PISCES: A Programmable, Protocol-Independent Software Switch

SIGCOMM 2016

Sean Choi
Outline

- Motivations and history of SDN
- Use cases of SDN
- SDN and the change in the networking stack
- What is P4 and Protocol Independent Packet Processing?
- Introducing PISCES
Outline

• Motivations and history of SDN
  • Use cases of SDN
  • SDN and the change in the networking stack
  • What is P4 and Protocol Independent Packet Processing?
• Introducing PISCES
What is Software-Defined Networking (SDN)?
Software Defined Network

A network in which the control plane is physically separate from the forwarding plane.

and

A single control plane controls several forwarding devices.

(That’s it)
Software Defined Network (SDN)
Intended consequences...
1. Put network owners and operators in control.
2. Networks that cost less: simpler, streamlined hardware.
3. Networks that cost less to operate (fewer features).
4. Networks that evolve faster.
Origins of SDN

Martin Casado
The Ethane Project
[SIGCOMM 2007]
How difficult is it to define all network operations in software, outside the data path?

Stanford campus

2006

35,000 users
10,000 new flows/sec
137 network policies

2,000 switches
2,000 switch CPUs
Crazy question: What if software decides whether to accept each flow, and how to route it?
How many $400 controller servers do we need to service 35,000 users?
Less than One
If we **can** define network behavior outside the data path, then eventually we **will**.
What happened next

SDN, OpenFlow, Open vSwitch, Network Virtualization, ...
About 250 startups so far.

Source: SDX Central
Outline

- Motivations and history of SDN
- Use cases of SDN
- SDN and the change in the networking stack
- What is P4 and Protocol Independent Packet Processing?
- Introducing PISCES
SDN use cases
Routing and NFV
function Dijkstra(Graph, source):
  for each vertex v in Graph:
    dist[v] := infinity;
    previous[v] := undefined;
  dist[source] := 0;
  Q := the set of all nodes in Graph;
  while Q is not empty: // The main loop
    u := vertex in Q with smallest distance in dist[];
    remove u from Q;
    if dist[u] = infinity:
      break;
    for each neighbor v of u:
      alt := dist[u] + dist_between(u, v);
      if alt < dist[v]:
        dist[v] := alt;
        previous[v] := u;
        decrease-key v in Q;
  return dist[], previous[];
end function
1. Figure out which routers and links are present.
2. Run Dijkstra’s algorithm to find shortest paths.

"If a packet is going to B, then send it to output 3"
1. Figure out which routers and links are present.
2. Run Dijkstra’s algorithm to find shortest paths.

Network Working Group
Request for Comments: 2328
STD: 54
Obsoletes: 2178
Category: Standards Track

OSPF Version 2

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (1998). All Rights Reserved.

Abstract

This memo documents version 2 of the OSPF protocol. OSPF is a link-state routing protocol. It is designed to be run internal to a single Autonomous System. Each OSPF router maintains an identical database describing the Autonomous System’s topology. From this database, a routing table is calculated by constructing a shortest-path tree.
Specialized Hardware

Dijkstra
Network Map
OS
95%
5%

Global Network Map
Network OS
Packet Forwarding
Packet Forwarding
Packet Forwarding
Packet Forwarding
Network Function Virtualization (NFV)
Network Function Virtualization (NFV)
Dijkstra

IS-IS

BGP

MPLS

NFV

Global Network Map

Network OS

Forwarding

Forwarding

Forwarding

Forwarding
Outline

• Motivations and history of SDN
• Use cases of SDN
• SDN and the change in the networking stack
• What is P4 and Protocol Independent Packet Processing?
• Introducing PISCES
Tens of millions of lines of code.
Closed, proprietary, outdated.
Billions of gates.
Power hungry and bloated.

Hundreds of protocols
7,000 RFCs
Vertically integrated
Closed, proprietary
Slow innovation

Horizontal
Open interfaces
Rapid innovation
Vertically integrated
Closed, proprietary
Slow innovation

Horizontal
Open interfaces
Rapid innovation
I can customize my networks!...

1. See what my forwarding plane is doing.
2. Quickly deploy new protocols
3. Put expensive middlebox functions into the network.
5. Try out beautiful new ideas. Tailor my network to meet my needs.

Not Really...
What about the fixed function switch?

“This is how I process packets”
Problems: Fixed function switch

1. Slow innovation
   Several months or years to add a new feature or protocol

2. Inefficient
   Match tables hard-wired to specific purpose

3. Complicated
   Switch implements superset of all features

4. Leads to bottom-up design
   Frustrating for programmers
What we want: Programmable Switch

“This is how the switch must process packets”

P4 can help us do this!
Outline

• Motivations and history of SDN
• Use cases of SDN
• SDN and the change in the networking stack
• What is P4 and Protocol Independent Packet Processing?
• Introducing PISCES
P4

