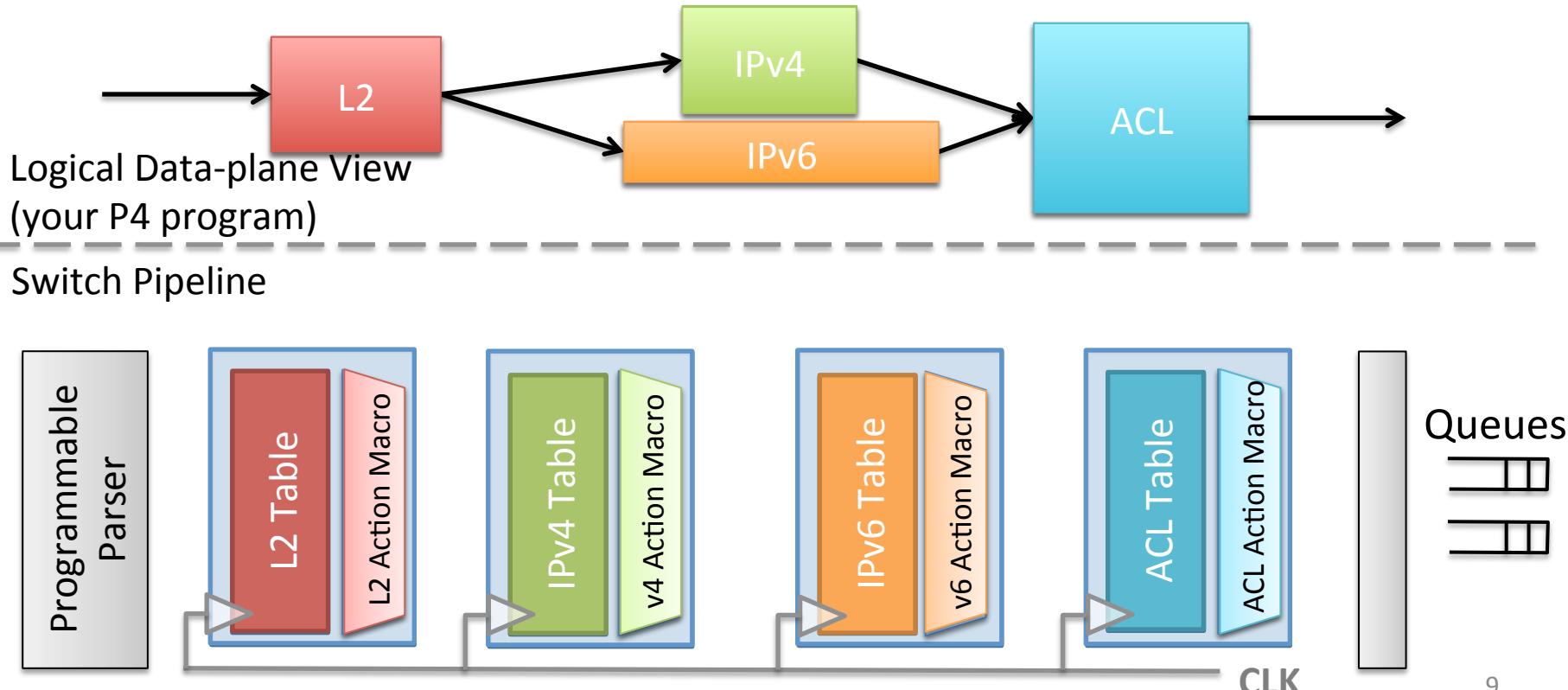
- To network device vendors
 - S/W programming practices and tools used in every phase
 - Extremely fast iteration and feature release
 - Differentiation in capabilities and performance
 - Can fix even data-plane bugs in the field
- To large on-line service providers
 - No more “black boxes” in the “white boxes”
 - Your devs can program, test, and debug your network devices all the way down
 - You keep your own ideas

