# Introducing P4TC
# A P4 Implementation on Linux Kernel Traffic Control

Euro P4 workshop (Dec 08/2023)
Paris, France

Jamal Hadi Salim[1], Deb Chatterjee[2], Victor Nogueira[1], Pedro Tammela[1], Tomasz Osinski[2, 3], Evangelos Haleplidis[1, 4], Sosutha Sethuramapandian[2], Balachandher Sambasivam[2], Usha Gupta[2], Komal Jain[2]

Mojatatu Networks[1], Intel[2], Warsaw University of Technology[3], University of Piraeus[4]

# Motivation

# Motivation

## Goal: Grow Network Programmability ecosystem

- Datapath definition using P4
  - Linux kernel native P4 implementation
  - Mundane developer knowledge automated into compiler
    - knowledge shift to system (and P4) from HTA kernel skills
    - Zero upstream effort
- Same interfaces for either s/ware or h/ware datapaths
  - TC offload functionality

# Motivation

- Why P4?
  - **<u>Only</u>** open/existing standardized (with h/w) language for describing datapaths
  - Commoditization happening with native P4 support on xPUS (Intel and AMD)
    - Intel Mev support in progress
  - Large consumers of NICs require at minimal P4 for <u>datapath behavioral description</u> 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

- 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)
  - Reuse existing TC interface
    - Consistent regardless of deployment being SW or HW

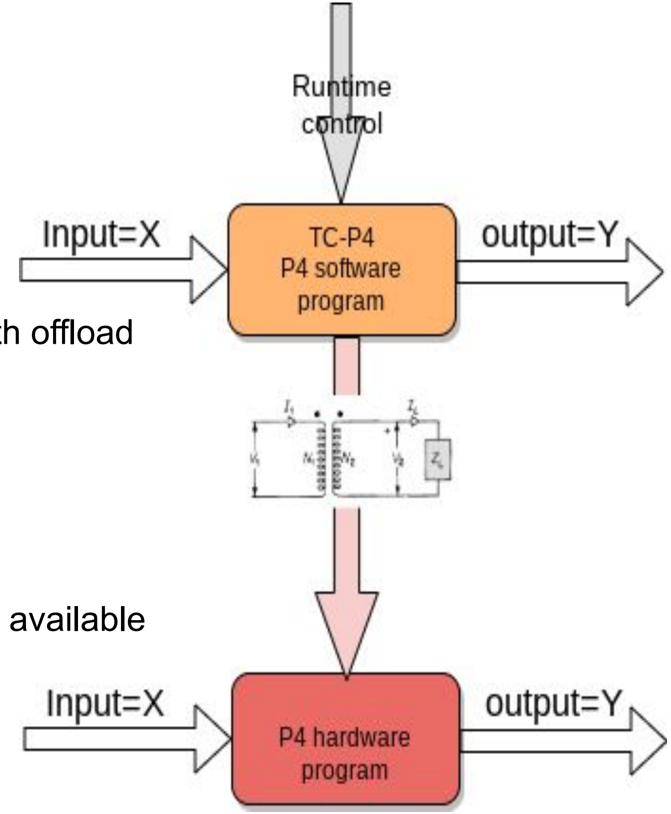
# P4TC Workflow And Runtime Architecture

