Domain Specific Run Time Optimization for Software Data Planes

Extended Abstract

Sebastiano Miano[1], Alireza Sanaee[1], Fulvio Risso[2], Gábor Rétvári[3], Gianni Antichi[1],

[1]Queen Mary University of London, UK

[2]Politecnico di Torino, IT

[3]MTA-BME Information Systems Research Group & Ericsson Research, HU

1. Motivation

Software Data Planes, packet processing programs implemented on commodity servers, are widely adopted in real deployments [19, 12, 10, 21, 22]. Network software is usually compiled using *static optimization*, yielding a binary that is agnostic to its run time behavior. Thus, the code may contain logic for protocols and features that may never be triggered in deployment, perform costly memory loads to access values that are only known at run time, or take difficult-to-predict branches conditioned on variable data.

Dynamic compilation enables program optimization based on invariant data computed at run time, producing code that is *specialized* to the input the program is processing [6, 2]. Generic dynamic compilation tools [4, 17], however, proved ineffective for *packet processing logic* whose performance critically depends on *highly variable domain-specific* traits, such as traffic patterns or match-action table content [16, 1, 7]. This calls for *domain-specific dynamic optimizations*, specifically tailored to the networking context. Our benchmarks, obtained with two common network applications, clearly demonstrate (Figure 1) this: *(i)* state-of-the-art generic tools bring minimal benefits on packet-processing performance; *(ii)* specializing networking code for slowly changing input, like flow-rules, ACLs and control plane policies (*Table Specialization* and *Run time Configuration* bar), substantially improves the performance; and *(iii)* for maximum performance, networking code must be specialized with respect to inbound traffic patterns[1] (*Fast Path* bar). The main challenge we tackle in our paper is to attain similar, or even higher, performance improvement by the automatic *dynamic compilation of network code*.

2. Limitations of the State of the Art

Online tracing. Dynamic compilation depends on timely information from the running data plane. Obtaining this, however, is difficult: lightweight *online* tracing tools (e.g., Linux `perf` [5]) do not provide enough insight to apply meaningful domain-specific optimizations, whereas tracing packet-level and instruction-level logs is prohibitively costly. As an example, GCC FDO instrumentation, when applied in this context, may easily incur ~900% mean overhead [13]. Therefore, existing dynamic optimization tools, designed either for generic software (AutoFDO [4, 11], Bolt [17]) or specifically for the networking domain [9, 7, 20], mainly perform offline optimizations using recorded execution traces. This requires operators to collect representative samples of match-action tables and predict traffic patterns from production deployments. We argue, instead, a dynamic compiler for networking code should work in a fully *unsupervised* mode where all tracing data needed for code specialization is collected *online*.

[1]Refer to §2 of the paper for more information.

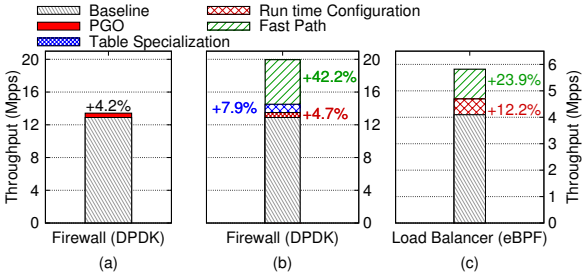

Figure 1: (a) Impact of AutoFDO+Bolt (PGO) and performance breakdown when applying a set of domain specific optimizations to both (b) the DPDK *firewall* [8] and (c) the Facebook's Katran eBPF *load balancer* [10]. We were unable to run PGO on the latter, since existing tools do not support eBPF targets.

Domain-specific optimization. ESwitch [16, 18] is the first functional framework for the unsupervised *dynamic* optimization of software data planes, but it targets only legacy (OpenFlow) code. PacketMill [9] and NFReducer [7] are more generic, leveraging the LLVM toolchain [14] instead of OpenFlow: PacketMill targets the FastClick datapath and NFReducer optimizes generic network code using symbolic execution. Our work, Morpheus, is strictly complementary to these works: *(i)* it applies similar optimizations but it also introduces a toolbox of new ones (e.g., branch injection or constant propagation for stable table entries to name a few); *(ii)* it detects packet-level dynamics and applies more aggressive optimizations depending on the specific traffic patterns; and *(iii)* it is data plane agnostic as it performs the optimizations at the IR-level using a portable compiler core.

3. Key Insights

Our main insight is that to squeeze out the maximum performance for a software network function, a compiler should be

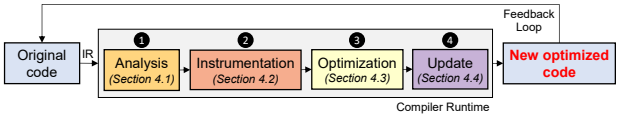

Figure 2: The Morpheus compiler pipeline.

extended with *domain-specific* insights that are meaningful only in the packet processing context. In particular, the system should be able to specialize the networking code for slowly changing input, like flow-rules, ACLs and control plane policies, alongside fast changes in input traffic patterns, which may open up the space for more aggressive optimizations.

