WE DON’T OFTEN THINK ABOUT THE NETWORK AS A “COMPUTING DEVICE”

For most of us, the network is just the Internet, or perhaps a virtually private cloud (VPC).

But modern networks are actually programmable.

How does this work, and what are cloud companies like Amazon and Microsoft using this for?
Network devices include
- Network interface cards (NICs)
- Switches (they support a 1:1 form of packet movement, very rigid)
- Routers (they look at the destination and send the packet on a good path to reach that destination. Very flexible because the routing table can be updated at runtime).

We are already “programming” a network if we configure a switch or load a routing table into a router.
OTHER INTERESTING DEVICES

Firewalls: They use “rules” to block attacks like DDoS traffic or spam.

Network address translation devices (NAT boxes): They map from one network address range to a different one, and might also map port numbers or even byte ranges.

VLAN boxes: They automatically create VPNs and VPCs.

Cryptography boxes: They encrypt and decrypt “on the wire”
Network operators use automated tools to visualize a cloud (datacenter) network and its load levels, and to reach in and reconfigure elements to address problems.

These tools often can issue program-triggered alarms: “Warning, network overload on segment T:5-3.B. Packet drop rate spiking!”

Automated tools might route around broken hardware to mask issues.
... BUT THEY CAN’T DO FANCIER KINDS OF PROGRAMS

Suppose that I wanted to do fine-grained monitoring of just the traffic to a specific VLAN, or even to some single microservice within my network.

Or I might want to move part of a MapReduce task right into the network itself and have it compute the “reduce” functions with no help from the host.

Or we might want to create a very flexible new form of routing that dynamically selects specific packets and sends them to particular machines.
A NUMBER OF PROPOSALS HAVE BEEN MADE

OpenFlow: A router-control API that can support fancier network behavior

P4: A new language for writing programs that run directly on the routers
WE WILL LOOK AT SLIDES ON THE P4 LANGUAGE

Mihai Budiu was a Cornell PhD student, but he moved with a faculty member who went to CMU and finished up there.

He helped create the Microsoft LINQ technology we learned about. Then when Microsoft Research Silicon Valley closed, he moved to VMWare.

At VMWare he leads a P4 research group.
P4: specifying data planes

http://P4.org

VMware Techtalk
March 30, 2017

Mihai Budiu
VMware Research Group
About Myself

• Ph.D. from Carnegie Mellon

• Researcher at Microsoft Research, Silicon Valley
  • Distributed systems, security, compilers, cloud platforms, machine learning, visualization

• Software engineer at Barefoot Networks
  • Design and implementation of P4

• Researcher at VMware Research Group
  • Big data, P4
• P4 & Programmable networks
  • Why should you care?
• An introduction to $\text{P4}_{16}$
  • P4 limitations
• Conclusions
Networking 101

Data packets

routers
Control and Data Planes

Switch architecture

Control plane

Data plane

Interfaces

packets
Traditional switch architecture

- Control plane
  - Control-plane CPU
  - Table management
- Data plane
  - Switch ASIC
  - Look-up tables (policies)
Software-Defined Networking

Policies/signaling

Controller

Dumb control plane

Data plane
The P4 world

Dumb control plane

Programmable data plane

Upload program

Policies/signaling

SW: P4
Not just for switches!

Programmable switches
FPGA switches
Programmable network cards
Software switches
Hypervisor switches
You name it…
How is this possible?

Most useful if you have your own network playground
Data-planes

• From now on in this presentation we only talk about the data-plane
• We expect that SDN will continue to evolve the control-plane
WHY SHOULD YOU CARE?
Isn’t Open-Flow Enough?

Open-flow has never been enough: it keeps changing to describe new protocols.
vmware has lots at stake

• NSX is about programmable networks
  • Flexibility in networking
  • We are an industry leader

• P4 will change the dynamics in the industry
  • Device manufacturer ≠ device programmer
  • Many network capabilities exposed to software
Protocols = programs

• VxLAN: 175 lines of P4
  • Took 4 years from proposal to wide availability
• NVGRE: 183 lines of P4

M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, J. Rexford, *PISCES: A Programmable, Protocol-Independent Software Switch*
SIGCOMM 2016
• 40 times reduction in the size of the OvS parser
• Much easier to add new protocols
• Same performance
Use only what you need

