PVPP: A Programmable Vector Packet Processor

Sean Choi, Xiang Long, Muhammad Shahbaz, Skip Booth, Andy Keep, John Marshall, Changhoon Kim

1. Problem Statement

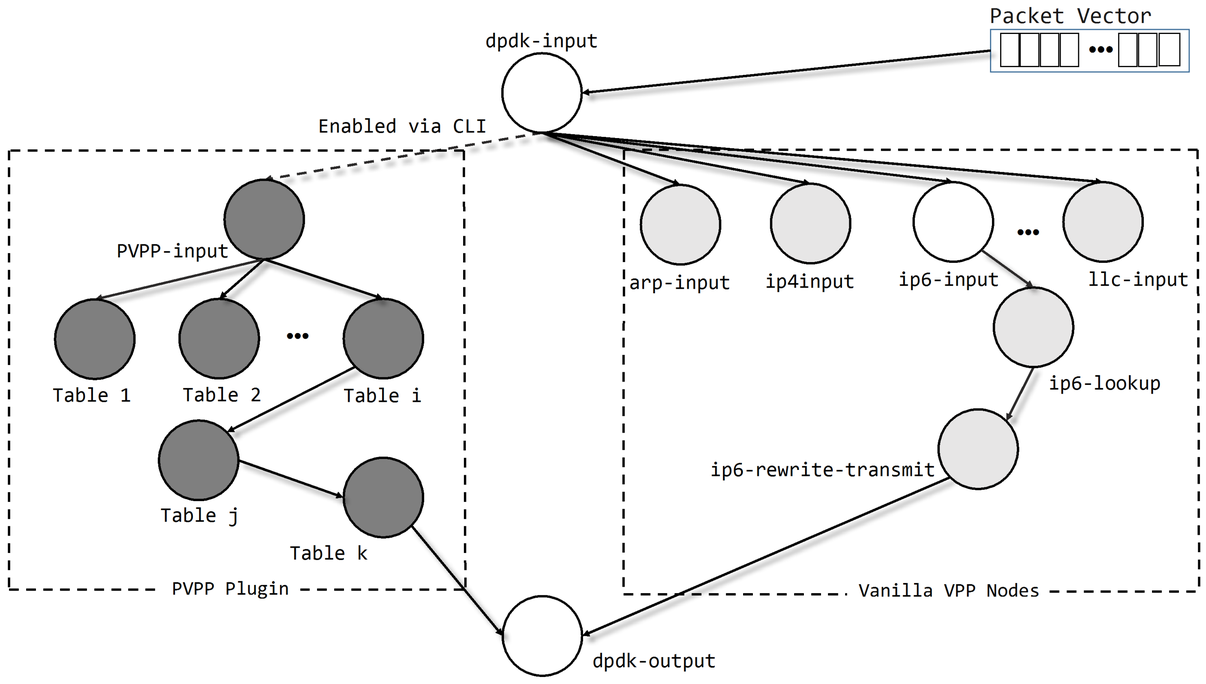
- **Match-action** abstractions are the de facto for compiling programmable data planes.
- Thus, switch targets expose just enough interfaces for customizing the match-action pipeline, but not a lot more.
- What if the switch target exposes more **complex interfaces** for interacting with the underlying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

Finer tuning of compilers for increased performance!

2. Approach

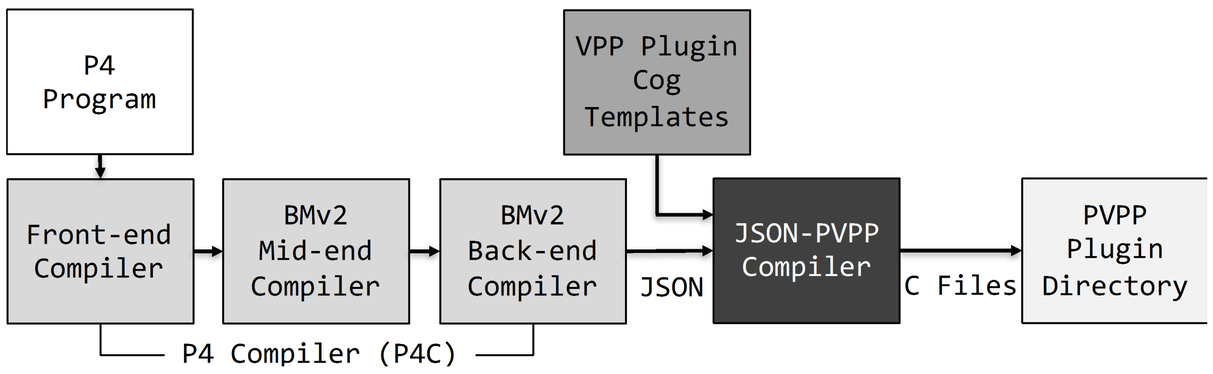
- We present a programmable switch with programmable node graph data plane abstraction called PVPP.
- PVPP is an extension for Vector Packet Processing(VPP).
- VPP exposes low-level interfaces for interacting directly with the CPU and the memory.
- VPP's unique node graph packet processing model allows 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



