P4TC: Kernel Implementation Approaches and Performance Evaluation

Jamal Hadi Salim (Mojatatu Networks)
Deb Chatterjee (Intel Corporation)
Victor Nogueira (Mojatatu Networks)
Pedro Tammela (Mojatatu Networks)
Tomasz Osinski (Intel Corporation)
Sosutha Sethuramapandian (Intel Corporation)
Balachandher Sambasivam (Intel Corporation)
Evangelos Haleplidis (Mojatatu Networks)

Introduction to P4TC

Expand P4 ecosystem to a wider audience

- P4 Linux kernel-native implementation
    - Kernel-based software datapath and Kernel-based HW datapath offload via TC
    - Use well understood and tested TC 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
- Kernel independence for P4 program (headers, parser, lookups, actions, etc)
    - No need to upstream any code for new P4 programs
        - Unlike other offload mechanisms like tc/flower
- P4 Architecture Independence
    - Allow for PSA, PNA, and new innovations on top
        - This is about progressing network programmability in addition to expanding P4 reach
- Scriptable
    - Simple operational model
        - Email the ascii P4 script to an operator

Status of P4TC

- Code available: https://www.p4tc.dev
  - Kernel code
  - Control plane via *iproute2::tc* (netlink)
  - Not public yet: P4C compiler code and driver offload code
- At this conference
  - See talk on driver interfaces offload
  - HW Offload Demo in the hallway
  - Workshop on the 26th at this location
- Other references: https://github.com/p4tc-dev/docs/blob/main/why-p4tc.md
- Ongoing discussions for upstreaming
  - We posted the first RFC to the netdev mailing list to gather initial feedback
    - Evaluating suggestions
      - eBPF software dataplane integration - which is the basis of this talk

Goals of Evaluation

Investigation of various approaches for the P4 software datapath integrating compiled ebpf datapath vs our posted scriptable datapath implementation

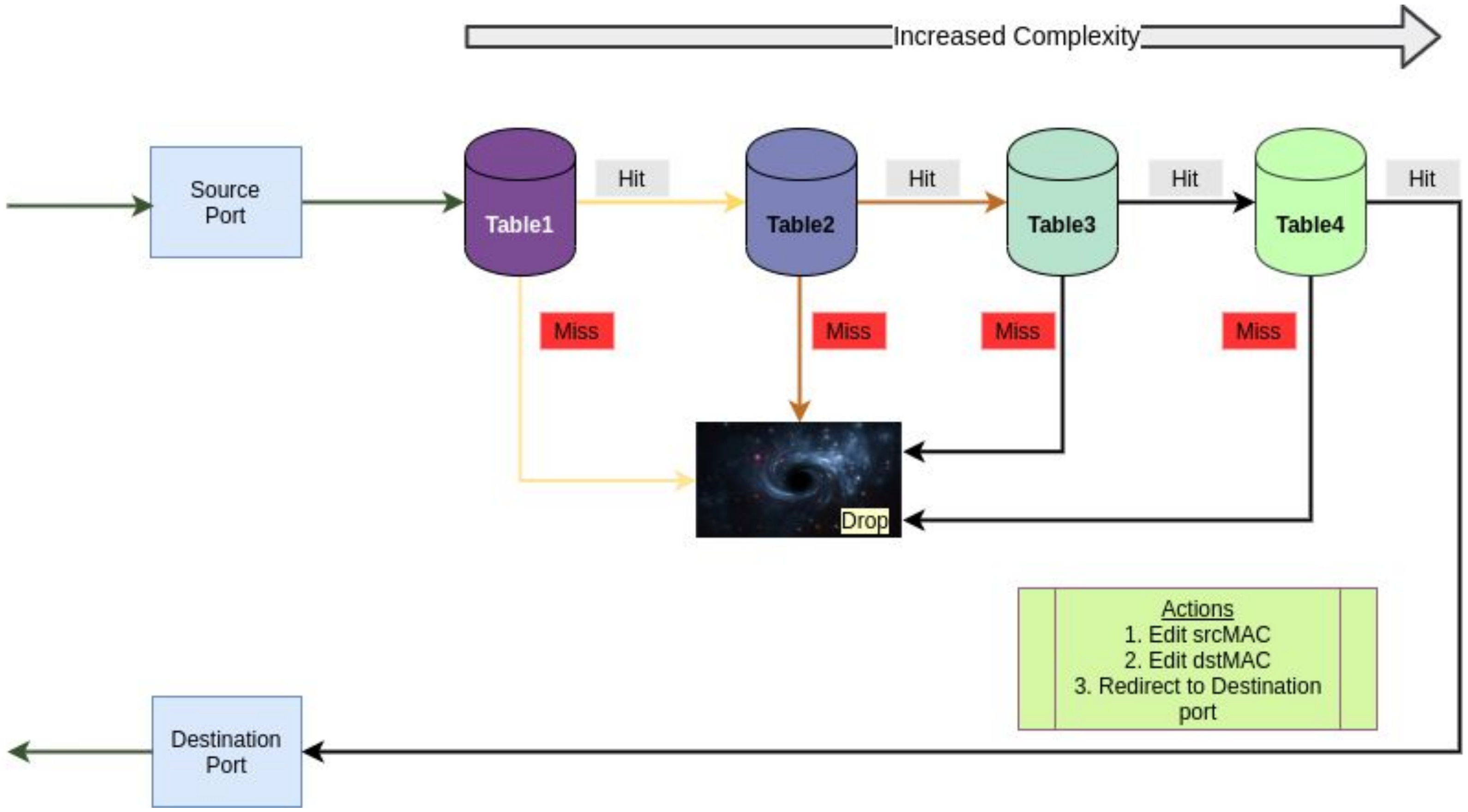
Key evaluation datapoints are:

- Performance of the s/w datapath
- Weighed out against: operational simplicity, debuggability, and ability to support all possible P4 programs and innovate new ideas

Iterations with the kernel community are expected

P4 Tests Complexity Setup

Adjust number of tables in the pipeline, per table entries, masks, how many entries to match traffic and increase computational complexity



P4 Test Program Complexity Variables

- The number of tables in a pipeline (varying from 1-4)
- The size of the table key (varying from 4-64 bytes)
- The number of entries(2-130K)
  - The number of entries being hit by traffic (2-130K)
- Number of masks for LPM(1-4) and ternary (4-192)
- The type of lookup: exact, LPM and ternary
- Computational complexity of the actions

Program-Under-Test Setup

Goals of Evaluation

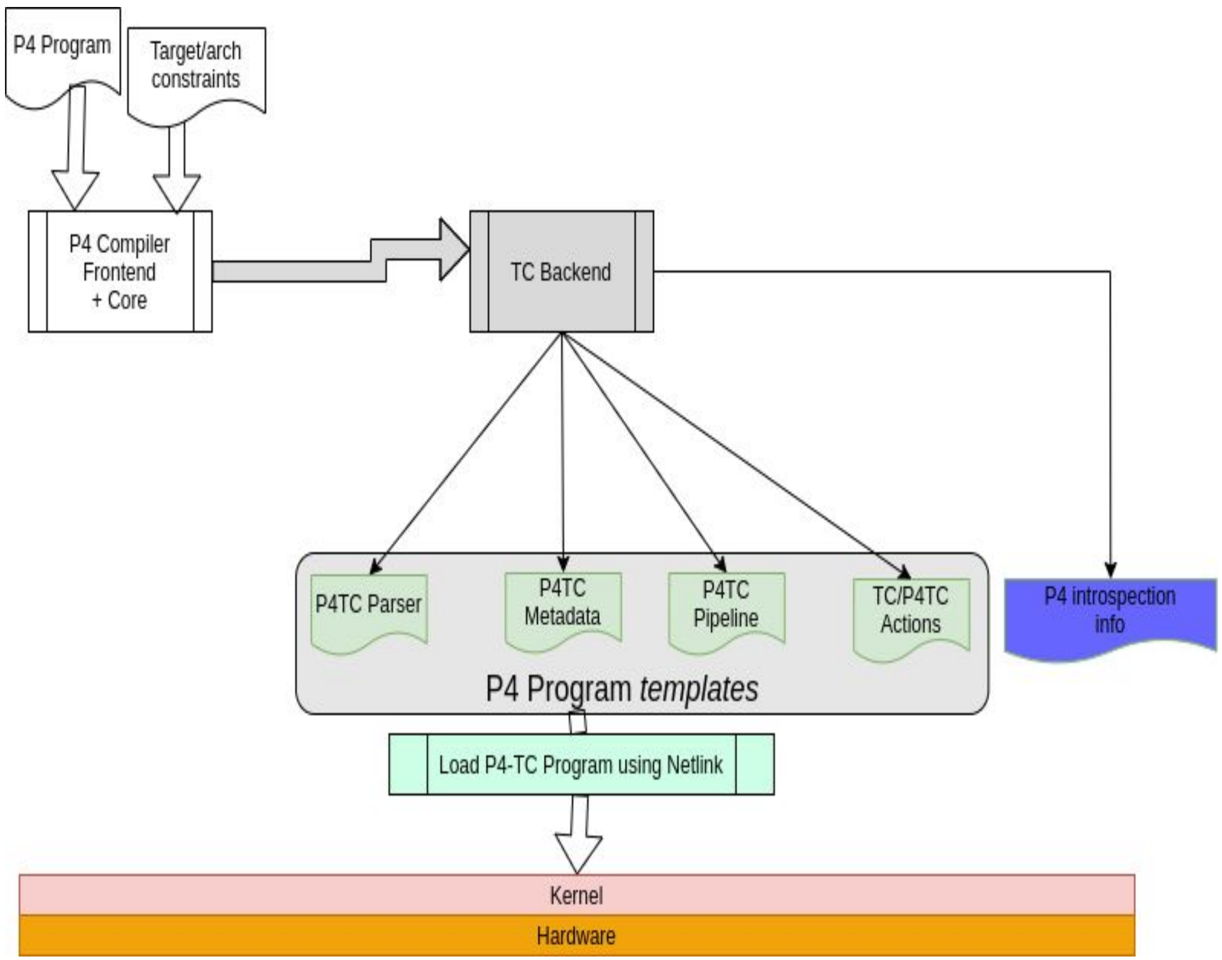
Investigate models of various implementation approaches for the P4 software datapath

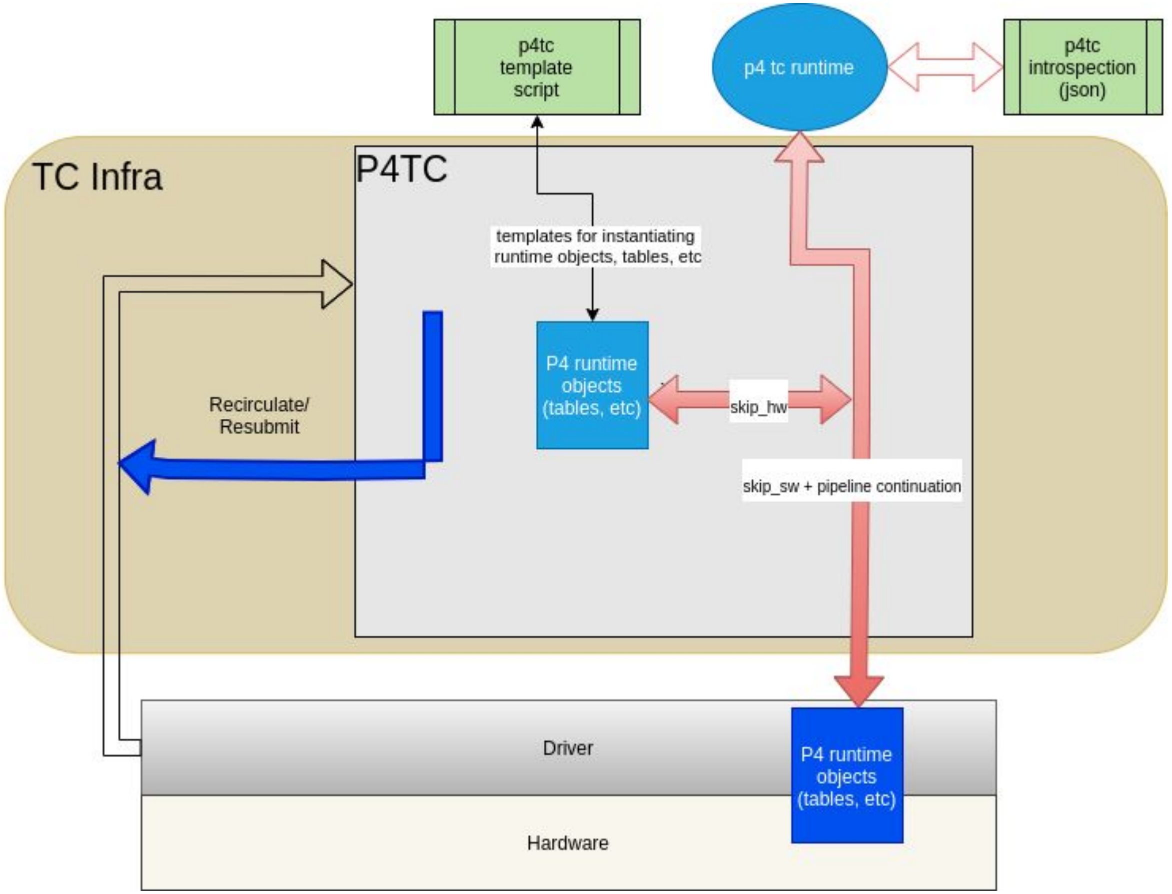
- Model 1: Scriptable P4TC
- Model 2: Partial TC/XDP eBPF(parser only) and P4TC being aware of parser
- Model 3: Plain eBPF at TC and XDP layers independent of P4TC
  - P4TC unaware of eBPF
  - eBPF limitations restrict range of testing
- Model 4: Hybrid approach of scriptable P4TC and eBPF at TC and XDP
  - P4TC aware of eBPF
  - eBPF limitations overcome with kfunc APIs provided by P4TC