# Introduction to P4TC

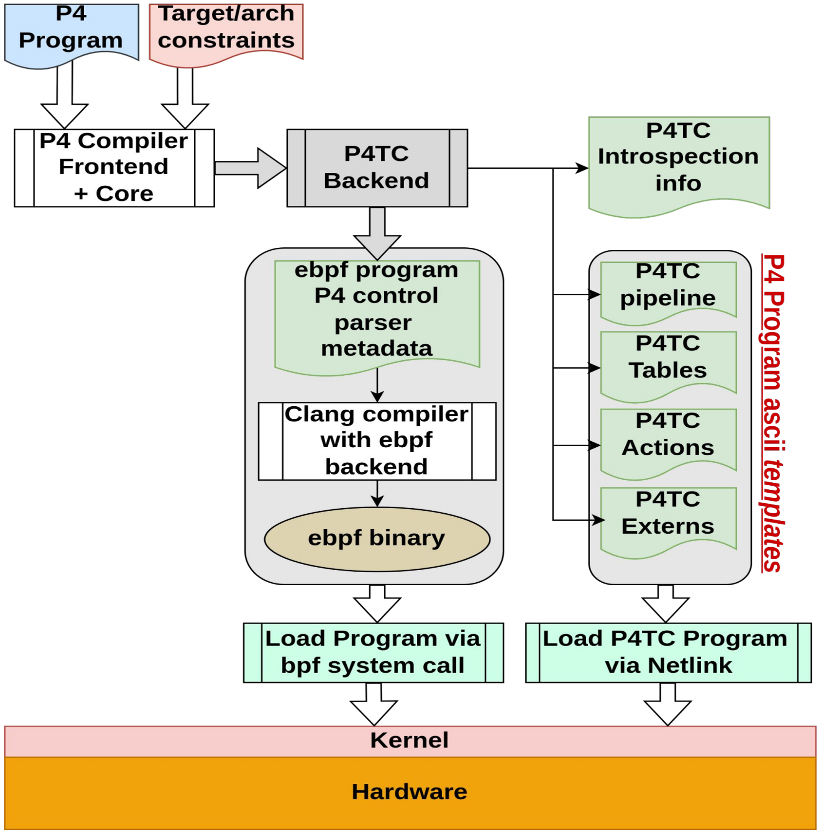
- TC based kernel-native P4 implementation
- Learn from previous experiences (tc flower, u32, switchdev, etc) and scale
    - Kernel independence
    - Control plane transaction rate and latency
- P4 Architecture Independence
    - Currently PNA with some extra “constructs”
        - Not hard to add other architectures
        - 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 userspace punting infrastructure (popularized by Cumulus)

# P4TC: Building On TC Offload

- Datapath definition using P4
  - Generate the datapath for 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
    - Understood Infra tooling which already has deployments
  - Seamless software and hardware symbiosis
  - Functional equivalence whether offloading or s/w datapaths
    - Bare Metal, VMs, or Containers
  - Ideal for datapath specification
    - test in s/w container, VM, etc) then offload when hardware is available



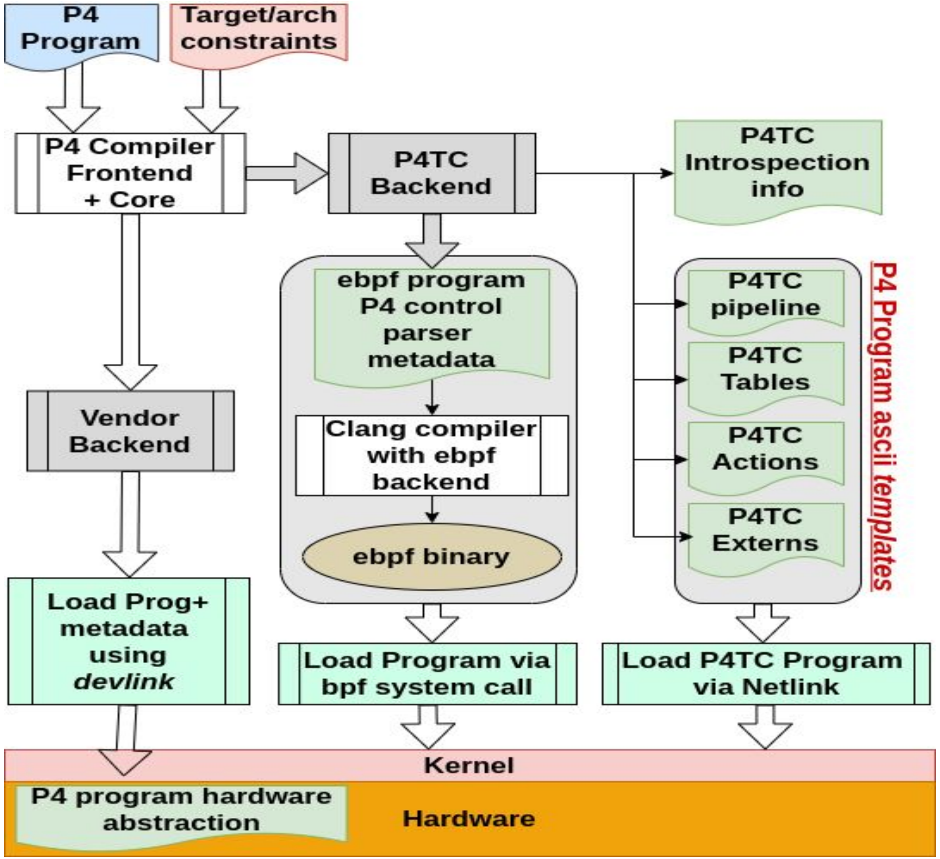
# P4TC Software Datapath Workflow




Generated

1. P4TC Template (Loaded via generated) script
2. P4TC Introspection json (used by CP)
3. eBPF s/w datapath (at tc and/or xdp level)
   *Per packet execution engine
   (compiled and loaded when instantiating)