Example of a P4 header to a PVPP C struct

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

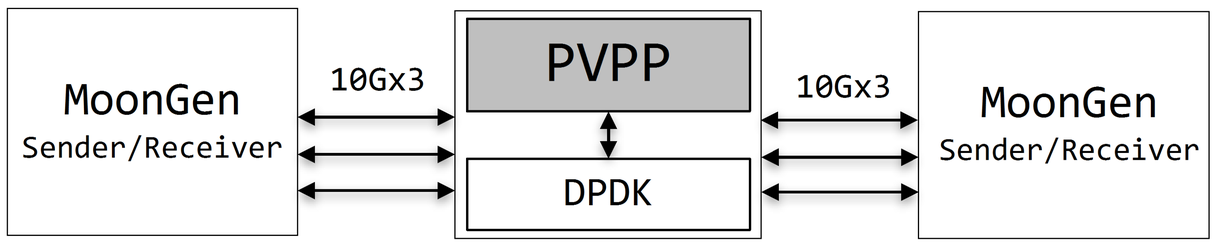
P4

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

PVPP

3. Experimental Setup

Experimental Platform Topology

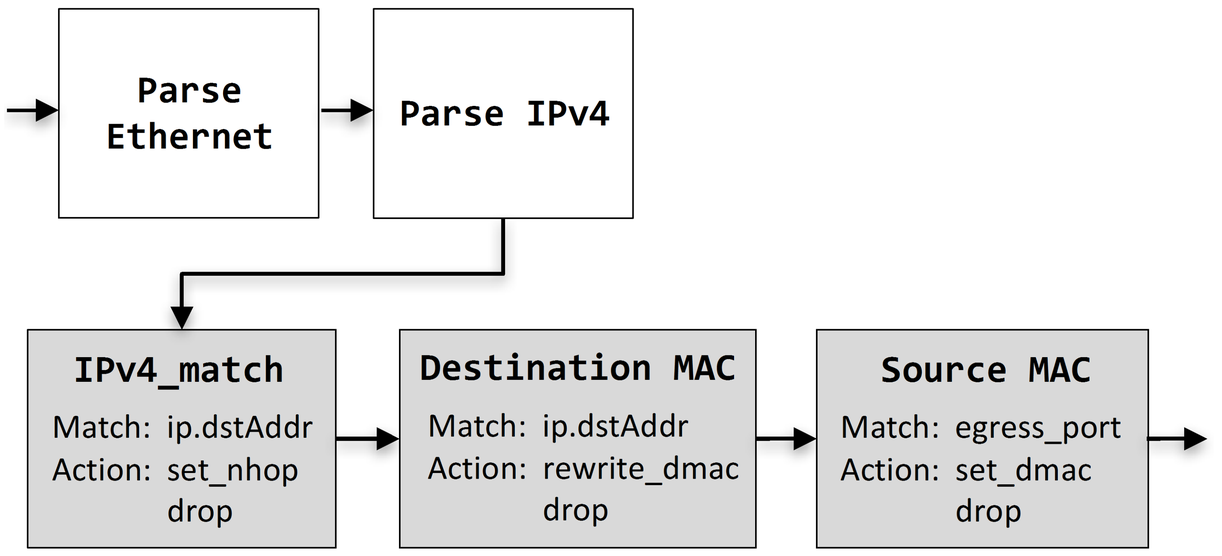


Experimental Server Specifications

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)

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

Throughput Comparison with P4-OvS (PISCES)

(*over all six 10G ports)*