**Why we call it
protocol-independent packet
processing**

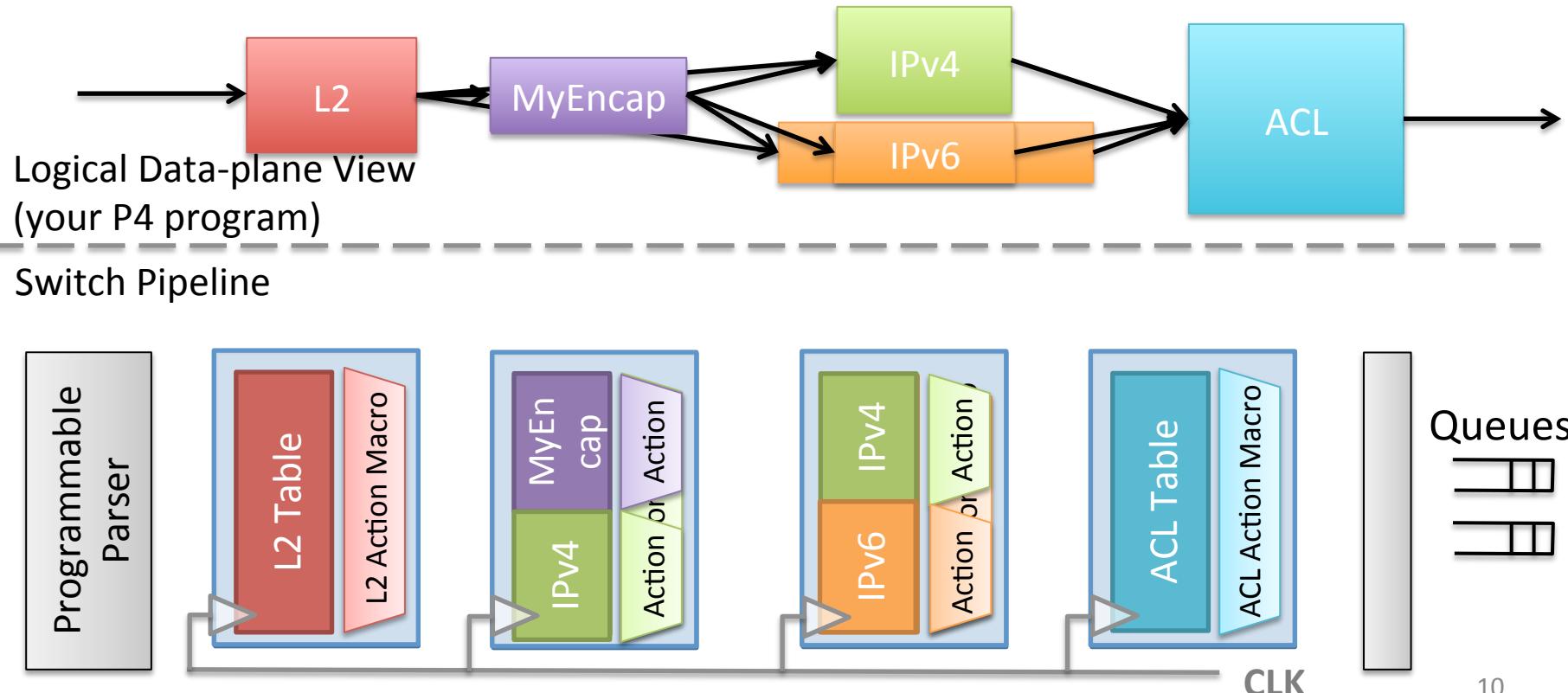
Device does not understand any protocols until it gets programmed



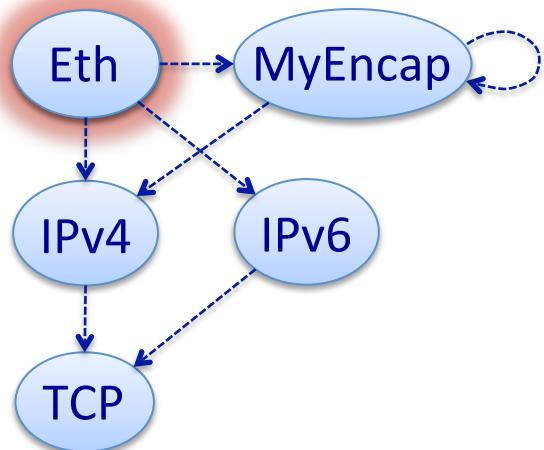
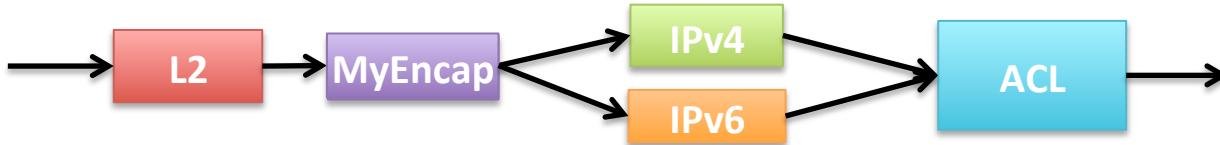
Mapping logical data-plane design to physical resources



