Introducing P4TC

P4 In The Linux Kernel

Agenda

- Introduction and motivation
- High level overview of development and deployment workflow
- Tie to OPI
- Status

Motivation

Goal: Grow Network Programmability ecosystem

- Datapath definition using P4
- P4 Linux kernel-native implementation
- Reduced Developer dependency

Motivation

Goal: Grow Network Programmability ecosystem

- Why P4?
  - _The_ Lingua Franca for describing hardware datapaths
  - Large consumers of NICs require at minimal P4 for datapath behavioral description if not implementation
    - Eg MS DASH
  - To Each, Their Itch
    - Conway's Law: Organizations model their datapath based on their needs
    - Ossification challenges: It's not just about traditional TCP/IP anymore

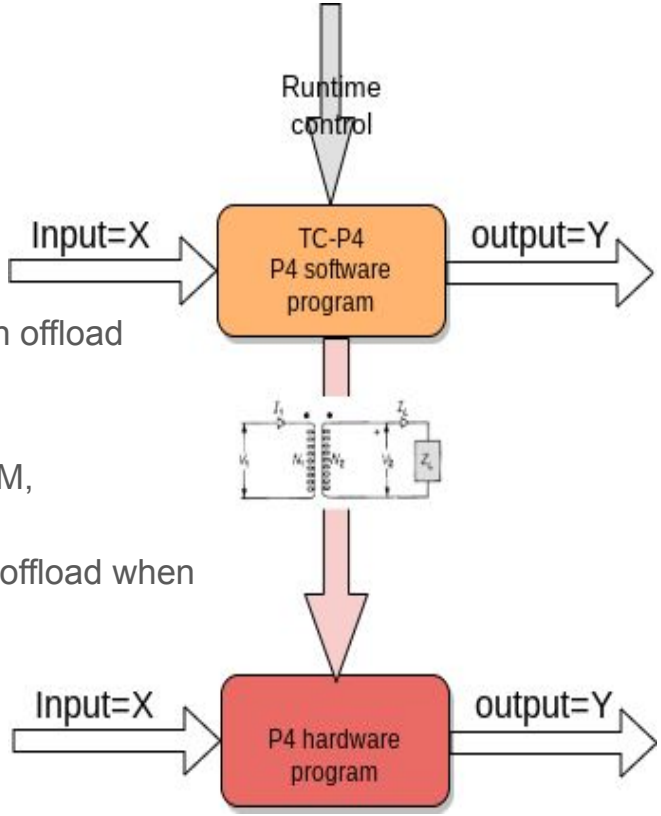
Motivation

Goal: Grow Network Programmability ecosystem

- Why Linux Kernel?
  - Mother of all networking infrastructure
    - If it beeps and/or has LEDs and maybe emits smoke it is more than likely running Linux
  - Singular API for offloads (via vendor driver)
  - Same consistent interface regardless of infrastructure deployment
    - SW or HW