To the best of our knowledge, Morpheus is the first dynamic data plane compiler able to optimize a software network function *on-the-fly*, without any traffic disruption, based on the network configuration and packet-level dynamics. In addition, Morpheus works without any *a priori* knowledge of the running program, which allows to decouple the system from the specific framework used by the underlying data plane.

4. Main Artifacts

We design, implement and evaluate a system, called Morpheus, capable of optimizing network code at run time using *domain-specific dynamic optimization techniques*. The different steps that involve the Morpheus compilation pipeline are shown in Figure 2. The pipeline is triggered periodically at given time slots to re-optimize the code for possibly changed traffic patterns and control plane updates.

The compiler first analyzes the code to understand the program control/data flow (§4.1 of the paper). Specifically, it performs a *signature-based call site analysis* to track operations on tables at the Intermediate Representation (IR) level, and then a combination of *memory dependency analysis* and *alias analysis* to distinguish stateful from stateless operations. The output of this stage is an IR code marked with debug information that is stored into the compiler's internal data structures. This is fed into the second stage that, starting from the received debug data, *instruments* the code to collect run time data that will drive the subsequent optimization passes (see §4.2 of the paper). To reduce the run time cost of profiling, Morpheus uses several dimensions of adaptation to record only the minimum set of information required for a complete understanding of the program behavior. Finally, Morpheus applies a number of optimizations to the code such as *(i)* improving constant propagation & constant folding; *(ii)* specializing data structures depending on their actual content; and *(iii)* creating a fast path for the most accessed entries in the code (see §4.3 of the paper). To guarantee the consistency of the data plane under any modification of the invariants the specialized code relies on, Morpheus injects *guards* at critical points to allow the execution to fall back to the unoptimized path whenever an invariant changes. At the last step, Morpheus replaces the running data plane with the new, optimized code on the fly (see §4.4 of the paper).

Implementation. Morpheus is implemented in about 6000 lines of C++ code and relies on the LLVM compiler toolchain for code manipulation and run time code generation, which makes it portable across different data plane frameworks and programming languages. Specifically, Morpheus is composed of *(i)* a portable *core*, containing the compiler passes, and *(ii)* a technology-specific *plugins* to interact with the underlying data plane framework (i.e., eBPF and DPDK). The plugins are abstracted via a *backend API*, which exports a set of functions for the code to identify match-action table access sites based on data plane specific call signatures, or compute cost functions for data structure specialization.

Evaluation. We applied Morpheus to a number of eBPF/XDP-based packet processing programs from the open-source eBPF network function virtualization framework Polycube [15] (i.e., *L2 Switch*, *Router*, *NAT*, *iptables*) and Facebook's Katran load-balancer [10]. We also tested Morpheus against the DPDK FastClick Router [3]. We evaluate our solution using both synthetic and real-world traffic traces, and compared it with against two optimization tools: eSwitch [16]) and PacketMill [9]. Moreover, we discuss the impact of the *adaptive instrumentation* mechanism, the time required to re-compile and inject the new data plane program, as well as the benefits of Morpheus under dynamic traffic pattern changes or with multi-cores programs.

5. Key Results and Contributions

Morpheus can increase up to 2x the packet-processing throughput of the targeted code, while halving its latency at the 99th percentile. At the micro-architectural scale, we show also that Morpheus can reduce the last-level CPU cache misses by up to 96% while effectively halving the instructions and branches executed per packet. When compared to state-of-the-art optimizers, our solution consistently delivers 5 to 10x improvement over eSwitch when input traffic patterns experience high locality and Morpheus can apply its most aggressive optimizations, while it essentially falls back to eSwitch for uniform traffic. When tested with DPDK, Morpheus produces a whopping 469% improvement over PacketMill in its best case scenario, while losing by about 9% in its worst.

Contributions. To summarize, in this paper we:

- demonstrate that tracking packet-level dynamics opens up new opportunities for network code specialization;
- design and implement Morpheus, a system working with standard compilers to optimize network code at run time;
- extensively evaluated Morpheus by applying it to two different I/O technologies (i.e., DPDK and eBPF), and a number of programs including production-grade software;
- plan to share the code in open source together with the scripts and traces used for out evaluation to foster reproducibility.

References

[1] Omid Alipourfard and Minlan Yu. Decoupling Algorithms and Optimizations in Network Functions. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets '18, page 71–77, New York, NY, USA, 2018. Association for Computing Machinery.