P4: Programming Protocol-Independent Packet Processors

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

Pat Bosshart, Glen Gibb, Martin Izzard, and Dan Talayco (Barefoot Networks), Dan Daly (Intel), Nick McKeown (Stanford), Cole Schlesinger and David Walker (Princeton), Amin Vahdat (Google), and George Varghese (Microsoft)

www.p4.org
Phases for Protocol-Independent Packet Processing

**Phase 0.** Initially, the switch does not know what a protocol is, or how to process packets (Protocol Independence)

**Phase 1.** We tell the switch how we want it to process packets (Configuration)

**Phase 2.** The switch runs (Run-time)
Three Goals

Protocol independence
  – Configure a packet parser
  – Define a set of typed match+action tables

Target independence
  – Program without knowledge of switch details
  – Rely on compiler to configure the target switch

Reconfigurability
  – Change parsing and processing in the field
The Abstract Forwarding Model

Initially, a switch is unprogrammed and does not know any protocols.
P4 in Detail

- Headers and Fields
- The Parser
- Match+Action Tables
- Control flow
Headers and Fields

Header Fields: Ethernet

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

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

Metadata

```c
header_type standard_metadata_t {
    fields {
        ingress_port : 32;
        packet_length : 32;
        ingress_timestamp : 32;
        egress_spec : 32;
        egress_port : 32;
        egress_instance : 32;
    }
}

metadata standard_metadata_t std_metadata;
```
The Parser

Parser: Ethernet

```c
parser parse_ethernet {
    extract(ethernet);
    return switch(latest.etherType) {
        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;
    }
}
```

Parser: IPv4

```c
parser parse_ipv4 {
    extract(ethernet);
    return switch(latest.etherType) {
        PROTO_TCP : parse_tcp;
        PROTO_UDP : parse_udp;
        ...
    }
}
```
Match+Action Tables

Specifies
- Which fields to examine in each packet
- Actions that may be applied (by rule)
- Table size (optional)

Match+Action Table: VLAN

```python
table port_vlan {
    reads {
        std_metadata.ingress_port : exact;
        vlan_tag[OUTER_VLAN].vid : exact;
    }
    actions {
        drop, ing_lif_extract;
    }
    size 16384;
}
```

Match+Action Table: Unicast RPF

```python
table urpf_check {
    reads {
        routing_metadata.bd : ternary;
        ipv4.dstAddr : ternary;
    }
    actions {
        urpf_clear, urpf_set;
    }
}
```
Actions

Built from primitives
– modify field (packet header or metadata)
– add/remove header
– clone/recirculate
– counter/meter/stateful memory operations

Parallel semantics

Actions: LIF Extract

```c
/* Ingress logical interface setup */
action ingress_lif_extract(i_lif, bd, vrf, v4term, v6term, igmp_snoop) {
    modify_field(route_md.i_lif, i_lif);
    modify_field(route_md.bd, bd);
    modify_field(route_md.vrf, vrf);
    modify_field(route_md.ipv4_term, v4term, 0x1);
    modify_field(route_md.ipv6_term, v6term, 0x1);
    modify_field(route_md.igmp_snoop, igmp_snoop, 0x1);
}
```
Control Flow

Control Flow: Ingress

control ingress {
    apply_table(port);
    apply_table(bcast_storm);
    apply_table(ip_sourceguard);
    if (valid(vlan_tag[0])) {
        apply_table(port_vlan);
    }
    apply_table(bridge_domain);
    if (valid(mpls_bos)) {
        apply_table(mpls_label);
    }
    retrieve_tunnel_vni();
    if (valid(vxlan) or valid(genv) or valid(nvgre)) {
        apply_table(dest_vtep);
        apply_table(src_vtep);
    }
    . . . .
}
The P4 View of the World

A compiler per target

Switch configuration

P4 Program/Library

Compiler

Switch

Packet Forwarding Engine
PISCES: First Ever P4 to vSwitch Compiler

P4 Program/Library

PISCES Compiler

Open vSwitch

Packet Forwarding Engine
Outline

• Motivations and history of SDN
• Use cases of SDN
• SDN and the change in the networking stack
• What is P4 and Protocol Independent Packet Processing?
• Introducing PISCES
Importance of Software Switches

Network switches are essential in modern data centers for efficient network traffic management. In this context, software switches play a critical role by providing flexibility and scalability. They are used in conjunction with hypervisors and virtual machines (VMs) to offer a robust and versatile network infrastructure.

The diagram illustrates the integration of software switches (OVS) with hypervisors and VMs, highlighting the importance of these components in network architecture. The switches are connected to the top-of-rack (ToR) switches, which in turn connect to core switches, forming a resilient network topology.
Importance of Software Switches
Ease of Customization?

Enable **Rapid Development** and **Deployment** of Network Features!

Is it REALLY the case?
Ease of Customization?

For example, OVS supports following tunneling protocols:

- **VXLAN**: Virtual Extensible LAN
- **STT**: Stateless Transport Tunneling
- **NVGRE**: Network Virtualization Generic Routing

What about adding new protocols?
Rapid Development & Deployment?
Rapid Development & Deployment?

Requires domain expertise in:

- Network protocol design
- Software development
  - Develop
  - Test
  - Deploy
  ...
  large, complex codebases.