Introduction to P4TC

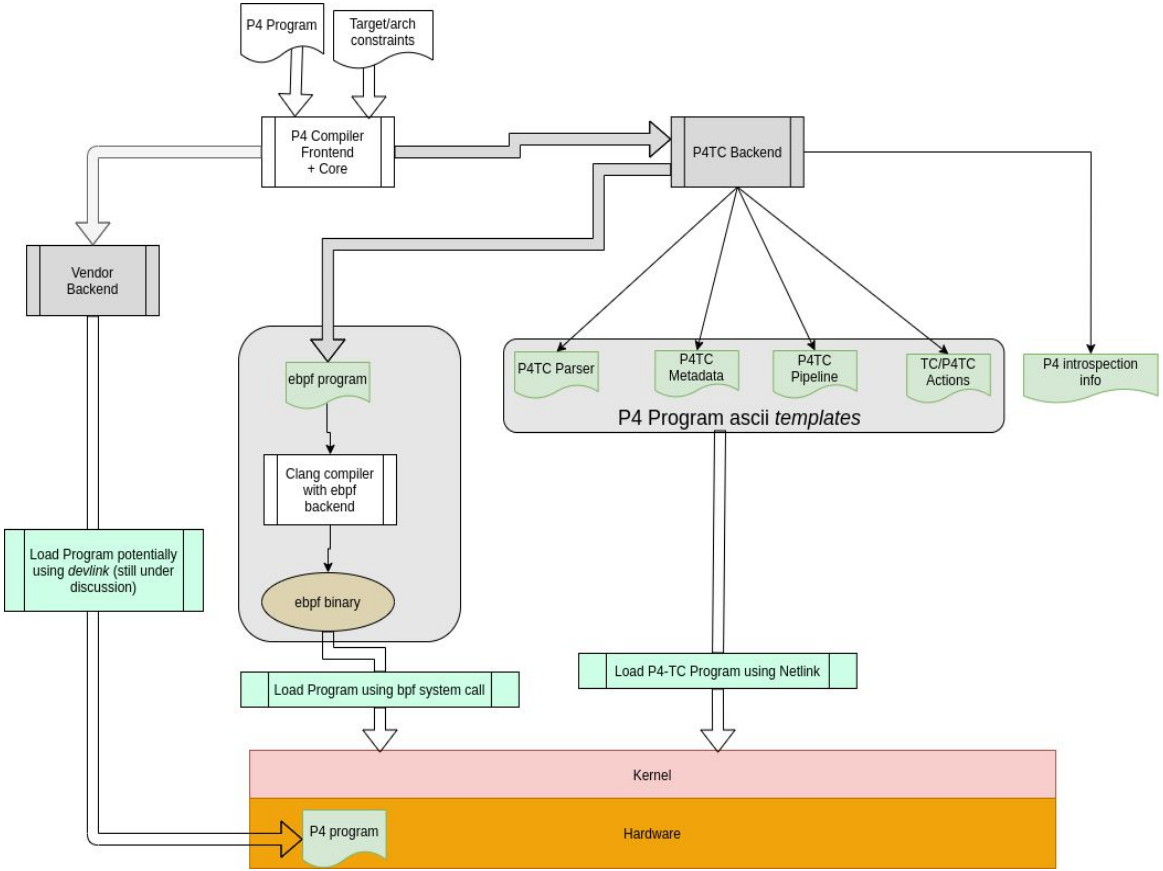
- Datapath definition using P4
  - Generate the datapath both s/w and vendor h/w
    - Functional equivalence between sw and hw
- P4 Linux kernel-native implementation
  - Kernel TC-based software datapath and Kernel-based HW datapath offload
    - Infra tooling which already has deployments
  - Seamless software and hardware symbiosis
  - Functional equivalence whether offloading or s/w datapaths (BM, VM, Containers)
  - Ideal for datapath specification (test in s/w container, VM, etc) then offload when hardware is available



Introduction to P4TC

- Kernel independence for P4 program
  - No need to upstream any code for new P4 programs
    - Unlike other offload mechanisms like tc/flower
- Learn from previous experiences (tc flower, u32, switchdev, etc) and scale
- P4 Architecture Independence
  - Allow for PSA, PNA, and new innovations on top
    - This is about progressing network programmability in addition to expanding P4 reach
- Vendor Independent interfacing
  - No need to deal with multiple vendor abstraction transformations (and multiple indirections)
  - No need for the (cumulus foo) punting infrastructure

