The Forwarding Plane: An Old New Frontier of Networking

Changhoon Kim

Stanford CS / Barefoot Networks, an Intel company
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

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

“Making programming networks as easy as programming computers.”

• Why would / 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 early on?
  – Any fundamental shifts happening?
Pre-SDN state of the network industry
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 ...
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 is not true anymore! 😊

RMT (now known as PISA) is real and will stay.
## Two Production Systems from a Well-known Switch Vendor

<table>
<thead>
<tr>
<th></th>
<th>Barefoot Tofino</th>
<th>Fixed Function Silicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2/L3 Throughput</td>
<td>6.4Tb/s</td>
<td>6.4Tb/s</td>
</tr>
<tr>
<td>Number of 100G Ports</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Availability</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Otherwise, both systems are identical:
- # of Ports
- CPU
- Power Supplies
Domain-specific processors

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

Graphics
- GPU

Signal Processing
- DSP

Machine Learning
- TPU

Networking
- Language Compiler

?
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
(it's paper name was RMT)
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
Device does not understand any protocols until it gets programmed.

Logical Data-plane View
(your P4 program)

Switch Pipeline

LAG
IPv4
IPv6
ACL

Programmable Parser

Match Table
Action ALUs

packet

Queues

CLK
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
ACL Action Macro

Queues

CLK
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

Deparser Program
- Field assembly

No: memory (pointers), loops, recursion, floating point
What exactly does a compiler do?

- PHV (Packet Header Vector)
- Programmable Parser
- Cross Bar
- Hash Gen
- Action & Instr Mem
- Match Table (SRAM or TCAM)
- ALUs
- PHV'
- Queues
- CLK
How is P4 programmability used?

1. Reducing complexity
Reducing complexity

IPv4 and IPv6 routing
- Unicast Routing
- Routed Ports & SVI
- VRF
- Unicast RPF
- Strict and Loose
- Multicast
- PIM SM/DM & PIM-Dcll

Ethernet switching
- VLAN Flooding
- MAC Learning & Aging
- STP state
- VLAN Translation

Load balancing
- ECMP & WCMP
- Resilient Hashing
- Flowlet Switching

Fast Failover
- LAG & ECMP

Tunneling
- IPv4 and IPv6 Routing & Switching
  - IP in IP (Gin/4, 4in4)
  - VXLAN, NVGRE, GENEVE & GRE
  - Segment Routing, ILA

MPLS
- LER and LSR
- IPv4/v6 routing (L3VPN)
- L2 switching (CoMPLS, VPLS)
- MPLS over UDP/GRE

ACL
- MAC ACL, IPv4/v6 ACL, RACL
- QoS ACL, System ACL, PBR
- Port Range lookups in ACLs

QoS
- QoS Classification & marking
  - Drop profiles/WRED
  - RoCE v2 & FCoE
  - CoPP (Control plane policing)

NAT and L4 Load Balancing

Security Features
- Storm Control, IP Source Guard

Monitoring & Telemetry
- Ingress Mirroring and Egress Mirroring
- Negative Mirroring
- SFlow
- INT

Counters
- Route Table Entry Counters
- VLAN/Bridge Domain Counters
- Port/Interface Counters

Protocol Offload
- BFD, OAM

Multi-chip Fabric Support
- Forwarding, QoS
Reducing complexity

My switch.p4

P4 Compiler

Switch OS

Driver

PISA Switch
Switch.p4 vs Existing 3.2T device

<table>
<thead>
<tr>
<th>Entry Type</th>
<th>Tofino</th>
<th>Existing 3.2T Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC</td>
<td>512k</td>
<td>136k</td>
</tr>
<tr>
<td>IPv4 LPM CLPM</td>
<td>904k</td>
<td>128k</td>
</tr>
<tr>
<td>IPv6 /64 CLPM</td>
<td>520k</td>
<td>86k</td>
</tr>
<tr>
<td>IPv6 /128 CLPM</td>
<td>240k</td>
<td>1k</td>
</tr>
<tr>
<td>IPv4 Host Routes*</td>
<td>1,240k</td>
<td>104k</td>
</tr>
<tr>
<td>IPv6 Host Routes*</td>
<td>296k</td>
<td>56k</td>
</tr>
<tr>
<td>ACLs</td>
<td>120k</td>
<td>8k</td>
</tr>
</tbody>
</table>
How is P4 programmability used?

2. Introducing new features rapidly
What we have seen so far:
Adding new networking features

1. New encapsulations and tunnels: e.g., PPP over Eth, load-balancing GTP tunnels or initiating/terminating them
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
6. NFV acceleration
7. Network slicing and packet-gateway acceleration for 5G
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
   – Developed and deployed in production by a large cloud-service provider

2. 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)
Extending PISA with external memory

• Can attach external DRAM to PISA
  – Via RDMA (e.g., TEA, GEM) or FPGA

• Significantly increase feature scale of PISA
  – Maintain $O(100M)$ connection-state entries
  – Increase the buffer size by three orders of magnitude

• P4 and compiler technologies allow us to abstract this kind of hybrid system into a “virtual chip”
How is P4 programmability used?

