Programming
The Network Data Plane

Changhoon Kim
Beautiful ideas: What if you could ... 

- Realize a small, but super-fast DNS cache
- Perform TCP SYN authentication for billions of SYNs per sec
- Build a replicated key-value store ensuring RW ops in a few usecs
- Improve your consensus service performance by ~100x
- Boost your Memcached cluster’s throughput by ~10x
- Speed up your DNN training dramatically by realizing parameter servers

... using *switches* in your network?
You couldn’t do any of those so far because …

• No DIY – must work with vendors at feature level
• Excruciatingly complicated and involved process to build consensus and pressure for features
• Painfully long and unpredictable lead time
• To use new features, you must get new switches
• What you finally get != what you asked for
This is very unnatural to developers

• Because you all know how to realize your own ideas by “programming” CPUs
  – Programs used in every phase (implement, test, and deploy)
  – Extremely fast iteration and differentiation
  – You own your own ideas
  – A sustainable ecosystem where all participants benefit

Can we replicate this healthy, sustainable ecosystem for networking?
Reality: Packet forwarding speeds

Gb/s (per chip)

- 1990
- 1995
- 2000
- 2005
- 2010
- 2015
- 2020

Switch Chip

6.4Tb/s
Reality: Packet forwarding speeds

Gb/s (per chip)


Switch Chip
CPU

6.4Tb/s
80x
What does a typical switch look like?

A switch is just a Linux box with a high-speed switching chip
Networking systems have been built “bottoms-up”

“This is roughly how I process packets ...”

Fixed-function switch
Turning the tables “top-down”

in P4

“This is precisely how you must process packets”

Switch OS

Programmable Switch
“Programmable switches are 10-100x slower than fixed-function switches. They cost more and consume more power.”

Conventional wisdom in networking
Evidence: Tofino 6.5Tb/s switch (arrived Dec 2016)

The world’s fastest and most programmable switch.
No power, cost, or power penalty compared to fixed-function switches.
An incarnation of PISA (Protocol Independent Switch Architecture)
Domain-specific processors

Computers
- Java Compiler
  - CPU

Graphics
- OpenCL Compiler
  - GPU

Signal Processing
- Matlab Compiler
  - DSP

Machine Learning
- TensorFlow Compiler
  - TPU

Networking
- Language Compiler
  - ?
Domain-specific processors

Computers
- Java Compiler
- OpenCL Compiler
- Matlab Compiler
- TensorFlow Compiler

Graphics
- OpenCL Compiler

Signal Processing
- Matlab Compiler

Machine Learning
- TensorFlow Compiler

Networking
- P4 Compiler

- CPU
- GPU
- DSP
- TPU
- PISA
PISA: An architecture for high-speed programmable packet forwarding
PISA: Protocol Independent Switch Architecture
PISA: Protocol Independent Switch Architecture
PISA: Protocol Independent Switch Architecture

Match Logic
(Mix of SRAM and TCAM for lookup tables, counters, meters, generic hash tables)

Action Logic
(ALUs for standard boolean and arithmetic operations, header modification operations, hashing operations, etc.)

Programmable Packet Generator

Programmable Parser

Ingress match-action stages (pre-switching)

Buffer

Egress match-action stages (post-switching)

Recirculation

CPU (Control plane)

Generalization of RMT [sigcomm’13]
Why we call it protocol-independent packet processing
Logical Data-plane View
(your P4 program)

Switch Pipeline

Device does not understand any protocols until it gets programmed.
Mapping logical data-plane design to physical resources

Logical Data-plane View (your P4 program)

Switch Pipeline

Programmable Parser

L2 Table

IPv4 Table

IPv6 Table

ACL Table

Queues

CLK
Re-program in the field

Logical Data-plane View
(your P4 program)

Switch Pipeline
P4: Programming Protocol-Independent Packet Processors

Pat Bosshart†, Dan Daly*, Glen Gibb†, Martin Izzard†, Jennifer Rexford**, Cole Schlesinger†, Dan Talayco†, Amin Vahdat¶, George Varghese§, David Walker**, Cole Schlesinger†, Dan Talayco†, Amin Vahdat¶, George Varghese§, David Walker**

Barefoot Networks *Intel **Stanford University *Princeton University *Google *Microsoft Research

ABSTRACT

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able to change the way switches process packets once they are deployed. (2) Protocol independence: Switches should not be tied to any specific network protocols. (3) Target independence: Programmers should be able to describe packet-processing functionality independently of the specifics of the underlying hardware. As an example, we describe how to use P4 to configure a switch to add a new hierarchical label.

1. INTRODUCTION

Software-Defined Networking (SDN) gives operators programmatic control over their networks. In SDN, the control plane is physically separate from the forwarding plane, and one control plane controls multiple forwarding devices. While forwarding devices could be programmed in many ways, having a common, open, vendor-agnostic interface (like OpenFlow) enables a control plane to control forwarding devices from different hardware and software vendors.

