Building Fast but Flexible Software Routers

Sebastian Gallenmüller, Paul Emmerich, Rainer Schönberger, Daniel Raumer, and Georg Carle
Technical University of Munich
Department of Informatics
Chair of Network Architectures and Services
{gallenmu|emmericp|schoenbr|raumer|carle}@in.tum.de

1. INTRODUCTION

Creating quick and dirty prototypes is a simple and effective way to demonstrate the feasibility of new ideas in network research. Though, small scale proof-of-concepts may lack the performance needed to apply them to real world test cases. Thanks to powerful packet processing frameworks such as netmap and DPDK, high-performance packet forwarding systems can be implemented in software today.

We present MoonRoute, a framework dedicated to developing powerful software routers. It is built on top of DPDK and utilizes a highly parallelized architecture to achieve high performance (see Section 2). MoonRoute offers methods to reuse existing libraries and a scripting interface for easy extensibility (see Section 3). An example implementation based on the MoonRoute framework is carefully evaluated to demonstrate the performance and compare it to other relevant software routers (see Section 4).

The entire MoonRoute framework including a reference implementation of a software router is available as free software under MIT license [2]. A technical report featuring details about our architecture and more profiling results is available [1].

2. HIGH-PERFORMANCE DESIGN

MoonRoute leverages hardware features of modern NICs such as receive side scaling to distribute packets according to specified parameters across a number of CPU cores. This functionality for almost perfect scaling across CPU cores forms the basis to implement scalable multi-core packet processing applications. To profit from such a hardware design, the software must be programmed accordingly. The threads must be able to run as independently as possible to not interfere with each other.

The basic architecture of MoonRoute is made up of two different kinds of threads or packet processing components – the *fast path* and the *slow path*. The fast path is concerned with processing many packets requiring rather simple processing, i.e., reaching a routing decision and forwarding packets – fast paths are simple & fast. Other more complex operations, such as generating an ICMP response for timed out IP packets, are handled by the slow path. These operations require a more complex control flow but occur less often than the simple packets – slow paths are versatile & slow. Building a router exclusively from slow path components would be possible, however, lots of functions would only be rarely used. To achieve high performance, our reference implementation uses both types of components: There are several fast path components distributed to different CPU cores and a single slow path component running on a separate core. All components utilize lock-free queues, avoid shared data structures where possible, and employ read-only data structures where information sharing between different threads is necessary. This optimized multi-thread design transfers the multi-core scalability provided by hardware into the software. A diagram of our architecture can be found on the accompanying poster or in [1].

An important method to increase the performance of software packet processing systems is batching. In MoonRoute functions called by fast path components accept, process, and return packets in batches to maximize throughput. Modules can exclude packets from further processing either by building new batches (*rebatching*) or keeping the batches but flagging the packets for subsequent functions to ignore them (*flagging*). Both methods have disadvantages, rebatching introduces additional overhead for building new batches and flagging leads to inefficiencies for large batches containing only a few packets to process. Our router implementation uses both methods. Flagging is used in between function calls in the fast path. Packets are rarely excluded from routing, therefore, flagging avoids the overhead for rebatching. Flagged packets need to be handled by the slow path. As the slow path usually handles only a few packets, sending whole batches with few flags introduces unnecessary load on the slow path. Therefore, packets are flagged and remain in the batches for the fast path, but are at the same time inserted into a queue to the slow path. We call this hybrid approach between flagging and rebatching *drop-out batching.*

3. FLEXIBLE ARCHITECTURE

One of the main goals of MoonRoute is to enable a high flexibility and easy extensibility of software routers. MoonRoute uses libmoon [4] which is a Lua wrapper built on top of DPDK and allows using the easy-to-learn scripting language Lua to implement software routers. With just-in-time compilation (provided by LuaJIT [5]) it is possible to process minimum sized packets at line rate (10 Gbit/s) [4].

ANCS '17 May 18–19, 2017, Beijing, China
© 2017 ACM. ISBN 123-4567-24-567/08/06. . . $15.00
DOI: 10.475/123_4

