Integrating eBPF Into P4TC

Netdev conf 0x17 (Nov 02/2023)
Vancouver, Canada

Jamal Hadi Salim
Deb Chatterjee
Victor Nogueira
Pedro Tammela
Tomasz Osinski
Evangelos Haleplidis
Sosutha Sethuramapandian
Balachandher Sambasivam
Usha Gupta
Komal Jain

Community Historical Perspective

- Many informal hallway and online discussions (2016)
- Netdev 2.2 (Seoul, 2017)
  - Matty Kadosh, "P4 Offload", https://legacy.netdevconf.info/2.2/slides/salim-tc-workshop04.pdf
  - Prem Jonnalagadda, "Mapping tc to P4", https://legacy.netdevconf.info/2.2/slides/salim-tc-workshop06.pdf
- ONF 5th Workshop(Stanford, 2018)
  - Jamal Hadi Salim, "What P4 Can Learn From Linux Traffic Control", https://opennetworking.org/wp-content/uploads/2020/12/Jamal_Salim.pdf
- First ever P4 TC workshop, Intel, Santa Clara, 2018
  - Many Speakers (Barefoot, Intel, Cumulus, Melanox, Vmware, Mojatatu, and others) https://files.netdevconf.info/d/5aa8c0ca61ea4e96bb46/
- Netdev 0x12 (Montreal, 2018)
  - Antonin Bas and R. Krishnamoorthy, "Instrumenting P4 in the Kernel" https://www.files.netdevconf.info/d/9535fba900604dcd9c93/files/?p=/Instrumenting%20P4%20in%20the%20Linux%20kernel.pdf
- Netdev 0x13 (Prague, 2019)
  - Marian Pritsak and Matty Kadosh, "P4 Compiler Backend for TC", https://legacy.netdevconf.info/0x13/session.html?p4-compiler-backend-for-tc

Motivation

Goal: Grow Network Programmability ecosystem

- Datapath definition using P4
  - P4 Linux kernel-native implementation
  - Compiler generates the rest
    - Mundane developer knowledge moved into compiler
    - Reduced developer dependency
    - Reduced upstream effort

Motivation

Goal: Grow Network Programmability ecosystem

- Why P4?
  - _Only_ standardized language for describing datapaths
  - Emergence of P4 Native NICs (Intel, AMD)
  - Large consumers of NICs require at minimal P4 for datapath behavioral description if not implementation
    - Eg MS DASH
    - In our case validate using kernel datapath
  - To Each, Their Itch
    - Conway's Law: Organizations model their datapath based on their needs
      - Burger King Philosophy: Have it your way
    - 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

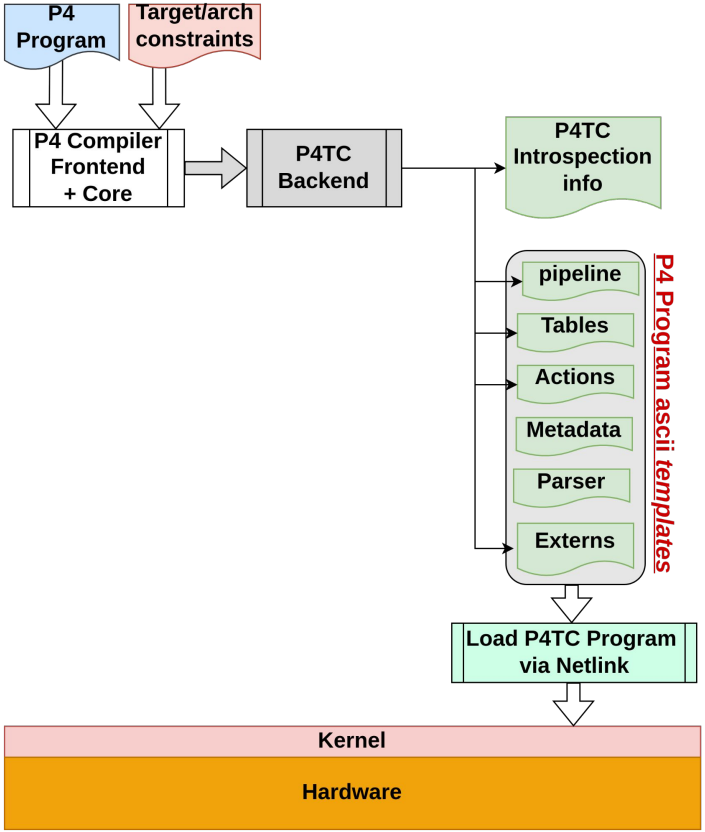
- Kernel independence for P4 program
  - No need to upstream any code for new P4 programs
    - Unlike other offload mechanisms like tc/flower
- Learn from previous experiences (tc flower, u32, switchdev, etc) and scale
  - Example control plane rate and latency
- P4 Architecture Independence
  - Allow 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

