The Case for a Flexible Low-Level Backend for Software Data Planes

Sean Choi¹, Xiang Long², Muhammad Shahbaz³,
Skip Booth⁴, Andy Keep⁴, John Marshall⁴, Changhoon Kim⁵

¹Cornell University ²Cornell University ³University of Chicago ⁴CISCO Systems ⁵Barefoot Networks
Why software data planes?

- VM hypervisors
- Cost savings with commodity general purpose processing units – where desired throughput < ~100 Gbps
- Prototyping protocol design
- Prototyping hardware DP architecture
Software Switch

PISCES

Software switch DSLs

```
table routing {
  reads {
    ipv4.dstAddr : lpm;
  }
  actions {
    do_drop;
    route_ipv4;
  }
  size: 2048;
}

control ingress {
  apply(routing);
}
```

High-level, close to protocol

Abstract forwarding model
Nice for programmers...

• Familiar and logical model in mind when programming, e.g. match/action pipelines

• Can specify packet data without worrying about implementation

• Portable code across platforms

• ...

• ...
Not so nice for compilers

• Abstract forwarding model not designed for e.g. CPU-based architectures

• Limited in expressiveness

• Insulated from underlying low-level APIs

• Result: Difficult to realize full performance potential of underlying hardware
Hypothesis

If software switches exposed more low-level characteristics to the data plane compiler

improvements are possible in performance and features
Our contribution

• Identify a software switch that can be programmed at low-level w.r.t to the hardware architecture
• Create compiler targeting that switch to allow it to support high-level data plane programs
• Compare performance
Target Switch: Vector Packet Processor (VPP)

- Open sourced by Cisco
- Can be programmed at low-level
- Part of the FD.io project
Vector Packet Processing (VPP) Platform

- Modular packet processing node graph abstraction
Vector Packet Processing (VPP) Platform

- Each node can execute almost arbitrary C code on vectors of packets
Vector Packet Processing (VPP) Platform

- Code is divided into nodes to optimize for i- and d-cache locality
Vector Packet Processing (VPP) Platform

- Extensible packet processing through first-class plugins
Vector Packet Processing (VPP) Platform

- Proven performance\(^1\)
  - 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

\(^1\) https://wiki.fd.io/view/VPP/What_is_VPP%3F
Vector Packet Processing (VPP) Platform

• Disadvantage: large burden on the programmer
• Requires knowledge from different fields: protocols, operating systems, processor architecture, C compiler optimization....
• Some Magic Required for good performance
Some Magic Required

Manually fetch two packets

Consequence of being low-level
Ease of programmability sacrificed for performance at low-level

Can a high-level DSL compiler help?

Programmable Vector Packet Processor (PVPP)
PVPP structure

Standard compiler optimizations are also applied, e.g. redundant table removal
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**

1. **Parse Ethernet/IPv4**
   - Action: Set Nhops drop

2. **IPv4_match**
   - Match: ip.dstAddr
   - Action: Set Nhops drop

3. **Destination MAC**
   - Match: ip.dstAddr
   - Action: Set Dmac drop

4. **Source MAC**
   - Match: egress_port
   - Action: Set Dmac drop
Baseline Performance

64 byte packets, single 10G port
Vector Packet Processing (VPP) Platform

- Each node can execute almost arbitrary C code on vectors of packets
Optimized Performance

64 byte packets, single 10G port

Throughput (Mpps)

- Baseline
- Removing Redundant Tables
- Reducing Metadata Access
- Loop Unrolling
- Bypassing Redundant Nodes
- Reducing Pointer Dereferences
- Caching Logical HW Interface

Comparison between Single Node and Multiple Node configurations.
Scalability

64 byte packets across 3 x 10G ports

- Throughput (Mpps) vs Number of CPU cores for Single Node and Multiple Node configurations.
Performance Comparison

Throughput (Mpps)

Packet Size (Bytes)

PVPP | PISCES (with Microflow) | PISCES (without Microflow)

64  | 59.53  | 63.49  | 30.22
128 | 49.31  | 47.23  | 30.22
192 | 34.71  | 34.72  | 30.20
256 | 26.78  | 26.78  | 26.78
Future work

• Microbenchmarking VPP to inform VPP-specific optimizations
• P4 compiler annotations for low-level constructs
• Explore when multi-node compilation is beneficial for PVPP
• Demonstrate use cases where OVS microflow cache is defeated – to show PVPP is just as programmable without resorting to separated fast/slow path
Summary

• High-level DSLs are great for programmers of software switches, but lack expressivity for optimizations.
• Low-level software switches such as VPP are performant but hard to program.
• We propose that best of both is possible with PVPP.
• Comparable to state-of-art performance achieved but still work in progress.