Re-configurability



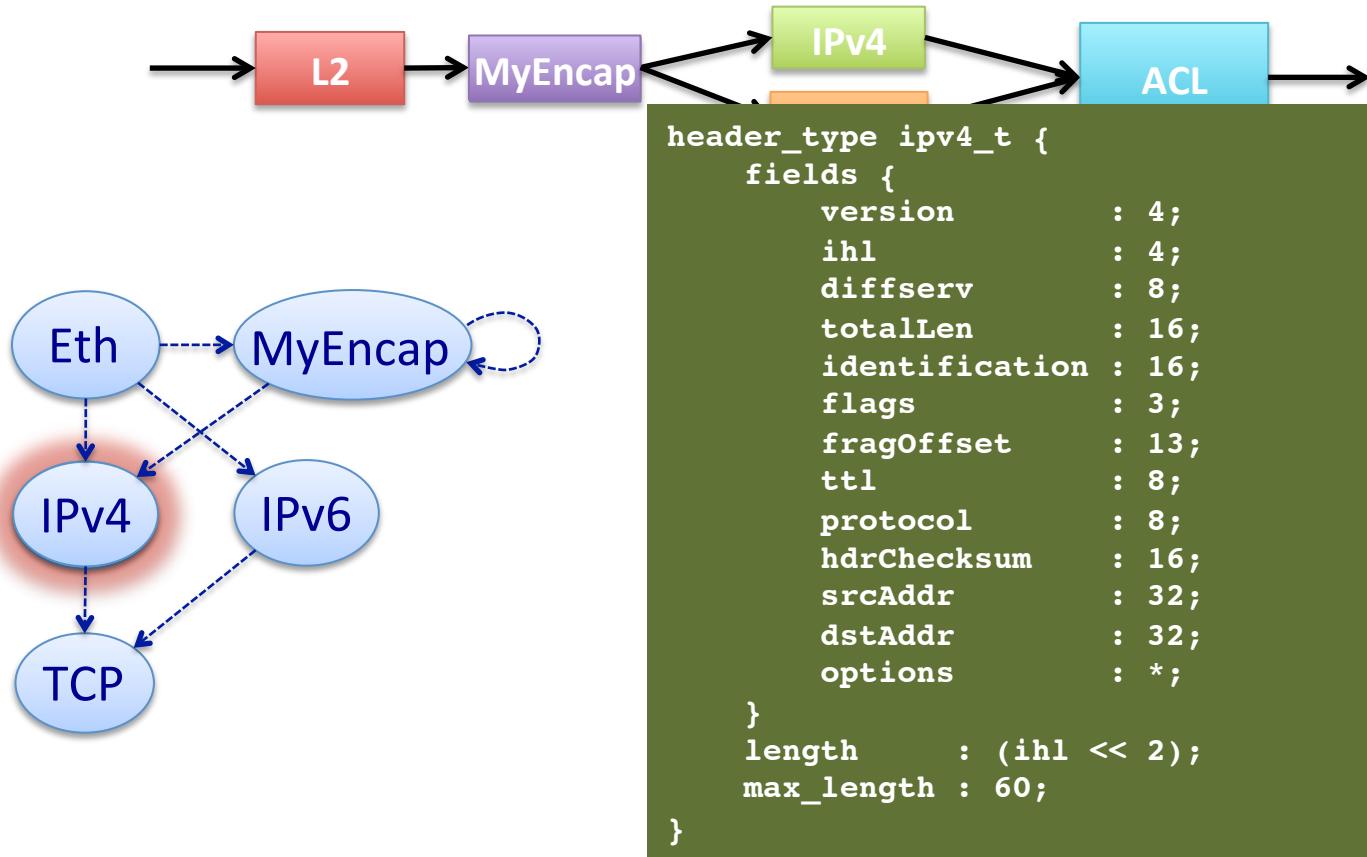
What does a P4 program look like?



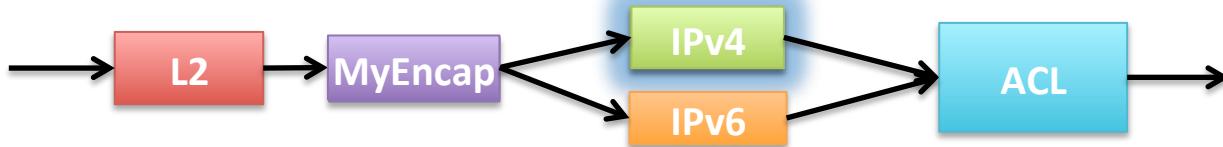
```
header_type ethernet_t {
    fields {
        dstAddr : 48;
        srcAddr : 48;
    }
}
parser parse_ethernet {
    extract(ethernet);
    return select(latest.etherType) {
        0x8100 + parse_vlan;
    }
}

header_type my_encap_t {
    fields {
        foo : 12;
        bar : 8;
        baz : 4;
        qux : 4;
        next_protocol : 4;
    }
}
```

What does a P4 program look like?



What does a P4 program look like?