3. *Network telemetry*
Which path did my packet take?

I visited Switch 1 @780ns, Switch 9 @1.3µs, Switch 12 @2.4µs

Which rules did my packet follow?

In Switch 1, I followed rules 75 and 250. In Switch 9, I followed rules 3 and 80.
3. "How long did my packet queue at each switch?"

4. "Who did my packet share the queue with?"

```
Delay: 100ns, 200ns, 19740ns
```
3. “How long did my packet queue at each switch?”
   "Delay: 100ns, 200ns, 19740ns"

4. “Who did my packet share the queue with?”

Aggressor flow!
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!
One example: In-band Network Telemetry (INT)

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

“Monitor every packet, and report only what’s worth!”
- Generate reports upon detecting changes
  • Flow initiation & termination
  • Path or latency changes
  • Special field values
- Change detectors are reset periodically (e.g., once every 500ms)
Flexibility matters

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

P4 code snippet: **switch ID, ingress and egress timestamp**

```p4
/* INT: add switch id */
action int_set_header_0() {
    add_header(int_switch_id_header);
    modify_field(int_switch_id_header.switch_id,
                        global_config_metadata.switch_id);
}

/* INT: add ingress timestamp */
action int_set_header_1() {
    add_header(int_ingress_tstamp_header);
    modify_field(int_ingress_tstamp_header.ingress_tstamp,
                        i2e_metadata.ingress_tstamp);
}

/* INT: add egress timestamp */
action int_set_header_2() {
    add_header(int_egress_tstamp_header);
    modify_field(int_egress_tstamp_header.egress_tstamp,
                        eg_intr_md_from_parser_aux.egress_global_tstamp);
}
```
Viewing Microbursts (to the nanosecond)
How is P4 programmability used?

4. Beyond packet forwarding
PISA: An architecture for high-speed programmable packet forwarding event processing
Beautiful ideas: What if you could …

• Realize a small, but super-fast DNS cache
  [Jaqen – Usenix Security’21]
• Perform TCP SYN authentication for billions of SYN per sec
  [Jaqen – Usenix Security’21]
• Build a replicated key-value store ensuring RW ops in a few usecs
  [NetChain – NSDI’18]
• Improve your consensus service performance by ~100x
  [P4xos – CCR’16, Eris – SOSP’17]
• Boost your Memcached cluster’s throughput by ~10x
  [NetCache - SOSP’17]
• Speed up your DNN training dramatically by realizing parameter servers
  [SwitchML – NSDI’21]

... using **Tofino servers**?
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

Throughput (BQPS)

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

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

A read-only cache handling hot keys directly!

Q: How **big** and **fast** the front-end cache should be?
For a front-end cache to be effective

• How big should it be?
  – Keep $O(N \log N)$ 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 ($N \cdot C$)
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 (hit)

1. Client → Server
2. Server → Client

Read Query (miss)

1. Client → Server
4. Server → Client
2. Client → Server
3. Server → Client

Write Query

1. Client → Server
4. Server → Client
2. Client → Server
3. Server → Client
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:**

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
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.
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
Some observations

• PISA and P4: The first attempt to define a machine architecture and programming models for networking in a disciplined way
• We’ll see a lot more P4-programmable devices, especially NICs and software targets
• 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
Some questions and comments

• I wonder if it would be reasonably easy to verify the RMT programs?

• “A physical stage could also accommodate multiple logical stages”. How come? They should be using different actions, but 1 physical stage could only execute 1 instruction per cycle.

• Scaling such arch by adding pipeline stages seems to be a bad idea. The paper uses a 28nm process (probably TSMC) which is a pretty old one. 10nm and 7nm have improved PPA (Power, Performance, Area) so much, and it would be able to increase clock freq. The problem is SerDes on such an advanced node because the analog part in SerDes does not scale very well than digital logic.
More questions and comments

• Limited capabilities and primitives of programmable switches. Because programmable switches need to catch extremely high line-rate (Tb/s), operations switches support are not very complex. Also, the memory RMT pipelines need to be very fast (SRAM level). Due to its expensive cost, its size is limited. Under these constraints, we struggle to shoehorn application functionalities into RMT switches.

• Cost-effective. High-end programmable switches are more expensive in terms of price and energy cost. Should we migrate some applications like load-balancer from x86 servers to RMT switches? There is a tradeoff between cost and perf.

• Complexity on developing and maintaining apps for RMT switches. At least until now, RMT switches require developers to use domain-specific languages like P4 to build apps. Also, developers need to know more about RMT switch architecture. These will increase burdens for network operators to migrate fancy application functionalities to RMT switches.
ANY MORE?
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 ...)