PVPP: A Programmable Vector Packet Processor

Sean Choi, Xiang Long, Muhammad Shahbaz, Skip Booth, Andy Keep, John Marshall, Changhoon Kim
Fixed Set of Protocols

- TCP
- IPv4
- IPv6
- Ethernet
- UDP
- HTTP
- BGP
- TLS

Fixed-Function Switch Chip

Broadcom Chip
Custom Protocols

- TCP
- IPv4
- IPv6
- Ethernet
- CUSTOM_P
- HTTP
- BGP
- TLS

Programmable Switching Chip
Software Switch

1 Physical Port

3 Virtual Ports
Approx. Number of Physical Ports vs. Virtual Ports

Custom Protocols

- TCP
- IPv4
- IPv6
- Ethernet
- CUSTOM_P
- HTTP
- BGP

Software Switch

PISCES\textsuperscript{[1]}

BMv2\textsuperscript{[2]}

\textsuperscript{[1]} PISCES. ACM SIGCOMM 2016.

\textsuperscript{[2]} https://github.com/p4lang/behavioral-model
Throughput on Eth + IPv4 + ACL benchmark application\[1\]

<table>
<thead>
<tr>
<th>Throughput (Gbps)</th>
<th>PISCES v0.1</th>
<th>PISCES v1.0</th>
<th>Native OVS</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 Packet Size (Bytes)</td>
<td>7.59</td>
<td>13.32</td>
<td>13.43</td>
</tr>
</tbody>
</table>

Performance overhead of < 2%

So... why ANOTHER P4 software switch?
Initially, the switching chip is not programmed and does not know any protocols.
Kernel

DPDK

Software Switch

Parser

Match-Action Pipeline

DPDK
PISCES
- P4 to OvS

BMv2
- P4 to a C++ custom switch
What’s wrong with this design?
• Not designed for **CPU based architectures**
• Limited in **expressiveness**
• **Limited APIs** to access low level constructs

=> Lot of room for improvements!
Vector Packet Processing (VPP) Platform

• Open source version of Cisco’s Vector Packet Processing technology

• Modular packet processing node graph abstraction

• Each node processes a vector of packets to reduce CPU I-cache thrashing

• Extensible and dynamically reconfigurable via plugins
Vector Packet Processing (VPP) Platform

• Proven Performance\textsuperscript{[1]}
  • Multiple MPPS from a single x86_64 core
    1 core: 9 MPPS ipv4 in+out forwarding
    2 cores: 13.4 MPPS ipv4 in+out forwarding
    4 cores: 20.0 MPPS ipv4 in+out forwarding
  • > 100Gbps full-duplex on a single physical host
  • Outperforms Open vSwitch in various scenarios

\textsuperscript{[1]} https://wiki.fd.io/view/VPP/What_is_VPP%3F
PVPP Overview

- Creates a plugin based on the input P4 program
- No changes to existing VPP code base
- Compiles either single node or multiple node plugin
- Multiple nodes are split by number of tables in the input P4 program
- P4 programs can be swapped dynamically
Packet Vector

Enabled via CLI

pvpp-input

Table 1

Table 2

Table i

Table j

Table k

Multi-Node PVPP Plugin

dpdk-input

ip4-input

ip6-input

llc-input

ip6-lookup

ip6-rewrite-transmit

Vanilla VPP Nodes

dpdk-output
Details of PVPP Plugin

• Headers are defined as C structs

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

typedef struct {
  u8 dstAddr[6];
  u8 srcAddr[6];
  u16 etherType;
} p4_type_Ethernet_h;

• Action interface takes pointers to all header, metadata, runtime data and compiler selects the correct pointer and set of primitives to perform on the data.
Details of PVPP Plugin

• A table definition contains two parts

  1. A match definition that defines the type of match (EXACT, LPM) and which fields to match with
  2. A action definition which contains set of action pointers corresponding to the match result
PVPP CLI

• Two CLIs are currently supported

1. Enable/Disable PVPP Pipeline
   
   $ pvpp [ingress interface name]

2. CLI to install match rules for a particular table
   
   $ pvpp insert-rule [table name]  
   [match value] [action name]  
   [runtime data]
Experimental Setup

CPU: Intel Xeon E5-2640 v3 2.6GHz
Memory: 32GB RDIMM, 2133 MT/s, Dual Rank
NICs: Intel X710 DP/QP DA SFP+ Cards
HDD: 1TB 7.2K RPM NLSAS 6Gbps
Benchmark Application

- **Parse Ethernet/IPv4**
  - Action: Match: ip.dstAddr
  - Action: Set_nhpop
  - Action: drop