Note:

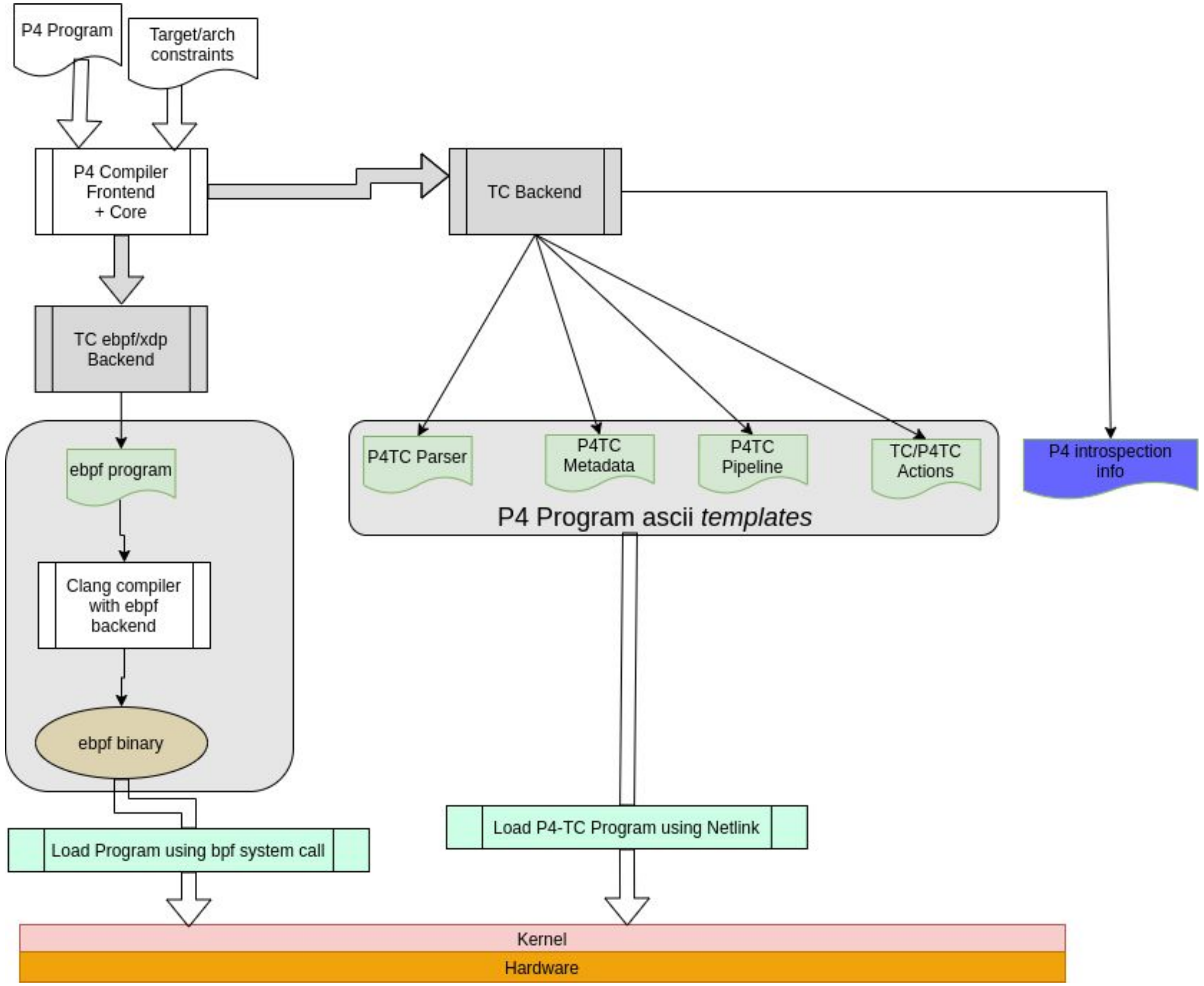
- Models 2 & 4 use modifications of P4C-eBPF (PSA backend) whereas 3 uses it unchanged
- Model 3 has two runtime control plane (s/w via eBPF and hardware via P4TC)

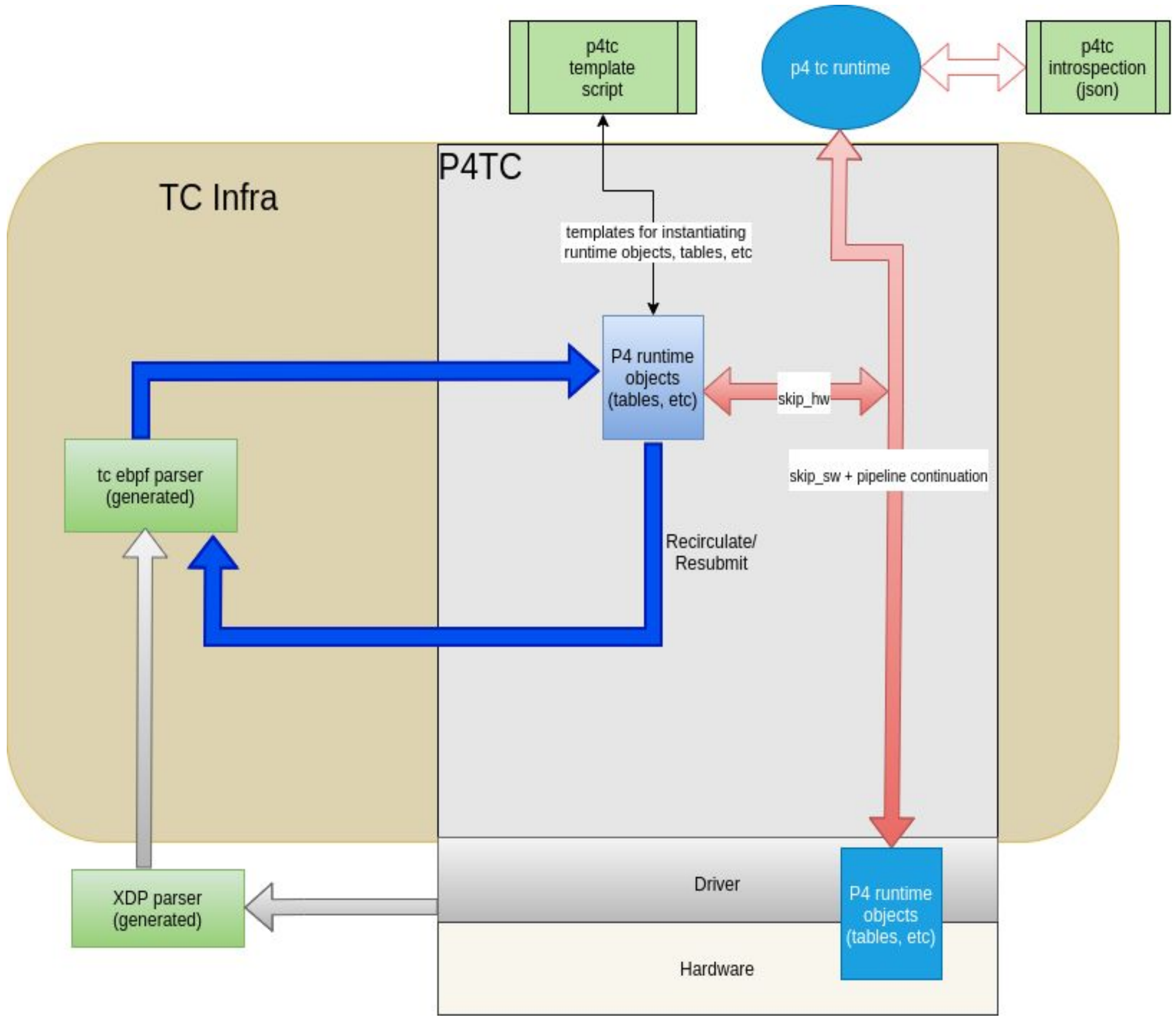
Model1: Scriptable P4TC (SW dpath via P4TC)



