The Forwarding Plane: An Old New Frontier of Networking Research

CS244, Spring 2019

Changhoon Kim
chang@barefootnetworks.com
Software-Defined Networking (SDN)

Definition

**What is SDN?** The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices.
What is SDN in plain English?

• Ideally at the level for college freshmen
  – Because, if you can’t, you are not really understanding it!
    [Feynman’s guiding principle]

“Making programming networks as easy as programming computers.”
Natural questions that follow

“Making programming networks as easy as programming computers.”

• Why should we program a network?
  – To realize some “beautiful ideas” easily, preferably on our own

• What are those “beautiful ideas”?
  – Any impactful or intriguing apps in particular?

• Why couldn’t we do this easily in the pre-SDN era?
  – Any fundamental shifts happening?
Pre-SDN state of the network industry

Network Owner

Network Equipment Vendor

Feature

Feature

Years

Years

Engineering Division

Software Team

ASIC Team
Compared to other industries, this is very unnatural

- Because we all know how to realize our own ideas by programming CPUs, GPUs, TPUs, etc.
  - Programs used in every phase (implement, verify, test, deploy, and maintain)
  - Extremely fast iteration and differentiation
  - We own our own ideas
  - A sustainable ecosystem where all participants benefit

Can we replicate this healthy, sustainable ecosystem for networking?
What SDN pioneers had realized ...
And, SDN started to unfold ...
And, SDN started to unfold ...

Innovation-deprived, ossified layer

Innovation-rich, programmable layer

Various Control-plane Projects

Days to Weeks

Features

Weeks to Months

Network Owner

Software Team

Network Forwarding-plane Vendor

ASIC Team

Software Team

Feature

Feature

Feature

Feature
Reality: Packet forwarding speeds

Gb/s (per chip)


Switch Chip

6.4Tb/s
Reality: Packet forwarding speeds

Unaccommodating, performance-dominated zone?

Gb/s (per chip)


Switch Chip

CPU

6.4Tb/s

80x
“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 or cost penalty compared to fixed-function switches. An incarnation of PISA (Protocol Independent Switch Architecture)
Domain-specific processors

Computers
Java Compiler

Graphics
OpenCL Compiler

Signal Processing
Matlab Compiler

Machine Learning
TensorFlow Compiler

Networking
Language Compiler

CPU
GPU
DSP
TPU

?
Domain-specific processors

Computers
- Java Compiler
- CPU

Graphics
- OpenCL Compiler
- GPU

Signal Processing
- Matlab Compiler
- DSP

Machine Learning
- TensorFlow Compiler
- TPU

Networking
- P4 Compiler
- PISA
  (Protocol-Independent Switch Architecture)
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.)

Ingress match-action stages (pre-switching)

Egress match-action stages (post-switching)

Generalization of RMT [sigcomm’13]
Why we call it protocol-independent packet processing
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
Re-program in the field

Logical Data-plane View (your P4 program)

Switch Pipeline
P4 language components

- **Parser Program**
  - State-machine;
  - Field extraction

- **Match Tables + Actions**
  - Table lookup and update;
  - Field manipulation;
  - Control flow

- **Control Flow**
  - Field assembly

No: memory (pointers), loops, recursion, floating point
Questions and critiques …?

- What does a compiler do?
- What’s the latest on P4? Have you heard of P4$_{16}$?
- How do you update tables at runtime?
- Why is it important to derive a runtime API from a P4 program?
- What about queueing, scheduling, and congestion control?
What exactly does a compiler do?
P4_{16}: Why and how?

- **Embrace target heterogeneity without language churns**
  - Architectural heterogeneity via architecture-language separation
  - Functional heterogeneity via extern types

- **Help reuse code more easily: portability and composability**
  - Standard architecture and standard library
  - Local name space, local variables, lexical scoping, parameterization, and sub-procedure-like constructs

- **Make P4 programs more intuitive and explicit**
  - Expressions, sequential execution semantics for actions, strong type, and explicit de-parsing
To recap: 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
What kind of “stunt” can you do by programming data planes?
Advanced network measurement, analysis, and diagnostics
- In-band Network Telemetry [SIGCOMM’15], Packet History [NSDI’14], FlowRadar [NSDI’16], Marple [SIGCOMM’17]

Advanced congestion control
- RCP, XCP, TeXCP, DCQCN++, Timely++

Novel DC network fabric
- Flowlet switching, CONGA [SIGCOMM’15], HULA [SOSR’16], NDP [SIGCOMM’17]

World’s fastest middleboxes
- L4 connection load balancing [SIGCOMM’17], TCP SYN authentication, etc.