Recent chip designs demonstrate that such flexibility can be achieved in custom ASICs at terabit speeds [1, 2, 3]. Programming this new generation of switch chips is far from easy. Each chip has its own low-level interface, akin to microcode programming. In this paper, we sketch the design of a higher-level language for Programming Protocol-independent Packet Processors (P4). Figure 1 shows the relationship between P4—used to configure a switch, telling it how packets are to be processed—and existing APIs (such as OpenFlow) that are designed to populate the forwarding tables in fixed function switches. P4 raises the level of abstraction for programming the network, and can serve as a future-proof standard.
What does a P4 program look like?

```
header_type ethernet_t {
  fields {
    dstAddr : 48;
    srcAddr : 48;
    etherType : 16;
  }
}
parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    0x8100 : parse_vlan;
    0x800  : parse_ipv4;
    0x86DD : parse_ipv6;
  }
}

header_type my_encap_t {
  fields {
    foo : 12;
    bar : 8;
    baz : 4;
    qux : 4;
    next_protocol : 4;
  }
}
```
What does a P4 program look like?

```

```

```
P4.org (http://p4.org)

- Open-source community to nurture the language
  - Open-source software – Apache license
  - A common language: P4₁₆
  - Support for various types of devices and targets
- Enable a wealth of innovation
  - Diverse “apps” (including proprietary ones) running on commodity targets
- With no barrier to entry
  - Free of membership fee, free of commitment, and simple licensing
So, what kinds of exciting new opportunities are arising?
The network should answer these questions

1. “Which path did my packet take?”
2. “Which rules did my packet follow?”
3. “How long did it queue at each switch?”
4. “Who did it share the queues with?”

PISA + P4 can answer all four questions for the first time. At full line rate. Without generating any additional packets!
In-band Network Telemetry (INT)

A read-only version of Tiny Packet Programs [sigcomm’14]

Add: SwitchID, Arrival Time, Queue Delay, Matched Rules, ...

Log, Analyze, Replay

Visualize

Original Packet
A quick demo of INT!
What does this mean to you?

• Improve your distributed apps’ performance with telemetry data
• Ask the four key questions regarding your packets to network admins or cloud providers
• Huge opportunities for Big-data processing and machine-learning experts
• “Self-driving” network is not hyperbole
PISA: An architecture for high-speed programmable packet forwarding event processing
What we have seen so far: Adding new networking features

1. New encapsulations and tunnels
2. New ways to tag packets for special treatment
3. New approaches to routing: e.g., source routing in data-center networks
4. New approaches to congestion control
5. New ways to manipulate and forward packets: e.g. splitting ticker symbols for high-frequency trading
What we have seen so far:
World’s fastest middle boxes

1. Layer-4 load connection balancing at Tb/s
   – Replace 100s of servers or 10s of dedicated appliances with one PISA switch
   – Track and maintain mappings for 5 ~ 10 million HTTP connections

2. Stateless firewall or DDoS detector
   – Add/delete and track 100s of thousands of new connections per second
   – Include other stateless line-rate functions
     (e.g., TCP SYN authentication, sketches, or Bloomfilter-based whitelisting)
What we have seen so far:
Offloading part of computing to network

1. DNS cache
2. Key-value cache [ACM SOSP’17]
3. Chain replication
4. Paxos [ACM CCR’16] and RAFT
5. Parameter service for DNN training
Example: NetCache

- Non-goal
  - Maximize the cache hit rate
- Goal
  - Balance the workloads of backend servers by serving only $O(N \log N)$ hot items -- $N$ is the number of backend servers
  - Make the “fast, small-cache” theory viable for modern in-memory KV servers [Fan et. al., SOCC’11]
- Data plane
  - Unmodified routing
  - Key-value cache built with on-chip SRAM
  - Query statistics to detect hot items
- Control plane
  - Update cache with hot items to handle dynamic workloads
Throughput vs. cache size.

Throughput vs. value size.

Yes, it’s Billion Queries Per Sec, not a typo 😊

One can further increase the value sizes with more stages, recirculation, or mirroring.
And its “not so boring” benefits

Throughput of a key-value storage rack with one Tofino switch and 128 storage servers.

NetCache provides 3-10x throughput improvements.
NetCache is a **key-value** store that leverages **in-network caching** to achieve:

- Billions of queries/sec
- A few usec latency
- Even under highly-skewed
- Rapidly-changing workloads.
Summing it up ...
Why data-plane programming?

1. **New features**: Realize your beautiful ideas very quickly
2. **Reduce complexity**: Remove unnecessary features and tables
3. **Efficient use of H/W resources**: Achieve biggest bang for buck
4. **Greater visibility**: New diagnostics, telemetry, OAM, etc.
5. **Modularity**: Compose forwarding behavior from libraries
6. **Portability**: Specify forwarding behavior once; compile to many devices
7. **Own your own ideas**: No need to share your ideas with others

“Protocols are being lifted off chips and into software”
– Ben Horowitz
My observations

- PISA and P4: The first attempt to define a machine architecture and programming models for networking in a disciplined way
- Network is becoming yet another programmable platform
- It’s fun to figure out the best workloads for this new machine architecture
Want to find more resources or follow up?

• Visit [http://p4.org](http://p4.org) and [http://github.com/p4lang](http://github.com/p4lang)
  – P4 language spec
  – P4 dev tools and sample programs
  – P4 tutorials

• Join P4 workshops and P4 developers’ days

• Participate in P4 working group activities
  – Language, target architecture, runtime API, applications

• Need more expertise across various fields in computer science
  – To enhance PISA, P4, dev tools (e.g., for formal verification, equivalence check, and many more ...)

43
Thanks.
Let’s develop your beautiful ideas in P4!