Programming

The Network Data Plane: What, How, and Why?

Changhoon Kim

P4.org / Barefoot Networks
Network systems have been built “bottoms-up”

"This is roughly how I process packets ..."
My Super Secret Source-Routing

P4 program

P4 Compiler

Programmable Switch
Turning the tables “top-down”

“This is precisely how you must process packets”
Programmable switches are 10-100x slower than fixed-function switches. They cost more and consume more power.

“Conventional wisdom in networking

Not true anymore! 😊
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
- OpenCL Compiler

Graphics

- Signal Processing
- Matlab Compiler
- TensorFlow Compiler

Machine Learning

Networking

- DSP Compiler
- Language Compiler

Networking

- TPU

CPU

GPU

DSP

TPU
Domain-specific processors

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

Graphics
- GPU

Signal Processing
- DSP

Machine Learning
- TPU

Networking
- P4 Compiler

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.)

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

Logical Data-plane View
(your P4 program)

Switch Pipeline
Mapping logical data-plane design to physical resources

Logical Data-plane View
(your P4 program)

Switch Pipeline

Programmable Parser

L2 Table
L2 Action Macro

IPv4 Table
v4 Action Macro

IPv6 Table
v6 Action Macro

ACL Table
ACL Action Macro

Queues

CLK
Re-program in the field

Logical Data-plane View
(your P4 program)

Switch Pipeline

Programmable Parser
L2 Table
L2 Action Macro
IPv4 Table
IPv4 Action
MyEncap
IPv6 Table
IPv6 Action
ACL Table
ACL Action Macro
Queues
CLK

myencap
L2
IPv4
IPv6
ACL
What exactly does the compiler do?

- Programmable Parser
- Cross Bar
- Hash Gen
- Match Table (SRAM or TCAM)
- Action & Instr Mem
- ALUs
- PHV (Pkt Hdr Vector)
- PHV'

Queues

CLK
From now on, they have the same speed, power and cost as fixed-function switch chips.

But, why now?
Switch chip area: Serial I/O

Silicon Area
Normalized

Area stays same
Area stays same

Serial I/O
Same area for each generation

Semiconductor generation

Switch chip area: Serial I/O

Silicon Area
Normalized

Area stays same
Area stays same

Serial I/O
Same area for each generation

Semiconductor generation
30% Serial I/O

25% Memory (Lookup tables)

25% Logic (Packet Processing)

20% Packet Buffer & TM
Switch chip area: Memory (Tables and Buffer)

Memory Same area for each generation

Silicon Area
Normalized

Semiconductor generation
Logic dictates whether “fixed function” or “programmable.”
Switch chip area: Packet Processing Logic

Siocomdor generation

Silicon Area

Logic Area goes down each generation

Normalized

1.0

0.5

10% new features

Area goes down

Moore's Law

10% new features
Logic dictates whether “fixed function” or “programmable”.
Packet Processing 30% Serial I/O

Logic (Packet Processing)

Memory (Lookup tables)

Packet Buffer & TM

Logic dictates whether “fixed function” or “programmable”
Logic dictates whether “fixed function” or “programmable”
Protocols and table complexity 30 years ago

- Ethernet
- IPv4
- IPX

Diagram:

- Ethernet
  - ethtype to IPv4
  - ethtype to IPX
Enterprise switch today
public switch.p4
Why we will have good programmability

1. **Chip technology**: The difference in chip area and power between “programmable” and “fixed function” is going away.

2. **Chip complexity**: There are now too many protocols to correctly hard-code in silicon.

3. **Chip architecture**: Instruction set is now known.

4. **Chip speed**: We can now make programmable switch chips as fast as fixed ones.

5. **New ideas**: Beautiful new ideas are owned by the programmer, not the chip designer.

So, how do we program PISA machines and other flexible packet processors?
What I mean by data-plane programming

• Dictating the followings for *every* packet a networking device processes
  – The structure of all possible headers and metadata
  – How to parse, re-parse, and de-parse (re-assemble)
  – How to forward: a sequence of custom match-actions
  – How to replicate or recirculate
  – How to structure, maintain, and apply all necessary info for match-actions
  – How to collect and export forwarding statistics

• I’d also like to prescribe the followings, hopefully soon
  – How to schedule packets
  – How to generate packets
  – How to apply even more complicated non-forwarding functions to packets
P4.org and P4\textsubscript{16}

• Open-source community to nurture the language
  – Open-source software – Apache license
  – A common language – P4\textsubscript{14} is based on the original P4 paper [CCR’14]. \textbf{P4\textsubscript{16} is the version revised by the P4 Language Design WG.}
  – Support for various devices – Physical & virtual SWs, host networking stacks, NICs, and middleboxes
  – Support for various targets – PISA chips, FPGAs, NPUs, and CPUs

• 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
P4_{16}: Why and how?