Original P4TC Implementation

Scriptable Datapath See:

https://netdevconf.info/0x16/sessions/talk/your-network-datapath-will-be-p4-scripted.html

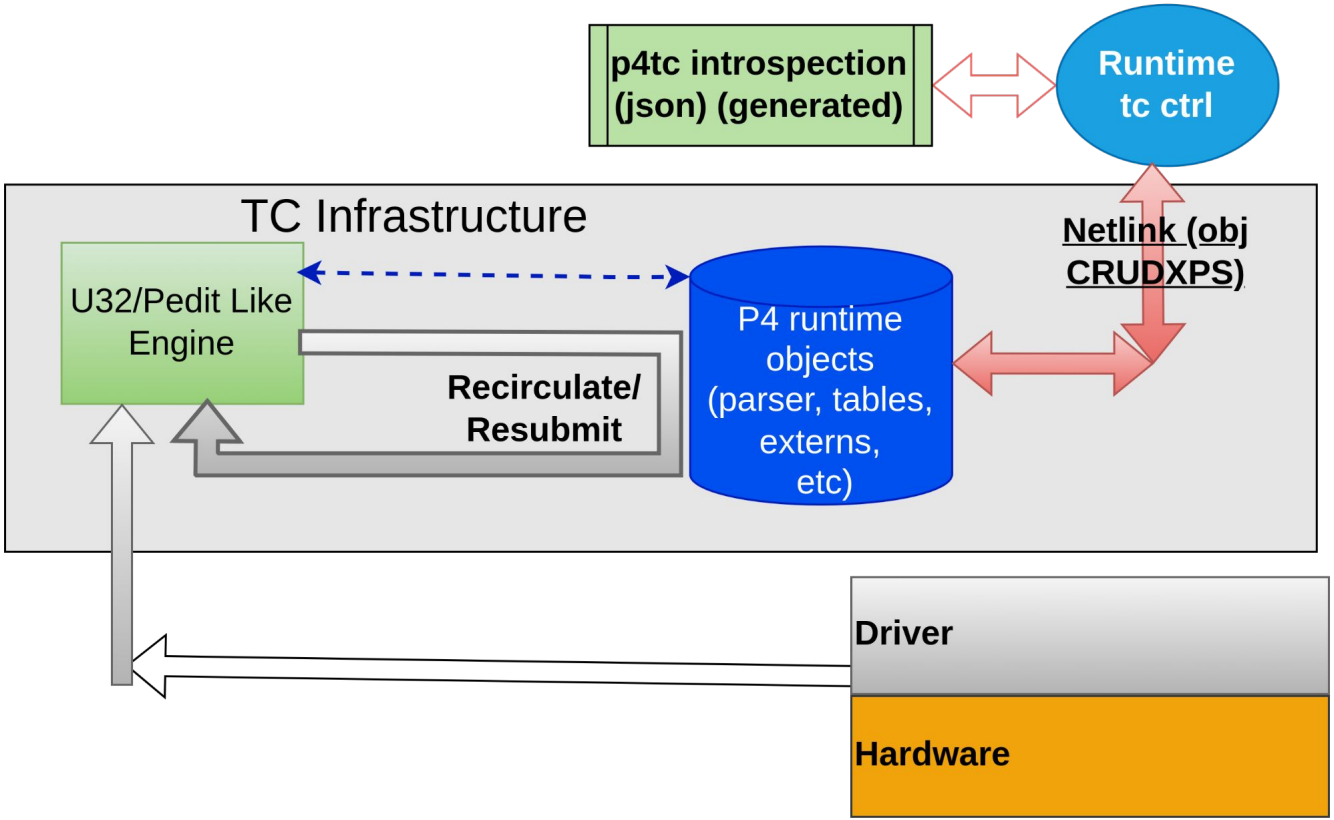
P4TC Original Workflow



Generated

1. P4TC Template (Pipeline, etc)
2. P4TC Introspection json

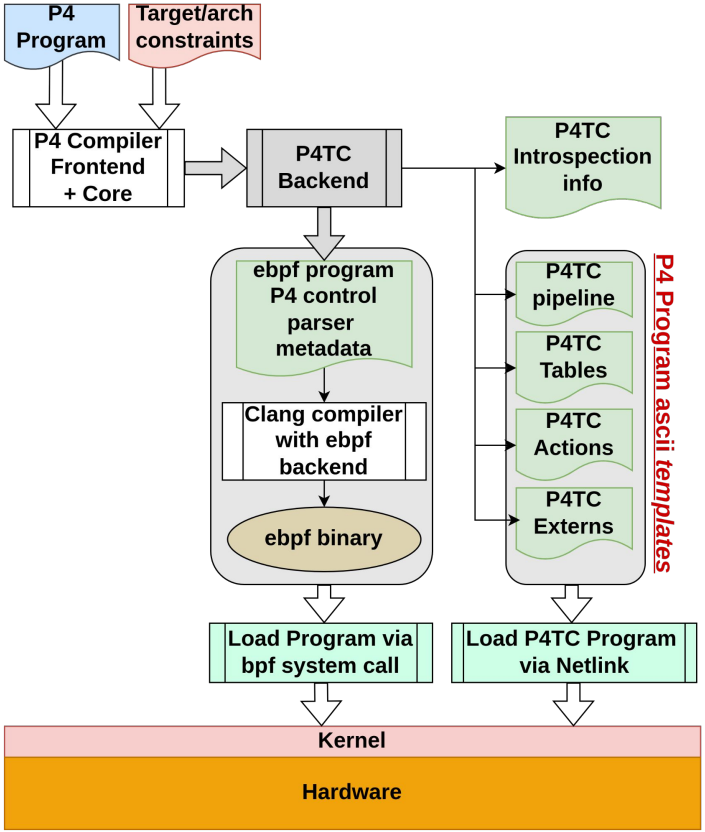
P4TC Original Datapath



Moving From Scriptable To eBPF….

- Feedback on the ML to move to eBPF, theory is:
  - eBPF datapath gives us more security (eg parser)
  - eBPF datapath gives us better performance
  - See reference [3] for evaluation
- Cost us 10 months of development and testing time!
- Patches focussing only on s/w datapath
  - Once merged will produce patches for h/w datapath
    - Ongoing effort

