APRIL 24-25
P4
WORKSHOP 2023

State of P4

Andy Fingerhut

P4’s raison d’être

- “reason for being”

- The same as it has been since it was created:
  - To program packet processing devices that achieve the best price/power/performance ratios that anyone knows how to build.
  - Hardware architectures are usually some variant of RMT, DRMT, or “array of small RISC cores”
- If P4 did not exist, many people would (and did) invent variants of it
  - But each even more target-specific than P4 is today

Is P4 feature complete?

- Do you see features missing?

- I do.
- Let us find ways to add them incrementally to what we have.

Missing features in 2020

- Add-on-miss
  - Add new entries to tables at high rate *in data plane*
- Auto-delete
  - Delete old entries *in the data plane* when they have been unmatched for configurable duration.
  - The timeout duration of an entry is *modifiable* at packet processing time.
- Packet encryption
  - Data plane APIs and P4 architecture flow for encryption & decryption

- Now part of the PNA specification

Missing features in 2023

- Data-plane-writable action data
  - e.g. maintain expected TCP sequence numbers independently for each table entry, in TCP connection tracking.
- High throughput control plane APIs
  - Adding millions of table entries per second to large tables.
- Configuring externs with P4Runtime API more consistently
  - See new GenericTable idea proposed in P4 API work group

- In active discussion in P4.org working groups now

Missing features in 2023 (cont’d)

- Updating P4 code with 0 down time
  - Implementation techniques are typically target-dependent, but sharing ideas on how is likely to make this more widely available.
- Support in Linux to load P4 code into kernel
  - Then offload into NICs that support it. See talk on P4-TC later in the workshop.
- Good IDE support
  - New open source repo: https://github.com/p4lang/p4analyzer

- In active discussion/development now

As specification

- P4 is also used as a specification language
  - Write behavioral specification of a not-P4-programmable device
  - Then use a combination of formal verification methods and testing to compare this spec to actual device, or desired network-wide behavior.
- One tool of note:
  - P4TestGen is now open source and part of https://github.com/p4lang/p4c (thanks to Nate Foster, Fabian Ruffy, and others)

Active participants today include

- AMD*
- Google*
- Intel*
- Keysight*
- Marvell
- Nvidia
- Vmware*

- Most device vendors use open source P4 front/mid-end compiler from https://github.com/p4lang/p4c
- * At least one P4.org work group co-chair works here

Thank You!

Andy Fingerhut