- IETF has issued thousands of RFCs
- Switch RAM and CPU is very expensive
- Network operators can remove protocols
- Simpler troubleshooting
Network monitoring

In-Band Network Telemetry (INT)

**Improving Network Monitoring and Management with Programmable Data Planes**

*By Mukesh Hira & LJ Wobker*
Optimize your network

- Push application functionality in the network
- High speed

_Paxos Made Switch-y_
Huynh Tu Dang, Marco Canini, Fernando Pedone, Robert Soulé
CCR April 2016
Network = software

• Use *software* engineering principles and tools
• Upgrade your network at any time
• Protocols = intellectual property
P4.org Consortium

Carriers, cloud operators, chip co.s, networking, systems, universities, start-ups
AN INTRODUCTION TO $P_{16}$
Language evolution

**P4: Programming Protocol-Independent Packet Processors**

Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, David Walker *ACM SIGCOMM Computer Communications Review (CCR). Volume 44, Issue #3 (July 2014)*


P4_{16} spec, reference implementation and tools released in December 2016.
P4 Community

• [http://github.com/p4lang](http://github.com/p4lang)
• [http://p4.org](http://p4.org)

• Mailing lists
• Workshops
• P4 developer days

• Academic papers (SIGCOMM, SOSR)
Available Software Tools

• Compilers for various back-ends
  • Netronome chip, Barefoot chip, eBPF, Xilinx FPGA (open-source and proprietary)
• Multiple control-plane implementations
  • SAI, OpenFlow
• Simulators
• Testing tools
• Sample P4 programs
• Tutorials
P4\textsubscript{16}

- Most recent revision of P4
- Similar to C; strongly typed
- Currently in draft form
P4_{16} data plane model

Programmable blocks

Data plane

Fixed function
Example packet processing pipeline

Packet (byte[]) → Programmable parser → Headers

Programmable match-action units → Metadata

Headers → Programmable reassembly → Packet

Payload → Queueing/switching
Language elements

- Programmable parser
  - State-machine;
  - bitfield extraction

- Programmable match-action units
  - Table lookup; bitfield manipulation;
  - control flow

- Programmable reassembly
  - Bitfield reassembly

- Data-types
  - Bitstrings, headers,
    structures, arrays

- Target description
  - Interfaces of programmable blocks

- External libraries
  - Support for custom accelerators

User target
Data Types

typedef bit<32> IPv4Address;

header IPv4_h {
    bit<4> version;
    bit<4> ihl;
    bit<8> tos;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    bit<8> protocol;
    bit<16> hdrChecksum;
    IPv4Address srcAddr;
    IPv4Address dstAddr;
}
// List of all recognized headers

struct Parsed_packet {
    Ethernet_h ethernet;
    IPv4_h ip;
}
Parsing = State machines

```
parser Parser(packet_in b, out Parsed_packet p) {
    state start {
        b.extract(p.ethernet);
        transition select(p.ethernet.type) {
            0x0800: parse_ipv4;
            default: reject;
        }
    }
    state parse_ipv4 {
        b.extract(p.ip);
        transition accept;
    }
}
```
Actions

• ~ Objects with a single method.
• Straight-line code.
• Reside in tables; invoked automatically on table match.

```
action Set_nhop(IPv4Address ipv4_dest, PortId port) {
    nextHop = ipv4_dest;
    outCtrl.outputPort = port;
}
```

```
class Set_nhop {
    IPv4Address ipv4_dest;
    PortId port;
    void run() {
        nextHop = ipv4_dest;
        outCtrl.outputPort = port
    }
}
```

Java/C++ equivalent code.
Tables

- HashMap<K, Action>

```
table ipv4_match() {
    key = { headers.ip.dstAddr: exact; }
    actions = { Drop_action; Set_nhop; }
    default_action = Drop_action;
}
```

Populated by the control plane

<table>
<thead>
<tr>
<th>dstAddr</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0</td>
<td>drop</td>
</tr>
<tr>
<td>10.0.0.1</td>
<td>Set_nhop(10.4.3.4, 4)</td>
</tr>
<tr>
<td>224.0.0.2</td>
<td>drop</td>
</tr>
<tr>
<td>192.168.1.100</td>
<td>drop</td>
</tr>
<tr>
<td>10.0.1.10</td>
<td>Set_nhop(10.4.2.1, 6)</td>
</tr>
</tbody>
</table>
Match-Action Processing