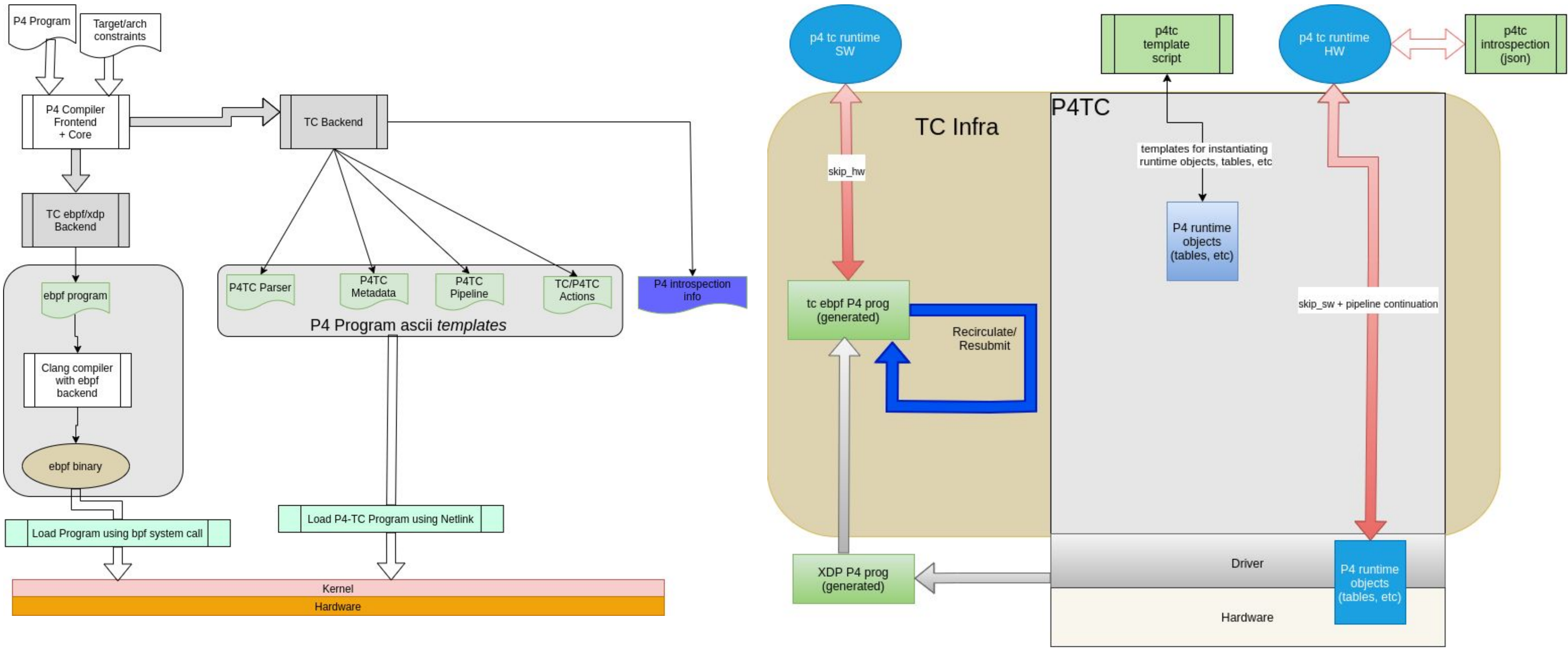


Model 2: eBPF Parser Only, rest of SW Dpath via P4TC

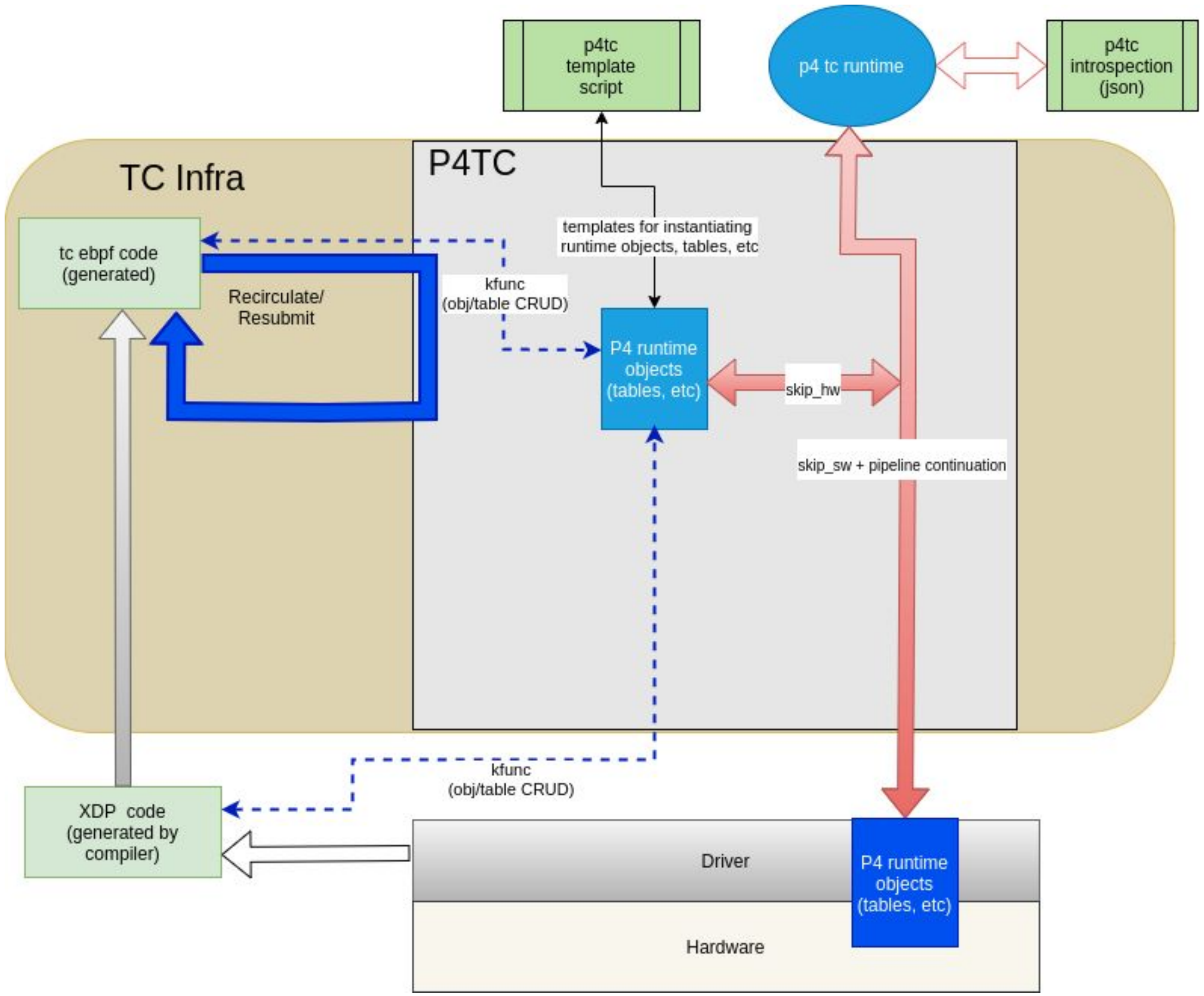




Model 3:SW dpath eBPF at TC+XDP independent of P4TC



Model 4: Integrated ebpf sw-dataplane P4TC control



Kfunc-able P4 objects:

1. Table (entries)
2. Registers
3. Counters
4. Meters
5. Multiple externs
   Checksum, hash, digest, etc
6. Set-value parser (reverse)

Tests

Test Constraints

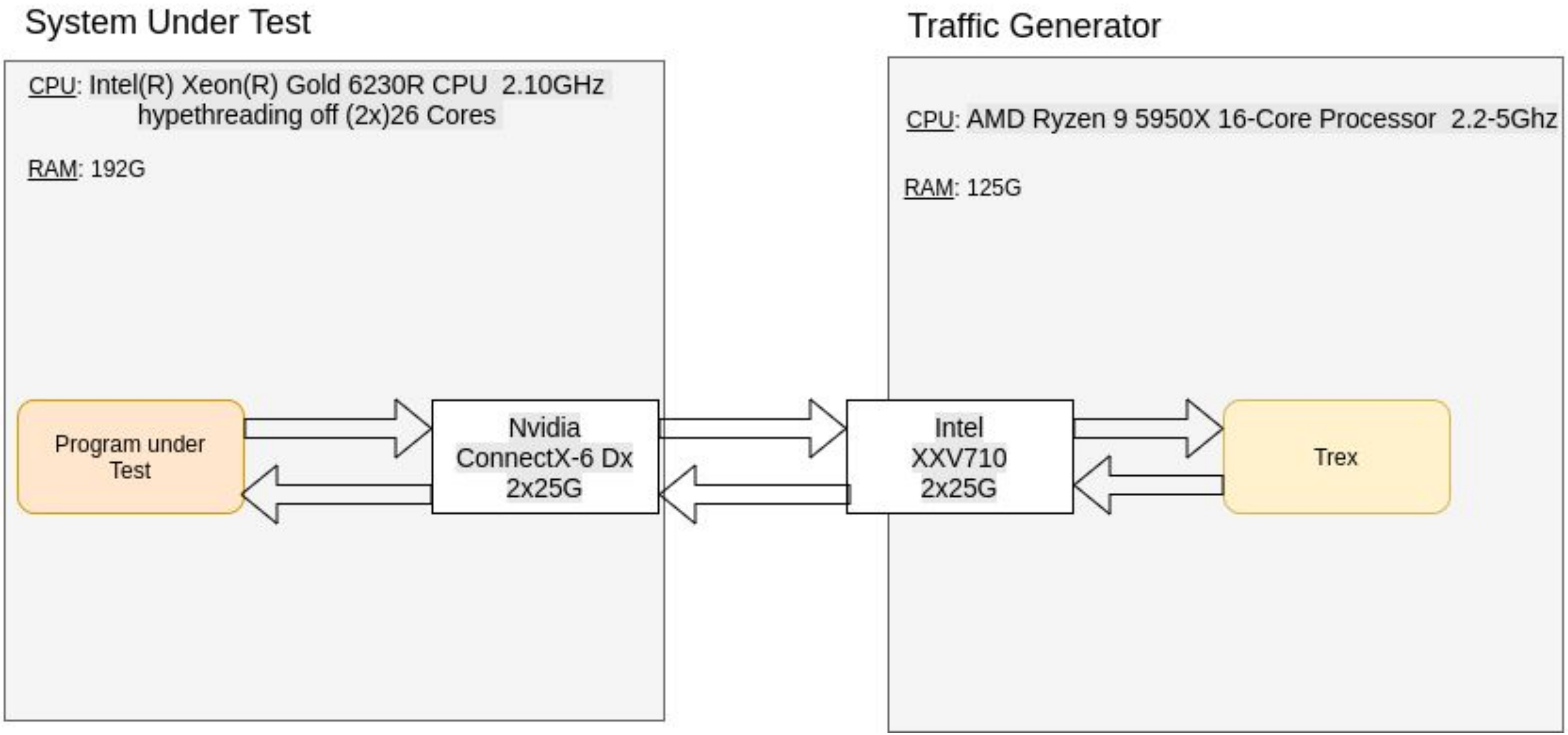
Test cases designed to be constrained by the capability of eBPF to map to P4 programs:

- Example limiting ternary masks to 192
- Don't do things XDP can't do, eg broadcasts, Large MTUs, LRO, etc