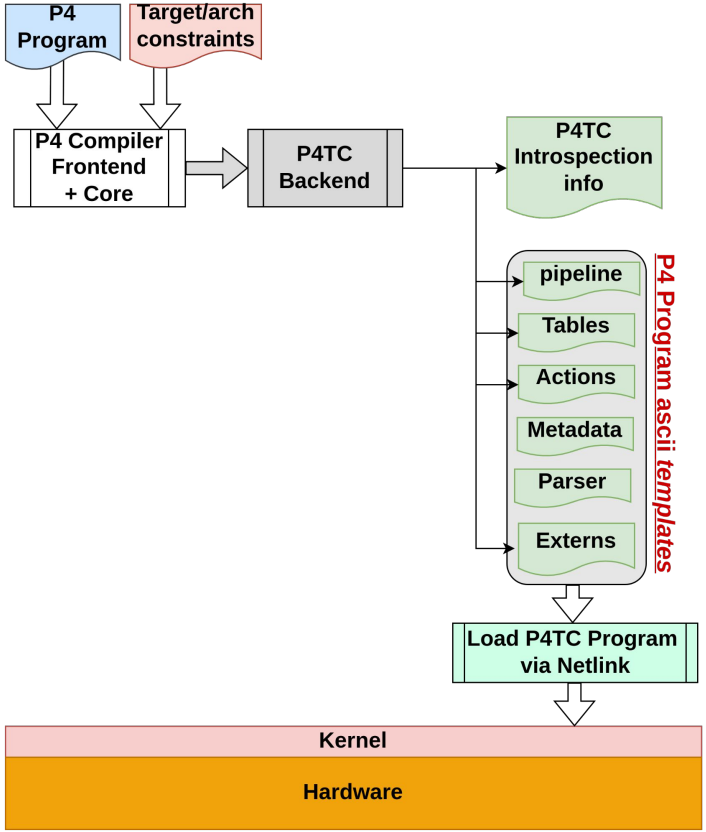
P4TC New Workflow

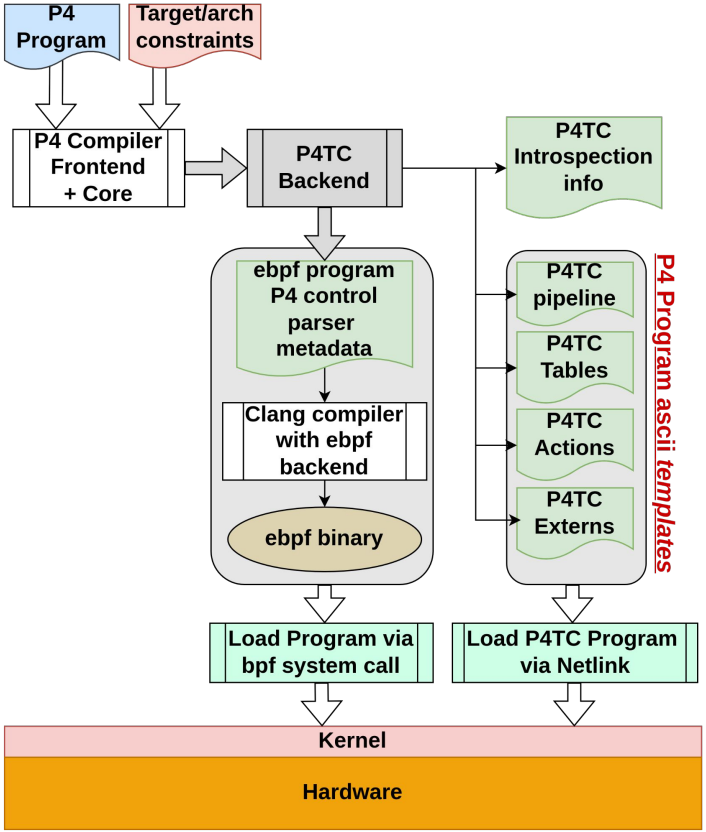


Generated

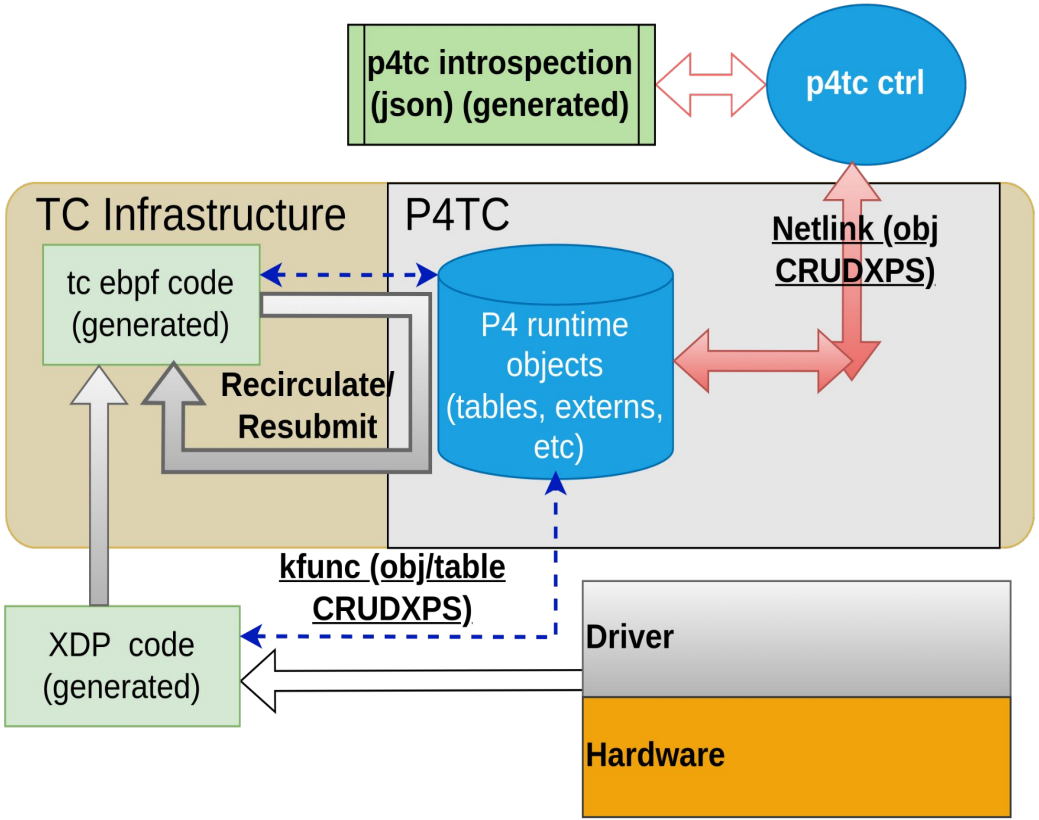
1. P4TC Template (minus parser, and metadata residing in generated eBPF)
2. P4TC Introspection json
3. eBPF s/w datapath (at tc and/or xdp level)
   *Per packet execution engine