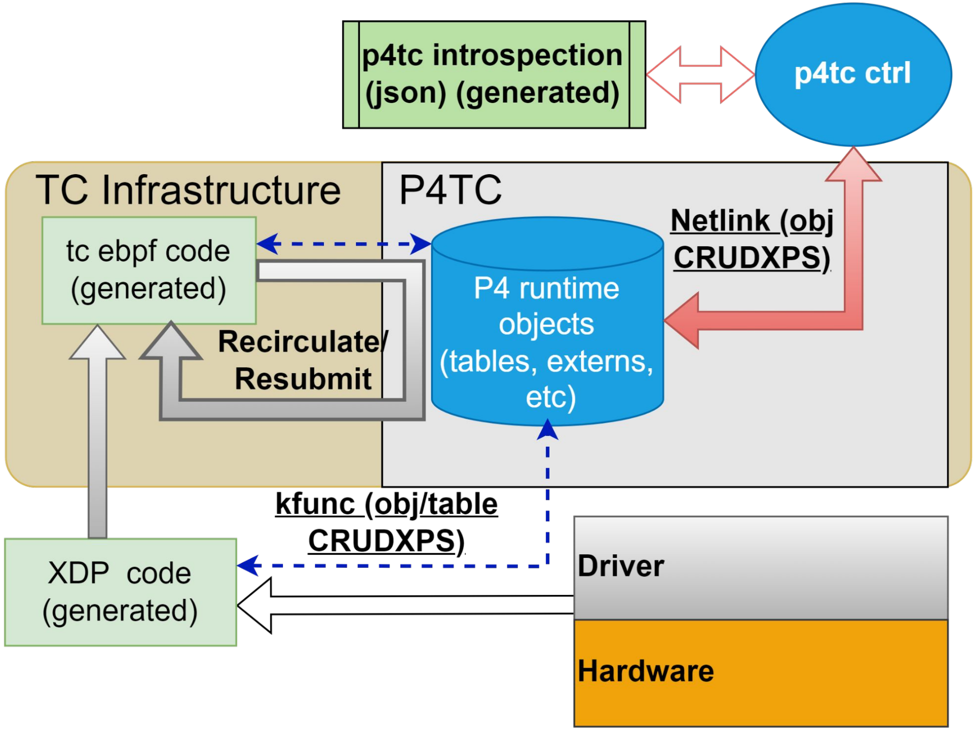
# P4TC Workflow With HW offload




HW offload path also generates:

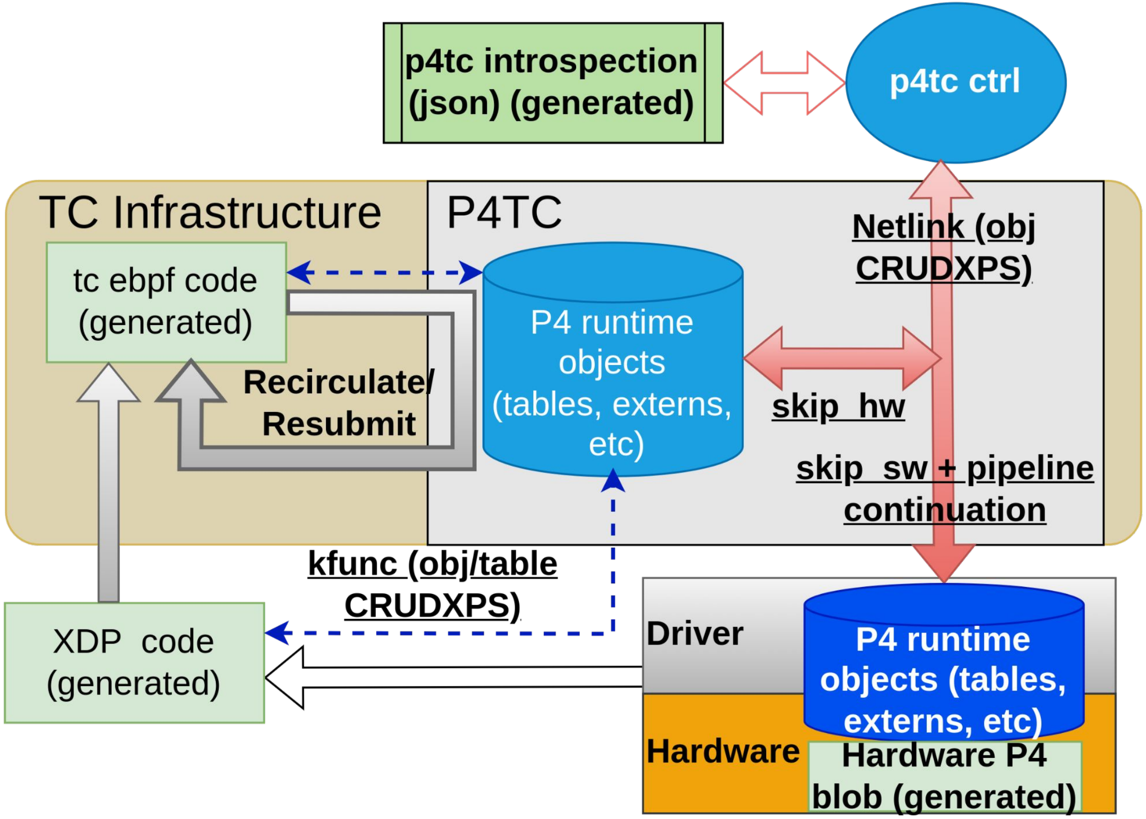
- **Binary hardware blob**
  - Compatible with vendor hardware
  - Loaded via firmware upload mechanisms

# P4TC Runtime S/w Datapath



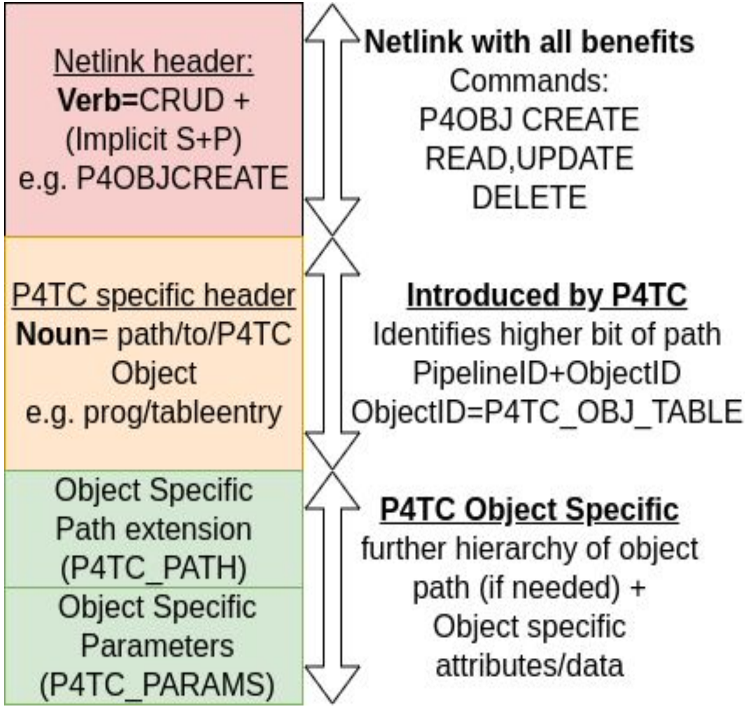

- eBPF serves as <u>per packet</u> exec engine
  - Parser, control block and deparser
- P4 objects that require control state reside in TC domain (attached to netns)
  - Actions, externs, pipeline, tables and their attributes (default hit/miss actions, etc)
  - Kfunc to access them from ebpf when needed