Restrict scope to one CPU

- Repeatable results

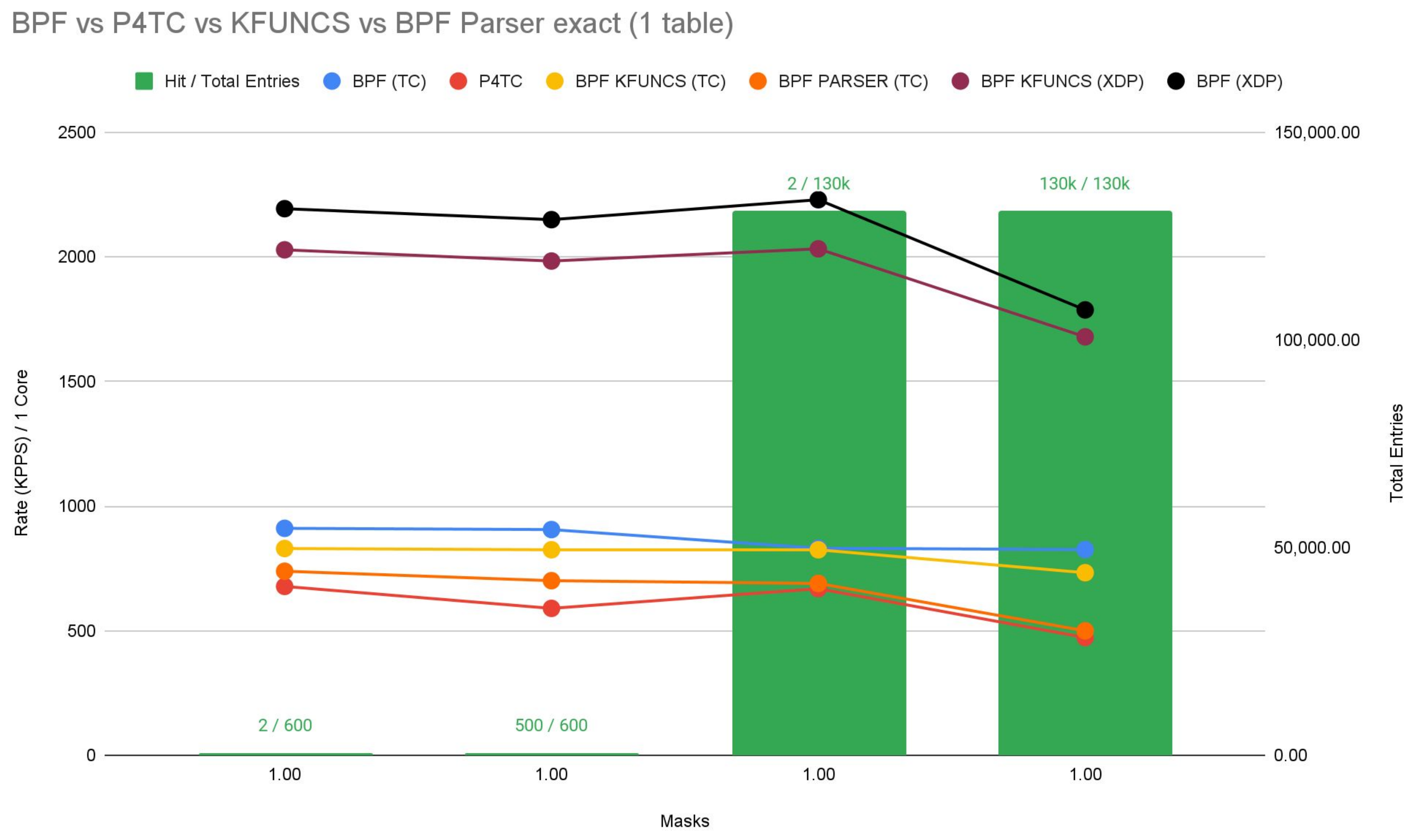
Test Environment Setup



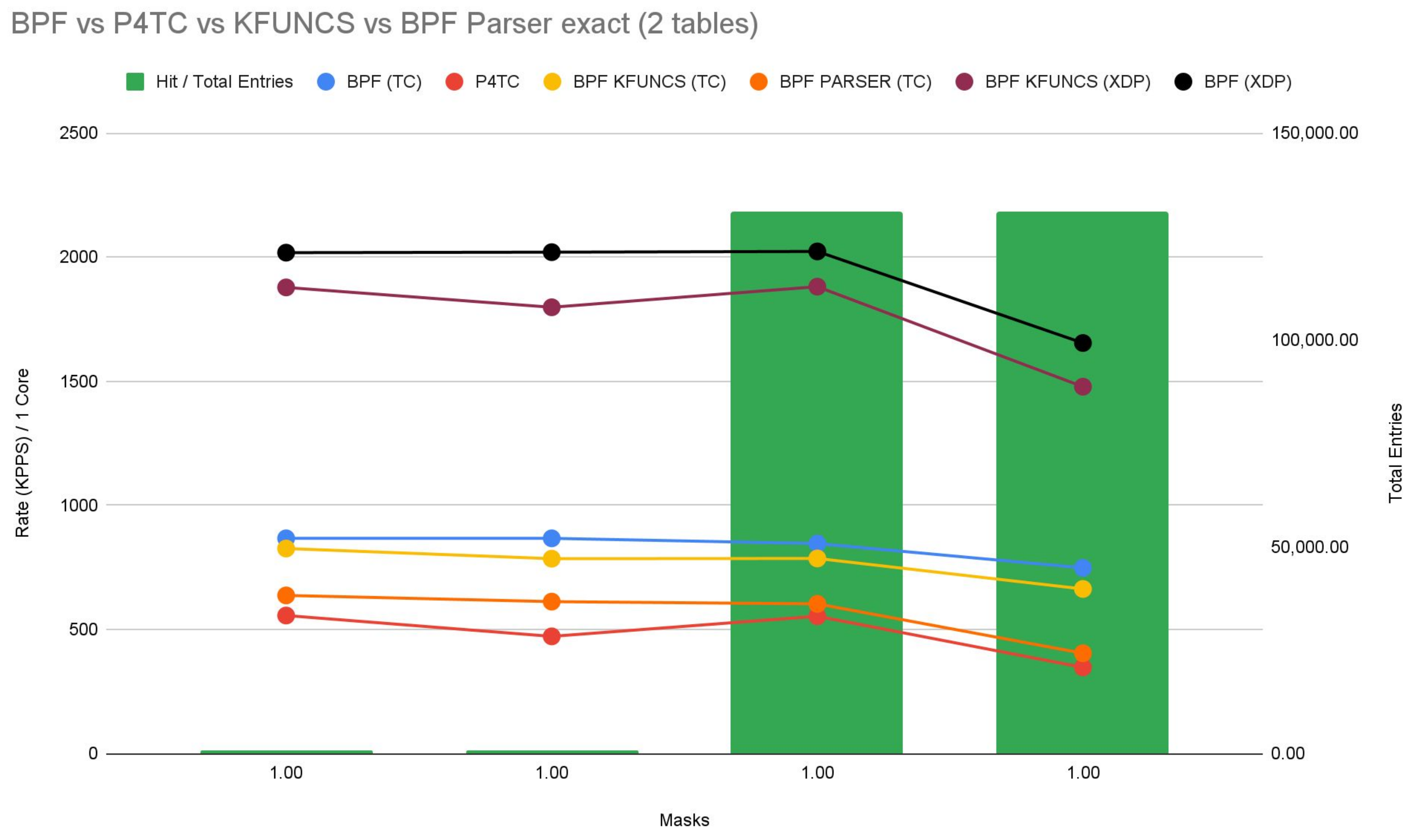
Setup And Result Collection

- SUT Setup
  - Net-next with P4TC patches
    - sudo sysctl -w net.core.netdev_budget=8000
    - sudo sysctl -w net.core.netdev_budget_usecs=8000
  - IRQ affinity to single core
  - Default DMA rings sizes (TX:1024, RX:1024)
  - Adaptive Tx/Rx interrupt mode
    - Interrupt either after receiving 128 frames or 8 microsecs
  - Performance mode scaling governor
- SUT Data Collected
  - CPU utilization
  - Perf profiling
  - Ethernet statistics
  - Memory statistics
- Traffic Generator
  - Controlled from SUT
    - Discover the non-drop rate (ndr) for each setup, then repeat the test at the ndr level
- Overall we run **1000+ tests** in total for all the setups

