PVPP: A Programmable Vector Packet Processor
Sean Choi, Xiang Long, Muhammad Shahbaz, Skip Booth, Andy Keep, John Marshall, Changhoon Kim

1. Problem Statement

- **Match-action** abstractions are the de facto for compiling programmable data planes.
- Thus, switch targets expose just enough interfaces for customizing the match-action pipeline, but not a lot more.
- What if the switch target exposes more complex interfaces for interacting with the underlying architecture?

Finer tuning of compilers for increased performance!

2. Approach

- We present a programmable switch with programmable node graph data plane abstraction called PVPP.
- PVPP is an extension for Vector Packet Processing (VPP).
- VPP exposes low-level interfaces for interacting directly with the CPU and the memory.
- VPP’s unique node graph packet processing model allows various number of packets to be processed arbitrarily at each node with separate and isolated instructions.

Node Graph Packet Processing Abstraction

- PVPP is programmed via P4, a domain specific language specially designed to easily describe data plane behavior.
- PVPP’s P4 compiler is highly configurable.
  - Number of nodes for the given P4 file.
  - Number of packets per vector or per iteration.
- P4 Headers map directly to PVPP structs for increased readability and performance.

P4 to VPP Compiler Flow Diagram

3. Experimental Setup

Experimental Platform Topology

Experimental Server Specifications

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

4. Results

L2-L3 Benchmark Application

- The experimental results on L2-L3 benchmark application show that optimized PVPP has comparable throughput performance with VPP and other P4 to software switch implementation.

Compiler Optimization Results

(64 byte packets over a single 10G port)

<table>
<thead>
<tr>
<th>Optimizations</th>
<th>Throughput (Mpps)</th>
<th>Increment (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PVPP Baseline</td>
<td>7.860</td>
<td>N/A</td>
</tr>
<tr>
<td>Exclude redundant tables</td>
<td>9.248</td>
<td>+17.7</td>
</tr>
<tr>
<td>Reducing metadata access</td>
<td>9.508</td>
<td>+2.81</td>
</tr>
<tr>
<td>Multiple packet processing</td>
<td>9.508</td>
<td>0</td>
</tr>
<tr>
<td>Bypassing interface-output node</td>
<td>9.583</td>
<td>+0.79</td>
</tr>
<tr>
<td>Reducing pointer dereference</td>
<td>10.008</td>
<td>+4.43</td>
</tr>
<tr>
<td>Caching logical HW interface</td>
<td>10.209</td>
<td>+2.01</td>
</tr>
<tr>
<td>Vanilla VPP Baseline</td>
<td>10.748</td>
<td></td>
</tr>
</tbody>
</table>

Throughput Comparison with P4-Ovs (PISCES)

(over all six 10G ports)

<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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example of a P4 header to a PVPP C struct

```c
typedef struct {
  u8 dstAddr[6];
  u8 srcAddr[6];
  u16 etherType;
  p4_type_Ethernet_h;
} p4_type_Ethernet_t;
```