• Embrace architectural and functional heterogeneity while keeping the language clean and stable
  – Architecture-language separation
  – Extern types

• Help reuse code more easily: portability and composability
  – Standard architecture and standard library
  – Local name space, local variables, and parameterization
  – Sub-procedures

• Make P4 programs more intuitive and explicit
  – Expressions
  – Sequential execution semantics for actions
  – Strong type
  – Explicit de-parsing
Architecture-language separation

**Switch Architecture Specification**

```
// “arch.p4”
// Architecture declaration
parser P<H>(in packet_in packet,
          out H headers);
control Ingress<H>(
    inout H headers,
    in intrinsic_metadata_in imi,
    out intrinsic_metadata_out imo
);
control Deparser<H>(in H headers,
                      out packet_out packet);
package Switch<H>(Parser<H> p,
                   Ingress<H> ingress,
                   Deparser<H> deparser);
```

**Switch Implementation (by user)**

```
// Program written by user
#include “arch.p4”

parser MyParser(...) { ... }
control MyIngress(...) { ... }
control MyDeparser(...) { ... }

// Top-level element instantiation
Switch(MyParser(),
      MyIngress(),
      MyDeparser()) MySwitch;
```
Fitting all these together

**P4.org**
- **Standard Architecture**
- **Standard Library**
  - Primitive actions
  - `extern` types

**Target Consumer (P4 Programmer)**
- **Architecture Selection**
- **Compile**
  - The ima
  - Target Data-plane Configuration
- **Auto-generated API**

**Target**
- **Target-specific Architecture(s)**
- **Target-specific Library**
  - Primitive actions
  - `extern` types

**Target Provider**
- **User Library**
  - P4 code
So, what kinds of exciting new opportunities are arising?
1. “Which path did my packet take?”

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

2. “Which rules did my packet follow?”

“In Switch 1, I followed rules 75 and 250. In Switch 9, I followed rules 3 and 80.”

Rule
1
2
3
...
75 192.168.0/24
...

3. “How long did my packet queue at each switch?”

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

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

“Delay: 100ns, 200ns, 19740ns”

Aggressor flow!

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

Original Packet

Log, Analyze, Replay

Visualize
A quick demo of INT!
Adding features: What we have seen so far

1. New encapsulations and tunnels
2. New ways to tag packets for special treatment
3. New approaches to routing: e.g., source routing in MSDCs
4. New approaches to congestion control
5. New ways to manipulate and forward packets: e.g. splitting ticker-symbols for HFT
To push more into and to take more out of the data plane (at the same time)

• Putting more into the data plane
  – Middle-box functions
    • Layer-4 load balancing [SilkRoad, sigcomm’17], network security features, DNS services, etc.
  – Part of distributed apps
    • NetPaxos [ccr’16], MoM [nsdi’15], SwitchKV [nsdi’16], NetChain, NetCache, etc.

• Taking more out of the data plane
  – Exposing packet metadata to upper layers – transport or even apps
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
  - Handle dynamic workloads
The “boring life” of a NetCache switch

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 $3\text{-}10x$ 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.
Experiences, lessons, and potential research opportunities – with a bit of personal musings
Why data-plane programming?

1. **New features**: Realize new protocols and behaviors 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 network**: No need to wait for next chips or systems

“Protocols are being lifted off chips and into software”
— Ben Horowitz
Subtle, but important benefits

- Simplify hardware micro-architecture
  - No more chasing game at the h/w level to catch up on intricate details and vagaries of particular protocols
  - Stop worrying about the unpredictability of protocol adoption

- Avoid unnecessary struggle for a common API (a.k.a., SDK)
  - Common, flexible, transparent, and robust API? Never heard of one.
  - Forget about the “common” part; automatically derive “your own” API – including its implementation – from your P4 program and keep reusing it
  - Faster integration between control and data planes
Network monitoring, analysis, and diagnostics

• Monitoring features can finally become first-class citizens
  – “Dear device vendors: No, thank you. You can now (happily) stop making critical feature trade-off decisions for me.”

• Network owners know what to monitor and how best
  – Network owners can retain and build on their improvement (i.e., the right monitoring features that work best for themselves)

• Powerful new approaches are emerging
  – INT (In-band Network Telemetry), mirroring dropped packets, reachability monitoring directly within the data plane, etc.
  – Joint-engineer control and data plane even at run time [Marple, sigcomm’17]
What’s next for P4?

• Modularity
  – Equivalent of static or even dynamic libraries

• Event-driven programming in P4 and for data plane?
  – Programmable packet generator + stateful processing can enable new types of apps

• Clarifying the standard architecture and its behavior?
  – Reference implementation with P4-defined interfaces, or a formal specification?

• Scheduler sub-language?
  – Programmable scheduler might appear in a few years, starting with lower-speed devices

Help P4.org with your suggestions and contributions
How far can we go?

• Can we auto-generate P4 programs? If yes, from what and when?
• How much can we verify when all aspects of networking are programmable?
• What kind of stateful data-plane algorithms are useful and feasible? What are the right development abstractions and tools for them [Domino, sigcomm’16]?
• Can we build a network without any switch-local control plane (aka Switch OS) at all?
But, really why? What are the intellectual merits?

The first attempt to define the machine architecture for networking in a disciplined way

Inherently multi-disciplinary

Figuring out the best workloads for this new type of machine
Thanks & enjoy data-plane programming!