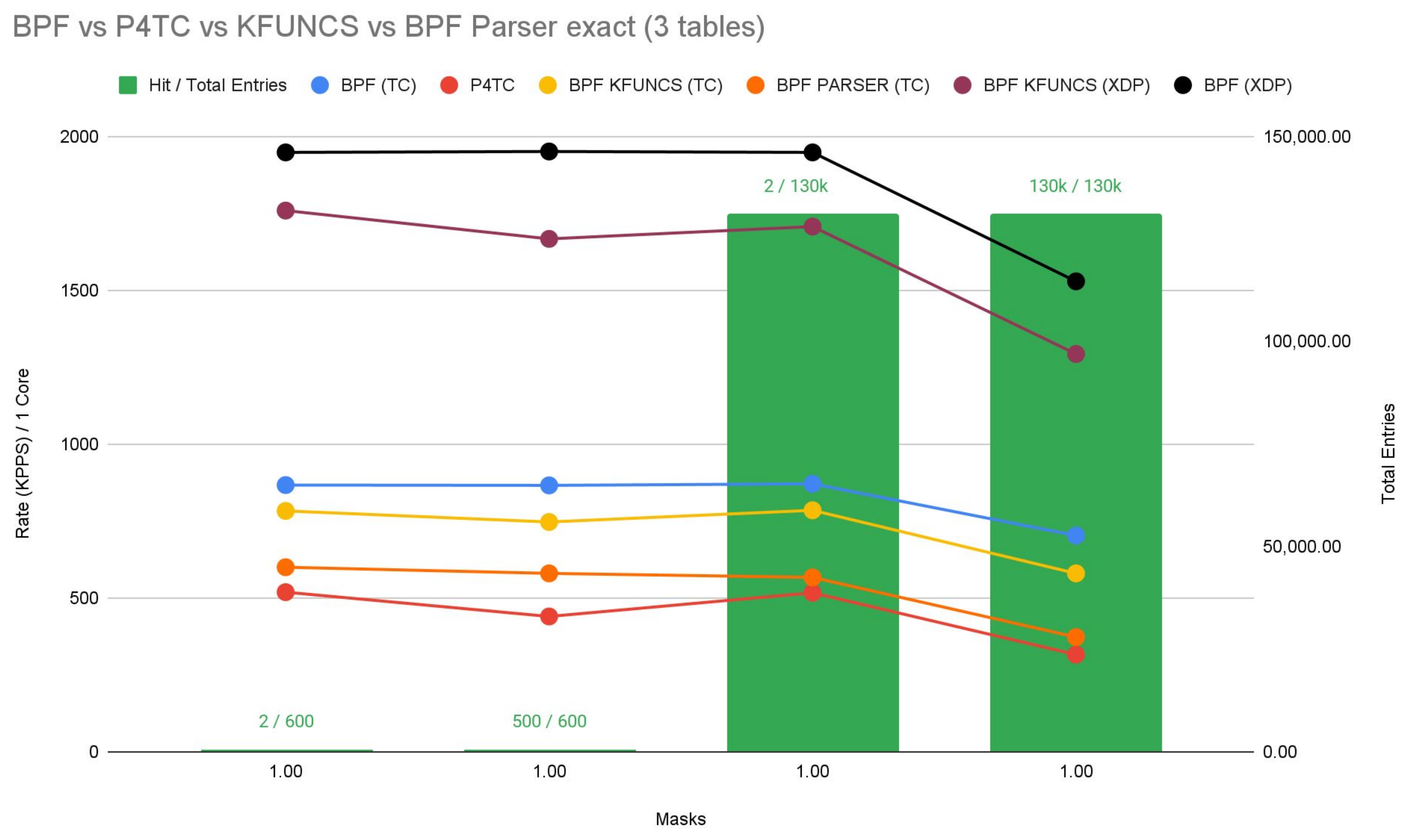
Exact 1 Table: L3 redirect



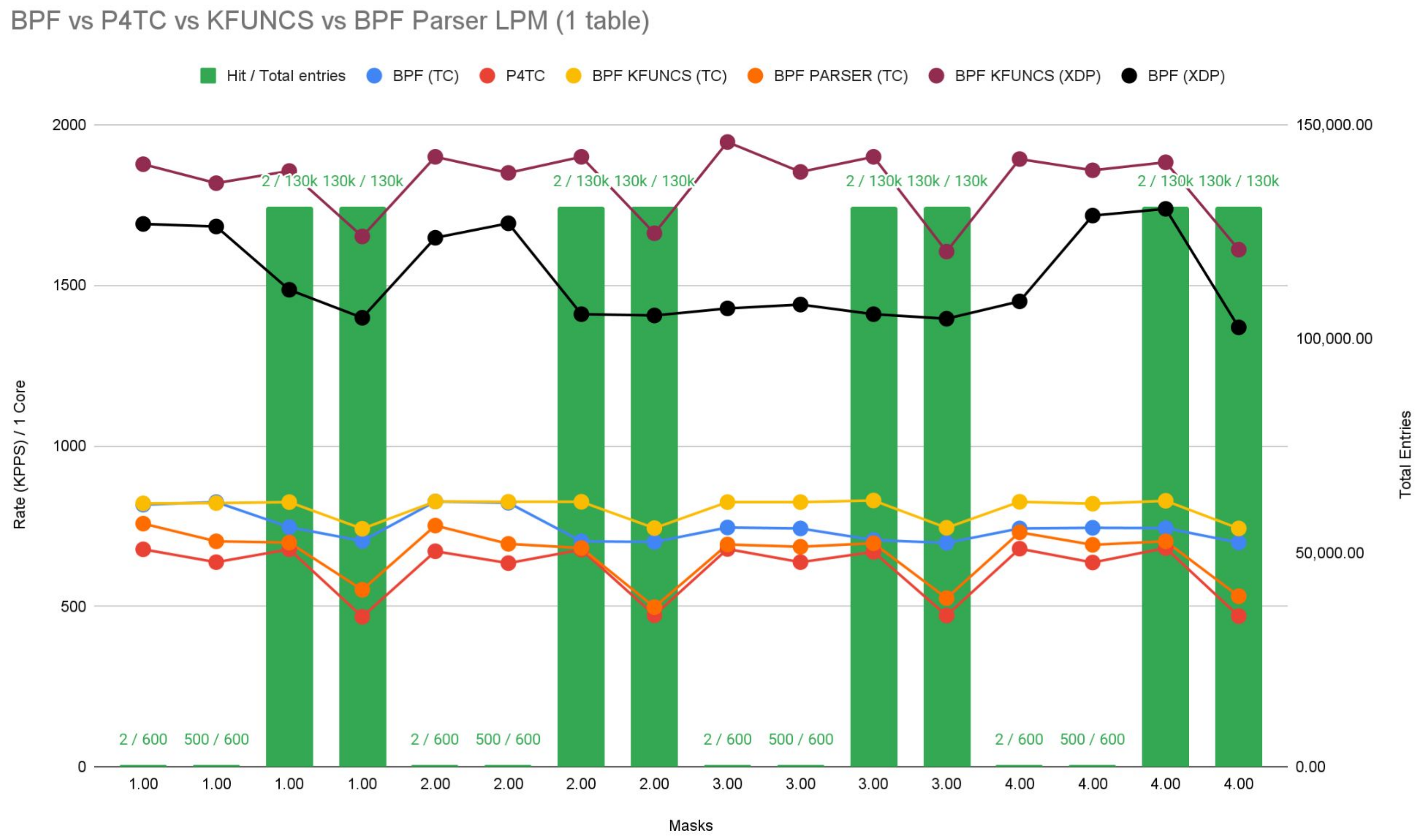
Exact 2 Table: L3 redirect



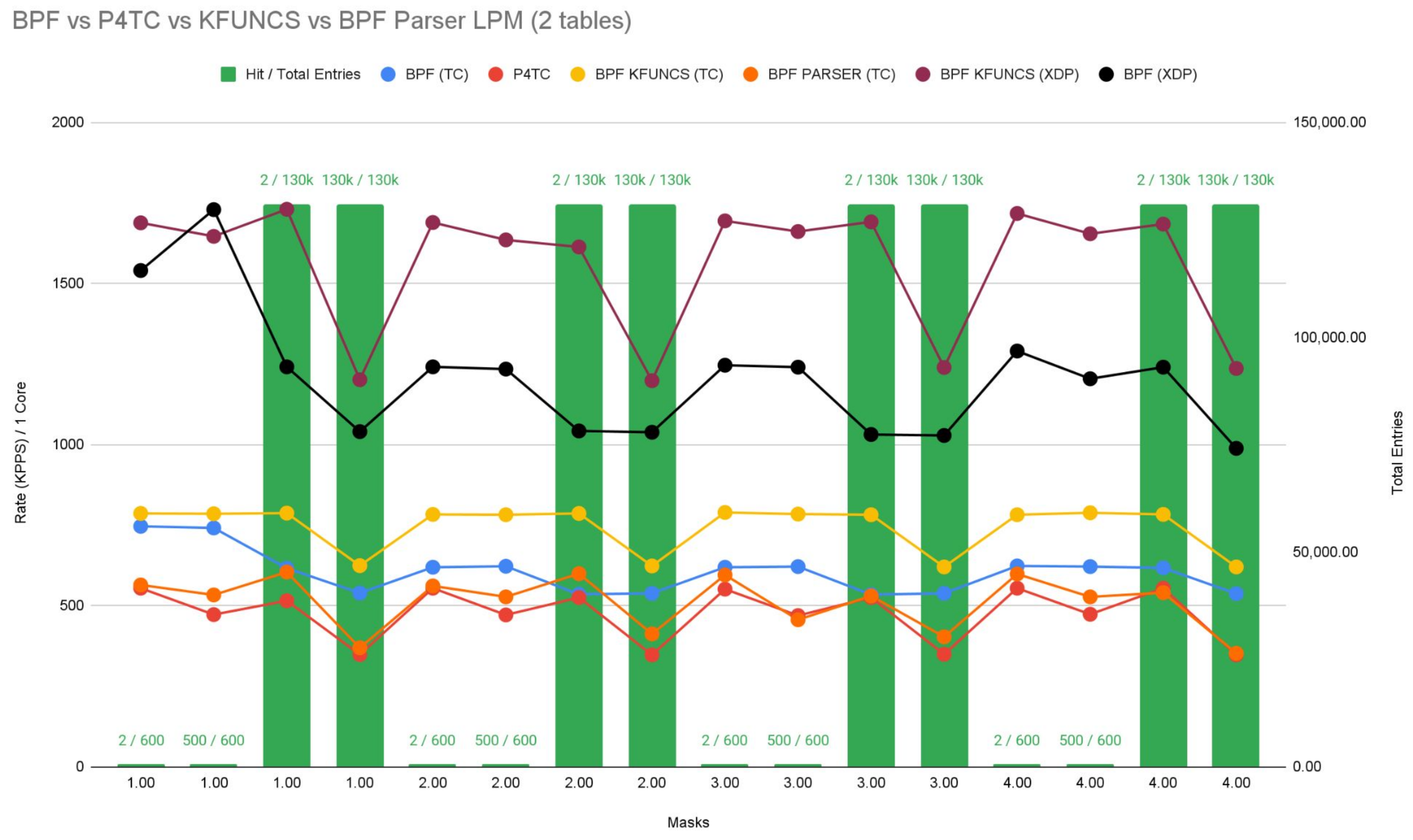
Exact 3 Table: L3 redirect



LPM 1 Table: L3 redirect

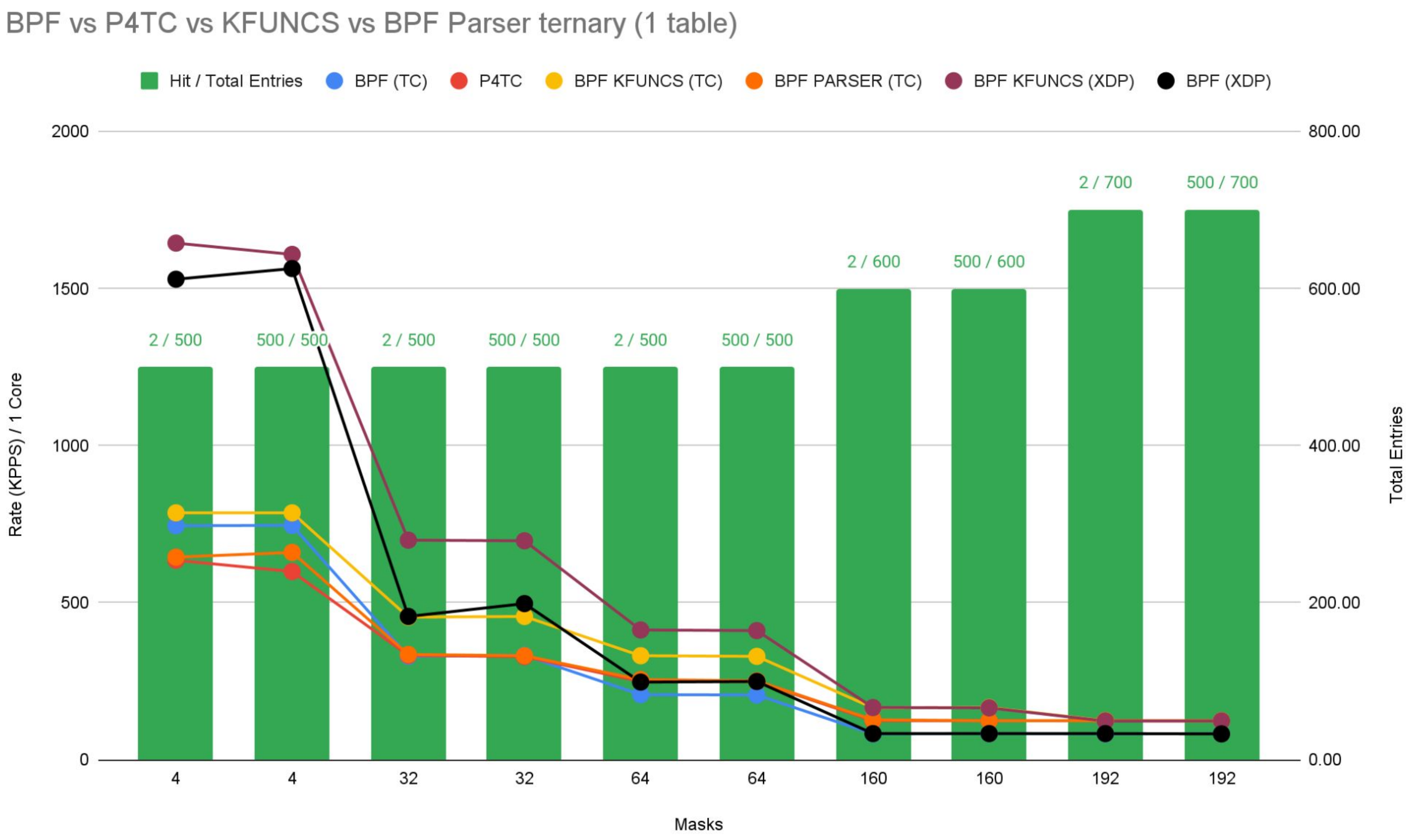


LPM 2 Table: L3 redirect

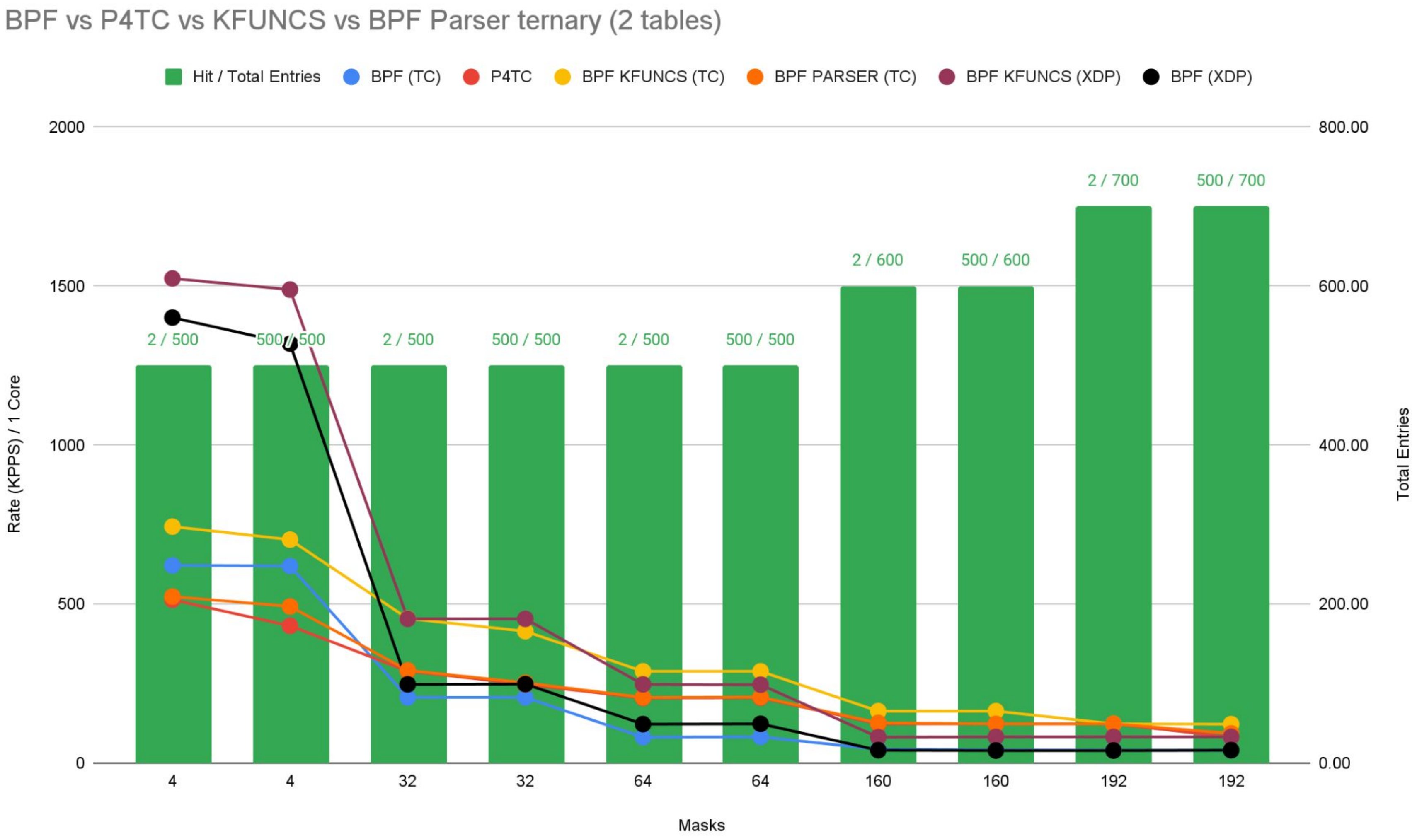


LPM 3 Table: L3 redirect

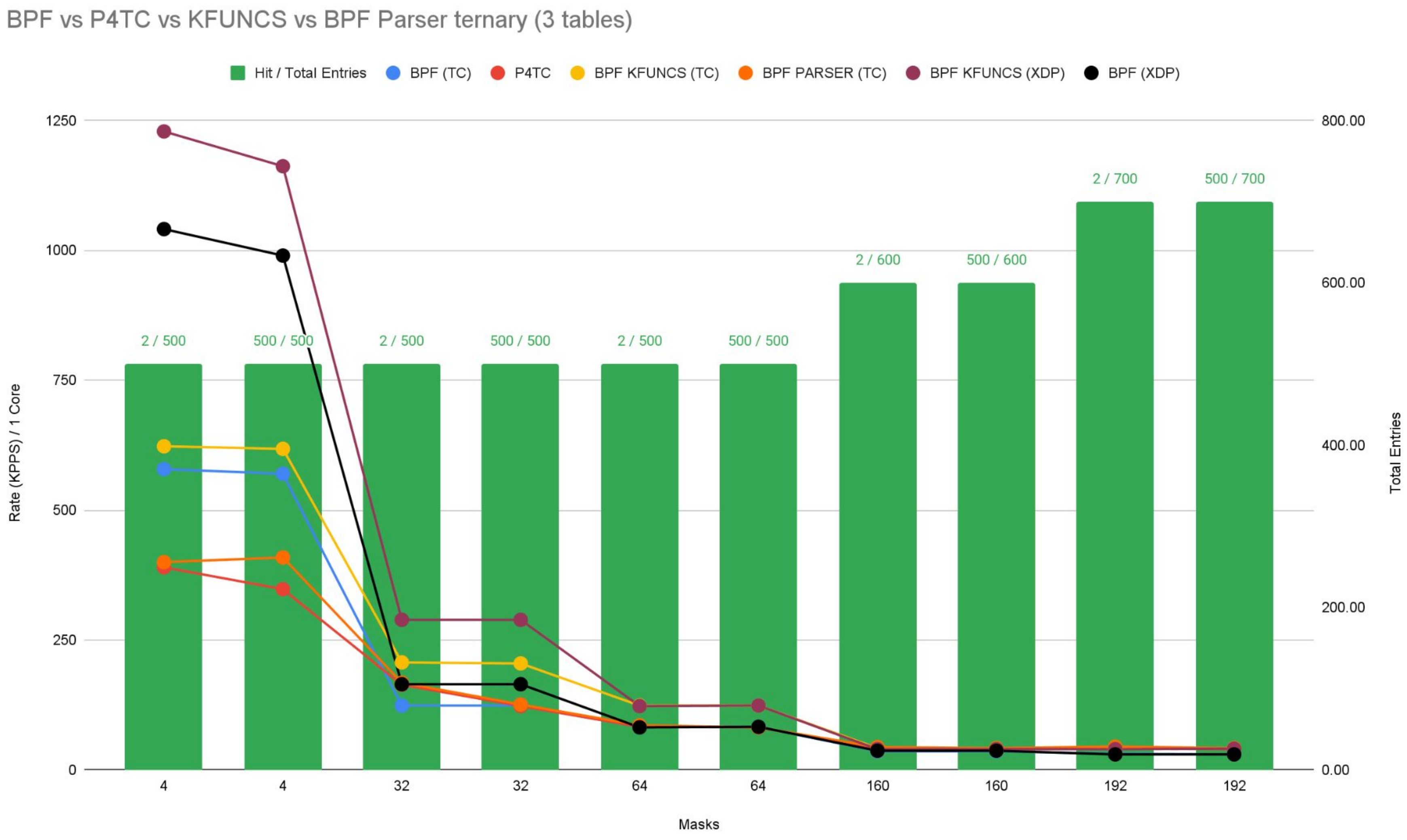
Ternary 1 Table: L3 redirect



Ternary 2 Table: L3 redirect



Ternary 3 Table: L3 redirect



Conclusions

- eBPF <u>does</u> improve performance over P4TC in most cases
  - In the simple case XDP/eBPF provides twice the performance of TC/eBPF
    - In the complex cases, the gap is much smaller and for computationally more intensive workloads like ternary (no observable difference between all 6 setups)
- eBPF with kfunc performance is better than plain eBPF for ternary and LPM lookups but not for exact matches
  - Needs investigation
- XDP performs better than TC for less computationally intensive applications
  - As complexity of P4 program increases the performance of XDP is not very different from tc
    - Infact at some point converges to scriptable P4TC