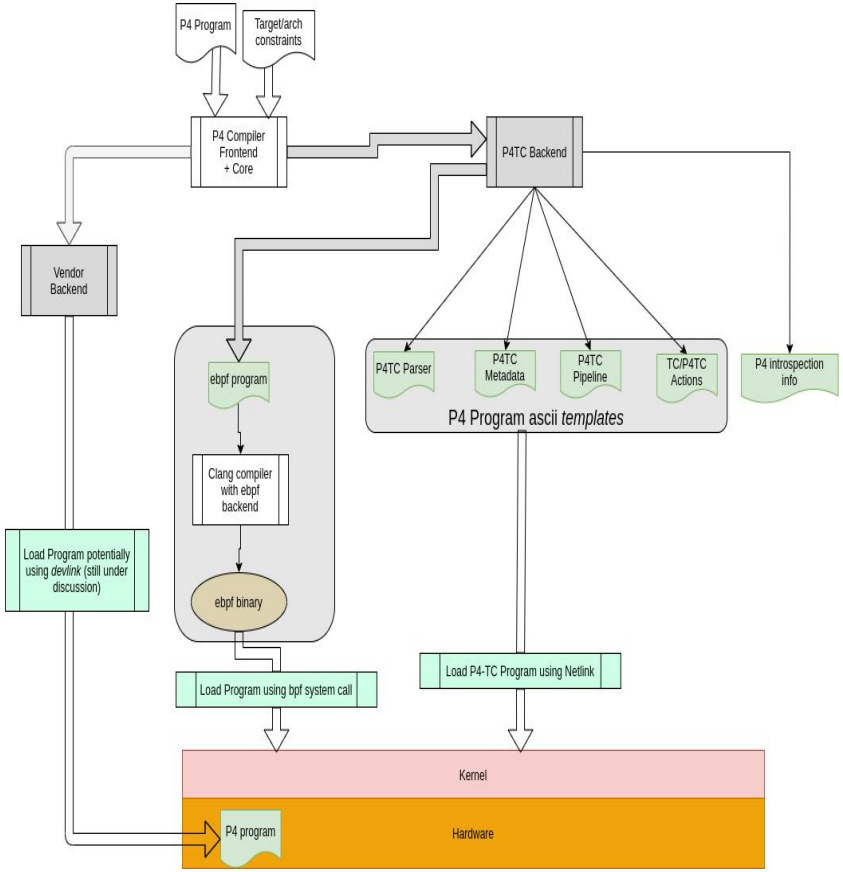
P4TC Workflow

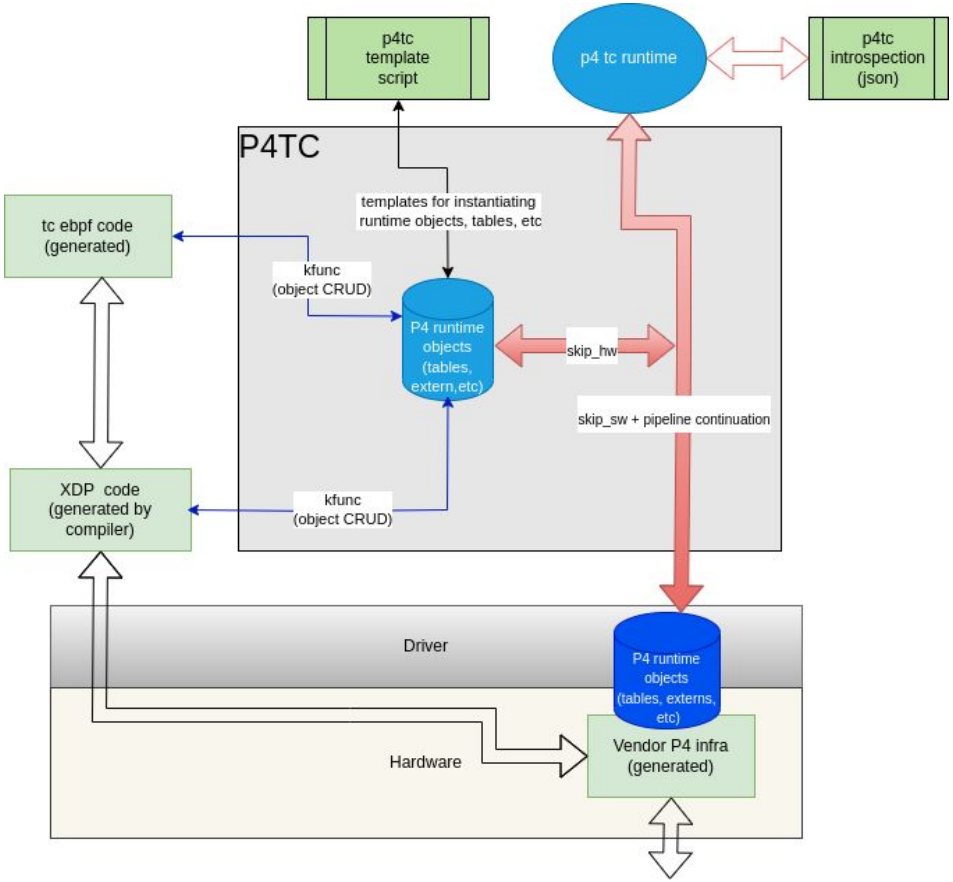


Generated

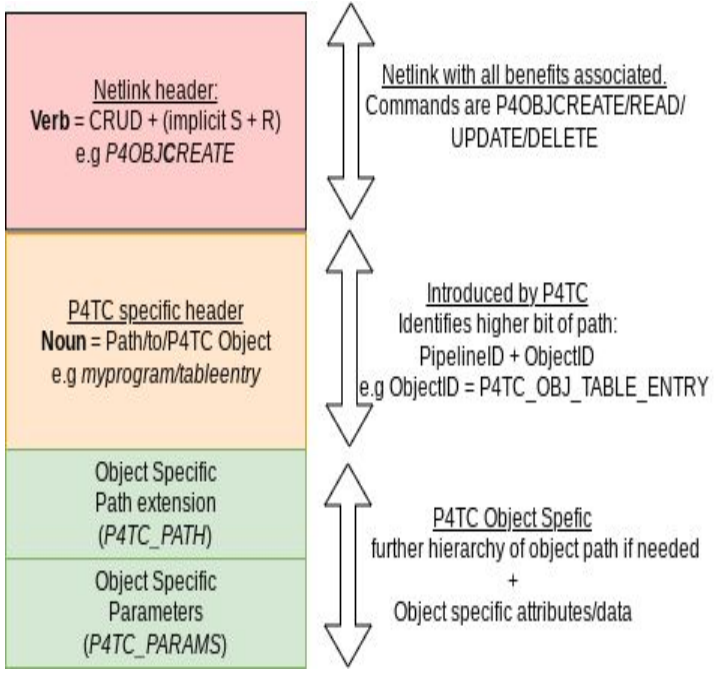
1. P4TC Template
2. P4TC Introspection json
3. eBPF s/w datapath
   At tc and/or xdp level
4. Vendor Specific HW Datapath