Arcane APIs
- Can take 3-6 months to get a new feature in.
- Maintaining changes across releases
Rapid Development & Deployment?
Rapid Development & Deployment?
Rapid Development & Deployment?

P4

Parser Match-Action Pipeline

Compile

OVS

Parser Match-Action Pipeline

DPDK

Native OVS

341 lines of code

14,535 lines of code
Rapid Development & Deployment?

Parser  Match-Action Pipeline

P4

Compile

Parser  Match-Action Pipeline

OVS

DPDK

Performance overhead!
What’s the **cost of programmability** on Performance?
PISCES: A Protocol-Independent Software Switch
PISCES: A Protocol-Independent Software Switch

P4

Compiler

parse match action

OVS

Runtime Flow Rules

Flow Rule Checker

Executable
PISCES: A Protocol-Independent Software Switch

- P4 and OVS packet forwarding models.
- Performance overhead of a naïve mapping from P4 to OVS.
- PISCES compiler optimizations to reduce the performance overhead.
P4 Forwarding Model (or Post-Pipeline Editing)
OVS Forwarding Model (or Inline Editing)
(Modified) OVS Forwarding Model

- Supports both editing modes:
  - Inline Editing
  - Post-pipeline Editing
Naïve Mapping from P4 to OVS

A naïve compilation of L2L3-ACL benchmark application

Performance overhead of
~ 40%
Causes of Performance Degradation

Ingress → Packet Parser → Match-Action Pipeline → Packet Deparser → Egress

CPU Cycles per Packet
Causes of Performance Degradation

- Factors affecting CPU cycles:
  - **Extra copy of headers** in the post-pipeline editing mode
  - **Fully-specified checksum** calculation
  - **Redundant parsing** of header fields and more ...
Causes of Performance Degradation

Factor #1: *Extra copy of headers*

<table>
<thead>
<tr>
<th>Editing Mode</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-Pipeline</td>
<td></td>
<td>Extra copy of headers</td>
</tr>
<tr>
<td>Inline</td>
<td>No extra copy of headers</td>
<td></td>
</tr>
</tbody>
</table>

- *Post-pipeline* editing consumes 2x more cycles than *inline* editing when parsing *VXLAN protocol*. 
Causes of Performance Degradation

Factor #1: **Extra copy of headers**

<table>
<thead>
<tr>
<th>Editing Mode</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-Pipeline</td>
<td><strong>Packets are adjusted once</strong></td>
<td>Extra copy of headers</td>
</tr>
<tr>
<td>Inline</td>
<td>No extra copy of headers</td>
<td><strong>Multiple adjustments to packet</strong></td>
</tr>
</tbody>
</table>

### Diagram

- **Inline editing**
  - Post-Pipeline Editing
  - Inline Editing

- **Post-pipeline editing**
  - Number of adjustments: Deparse, x1, x2, x4, x8, x16
  - Cycles per Packet: 0, 200, 400, 600, 800
Causes of Performance Degradation

Factor #2: **Fully-Specified Checksums**

```
Checksum
Incremental-Checksum
(ttl)
```

Ingress  | Packet Parser  | decrement(ttl)  | Packet Deparser  | Egress
Causes of Performance Degradation

Factor #3: **Redundant parsing of headers**
Optimizing for CPU Cycles

<table>
<thead>
<tr>
<th>Optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline vs. post-pipeline editing</td>
</tr>
<tr>
<td>Incremental checksum</td>
</tr>
<tr>
<td>Parser specialization</td>
</tr>
<tr>
<td>Action specialization</td>
</tr>
<tr>
<td>Action coalescing</td>
</tr>
</tbody>
</table>
## Optimizing for CPU Cycles

<table>
<thead>
<tr>
<th>Optimizations</th>
<th>Extra Copy of Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline vs. post-pipeline editing</td>
<td>Fully-Specified Checksum</td>
</tr>
<tr>
<td>Incremental checksum</td>
<td>Redundant Parsing</td>
</tr>
<tr>
<td>Parser specialization</td>
<td></td>
</tr>
<tr>
<td>Action specialization</td>
<td></td>
</tr>
<tr>
<td>Action coalescing</td>
<td></td>
</tr>
</tbody>
</table>
## Optimizing for CPU Cycles

<table>
<thead>
<tr>
<th>Optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline vs. post-pipeline editing</td>
</tr>
<tr>
<td>Incremental checksum</td>
</tr>
<tr>
<td>Parser specialization</td>
</tr>
<tr>
<td>Action specialization</td>
</tr>
<tr>
<td>Action coalescing</td>
</tr>
</tbody>
</table>
Optimized Mapping from P4 to OVS

All optimizations together

Performance overhead of < 2%
Next Steps

- Support for **stateful memories** and **INT**
- **Integration** with the **mainline OVS**
Summary

- SDN brought huge changes to how networks operate

- There is still a missing gap between the current state of SDN and a fully programmable network

- P4 is a new tool fill in the missing gap

- PISCES is first to show that programmability using P4 comes at a very small overhead while bringing huge benefits
Questions?