Offloading parts of the distributed apps
- NetCache [SOSP’17], NetChain [NSDI’18], SwitchPaxos [SOSR’15, ACM CCR’16]

Jointly optimizing network and the apps running on it
- Mostly-ordered Multicast [NSDI’15, SOSP’15]

And many more ... -- we’re just starting to scratch the surface!
PISA: An architecture for high-speed programmable packet forwarding I/O event processing
What we have seen so far:
Accelerating part of computing with PISA

1. DNS cache
2. Key-value cache [NetCache - SOSP’17]
3. Key-value replication [NetChain - NSDI’18]
5. Parameter service for distributed deep learning
6. Pub-sub service
7. String searching [PPS – SOSR’19]
8. Pre-processing DB queries and streams
NetCache: Accelerating KV caching
Suppose a KV cluster coping with a highly-skewed & rapidly-changing workload.
Suppose a KV cluster coping with a highly-skewed & rapidly-changing workload

Q: How can you ensure a high throughput and bound tail latency?
Here comes the problem

![Diagram showing throughput for different workload distributions]

- uniform
- zipf-0.9
- zipf-0.95
- zipf-0.99

Throughput (BQPS)

Workload Distribution
What if we had a very fast front-end server?

Q: How **big** and **fast** the front-end cache should be?

A read-only cache handling hot keys directly!
For a front-end cache to be effective

• How big should it be?
  – Keep $O(N \cdot \log N)$ hot keys where $N$ is the number of KV servers
  – Theory proves that such a front-end cache bounds the variance of KV server utilization *irrespective* of the total number of keys

• How fast should it be?
  – At least as large as the aggregated throughput of all KV servers ($N \cdot C$)
Why is this relevant now?

Cache needs to provide the aggregate throughput of the storage layer

Storage layer
- Flash/disk
  - Each: $O(100)$ KQPS
  - Total: $O(10)$ MQPS

Cache layer
- In-memory
  - Each: $O(10)$ MQPS
  - Total: $O(1)$ BQPS

In-memory
- Cache
  - $O(10)$ MQPS

Why? $O(1)$ BQPS
Why is this relevant now?

Cache needs to provide the aggregate throughput of the storage layer

**Storage layer**
- Flash/disk
  - Each: $O(100)$ KQPS
  - Total: $O(10)$ MQPS

**Cache layer**
- In-memory
  - Each: $O(10)$ MQPS
  - Total: $O(1)$ BQPS

Small on-chip memory?
- Only cache $O(N \log N)$ small items

- **PISA (real-time I/O machine)**
  - $O(1)$ BQPS
A conventional switch built with PISA

Control plane (CPU)

Data plane (ASIC)

Programmable Parser

Programmable Match-Action Pipeline
A front-end KV cache built with PISA

- **Data plane**
  - Key-value store to serve queries for cached keys
  - Query statistics to enable efficient cache updates

- **Control plane**
  - Insert hot items into the cache and evict less popular items
  - Manage memory allocation for on-chip key-value store
Line-rate query handling in the data plane

Read Query (cache hit)

Read Query (cache miss)

Write Query

Client

Server

1

2

3

4

Hit

Cache

Update

Stats

Miss

Cache

Update

Stats

Invalidate

Cache

Stats
Packet format

- Application-layer protocol; compatible with existing L2-L4 layers
- Only the front-end cache needs to parse NetCache fields
Key-value store using register array in network ASIC

```python
action process_array(idx):
    if pkt.op == read:
        pkt.value ← array[idx]
    elif pkt.op == cache_update:
        array[idx] ← pkt.value
```

Register Array

```
0 1 2 3
```

- 0
- 1
- 2
- 3
Key-value store using register array in network ASIC

<table>
<thead>
<tr>
<th>Match</th>
<th>pkt.key == A</th>
<th>pkt.key == B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>process_array(0)</td>
<td>process_array(1)</td>
</tr>
</tbody>
</table>

action process_array(idx):

if pkt.op == read:
    pkt.value ← array[idx]

elif pkt.op == cache_update:
    array[idx] ← pkt.value

Register Array

0 1 2 3

 pkt.value: A B
Variable-length key-value store in network ASIC?

<table>
<thead>
<tr>
<th>Match</th>
<th>pkt.key == A</th>
<th>pkt.key == B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>process_array(0)</td>
<td>process_array(1)</td>
</tr>
</tbody>
</table>

**Key Challenges:**

- No loop or string due to strict timing requirements
- Need to minimize hardware resources consumption
  - Number of table entries
  - Size of action data for table each entry
  - Size of intermediate metadata across tables
Combine outputs from multiple arrays

**Lookup Table**

- **Match**: pkt.key == A
- **Action**: bitmap = 111, index = 0
- pkt.value: A0 | A1 | A2

**Value Table 0**

- **Match**: bitmap[0] == 1
- **Action**: process_array_0(index)

**Value Table 1**

- **Match**: bitmap[1] == 1
- **Action**: process_array_1(index)

**Value Table 2**

- **Match**: bitmap[2] == 1
- **Action**: process_array_2(index)

**Bitmap** indicates arrays that store the key’s value.

**Index** indicates slots in the arrays to get the value.

**Minimal hardware resource overhead**

0 1 2 3

<table>
<thead>
<tr>
<th>Register Array 0</th>
<th>A0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Register Array 1</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>Register Array 2</td>
<td>A2</td>
<td></td>
</tr>
</tbody>
</table>
## Combine outputs from multiple arrays