P4TC Original To New Workflow



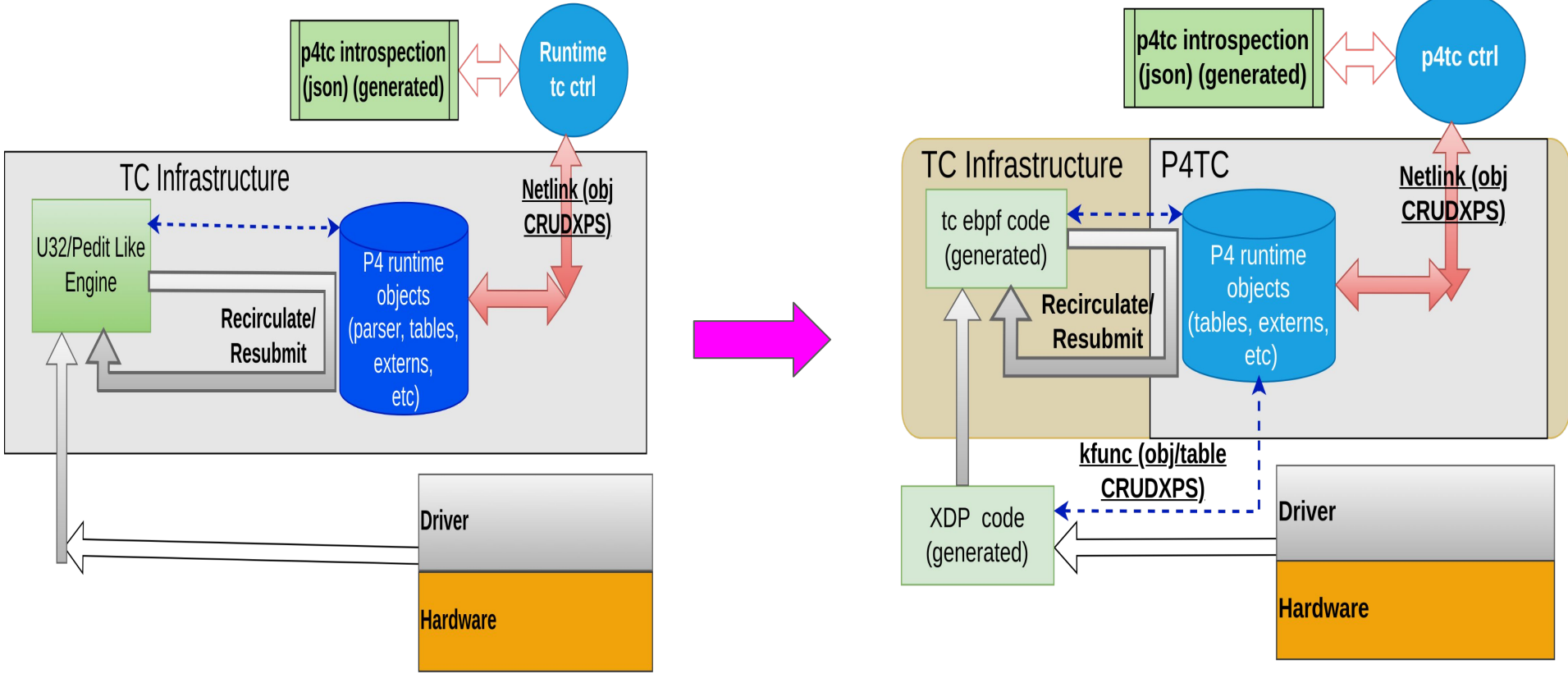


P4TC New Datapath

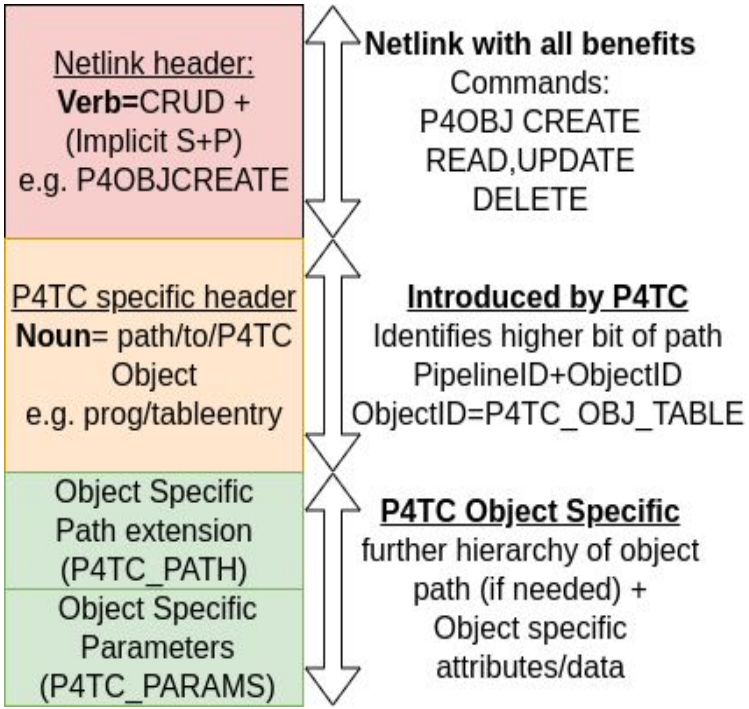


- eBPF serves as per packet exec engine
  - Parser, control block and deparser
- P4 objects that require control state are unchanged, (attached to netns)
  - Actions, externs, pipeline, tables and their attributes (default hit/miss actions, etc)
  - Kfunc to access them if needed

P4TC Original To New Datapath



Control Plane Runtime CRUDXPS Interface



Goal: Very High throughput and Low Latency interface

<VERB> <NOUN [OPTIONAL DATA]>+

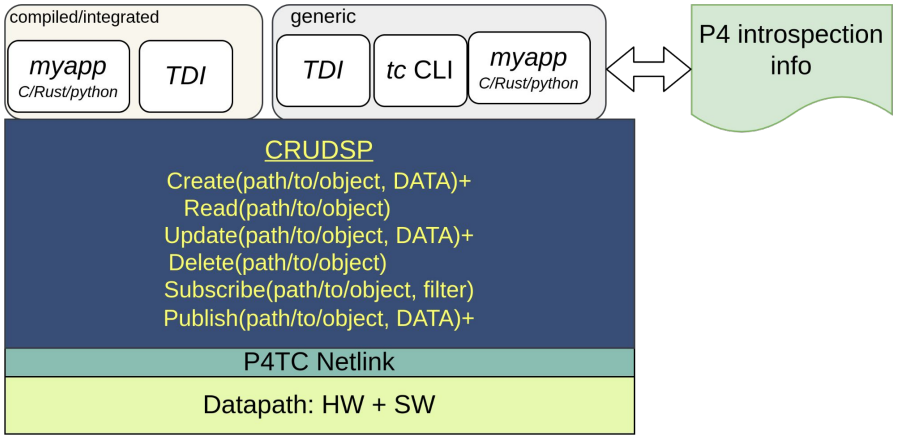
```
#Read a single Table entry
tc p4ctrl get myprog/table/control1/mytable ip/dstAddr 1.1.1.1/32 prio 16

#Read/Dump a whole Table
tc p4ctrl get myprog/table/control1/mytable

#create a single table entry
tc p4ctrl create myprog/table/control1/mytable ip/dstAddr 1.1.1.1/32 prio 16 \
action myprog/control1/drop

#create many entries
tc p4ctrl create myprog/table/control1/mytable \
   entry ip/dstAddr 10.10.10.0/24 prio 16 action myprog/control1/drop \
   entry ip/dstAddr 1.1.1.1/32 prio 32 action myprog/control1/drop \
   entry ip/dstAddr 8.8.8.8/32 prio 64 action myprog/control1/drop
```

