Honey for the Ice Bear – Dynamic eBPF in P4

Manuel Simon, Henning Stubbe, Sebastian Gallenmüller, Georg Carle

Sunday 4th August, 2024

Chair of Network Architectures and Services
School of Computation, Information and Technology
Technical University of Munich
Motivation

- Interrupt-free, dynamic updates increase network resilience
  - application migration
  - tenant-specific processing

- P4 and eBPF are well-established languages for programmable packet processing
  - P4: restricted, simple language, optimized for high performance
  - eBPF: JIT compiled, more high-level language features

- Both languages bring advantages for specific use-cases
  - eBPF programs as well-defined API for P4 externs to extend functionality
Dynmic eBPF in P4

- Extension of P4 pipeline with updatable eBPF modules
  - Fixed position
  - Extern
- Allows runtime re-programmability
  - Exchange using pre-compiled byte code
  - JIT compiled to machine code
- Extends P4 functionality with well-defined API
Dynamic modes

Static
- Fixed, non-changeable functionality

Pre-defined
- Pre-implemented, fixed set of functionality
- Defined before *initialization*, switchable during *runtime*

Extensible
- New functionality is sent as source or byte code
- JIT compiled and bound during *runtime*
Related Work

Reprogrammable P4:

- Das et al., ActiveRMT [1]: Instruction set in P4 allowing changeable functionality
- Xing et al., FlexCore [6]: Runtime partial reprogrammable switch architecture
- Feng et al., In-situ Programmability Data Plane [3]: Switch architecture and reconfigurable P4 (rP4) for runtime updates
  
  ⇒ single-language P4 approaches

P4/eBPF:

- P4 to eBPF [4]: Translation of P4 program to eBPF [4]
Implementation

- Implementation for software target T4P4S [5]
- eBPF execution using DPDK rte_bpf library
  - batched tx/rx eBPF callback execution for fixed position
  - non-batched execution for flexible externs
- User space eBPF execution
- Optional BLAKE3-based MACs ensuring authenticity of code updates
Evaluation

Overhead of eBPF execution at different positions (Throughput)

Three programs for basic overhead:

- **dummy**: returns 0
- **filter**: filters for one UDP port and IP address
- **change**: changes a header field
Evaluation

Overhead of eBPF execution at different positions (modeled per-packet CPU cycles)

Three programs for basic overhead:
- **dummy**: returns 0
- **filter**: filters for one UDP port and IP address
- **change**: changes a header field

Cost model:

\[
C = \frac{f_{CPU}}{r_{testbase}} - \frac{f_{CPU}}{r_{baseline}}
\]
Evaluation

Median costs of dynamic updates—ten runs (100 Mbit/s)

⇒ Update of fixed-position functionality more expensive
⇒ Dynamic eBPF byte code installation at reasonable costs
⇒ Authentication possible
Conclusion

- eBPF offers fixed API for P4 externs
- eBPF hardware offloading solutions exist
- eBPF execution within P4 allows additional applications
- Functionality can be updated during runtime (200 µs)

Read the paper if you want more information about:
- Security considerations
- Discussion of different processor positions
- Detailed analysis of program change

Acknowledgments

The authors thank Timon Tsiolis for his contributions and the fruitful discussions, and the reviewers for their valuable feedback. This work has partially been funded by the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 870470) and the German Federal Ministry of Education and Research (BMBF) under the project 6G-life (16KISK002) as well as the German Research Foundation (DFG) as part of the HyperNIC (CA595/13-1) project.
Memory management in activermt: Towards runtime-programmable switches.  

Moongen: A scriptable high-speed packet generator.  

Enabling in-situ programmability in network data plane: From architecture to language.  

GitHub p4c/backends/ebpf - eBPF Backend, 2024.  

T4P4S: a target-independent compiler for protocol-independent packet processors.  

Runtime programmable switches.  
Fixed position

- Pre-, Mid-, or Postprocessor
- Processes every packet
- Access to whole packet
- Potentially easier implementation
- E.g., prefilter, preprocessing, hashing/crypto
Dynamic eBPF in P4

Extern
- Flexible position as P4 extern
- Conditional execution
- Return value usable
- Access to whole packet or restricted to selected header fields
Setup

**DuT**
- Intel Xeon D-1518 2.2 GHz, 32 RAM
- Latency optimized \textit{T4P4S} → batch size of one

**LoadGen**
- MoonGen [2] is used to generate traffic
- Packet size 84 B

**Timestampmer**
- Packet streams duplicated using optical splitter
- Timestamps each packet incoming packet
- Resolution: 12.5 ns
eBPF byte code swapping during runtime possible without packet loss