### Lookup Table

<table>
<thead>
<tr>
<th>Match</th>
<th>pkt.key == A</th>
<th>pkt.key == B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action Bitmap</td>
<td>bitmap = 111</td>
<td>bitmap = 110</td>
</tr>
<tr>
<td>Index</td>
<td>index = 0</td>
<td>index = 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pkt.value:</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>B0</th>
<th>B1</th>
</tr>
</thead>
</table>

### Value Table 0

<table>
<thead>
<tr>
<th>Match</th>
<th>bitmap[0] == 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>process_array_0(index )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>B0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register Array 0

### Value Table 1

<table>
<thead>
<tr>
<th>Match</th>
<th>bitmap[1] == 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>process_array_1(index )</td>
</tr>
</tbody>
</table>

| A1 | B1 |

Register Array 1

### Value Table 2

<table>
<thead>
<tr>
<th>Match</th>
<th>bitmap[2] == 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>process_array_2(index )</td>
</tr>
</tbody>
</table>

| A2 |

Register Array 2

---

Combine outputs from multiple arrays
## Combine outputs from multiple arrays

<table>
<thead>
<tr>
<th>Lookup Table</th>
<th>pkt.value:</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>pkt.key == A</td>
<td>bitmap = 111, index = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pkt.key == B</td>
<td>bitmap = 110, index = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pkt.key == C</td>
<td>bitmap = 010, index = 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Table 0</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bitmap[0] == 1</td>
<td>process_array_0 (index )</td>
</tr>
</tbody>
</table>

Value Table 0: A0 B0

<table>
<thead>
<tr>
<th>Value Table 1</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bitmap[1] == 1</td>
<td>process_array_1 (index )</td>
</tr>
</tbody>
</table>

Value Table 1: A1 B1 C0

<table>
<thead>
<tr>
<th>Value Table 2</th>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bitmap[2] == 1</td>
<td>process_array_2 (index )</td>
</tr>
</tbody>
</table>

Value Table 2: A2

Register Arrays:
- Register Array 0: A0 B0
- Register Array 1: A1 B1 C0
- Register Array 2: A2

Combine outputs from multiple arrays: A0 A1 A2 B0 B1 C0
# Combine outputs from multiple arrays

<table>
<thead>
<tr>
<th>Lookup Table</th>
<th>Action</th>
<th>pkt.key == A</th>
<th>pkt.key == B</th>
<th>pkt.key == C</th>
<th>pkt.key == D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bitmap</td>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bitmap</td>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bitmap</td>
<td>010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bitmap</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>index</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pkt.value:</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>B0</th>
<th>B1</th>
<th>C0</th>
<th>D0</th>
<th>D1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Value Table 0</th>
<th>Match</th>
<th>bitmap[0] == 1</th>
<th>Action</th>
<th>process_array_0 (index )</th>
<th>0 1 2 3</th>
<th>A0 B0 D0 Register Array 0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Value Table 1</th>
<th>Match</th>
<th>bitmap[1] == 1</th>
<th>Action</th>
<th>process_array_1 (index )</th>
<th>A1 B1 C0 Register Array 1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Value Table 2</th>
<th>Match</th>
<th>bitmap[2] == 1</th>
<th>Action</th>
<th>process_array_2 (index )</th>
<th>A2 D1 Register Array 2</th>
</tr>
</thead>
</table>

Combine outputs from multiple arrays
Cache insertion and eviction

- Challenge: Keeping the **hottest** $O(N \log N)$ items in the cache
- Goal: React quickly and effectively to workload changes with **minimal updates**

![Diagram](image)

1. Data plane reports hot keys
2. Control plane compares loads of new hot and sampled cached keys
3. Control plane fetches values for keys to be inserted to the cache
4. Control plane inserts and evicts keys
Query statistics in the data plane

- **Cached key**: per-key counter array
- **Uncached key**
  - Count-Min sketch: report new hot keys
  - Bloom filter: remove duplicated hot key reports
The “boring life” of a NetCache switch

Throughput vs. value size.

Throughput vs. cache 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.**
Questions and critiques …?

• Conflating “in-network computing” with “accelerating apps using PISA”

• What are the common and unique strengths of PISA for those offload-able apps?

• Will the “stunt” become a main stream approach? What would be the triggers for that?

• Is P4 the right way of implementing PISA-server apps?
Some observations

• PISA and P4: The first attempt to define a machine architecture and programming models for networking in a disciplined way

• Inherently multi-disciplinary; we need more expertise across various fields in computer science

• It’s super fun to figure out the best workloads for this new machine architecture
Want to find more resources or follow up?

  - P4 language spec
  - P4 dev tools and sample programs
  - P4 tutorials
  - List of papers regarding PISA, PISA Apps, and P4
- 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, automated test generation, and many more ...)

58