Next Steps

- Another RFC
  - Another round of feedback from the kernel community
- Dive into the offload side of P4TC
  - Define a common driver interface for multiple vendors (see talk on topic)
    - Subject of discussion of the workshop
  - We have a small demo in the hallway that demonstrates scriptable offloading
- Make a decision on software dataplane
  - Weigh the pros and cons and decide on which implementation to move forward
  - Iterations with the kernel community are expected
  - Please, please provide us feedback

References

1. https://github.com/p4tc-dev/docs/blob/main/why-p4tc.md
2. https://netdevconf.info/0x16/session.html?Your-Network-Datapath-Will-Be-P4-Scripted
3. https://netdevconf.info/0x16/session.html?P4TC-Workshop
4. https://github.com/p4lang/p4c
5. https://man7.org/linux/man-pages/man8/tc-flower.8.html
6. https://dl.acm.org/doi/abs/10.1145/3555050.3569117
7. https://docs.kernel.org/bpf/kfuncs.html
8. https://github.com/p4lang/p4c/tree/main/backends/ebpf/psa
9. https://github.com/NIKSS-vSwitch/niks

Backslides

Vetting Against Requirements*

Feature	P4TC plain	Parser eBPF	eBPF sw P4TC hw	eBPF kfunc P4TC
Operational Usability	10	8	5	8
Debuggability	9	7	6	7
SW Performance	5	7	10	10
Tooling + Interface Stability	9.5	8	8	7
Ease of Innovation	10	9.5	5	6
% support of set of all P4 programs	10	10	8	10
Total	53.5	49.5	42	48

Pros of eBPF vs Scriptable P4TC

- Performance improvement in most cases
  - JIT compiled code
    - Native control flow, arithmetic and logical instructions as opposed to scripted
- Access to XDP
  - Fast packet processing in software
  - AF_XDP + io_uring if more processing is needed in userspace
  - Note: XDP (as illustrated) does not improve much when compute bound dominates

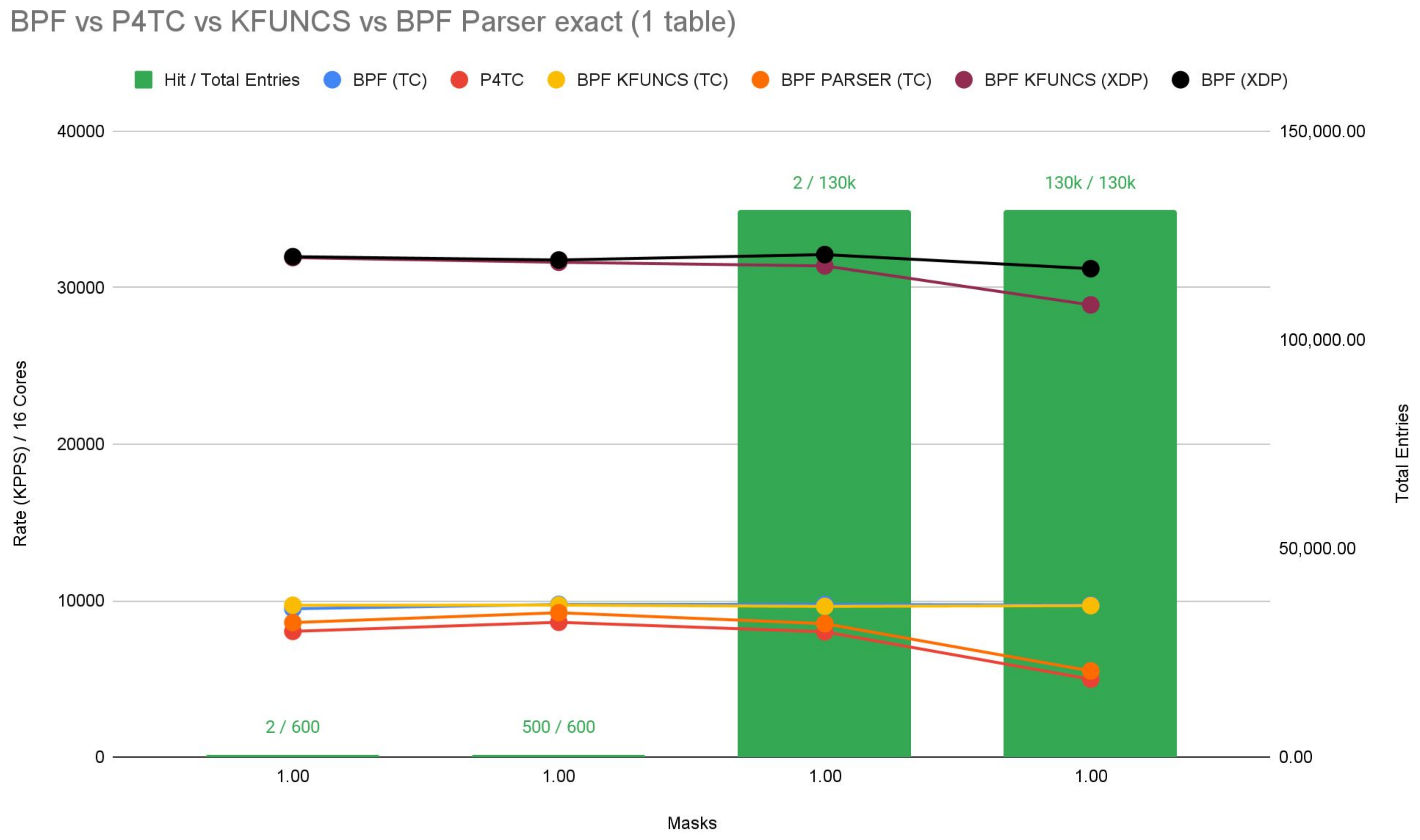
<u>Note:</u>

eBPF at both TC and XDP can be independent (of P4TC) as per traditional way of writing of eBPF today

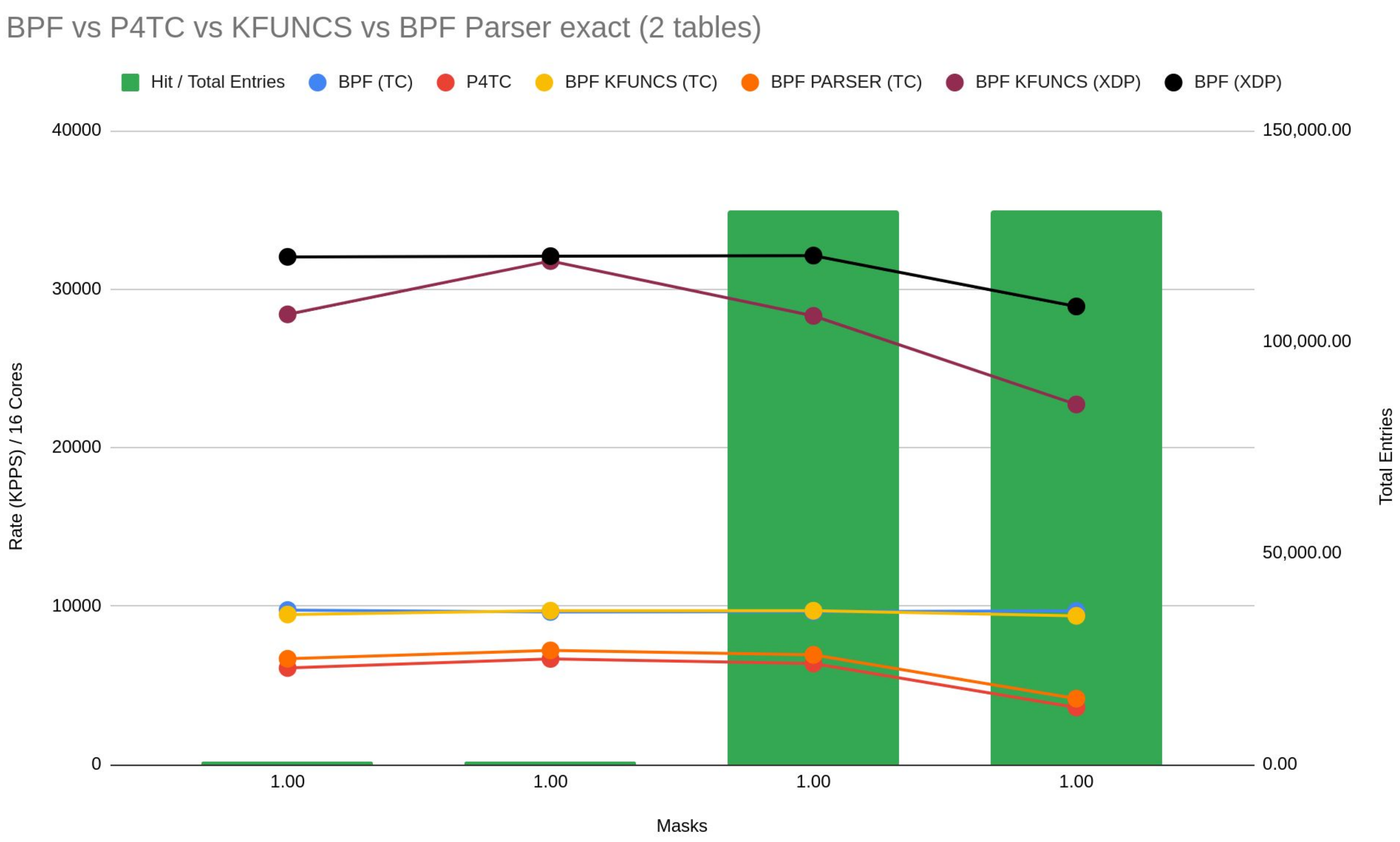
Cons of eBPF vs Scriptable P4TC

- Loss of operational simplicity
  - Two programs to load
  - Two very different subsystems to debug when something goes wrong
    - TC folks vs ebpf folks are very different domains
- Cannot recreate P4 program from what's taught to the kernel
- Has gaps in mapping a set of P4 programs into eBPF
  - Will work most of the time in square-hole-round-peg format (we are ok with such an approach since it doesn't affect performance much)
    - Eg default actions can be emulated with a separate table with a single entry
  - Will not work some of the time
    - eBPF verification and instruction limit can get in the way
  - Kfunc API helps fill in these gaps
    - Note: kfunc API stability is under our direct control (as opposed to helpers which are not not under our control)
- Heavily kernel and toolchain version dependent
  - New eBPF features require a full toolchain upgrade (kernel + compiler)
  - Backwards compatible deployments is sometimes challenge
    - Susceptible to breaks when kernel/toolchain upgrades