P4TC Control API Abstraction



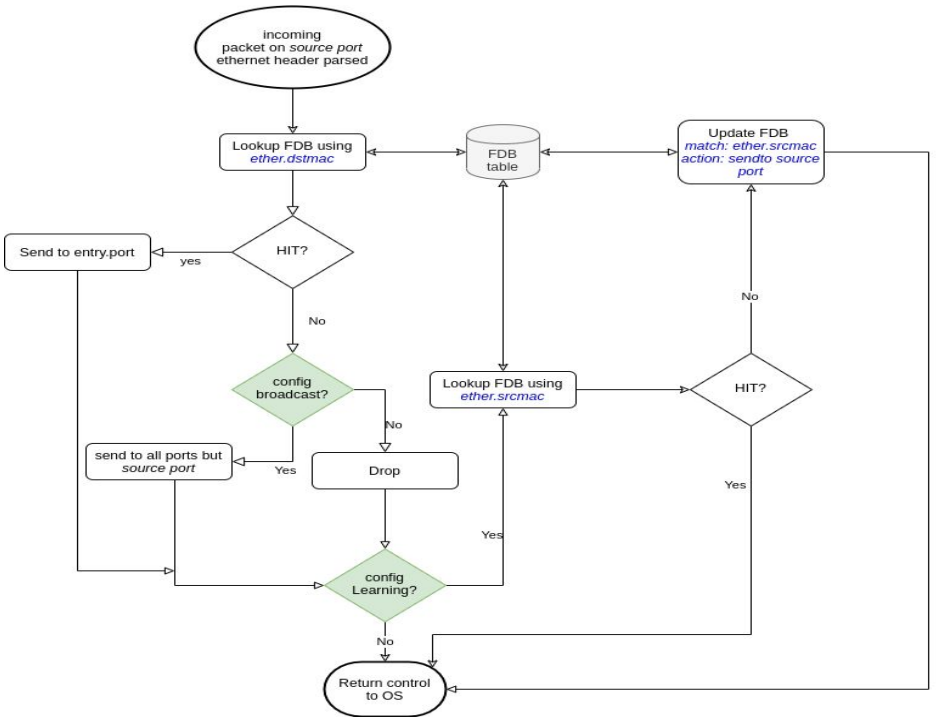
Interface Goals:

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

Some Sample Progs

Sample Programs

1. Trivial program with one table looks up based on IP address, rewrites Src + dst MAC address, send to a port
2. A ridiculous calculator program
   - Send packet with two operands + operator
     - Does math in datapath and responds
3. A stateful example (see figure)
   - mimics basic bridge
     - Can broadcast, learn etc
4. Can define very complex progs
   - 5G setup etc



Some Performance Numbers

Data path: Intel Cascade Lake CPU, NVIDIA 25Gbps CX6 card

- 64 byte packets achieved 10M packets per core and 35M on 6 cores

Control path VM on AMD Ryzen 4800H (4 allocated CPUs)

- “Worst Case” implies action params were allocated and “Best case” implies actions are preallocated
- Test case adds 1M entries as fast as possible
- Results
  - Best case 641k entries per second on 1 core
  - Worst case 463k entries per second on 1 core
  - Best case on 4 cores 1.78M entries per second
  - Worst case on 4 cores 1.64M entries per second

Testing…

- Code has been ready for some time now
  - Checkpath
  - Sparse
  - Syzkaller
  - Compilation patch-by-patch (clang and gcc) in x86, arm64 and s390 (using tuxmake)
  - Compilation patch-by-patch 32-bit (using tuxmake)
  - Control path TDC testing
    - > 300 test cases
  - Coverity
  - Smatch
  - Coccinelle
  - Future
    - Datapath test case generation via P4C compiler
    - TDC control path test case generation via P4C compiler

Status

- Code has been ready for some time now
- Issued 6 RFCs
  - Some reviews and a few reviewed-bys
- Version 7 removed the RFC unfortunately it upset patchwork (CICD didn't test properly with C=1 W=1)
- Sending V8 after net-next reopens
  - Please review!
- Kernel: https://github.com/p4tc-dev/linux-p4tc-pub
- Iproute2: https://github.com/p4tc-dev/iproute2-p4tc-pub
- Compiler: https://github.com/p4lang/p4c/tree/main/backends/tc