P4TC Runtime Control And Datapath





Control Plane Runtime CRUD Interface



Goal: Very High throughput and Low Latency interface

<VERB> < <NOUN> [OPTIONAL DATA] >+

single entry:

```
tc p4runtime get ptables/table/control1/mytable \
    ip/dstaddr 1.1.1.1/32
```

vs dump the whole table

```
tc p4runtime get ptables/table/control1/mytable
```

single entry:

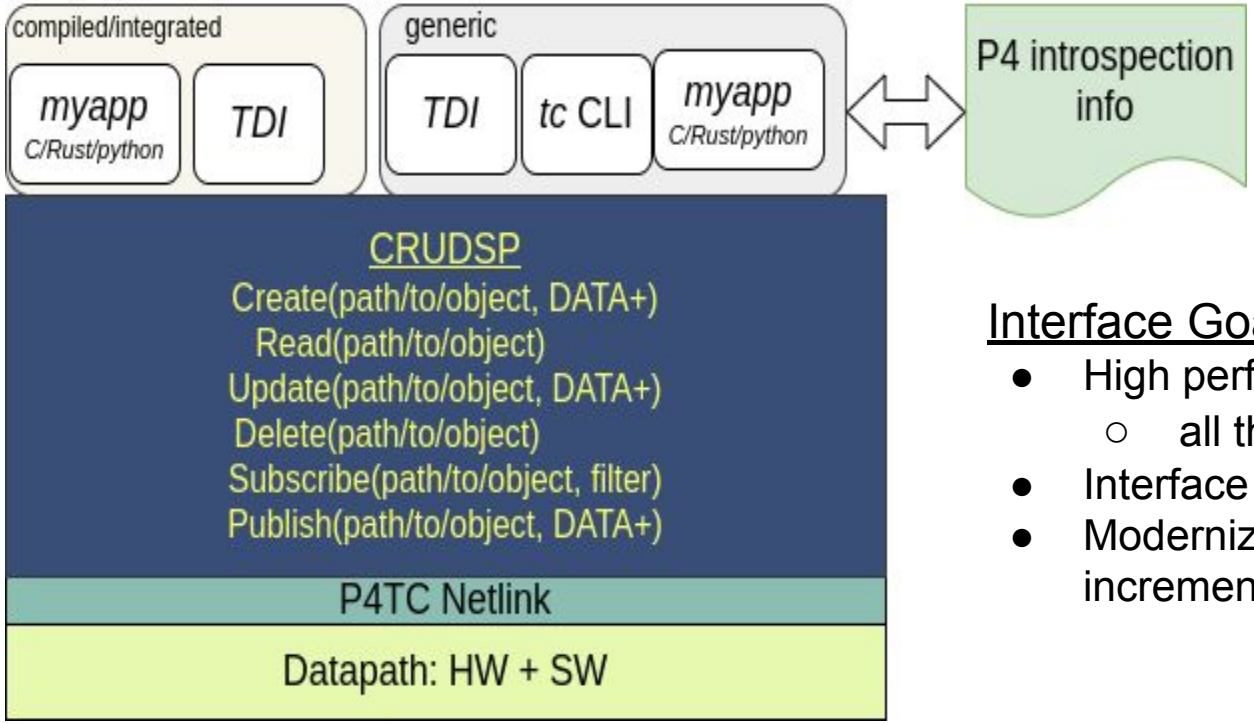
```
tc p4runtime create ptables/table/control1/mytable \
    ip/dstaddr 10.10.10.0/24 prio 16 action drop
```

Data

vs batch:

```
tc p4runtime create ptables/table/control1/mytable \
    ip/dstaddr 10.10.10.0/24 prio 16 action drop \
    ip/dstaddr 1.1.1.1/32 prio 32 action drop \
    ip/dstaddr 8.8.8.8/32 prio 64 action drop
```

P4TC OPI Interface



Interface Goals:

- High performance 1M/s + transactions
  - all the way to HW
- Interface with standard linux tooling (tc)
- Modernized Control approach to handle incremental operations

Status

- Ongoing effort to upstream
  - Moved from scriptable to ebpf for sw datapath based on feedback
    - Slowed us down a bit
  - Latest kernel patches
    - https://lore.kernel.org/netdev/20230801113807.85473-1-jhs@mojatatu.com/#r
- Ongoing vendor discussions for hw integration
  - Biweekly meetings
  - Mailing list
  - Multiple vendors involved
    - Intel most advanced
- More info
  - https://p4tc.dev