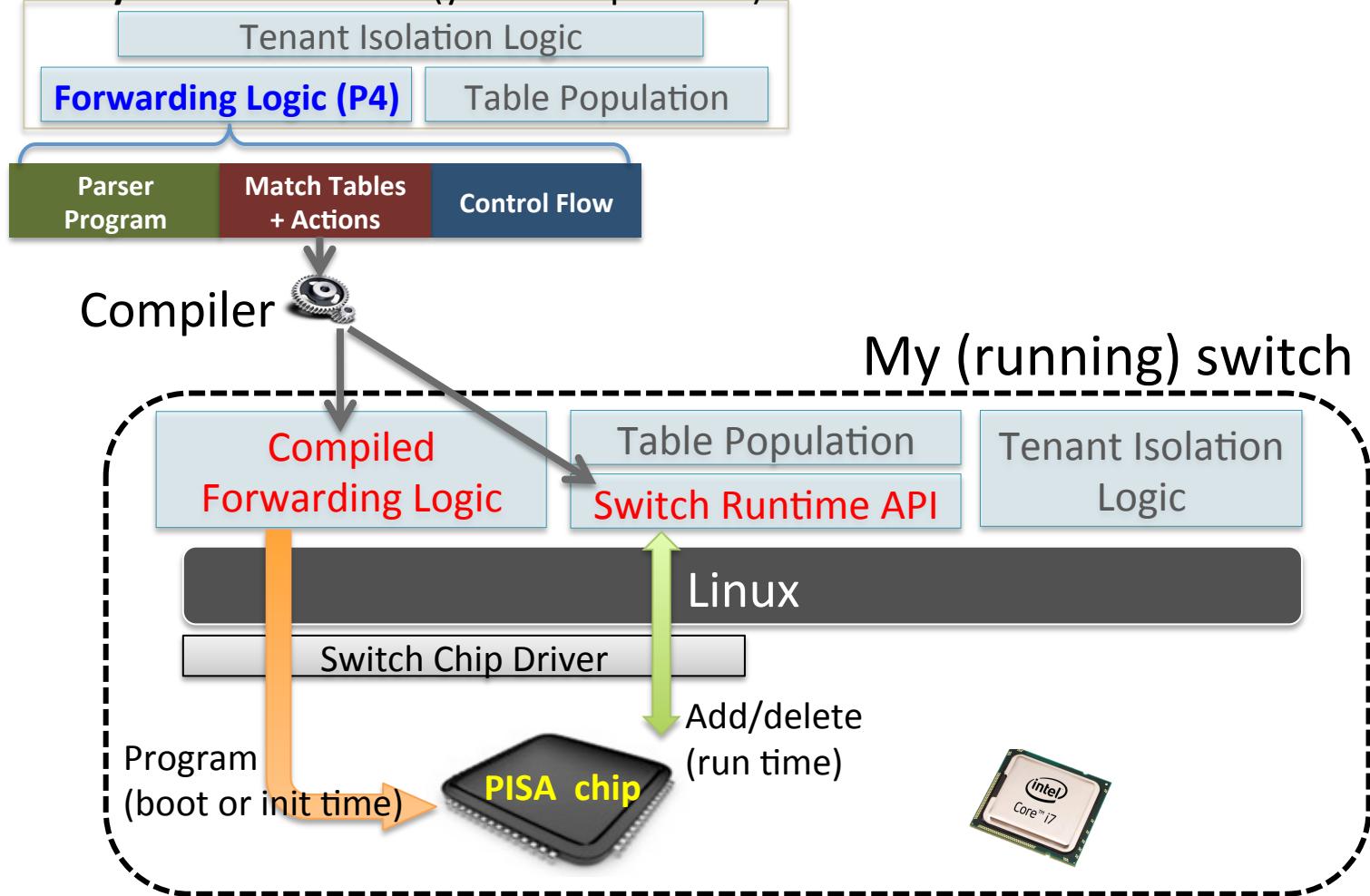


```
table ipv4_lpm
{
    reads {
        ipv4.dstA
    }
    actions {
        set_next_
    }
}
```

```
action set_next_hop(n)
{
    modify_field(meta)
    modify_field(stan)
    add_to_field(ipv4.ttl, -1);
}
```

```
control ingress
{
    apply(l2);
    apply(my_encap);
    if (valid(ipv4)) {
        apply(ipv4_lpm);
    } else {
        apply(ipv6_lpm);
    }
    apply(acl);
    output("eth0", ip_src_ip, ip_dst_ip);
}
```

MyNetVirtualization (your new protocol)



DEMO!

In-band Network Telemetry (INT)

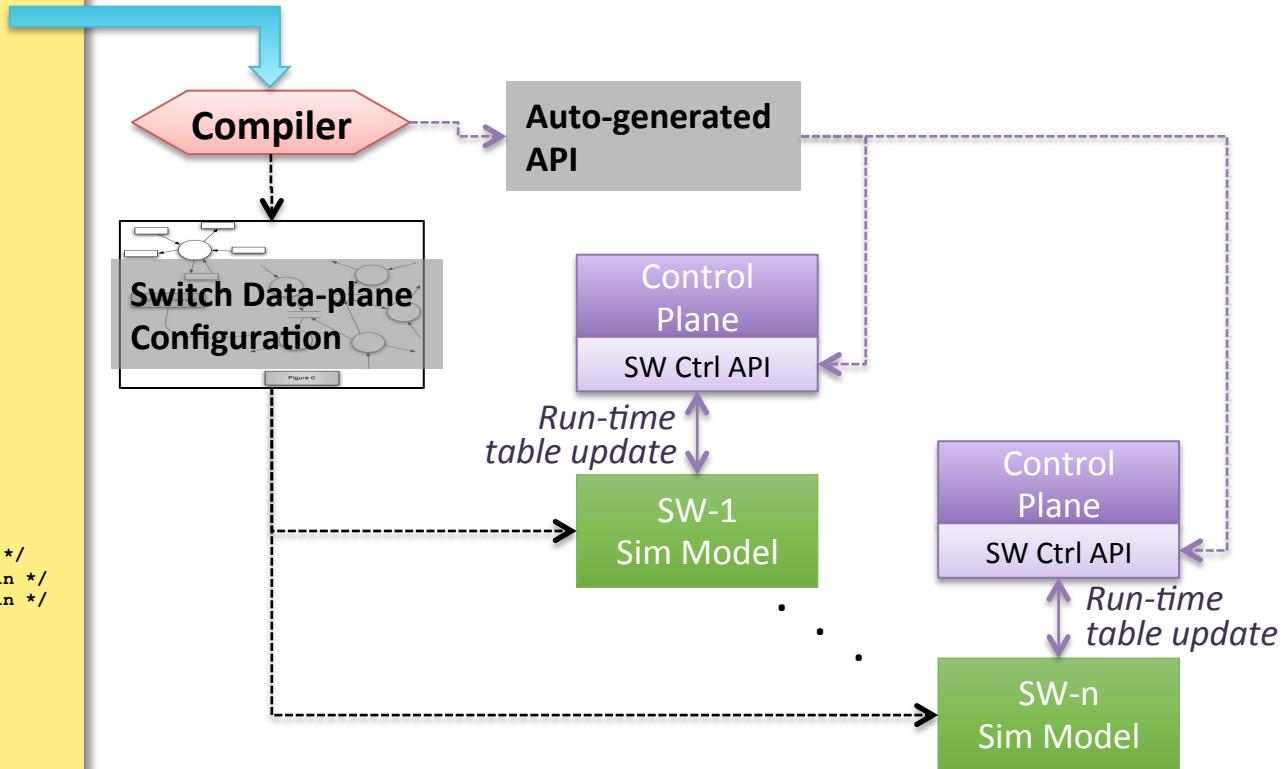
<http://p4.org/p4/inband-network-telemetry/>