References

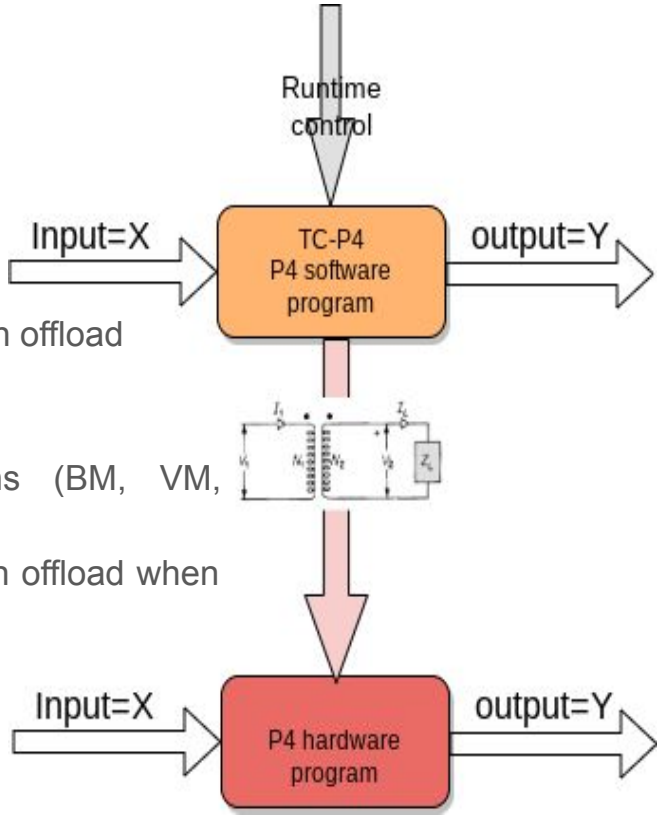
1. https://netdevconf.info/0x16/sessions/talk/your-network-datapath-will-be-p4-scripted.html
2. https://netdevconf.info/0x16/sessions/workshop/p4tc-workshop.html
3. https://github.com/p4tc-dev/docs/blob/main/p4-conference-2023/2023P4WorkshopP4TC.pdf
4. https://github.com/p4tc-dev/docs/blob/main/why-p4tc.md#historical-perspective-for-p4tc
5. https://2023p4workshop.sched.com/event/1KsAe/p4tc-linux-kernel-p4-implementation-approaches-and-evaluation
6. https://github.com/p4tc-dev/docs/blob/main/why-p4tc.md#so-why-p4-and-how-does-p4-help-here
7. https://github.com/p4lang/p4c/tree/main/backends/tc
8. https://p4.org/
9. https://www.intel.com/content/www/us/en/products/details/network-io/ipu/e2000-asic.html
10. https://www.amd.com/en/accelerators/pensando
11. https://github.com/sonic-net/DASH/tree/main

Integrating HW offload

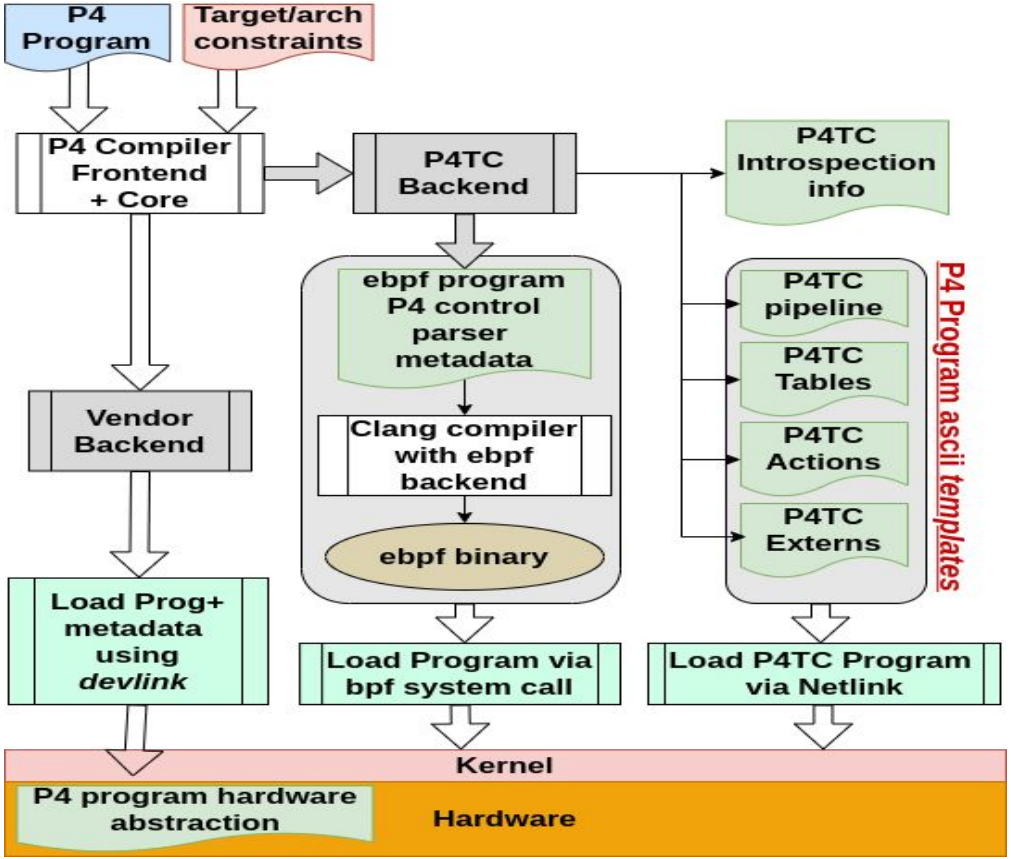
Ongoing effort (not part of patchset)

Recapping 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



P4TC New Workflow With HW offload



P4TC New Datapath With HW offload