Programming The Network Data Plane

Changhoon Kim

P4.org / Barefoot Networks

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 and de-parse (re-assemble)
  - How to forward: a sequence of custom match-actions
  - How to replicate, resubmit, 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

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 mirco-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

P4.org

- Open-source community to nurture the language
  - Open-source software – Apache license
  - A common language – $P4_{14}$ is currently widely used, $P4_{16}$ soon to be ready
  - 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
  - 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

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);
```

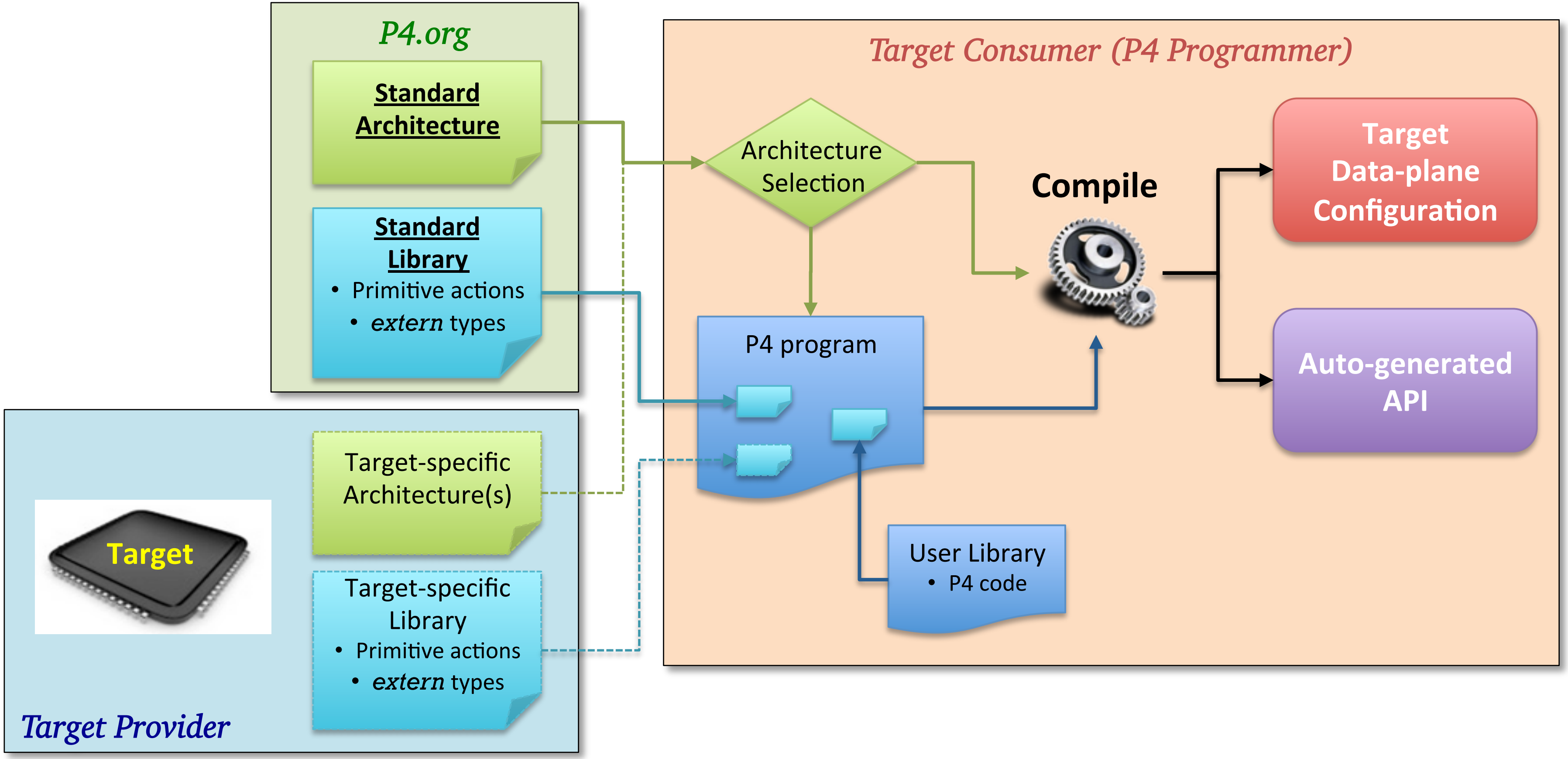
Switch Implementation (by user)

```
// 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



A few more points about $P4_{16}$

- Sequential execution semantics for an action
- Expressions
- Strong types
- Explicit deparsing
- Backwards-compatibility is a must with $P4_{16}$ and onward

Don’t worry; a $P4_{14}$-to-$P4_{16}$ translator is available

What's next for P4?

- Packet-generator sub-language?
  - When used with stateful processing, packet generator can enable event-driven programming style
- Clarifying the standard architecture?
  - 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

P4.org always welcome your suggestions and contributions

So, what kinds of exciting new opportunities are arising?

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.

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, network security features, etc.
  - Part of distributed apps
    - NetPaxos [CCR'16], MoM [NSDI'15], and SwitchKV [NSDI'16]
- Taking more out of the data plane
  - Exposing per-packet metadata to upper layers – transport or even apps

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?
- Can we build a network without any switch-local control plane at all?

Thanks & happy P4 coding!