- **IPv4_match**
  - Match: ip.dstAddr
  - Action: Set_dmac
  - Action: drop

- **Destination MAC**
  - Match: ip.dstAddr
  - Action: Set_dmac
  - Action: drop

- **Source MAC**
  - Match: egress_port
  - Action: Set_dmac
  - Action: drop
Baseline Performance

Throughput (Mpps)

<table>
<thead>
<tr>
<th>Packet Size (Bytes)</th>
<th>Single Node</th>
<th>Multiple Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>7.86</td>
<td>7.05</td>
</tr>
</tbody>
</table>
Compiler optimizations

- Remove redundant tables
- Reducing metadata access
- Bypassing redundant VPP nodes
- Reduce pointer dereference
- Caching logical HW interfaces
- Unrolling loops for multiple packet processing
Loop Unrolling

Manually fetches two packets
Optimized Performance

64 byte packets, single 10G port

Baseline: Throughput (Mpps)
- Single Node: 7.86
- Multiple Node: 7.05

Removing Redundant Tables: Throughput (Mpps)
- Single Node: 9.25
- Multiple Node: 8.38

Reducing Metadata Access: Throughput (Mpps)
- Single Node: 9.51
- Multiple Node: 8.50

Loop Unrolling: Throughput (Mpps)
- Single Node: 9.51
- Multiple Node: 8.80

Bypassing Redundant Nodes: Throughput (Mpps)
- Single Node: 9.58
- Multiple Node: 8.89

Reducing Pointer Dereferences: Throughput (Mpps)
- Single Node: 10.01
- Multiple Node: 9.02

Caching Logical HW Interface: Throughput (Mpps)
- Single Node: 10.21
- Multiple Node: 9.20

Throughput (Mpps)
Optimized Performance

Throughput (Mpps)

Packet Size (Bytes)

<table>
<thead>
<tr>
<th>Packet Size (Bytes)</th>
<th>Single Node</th>
<th>Multiple Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>10.21</td>
<td>9.20</td>
</tr>
<tr>
<td>128</td>
<td>8.07</td>
<td>8.07</td>
</tr>
<tr>
<td>192</td>
<td>5.63</td>
<td>5.65</td>
</tr>
<tr>
<td>256</td>
<td>4.38</td>
<td>4.38</td>
</tr>
</tbody>
</table>
Optimized Performance

Throughput (Mbps)

Packet Size (Bytes)

<table>
<thead>
<tr>
<th>Packet Size (Bytes)</th>
<th>Single Node</th>
<th>Multiple Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>5000</td>
<td>4500</td>
</tr>
<tr>
<td>128</td>
<td>8000</td>
<td>7500</td>
</tr>
<tr>
<td>192</td>
<td>9000</td>
<td>8500</td>
</tr>
<tr>
<td>256</td>
<td>9000</td>
<td>9000</td>
</tr>
</tbody>
</table>
Optimized Performance

Average CPU Cycles per Packet

Packet Size (Bytes) | Single Node | Multiple Node
---|---|---
64 | 133.00 | 159.00
128 | 149.00 | 172.30
192 | 171.00 | 222.30
256 | 194.00 | 255.20
Scalability

64 byte packets across 3 x 10G ports

<table>
<thead>
<tr>
<th>Number of CPUs</th>
<th>Single Node</th>
<th>Multiple Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.52</td>
<td>8.14</td>
</tr>
<tr>
<td>2</td>
<td>17.03</td>
<td>16.57</td>
</tr>
<tr>
<td>3</td>
<td>26.40</td>
<td>24.14</td>
</tr>
<tr>
<td>4</td>
<td>35.83</td>
<td>33.41</td>
</tr>
<tr>
<td>5</td>
<td>44.23</td>
<td>40.69</td>
</tr>
<tr>
<td>6</td>
<td>53.11</td>
<td>49.34</td>
</tr>
</tbody>
</table>
Performance Comparison

<table>
<thead>
<tr>
<th>Packet Size (Bytes)</th>
<th>PVPP</th>
<th>PISCES (with Microflow)</th>
<th>PISCES (without Microflow)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>59.53</td>
<td>63.49</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>49.31</td>
<td>47.23</td>
<td>30.22</td>
</tr>
<tr>
<td>192</td>
<td>34.71</td>
<td>34.72</td>
<td>30.20</td>
</tr>
<tr>
<td>256</td>
<td>26.78</td>
<td>26.78</td>
<td>26.78</td>
</tr>
</tbody>
</table>
Future Work

- Automated node splits based on the input program
- More compiler annotations for low level constructs
- Extending P4 support such as *data plane states*
- VPP specific P4_16 backend compiler
- Extending PVPP CLI features
Summary

- A performant and dynamically reconfigurable P4 switch based on a different packet processing abstraction

- More improvements planned over the summer prior to public release
Questions?