[2] Joel Auslander, Matthai Philipose, Craig Chambers, Susan J. Eggers, and Brian N. Bershad. Fast, Effective Dynamic Compilation. In *Proceedings of the ACM SIGPLAN 1996 Conference on Programming Language Design and Implementation*, PLDI '96, page 149–159, New York, NY, USA, 1996. Association for Computing Machinery.

[3] Tom Barbette, Cyril Soldani, and Laurent Mathy. Fast Userspace Packet Processing. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '15, page 5–16, USA, 2015. IEEE Computer Society.

[4] Dehao Chen, David Xinliang Li, and Tipp Moseley. AutoFDO: Automatic Feedback-Directed Optimization for Warehouse-Scale Applications. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, CGO '16, page 12–23, New York, NY, USA, 2016. Association for Computing Machinery.

[5] Wikipedia contributors. Perf (linux). https://en.wikipedia.org/wiki/Perf_(Linux), 2018. [Online; accessed 07-August-2021].

[6] Timothy Cramer, Richard Friedman, Terrence Miller, David Seberger, Robert Wilson, and Mario Wolczko. Compiling java just in time. *IEEE Micro*, 17(3):36–43, may 1997.

[7] Bangwen Deng, Wenfei Wu, and Linhai Song. Redundant Logic Elimination in Network Functions. In *Proceedings of the Symposium on SDN Research*, SOSR '20, page 34–40, New York, NY, USA, 2020. Association for Computing Machinery.

[8] DPDK. L3 forwarding with access control sample application. https://doc.dpdk.org/guides/sample_app_ug/l3_forward_access_ctrl.html, 2021. [Online; accessed 07-August-2021].

[9] Alireza Farshin, Tom Barbette, Amir Roozbeh, Gerald Q. Maguire Jr., and Dejan Kostić. PacketMill: Toward per-Core 100-Gbps Networking. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS 2021, page 1–17, New York, NY, USA, 2021. Association for Computing Machinery.

[10] Christian Hopps. Katran: A high performance layer 4 load balancer. September 2019. https://github.com/facebookincubator/katran.

[11] Google Inc. Propeller: Profile Guided Optimizing Large Scale LLVM-based Relinker. https://github.com/google/llvm-propeller, Oct 2019. [Online; accessed 07-August-2021].

[12] James Kempf, Bengt Johansson, Sten Pettersson, Harald Luning, and Tord Nilsson. Moving the mobile evolved packet core to the cloud. In *Proceedings of the 2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, WIMOB '12, page 784–791, USA, 2012. IEEE Computer Society.

[13] Tanvir Ahmed Khan, Ian Neal, Gilles Pokam, Barzan Mozafari, and Baris Kasikci. DMon: Efficient Detection and Correction of Data Locality Problems Using Selective Profiling. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 163–181. USENIX Association, July 2021.

[14] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*, CGO '04, page 75, USA, 2004. IEEE Computer Society.

[15] S. Miano, F. Risso, M. V. Bernal, M. Bertrone, and Y. Lu. A Framework for eBPF-based Network Functions in an Era of Microservices. *IEEE Transactions on Network and Service Management*, pages 1–1, 2021.

[16] László Molnár, Gergely Pongrácz, Gábor Enyedi, Zoltán Lajos Kis, Levente Csikor, Ferenc Juhász, Attila Kőrösi, and Gábor Rétvári. Dataplane Specialization for High-Performance OpenFlow Software Switching. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 539–552, New York, NY, USA, 2016. Association for Computing Machinery.

[17] Maksim Panchenko, Rafael Auler, Bill Nell, and Guilherme Ottoni. BOLT: A Practical Binary Optimizer for Data Centers and Beyond. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, CGO 2019, page 2–14. IEEE Press, 2019.

[18] Gábor Rétvári, László Molnár, Gábor Enyedi, and Gergely Pongrácz. Dynamic Compilation and Optimization of Packet Processing Programs. *ACM SIGCOMM NetPL*, 2017.

[19] Sourcefire. Snort - Network Intrusion Detection & Prevention System. https://www.snort.org/, nov 2020. [Online; accessed 07-August-2021].

[20] Patrick Wintermeyer, Maria Apostolaki, Alexander Dietmüller, and Laurent Vanbever. P2GO: P4 Profile-Guided Optimizations. In *Hot Topics in Networks (HotNets)*. ACM, 2020.

[21] David Wragg. Unimog - Cloudflare's edge load balancer. sep 2020.

[22] Mathieu Xhonneux, Fabien Duchene, and Olivier Bonaventure. Leveraging EBPF for Programmable Network Functions with IPv6 Segment Routing. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, page 67–72, New York, NY, USA, 2018. Association for Computing Machinery.