# P4TC Datapath With HW offload



# Control Plane Integration

# Control Plane Runtime CRUDXPS Interface




Goal: Very High throughput and Low Latency interface

<VERB> <NOUN [OPTIONAL DATA]>+

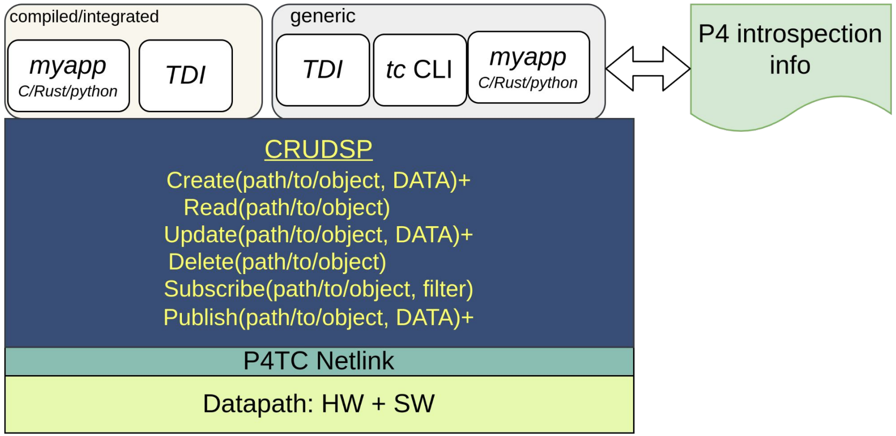
#Read a single Table entry
tc p4ctrl get myprog/table/control1/mytable ip/dstAddr 1.1.1.1/32 prio 16

#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

# Performance

# Some S/Ware Performance Numbers

Simple l3 forwarding app

- 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
        - 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

# Challenges And Opportunities

# Some Challenges And Opportunities (1)

- Kernel Challenges
  - Assumptions of statically defined  objects like P4 match actions
    - Introduced templating DSL to teach the kernel how to manifest a P4 pipeline
  - eBPF non-turing completeness
    - Used kfuncs
  - Social challenges in upstream process
    - Scriptable Version 1 met huge resistance from the eBPF folks
      - Took us 10 months of multiple people effort to convert to eBPF

# Some Challenges And Opportunities (2)

- P4 not well suited for defining control constructs
    - We worked around things by introducing annotations
- P4 constructs being hardware biased
    - Eg deparser emit centres around headers vs payload splitting
        - Ok for HW. SW has the full payload and dont need to emit headers when no header edit
- P4 Const definitions for tables and default actions to make them read-only
    - Opportunity: We extended to allow for a more refined approach for runtime objects
        - “CRUDXSP” Permissions to describe what the control plane or datapath is allowed to do
- Externs
    - P4 provides signature definitions for externs
    - Work the same way from a control plane perspective as any other object using annotations
    - <u>User defined custom externs</u> can be written as kernel modules
        - C or Rust, and interfaced with generated kfuncs from eBPF
        - Simple custom externs dont require any code

# Future Work And Status

# Ongoing and Future work

- **Ongoing work**
  - Improvement and stabilization of generated code
    - We may be missing some missing features
  - More refinement of externs
  - Generating datapath test cases using p4testgen
  - Generating of control plane test cases
  - Add other P4 architectures
    - Should not require kernel changes
- **Future work**
  - Go beyond P4: experiment then push for P4 standardization
  - Teach or build a new compiler to generate “distributed pipelines”

# Status

- Code has been ready for some time, most effort is spent juggling with upstream folks!
  - Sent V9 last week
    - 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
- Vagrant Tutorial Link
  - https://github.com/p4tc-dev/p4tc-tutorial-pub/tree/main
- Examples link
  - https://github.com/p4tc-dev/p4tc-examples-pub.git
- Good central link:
  - https://www.p4tc.dev

# Small Demo

# References


1. https://netdevconf.info/0x17/sessions/talk/integrating-ebpf-into-the-p4tc-datapath.html
2. https://netdevconf.info/0x16/sessions/talk/your-network-datapath-will-be-p4-scripted.html
3. https://netdevconf.info/0x16/sessions/workshop/p4tc-workshop.html
4. https://github.com/p4tc-dev/docs/blob/main/p4-conference-2023/2023P4WorkshopP4TC.pdf
5. https://github.com/p4tc-dev/docs/blob/main/why-p4tc.md#historical-perspective-for-p4tc
6. https://2023p4workshop.sched.com/event/1KsAe/p4tc-linux-kernel-p4-implementation-approaches-and-evaluation
7. https://github.com/p4tc-dev/docs/blob/main/why-p4tc.md#so-why-p4-and-how-does-p4-help-here
8. https://github.com/p4lang/p4c/tree/main/backends/tc
9. https://p4.org/
10. https://www.intel.com/content/www/us/en/products/details/network-io/ipu/e2000-asic.html
11. https://www.amd.com/en/accelerators/pensando
12. https://github.com/sonic-net/DASH/tree/main