Demo Setup

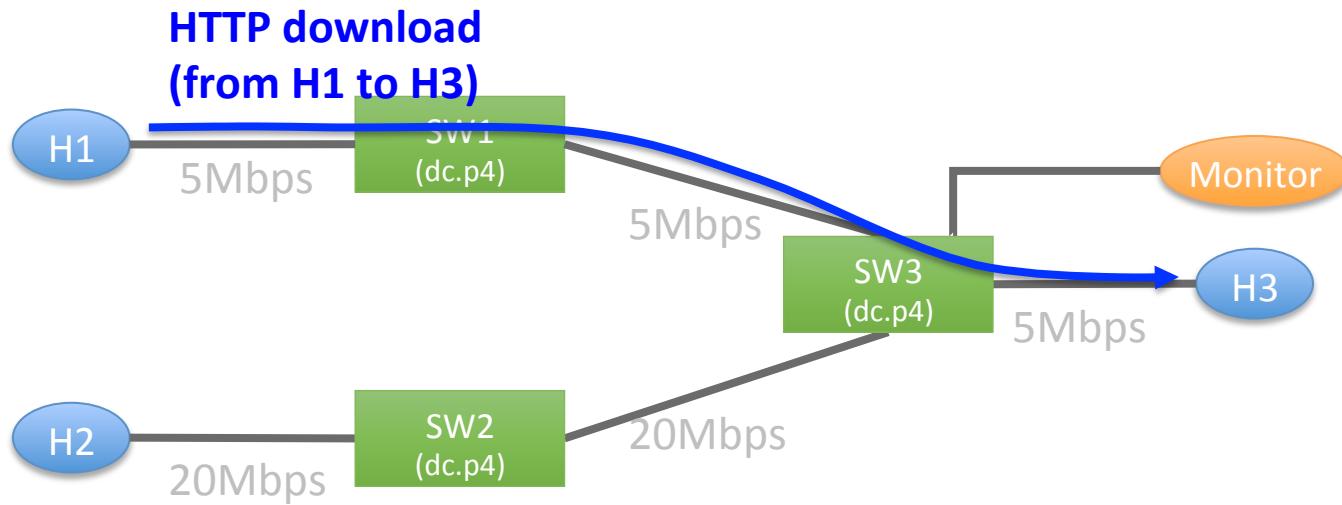
```
/* Reference p4 program
switch.p4
*/
#include "includes/headers.p4"
#include "includes/parser.p4"
/******************
/* Device Personality Selection */
******************/

// #define P_MINIMAL
#define P_BASE
// #define P_MSDC_SPINE
// #define P_MSDC_TOR
// #define P_MSDC_BL
// #define P_INTERNET_ROUTER
...
// Flexible Tables
#define LPM_TABLE_SIZE 16384
#define IPV6_LPM_TABLE_SIZE 4096
#define HOST_TABLE_SIZE 131072
#define IPV6_HOST_TABLE_SIZE 32768
...
header_type routing_metadata_t {
    fields {
        bd : BD_BIT_WIDTH; /* bridge domain */
        vrf : VRF_BIT_WIDTH; /* routing domain */
        v6_vrf : VRF_BIT_WIDTH; /* routing domain */
    }
}

table bridge_domain {
    reads {
        routing_metadata.bd : exact;
    }
    actions {
        nop;      // Not used
        bd_set;
    }
}
```



Scenario: Debugging Long Latency Tail



In-band Network Telemetry in P4

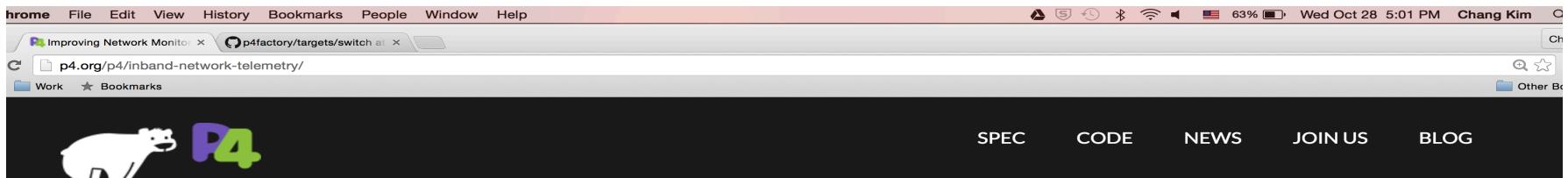
```
table int_table {
    reads {
        ip.protocol;
    }
    actions {
        export_queue_latency;
    }
}

action export_queue_latency (sw_id) {
    add_header(int_header);
    modify_field(int_header.kind, TCP_OPTION_INT);
    modify_field(int_header.len, TCP_OPTION_INT_LEN);
    modify_field(int_header.sw_id, sw_id);
    modify_field(int_header.q_latency,
                 intrinsic_metadata.deq_timedelta);
    add_to_field(tcp.dataOffset, 2);
    add_to_field(ipv4.totalLen, 8);
    subtract_from_field(ingress_metadata.tcpLength,
                       12);
}
```