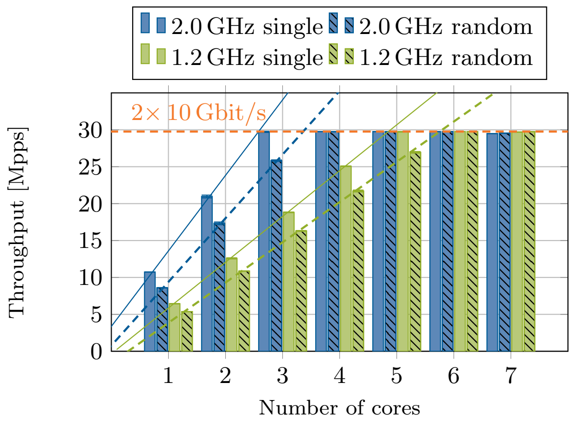

Figure 1: Scaling with the number of CPU cores

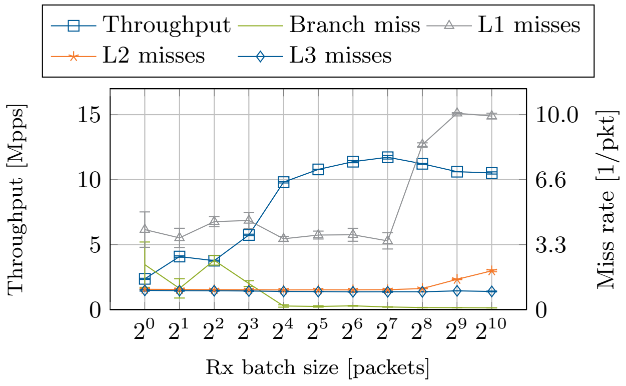

Figure 2: Hardware events (3.2 GHz)

The functionality used by the router is usually provided by modules, these modules represent a specific step in the packet processing task, such as the routing table lookup. These modules take an array of packet buffers and vector of flags as input. An optional parameter for these modules is the queue to a slow path component, where packet buffers can be inserted by the module if necessary. Such a module returns the array of packet buffers and an output vector of flags, signaling the packets to be processed in the next step.

The supported languages to write modules are C/C++ and Lua. The usage of the Lua language is intended for the initial implementation of quick, low-effort prototypes. It is possible to switch the prototype implementation for high-performance C/C++ libraries at a later point in time or to reuse already existing libraries conveniently through an integrated foreign function interface (FFI) provided by LuaJIT [5]. The included reference router has its main loop, which connects all modules in a component, written in Lua, i.e., Lua code extensions are simple to add. Most of the performance-critical functionality is handled by C libraries included in DPDK. The libraries are wrapped by a light-weight Lua wrapper calling the C library via the FFI.

4. EVALUATION

For evaluation, we use the default router integrated into MoonRoute. The test servers are equipped with an Intel X520-T2 NIC offering a bandwidth of 2× 10 Gbit/s. Upcoming many-core CPU architectures make parallelization and scalability the key aspects of high-performance designs. Figure 1 shows the scaling with the number of CPU cores (Intel 2.0 GHz Xeon E5-2640 v2 8-core CPU), indicating a linear trend with multiple cores. The importance of scalability is likely to grow for future many-core systems. Further, we run extensive profiling tests with hardware events to characterize the performance and find sweet spots for parameters such as batch size. Figure 2 shows how the performance peaks at a receive batch size of 128 as higher values overload the L1 cache and smaller values affect branch prediction.

Router	Mpps	Relative
MoonRoute	14.6	100%
FastClick (DPDK 2.2)	10.4	72%
Click (DPDK 2.2)	4.3	29%
Linux 3.7	1.5	10%

Table 1: Single core router performance

Table 1 demonstrates the single core performance of MoonRoute compared to other software routers, such as the Linux Router, the modular software router Click and FastClick both using DPDK as backend. FastClick [3] extends Click with performance enhancing techniques, e.g., batch processing. MoonRoute can almost saturate a 10 Gbit/s link utilizing a single core, improving performance between 30% – 90% compared to its contestants. These tests demonstrate the optimal throughput for the respective software routers each containing only a single routing table entry.

5. CONCLUSION

Modularity and high performance – often considered as conflicting goals for optimization – are achieved by MoonRoute's careful design choices: the two-path design separating high from low priority tasks, improved performance with batching techniques and multi-thread optimized data structures. The just-in-time compilation for Lua allows leveraging the flexibility of a scripting language without sacrificing performance, as well as employing a convenient foreign function interface to allow easy code reuse. The performance evaluation of our reference router offers insights in its scalability, the positive effects of batching, and profiling data of a realistic complex packet forwarding system.

6. REFERENCES

[1] MoonRoute – full paper. https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/MoonRoute_draft1.pdf.

[2] MoonRoute development snapshot, scripts, raw data, and click configurations. https://github.com/emmericp/MoonRoute-data, 2017.

[3] T. Barbette, C. Soldani, and L. Mathy. Fast Userspace Packet Processing. In *Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2015.

[4] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *Internet Measurement Conference 2015 (IMC'15)*, Tokyo, Japan, Oct. 2015.

[5] M. Pall. LuaJIT. http://luajit.org/.