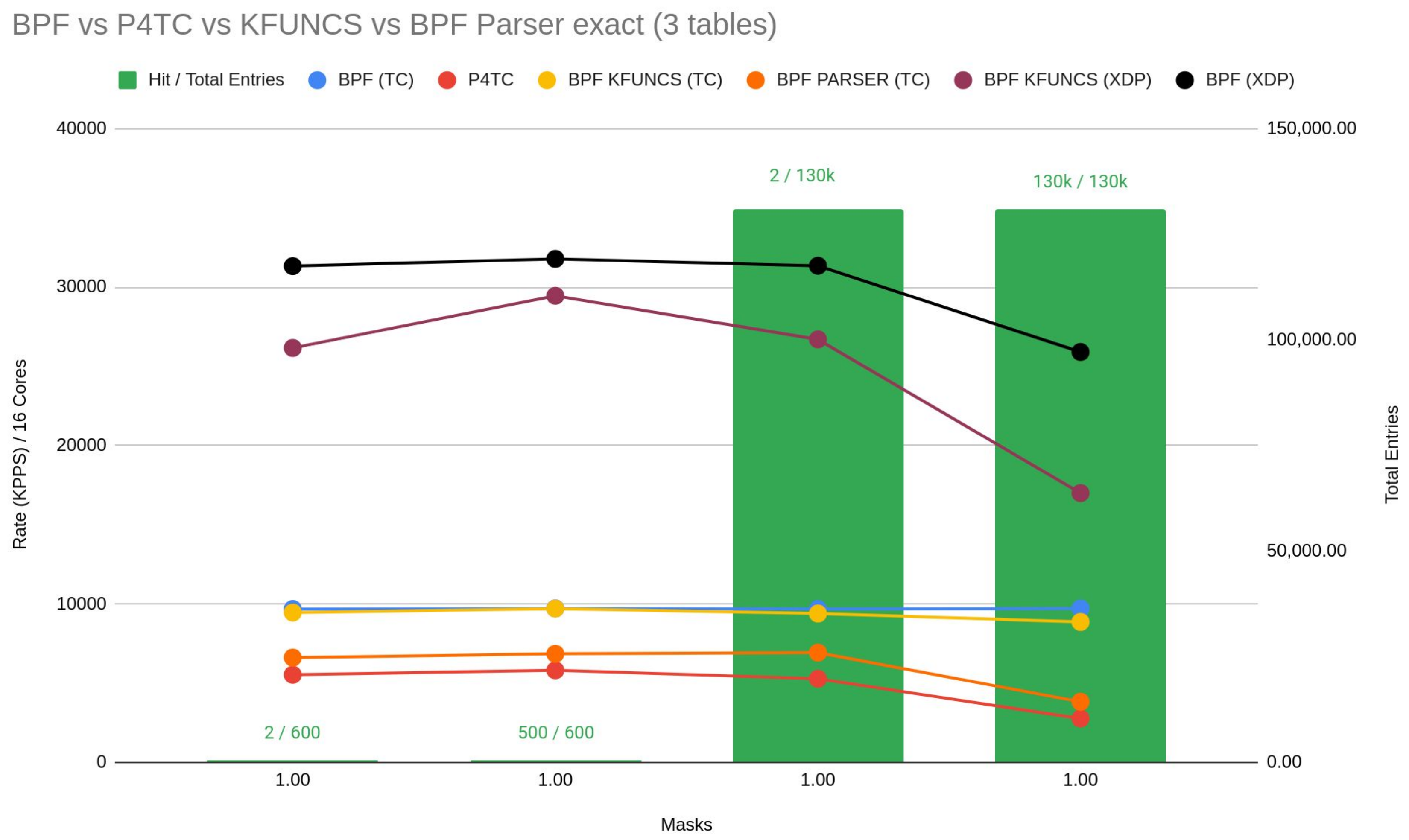
Exact 1 table 16 cores: L3 redirect



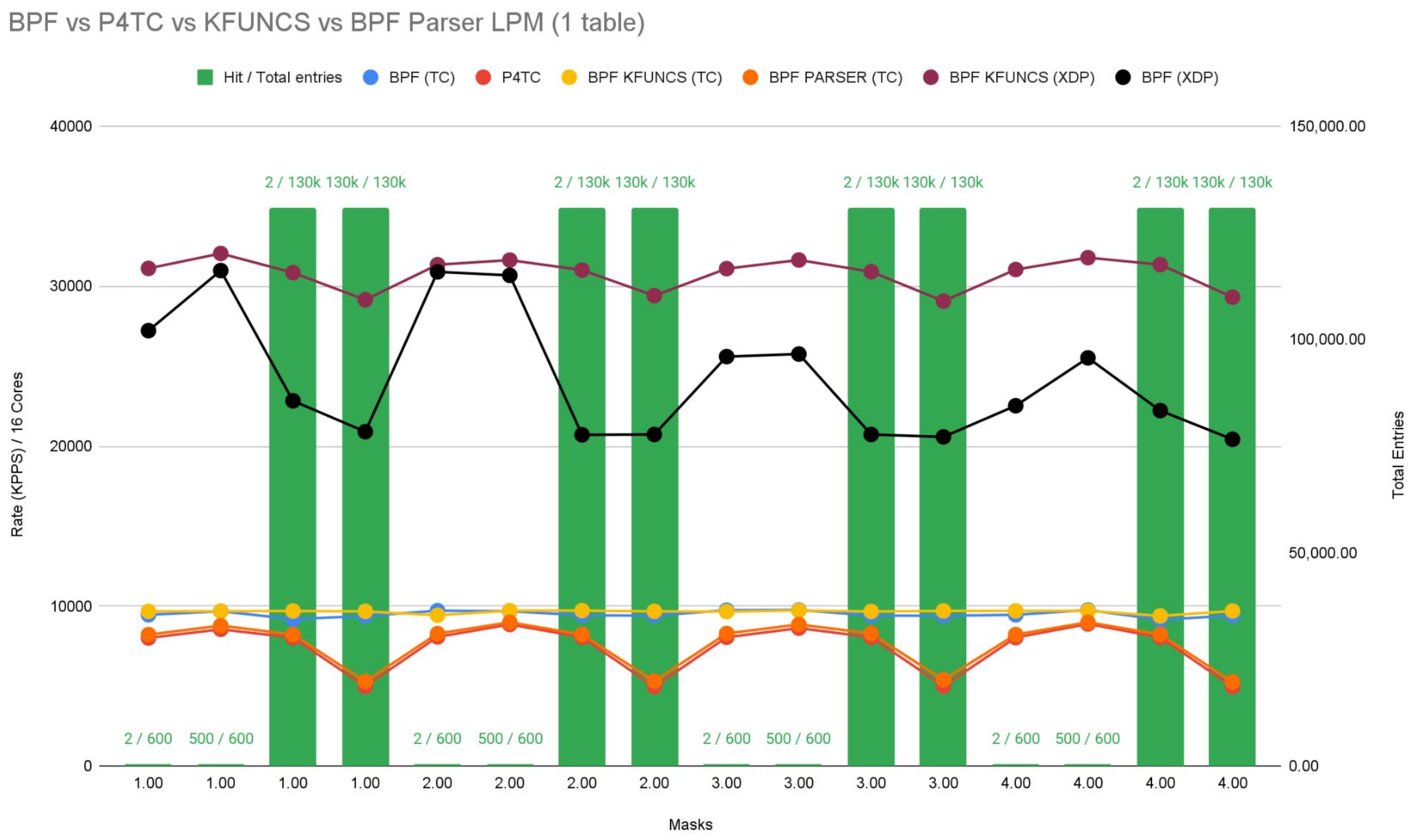
Exact 2 tables 16 cores: L3 redirect



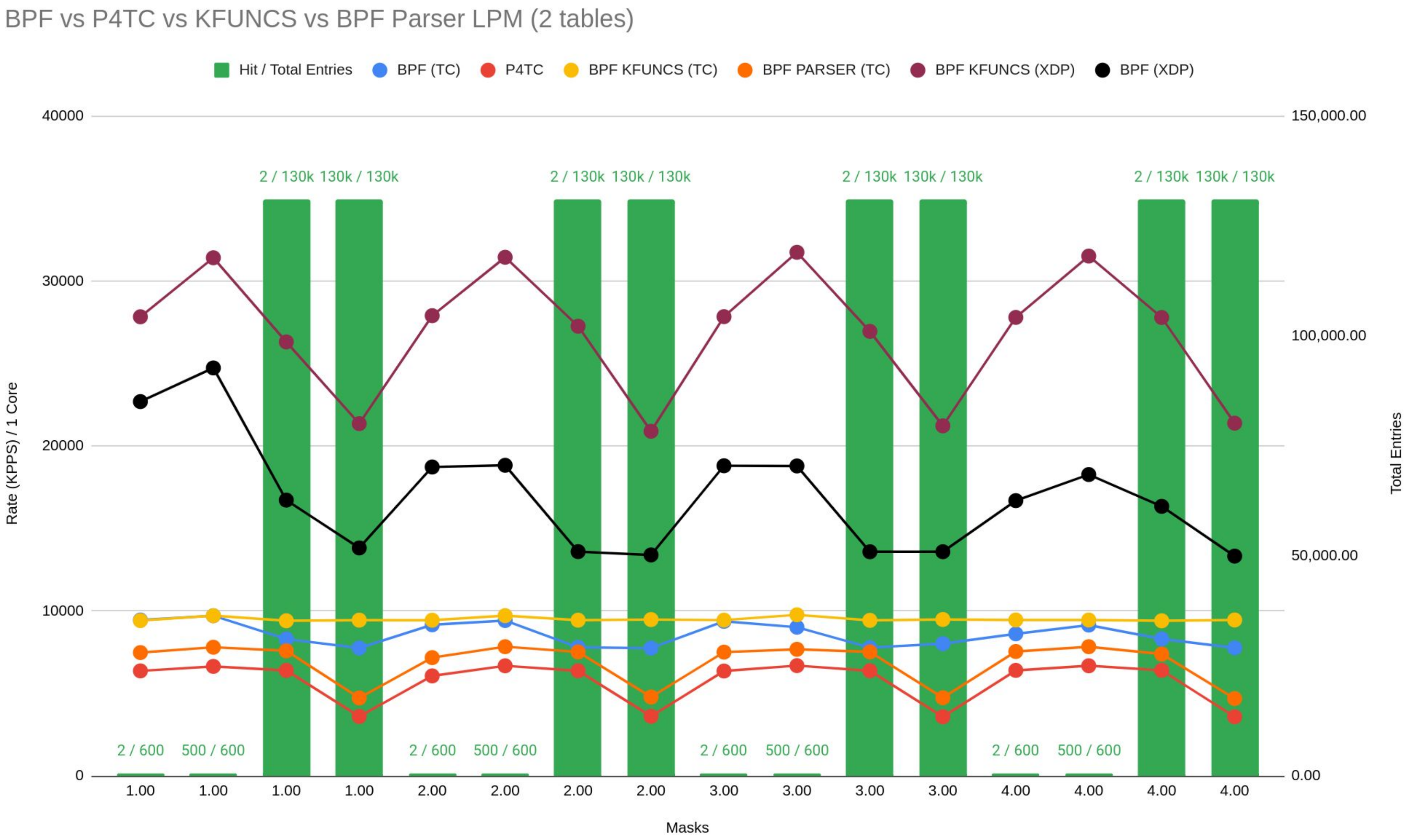
Exact 3 tables 16 cores: L3 redirect



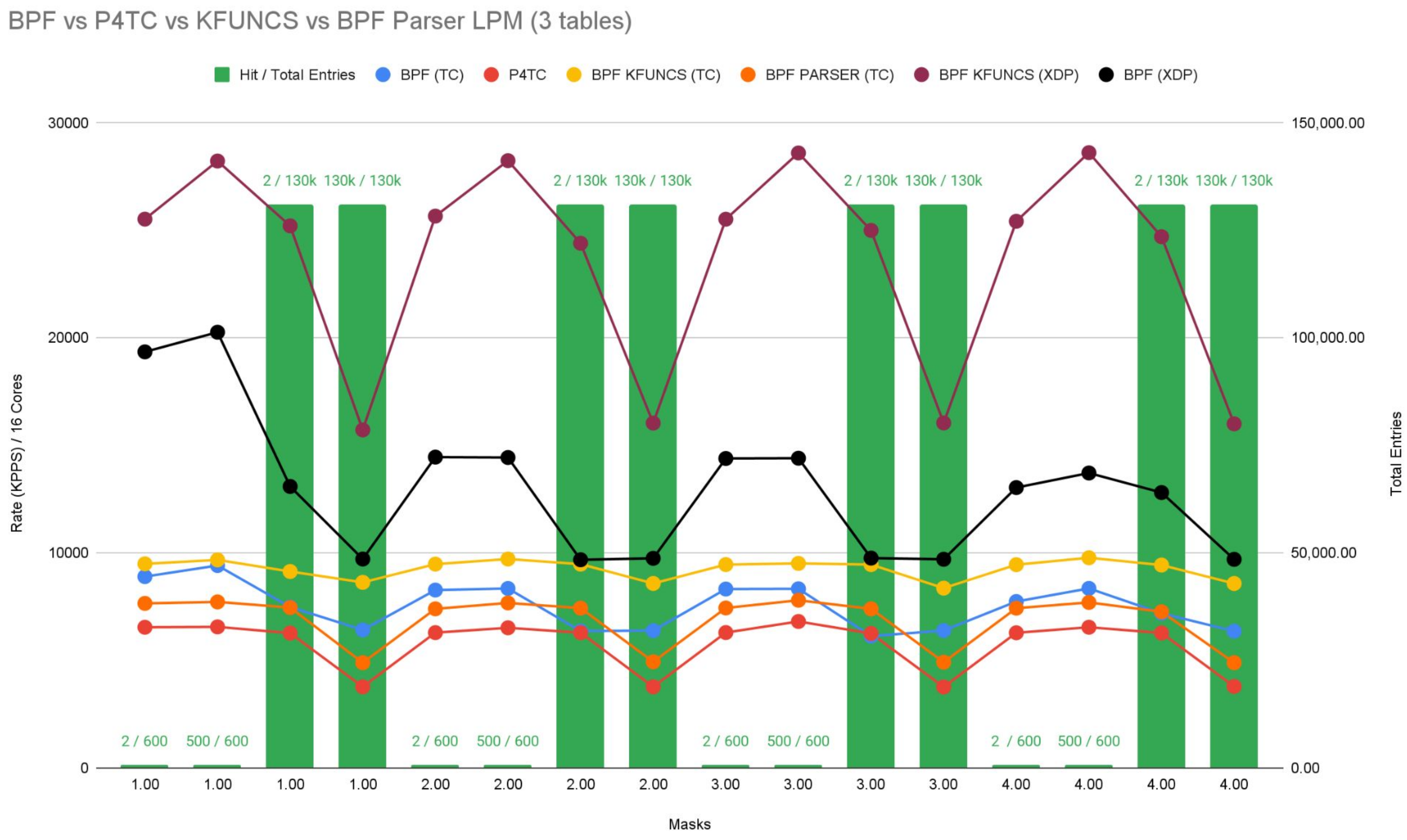
LPM 1 table 16 cores: L3 redirect



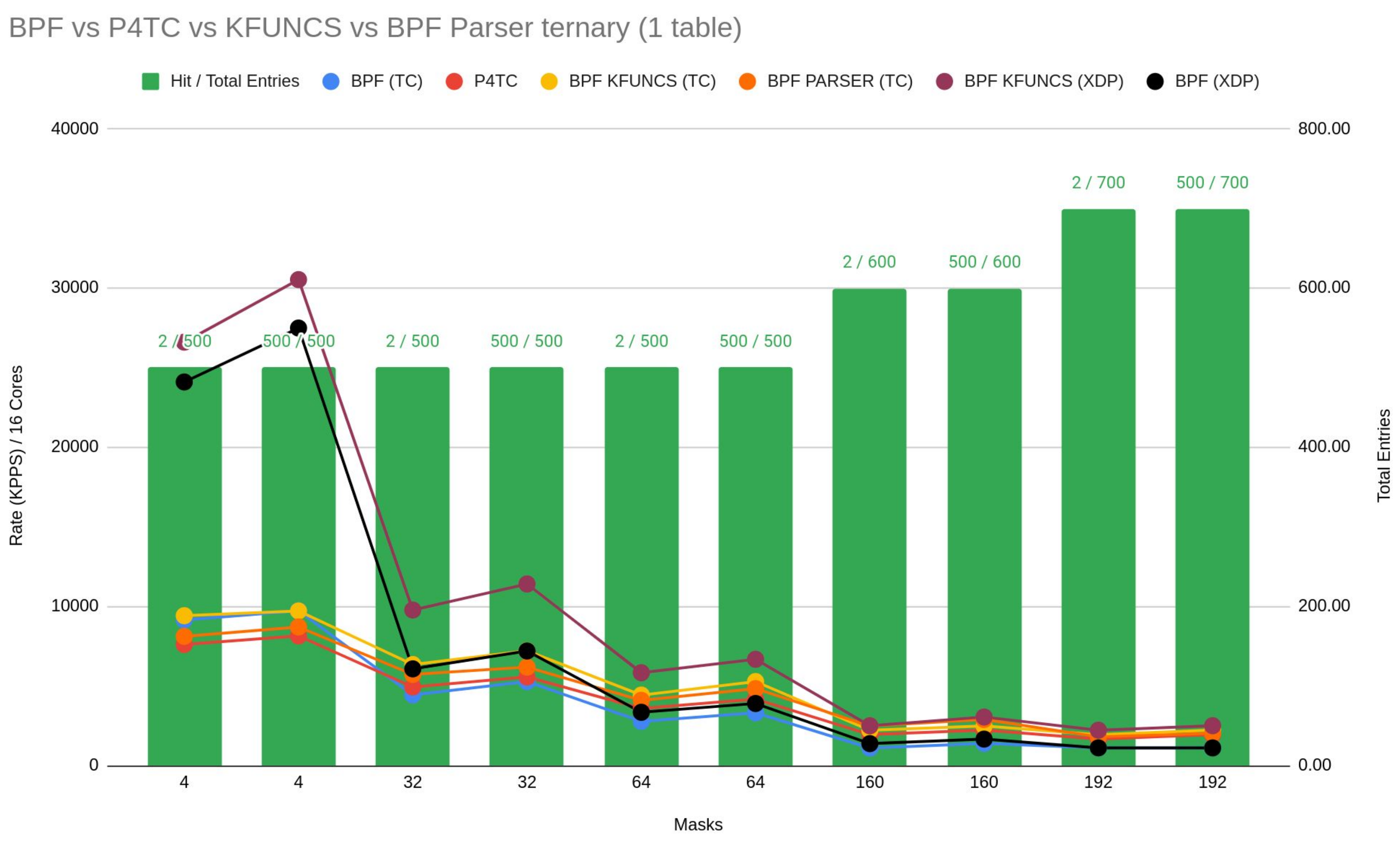
LPM 2 tables 16 cores: L3 redirect



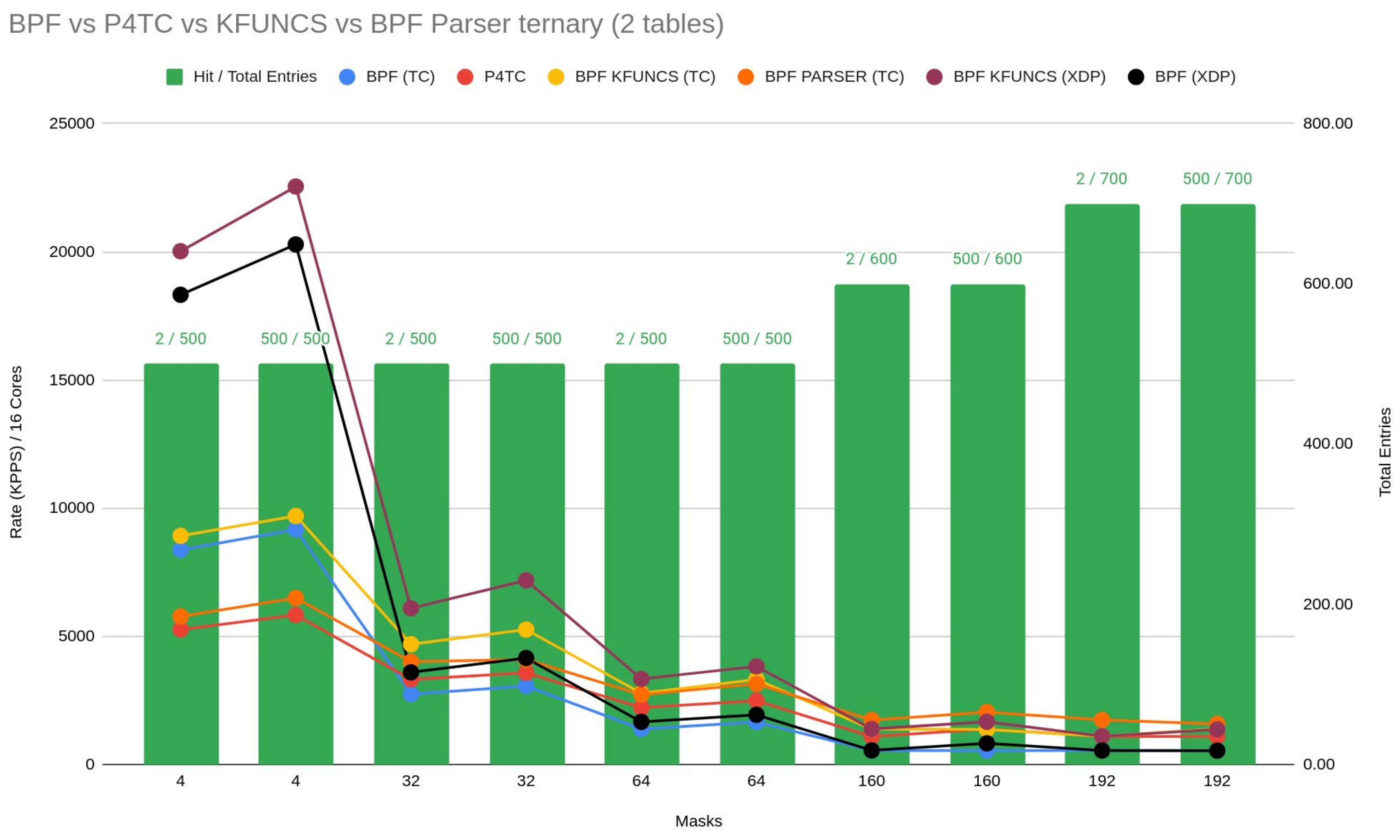
LPM 3 tables 16 cores: L3 redirect