Control plane

headers & metadata

Lookup key

Lookup table

key

action

Code & data

Action

action code

Execute

action data

headers & metadata
Control-Flow

```
control Pipe(inout Parsed_packet headers,
    in InControl inCtrl, // input port
    out OutControl outCtrl) { // output port
    IPv4Address nextHop; // local variable

    action Drop_action() { ... }
    action Set_nhop(...) { ... }
    table ipv4_match() { ... }
    ...

    apply { // body of the pipeline
        ipv4_match.apply();
        if (outCtrl.outputPort == DROP_PORT) return;
        dmac.apply(nextHop);
        if (outCtrl.outputPort == DROP_PORT) return;
        smac.apply();
    }
}
```
Packet Reassembly

Convert headers back into a byte stream. Only valid headers are emitted.

```
control Deparser(in Parsed_packet p, packet_out b) {
  apply {
    b.emit(p.ethernet);
    b.emit(p.ip);
  }
}
```
P4 Program structure

#include <core.p4> // core library
#include <target.p4> // target description
#include "library.p4" // library functions
#include "user.p4" // user program
Architecture declaration

Provided by the target manufacturer

```plaintext
struct input_metadata {
  bit<12> inputPort;
}
struct output_metadata {
  bit<12> outputPort;
}

parser Parser<H>(packet_in b, out H headers);

H = user-specified header type

control Pipeline<H>({inout H headers,
  in input_metadata input,
  out output_metadata output});

control Deparser<H>({in H headers, packet_out p});

package Switch<H>({Parser<H> p, Pipeline<H> p, Deparser<H> d});
```
Support for custom “accelerators”

```c
extern bit<32> random();

extern Checksum16 {
    void clear();              // prepare unit for computation
    void update<T>(in T data); // add data to checksum
    void remove<T>(in T data); // remove data from checksum
    bit<16> get();             // get the checksum for data added
}
```

External function

External object with methods. Methods can be invoked like functions. Some external objects can be accessed from the control-plane.
Execution model

- When a block is invoked (parser, control) it executes to completion on a separate thread
  - All local variables are thread-local
  - Only inter-thread communication possible through extern objects and functions
- Execution triggered by outside event (e.g., packet arrival)
- Actions execute atomically
  - @atomic annotation for further user-level control
P4 software workflow

- P4 program
- P4 architecture model
- Data plane
  - Table
  - External objects
- Control-plane
  - API
  - Control signals
- Manufacturer supplied
- User-supplied

P4 compiler
Dataplane runtime
API
LOAD
P4 LIMITATIONS
Limitations of P4\textsubscript{16}

- The core P4 language is very small
  - Highly portable among many targets
  - But very limited in expressivity
- Accelerators can provide additional functionality
  - May not be portable between different targets
  - Under construction: library of standard accelerators
What is missing

• Floating point
• Pointers, references
• Data structures, recursive data types
• Dynamic memory management
• Loops, iterators (except the parser state-machine)
• Recursion
• Threads

• => Constant work/byte of header
What cannot be done in (pure) P4

- Multicast or broadcast
- Queueing, scheduling, multiplexing
- Payload processing: e.g., encryption
- Packet trailers
- Persistent state across packets
- Communication to control-plane
- Inter-packet operations (fragmentation and reassembly)
- Packet generation
- Timers
How are these done?

• Multicast, broadcast, queueing, scheduling, multiplexing
  • By target device, controlled by P4 metadata
• Persistent state across packets (e.g. per-flow state)
  • External objects: registers, counters, meters
• Communication to control-plane
  • External objects: learning providers
• Packet generation
  • Control-plane, or external objects
• Reassembly, trailers:
  • Not currently done
P4 is not...

- Active networking: a way for packets to inject new code
- Programming the control plane: that is Software-Defined Networking
- A tool for third parties to program the network
- A language for:
  - distributed computations
  - network middleboxes (NFV)
  - network operating systems
Why use P4?

• **It is a language:**
you can specify the data-plane precisely

• **Expressive:**
express many packet-forwarding policies

• **High-level, type-safe:**
compiler-managed abstract resources

• **Software:**
treat network devices like software

• **Killer app:**
network monitoring
Creating a Programming Language Interface in a place where there wasn’t one.