Add TCP Options &
copy switch ID and queue latency
Into the options

INT Open-source and Spec

- <http://p4.org/p4/inband-network-telemetry/>



Improving Network Monitoring and Management with Programmable Data Planes

By Mukesh Hira & LJ Wobker

Quicklinks : [The INT specification](#) - - - [INT GitHub repository](#) - - - [INT demo video](#)

Compute virtualization and the widespread deployment of virtual machines has led to an extension of the network into the hypervisor. New network virtualization solutions have emerged that enable rapid provisioning of network services – logical switches, load balancers, virtual machines, and network interfaces. Virtual machines can be decoupled from the physical network so as to enable deployment of virtual services over any physical network infrastructure, the only requirement from the physical network being IP connectivity between the hypervisors.

While the decoupling of physical and virtual topologies has advantages, it is important to have some interaction between the physical and virtual switches to allow for end-to-end monitoring of the entire physical + virtual network from a “single pane of glass” and to help in troubleshooting and fault isolation in complex physical + virtual topologies.

We propose methods for various network elements to collect and report their state in real-time, allowing for improved cooperation between the virtual and physical layers without requiring intermediate layers such as CPU driven control planes. The general term we have applied to these methods is INT: Inband Network Telemetry.

Spec

Ref P4 code

Demo

Search the Site...

Search

Recent Posts

- [Improving Network Monitoring and Management with Programmable Data Planes](#)
- [P4 goes to England: SIGCOMM 2015](#)
- [P4 language evolution](#)
- [P4 and Open vSwitch](#)
- [1st P4 Workshop on June 4, 2015](#)

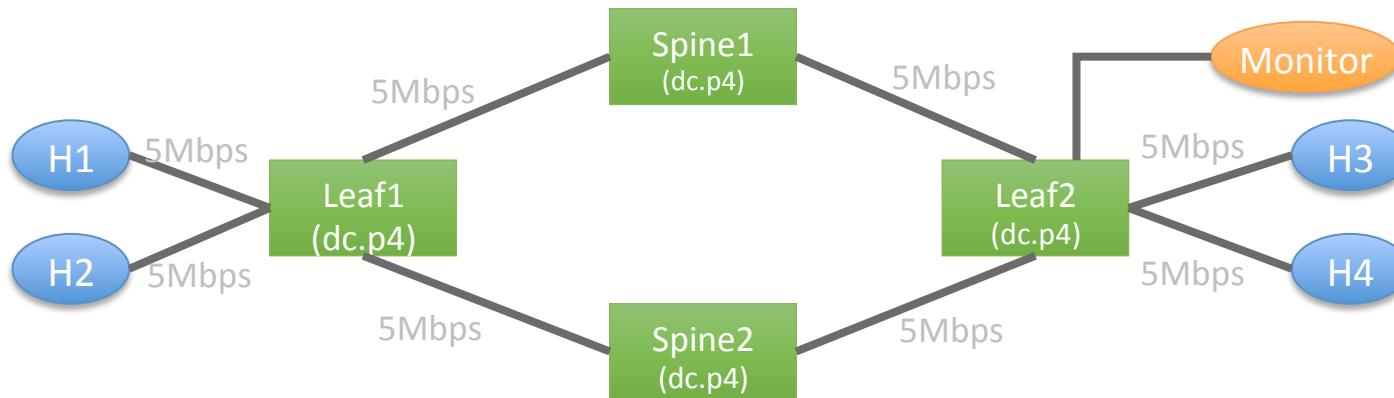
Archives

[September 2015](#)

[July 2015](#)

[June 2015](#)