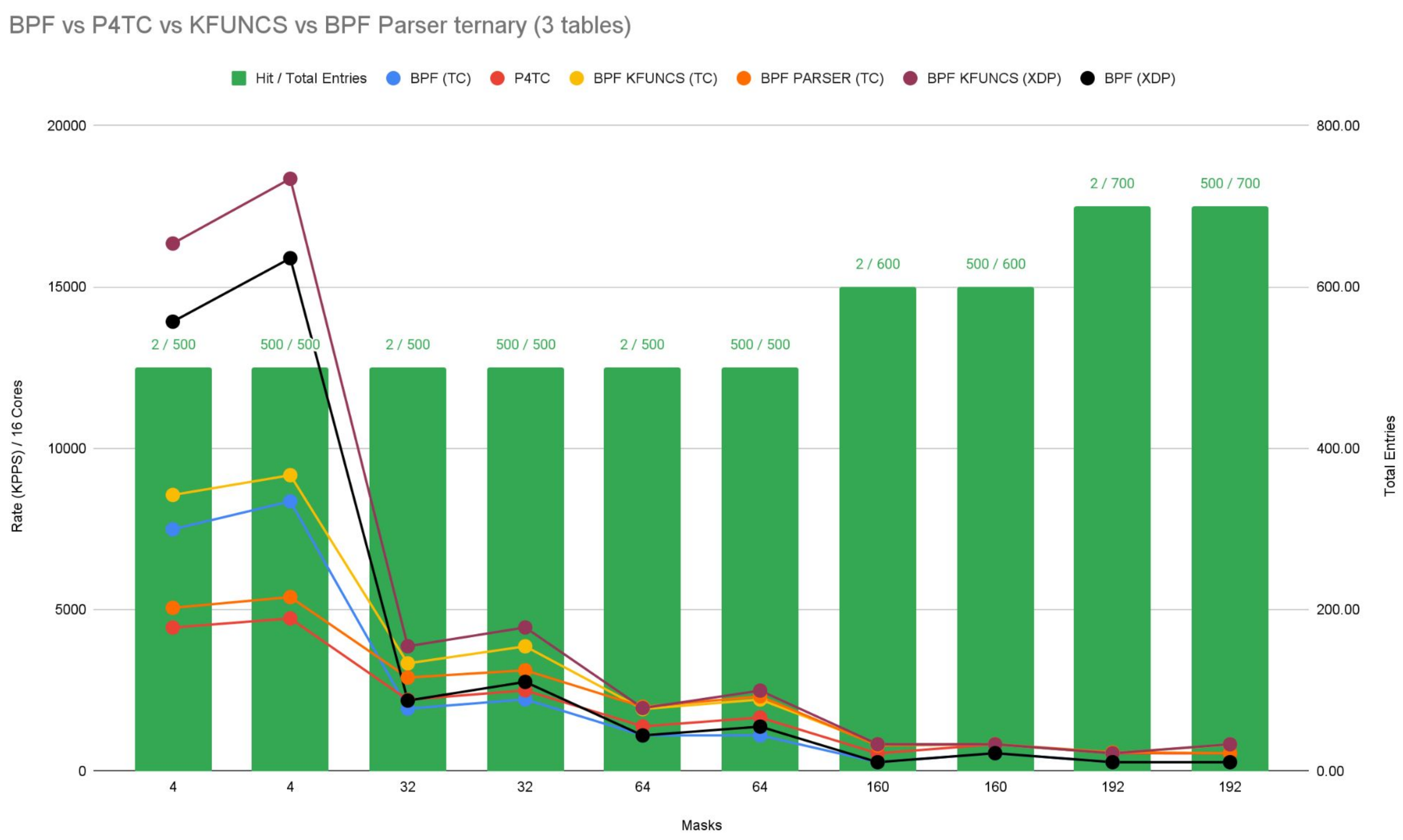
Ternary 1 table 16 cores: L3 redirect



Ternary 2 tables 16 cores: L3 redirect



Ternary 3 tables 16 cores: L3 redirect



APRIL 24-25 P4 WORKSHOP 2023

Thank You!

Jamal Hadi Salim (Mojatatu Networks)
Deb Chatterjee (Intel Corporation)
Victor Nogueira (Mojatatu Networks)
Pedro Tammela (Mojatatu Networks)
Tomasz Osinski (Intel Corporation)
Sosutha Sethuramapandian (Intel Corporation)
Balachandher Sambasivam (Intel Corporation)
Evangelos Haleplidis (Mojatatu Networks)