Paper Review November 12th

Jiaqi Yang (jxy530)

Dataplane Specialization for High-performance OpenFlow Software Switching

---

What problem is the paper solving and why is it important?

The problem that this paper tries to solve is that although OpenFlow is an amazingly expressive dataplane programming language, its expressiveness comes at severe performance prices, in terms of excessive packet classification in the fast path. Since OpenFlow is prevalently used in current software switch architecture and largely dependent on flow caching, this imposes intricate limitations on the workloads that can be supported efficiently and may even open the door to malicious cache overflow attacks. Therefore, the problem this paper addressed is important.

What is the main idea of the paper?

The main idea is to solve the problem described above: the authors argue that instead of enforcing the same universal flow cache semantics to all OpenFlow applications and optimize for the common case, a switch should rather automatically specialize its dataplane piecemeal with respect to the configured workload. The main insight of this paper is that by proposed eswitch, a novel switch architecture that uses on-the-fly template-based code generation to compile any OpenFlow pipeline into efficient machine code, which can then be readily used as fast path, because OpenFlow software packet classification still remains too expensive for today's line rates. Therefore, the authors of this paper argues that instead of relying on an overly general-purpose fast path, an OpenFlow switch should rather automatically specialize itself for the actual workload into an optimal exact matching switch when the flow tables specify pure L2 MAC forwarding, an LPM engine for L3 routing, or a fast, optimized packet classifier for L4 ACLs, and a reasonable combination of these building blocks whenever the OpenFlow pipelines matches heterogeneous protocol header fields.

How does the paper differ from previous work?

Although there are handful of researches and developments to address the problem described in part 1, the new OpenFlow switch architectures revolve around the organization of functionality inside the software pipeline implementation in order to permit processing flow entries and applying actions as fast as possible, while maintaining OpenFlow's inherent expressiveness. As a result, the vast majority of researches falls into two coarsely defined categories: direct datapath which performs packet classification right on the flow tables and indirect datapath which adapts the venerable fast-path/slow-path separation principle, like the most popular implementation: Open vSwitch. However, there are a number of problems with the current implementation, even with the most sophisticated and popular OVS design: unpredictable packet processing, performance artifacts, vexing cache management complexity, vulnerability to malicious attacks, brittle architectural constraints; the new proposal eswitch aims to solve these problems by occupying the opposite extreme by fully embracing the concept of dataplane specification.

Are there any flaws in the paper? How would you improve the paper or build on it in future work?

Although it is really attractive to automatically specialize its dataplane piecemeal with respect to the configured workload by proposing eswitch, which basically compiles, or optimizes the given OpenFlow code to more efficient machine code. However, due to the change in the code structure, it will possibly change the behavior of the program and introduce unwanted features. Furthermore, since the compiler compiles the code into different structure, it might be hard to debug; if the compiler is flawed, it could make the switches highly-vulnerable to malicious attack, or introduce bugs in the compiled machine code.