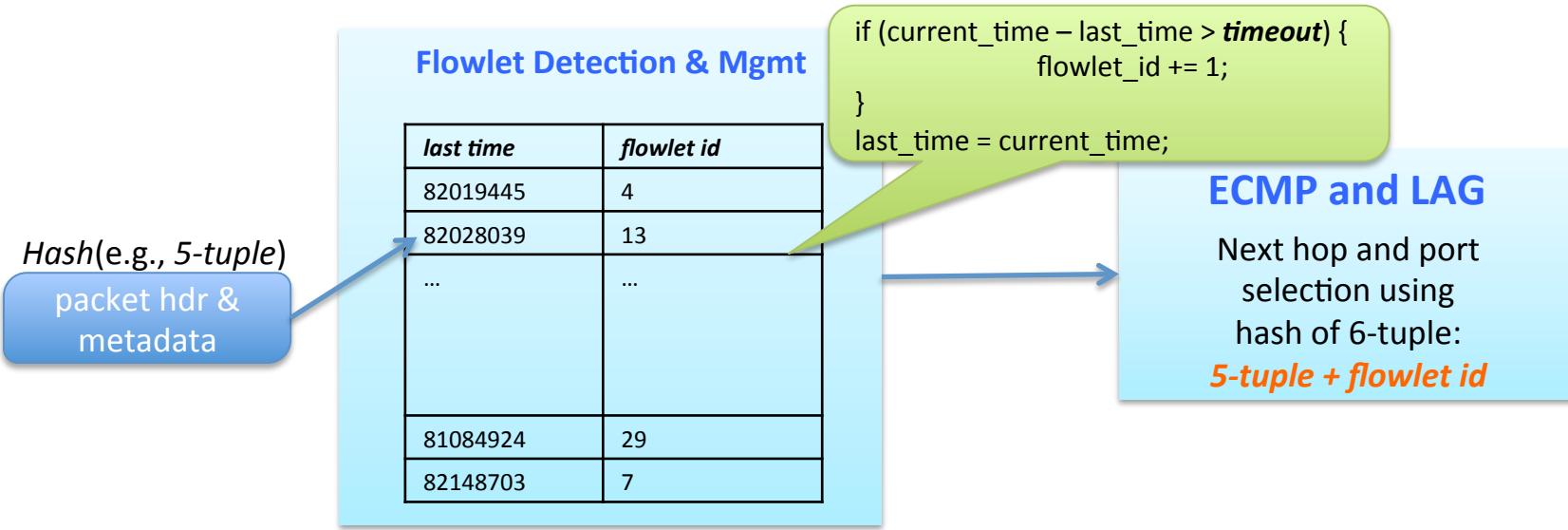
Scenario: Resolving Traffic Split Problem



Traffic Demand

H1 → H3	TCP
H2 → H4	TCP

Simple Flowlet Switching



- Hash collisions are **infrequent**
 - Flowlets are very short, and hence # of active flowlets is only up to tens of thousands
- Hash collisions are **inconsequential**
 - Flowlets are very small; sub-optimality caused by collisions is negligible
- Time out is **configurable**
 - Range can be as low ~1 us to ~100ms; a few hundred usec seems most suitable for DCs

Flowlet Switching in P4

```
field_list_calculation flowlet_map_hash {
    input {
        l3_hash_fields;
    }
    algorithm : crc16;
    output_width : FLOWLET_MAP_SIZE;
}

register flowlet_lasttime {
    width : 32;
    instance_count : 8192;
}

register flowlet_id {
    width : 16;
    instance_count : 8192;
}

action lookup_flowlet_map() {
    modify_field_with_hash_based_offset(ingress_metadata.flowlet_map_index, 0,
        flowlet_map_hash, FLOWLET_MAP_SIZE);

    register_read(ingress_metadata.flowlet_id,
        flowlet_id, ingress_metadata.flowlet_map_index);

    modify_field(ingress_metadata.flow_ipg, intrinsic_metadata.ingress_global_timestamp);
    register_read(ingress_metadata.flowlet_lasttime,
        flowlet_lasttime, ingress_metadata.flowlet_map_index);
    subtract_from_field(ingress_metadata.flow_ipg, ingress_metadata.flowlet_lasttime);

    register_write(flowlet_lasttime, ingress_metadata.flowlet_map_index,
        intrinsic_metadata.ingress_global_timestamp);
}

table flowlet {
    actions {
        lookup_flowlet_map;
    }
}
```

~ 100 lines of P4 code

```
field_list flowlet_l3_hash_fields {
    ingress_metadata.lkp_ipv4_sa;
    ingress_metadata.lkp_ipv4_da;
    ingress_metadata.lkp_ip_proto;
    ingress_metadata.lkp_l4_sport;
    ingress_metadata.lkp_l4_dport;
    ingress_metadata.flowlet_id;
}

field_list_calculation flowlet_ecmp_hash {
    input {
        flowlet_l3_hash_fields;
    }
}

action set_ecmp_select(ecmp_base, ecmp_count) {
    modify_field_with_hash_based_offset(ingress_metadata.ecmp_offset, ecmp_base,
        ecmp_hash, ecmp_count);
}

action set_flowlet_ecmp_select(ecmp_base, ecmp_count) {
    modify_field_with_hash_based_offset(ingress_metadata.ecmp_offset, ecmp_base,
        flowlet_ecmp_hash, ecmp_count);
}

table ecmp_group {
    reads {
        ingress_metadata.ecmp_index : exact;
    }
    actions {
        set_ecmp_select;
        set_flowlet_ecmp_select;
    }
    size : ECMP_GROUP_TABLE_SIZE;
}
```

This will accelerate innovations

- Areas of innovation (just to name a few)
 - Reducing feature set → “*biggest bang for buck*”
 - Network monitoring, analysis, and diagnostics
 - Tunnel-splicing gateways
 - Load balancing
 - Attack detection and mitigation
 - Host-stack offloading
- Across various types of network devices
 - Hardware and software devices (OVS, eBPF, etc.)
 - Switches, NICs, middle-boxes, etc.

This will create lots of significant R&D opportunities

- Novel network and protocol designs
 - Take advantage of unprecedented amount of visibility and control into the network
 - Joint design and optimization between hosts and network
- Development tools
 - Compilers, debuggers, static analyzers, profilers, etc.
 - P4-programming assistance tools
- Network verification, test, and simulation
- Many more ...

P4.org – P4 Language Consortium

The screenshot shows the P4.org website. At the top, there's a navigation bar with icons for back, forward, search, and user profile. The address bar shows 'p4.org'. Below the bar is a header with a polar bear icon and the text 'P4'. A blue callout bubble points from the right towards the center of the page, containing the text 'Maintains the language spec'. The main content area features a large headline 'It's time to say "Hello Network"' over a blurred background image of network equipment. To the left, there are three sections with purple headers: 'Protocol Independent', 'Target Independent', and 'Field Reconfigurable', each with a brief description. To the right, there's a code snippet in P4 language and a green 'TRY IT' button.

Maintains the language spec

Protocol Independent
P4 programs specify how a switch processes packets.

Target Independent
P4 is suitable for describing everything from high-performance forwarding ASICs to software switches.

Field Reconfigurable
P4 allows network engineers to change the way their switches process packets after they are deployed.

```
table routing {  
    reads {  
        ipv4.dstAddr : lpm;  
    }  
    actions {  
        do_drop;  
        route_ip4;  
    }  
    size: 2048;  
}  
  
control ingress {  
    apply(routing);  
}
```

TRY IT Get the code from GitHub

P4.org – P4 Language Consortium

The screenshot shows the P4.org website with a blue callout bubble overlaid on the center-left area. The callout bubble contains the text: "Maintains key dev tools under Apache license". Inside the main content area, there is a list of five items:

- Reference P4 programs
- Compiler
- P4 software switch
- Test framework

The background of the website shows a blurred image of a network switch and some text. At the bottom right of the website, there is a green button with white text that says "GET IT Get the code from GitHub".

P4.org – P4 Language Consortium

The screenshot shows the P4.org website interface. At the top, there's a navigation bar with links for SPEC, CODE, NEWS, JOIN US (which is highlighted with a blue border), and BLOG. Below the navigation is a section titled "Hello Network" featuring a snippet of P4 code:

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

A large blue callout bubble in the center-right area contains the text: "Open for participation by any individuals or organizations".

INDUSTRY MEMBERS

Logos for the following companies are displayed in a grid:

- AEP NYX™
- Alibaba Group 阿里巴巴集团
- Atomic Rules
- at&t
- Baidu 百度
- BAREFOOT NETWORKS
- BROADCOM 做任何事情
- BROCADE
- CAVIUM
- CISCO
- CORSA
- DELL
- EZchip
- HUAWEI
- INTEL
- JUNIPER NETWORKS
- MARVELL
- Mellanox TECHNOLOGIES
- Microsoft
- NETRONOME
- PLUMgrid
- XILINX

UNIVERSITY MEMBERS

Logos for the following universities are shown:

- PRINCETON UNIVERSITY
- Stanford University
- UNIVERSITÉ DU LUXEMBOURG
- Università della Svizzera Italiana

P4.org – P4 Language Consortium

The screenshot shows a web browser window for the P4.org website. The header includes a logo of a white bear with the letters 'P4' next to it, and navigation links for SPEC, CODE, NEWS (which is highlighted with a blue border), JOIN US, and BLOG. Below the header, a large image features the text "It's time to say 'Hello Network!'" and a subtext explaining P4 as a domain-specific programming language. A blue callout box in the foreground contains the text "Hosts P4 workshops and boot camps" and a bulleted list: "• 2nd P4 Workshop on Nov/18 at Stanford" and "• 1st P4 Boot camp in the same week". At the bottom right, there is a green button labeled "TRY IT" with a download icon, and the text "Get the code from GitHub".

It's time to say "Hello Network!"

P4 is a domain-specific programming language to describe the behavior of network hardware. It's designed to be simple, yet powerful enough to express complex behaviors.

Hosts P4 workshops and boot camps

- 2nd P4 Workshop on Nov/18 at Stanford
- 1st P4 Boot camp in the same week

SWITCHES PROCESS PACKETS AFTER THEY ARE DEPLOYED.

TRY IT Get the code from GitHub

Switch.p4: A typical L2/L3 switch in P4

- Feature set
 - Basic L2 switching: MAC learning, VLAN, flooding, and STP
 - Basic L3 routing: IPv4 and IPv4 routing with VRF
 - LAG and ECMP
 - Tunneling: VXLAN, NVGRE, Geneve, and GRE
 - Basic ACL: MAC and IP ACLs
 - Unicast RPF
 - MPLS: LER, LSR, IPVVPN, VPLS, and L2VPN
 - Host interface
 - Mirroring: Ingress and egress mirroring with ERSPAN
 - Counters/Statistics
- More features coming soon
 - IP Multicast, NAT, QoS, Ingress policing, etc.

Summary

- In next few years **data-plane programmability** will quickly become commonplace
- This will accelerate innovations in networking
- No more “black boxes” in “white boxes”
- As **the common industry-wide forwarding language, P4** will play crucial roles
- Lots of R&D and business opportunities opening up – join and contribute!