Jive: Performance Driven Abstraction and Optimization for SDN

Aggelos Lazaris[+] Daniel Tahara[×]
Xin Huang[⋆] Li Erran Li[†] Andreas Voellmy[×] Y. Richard Yang[×] Minlan Yu[+]
[†]Bell Labs [⋆]CYAN [+]USC [×]Yale

Introduction. A major benefit of software-defined networking (SDN) over traditional networking is simpler and easier programming of networks. In particular, the emergence of OpenFlow (OF) [1] has provided a standard, centralized approach for a network controller to install forwarding rules at the forwarding engines (called flow tables) of a heterogenous set of network switches, substantially reducing controller-switch dependency, and hence programming complexity.

One key issue that such a controller-switch protocol cannot resolve, however, is the diversity of switch implementations, capabilities, and behaviors. For example, hardware-based switches can have significant differences in their physical structures such as TCAM size, which significantly affects forwarding throughput over large sets of rules. Their software behaviors, such as their TCAM cache replacement algorithms and flow installation efficiency, also differ drastically. Since such diversity reflects many factors, including real-life, random, idiosyncratic designs as well as systematic switch vendor engineering exploration (which can be essential to foster switch vendor innovation), it is inconceivable that all switches will have the same capabilities and behaviors.

The presence of diverse switch capacities and behaviors can make a network much harder to understand and/or to control. For example, consider two switches with the same TCAM size, but one adds a software flow table on top. Then, insertion of the same sequence of rules may result in a rejection in one switch (TCAM full), but unexpected low throughput in the other (ended up in software flow table). Now consider that the two switches have the same TCAM and software flow table sizes; but they introduce different cache replacement algorithms on TCAM: one uses FIFO but the other traffic dependent. Then, insertion of the same sequence of rules may again produce different configurations of flow tables entries: which rules will be in the TCAM will be switch dependent. Whether a rule is in TCAM, however, can have a significant impact on its throughput, and hence QoS.

In this project, we design Jive, the first SDN programming system that systematically explores the issues of understanding and optimization of SDN programming in the presence of diverse switch capacities and behaviors. The basic idea of Jive is novel, simple, and yet quite powerful. In particular, different from all previous SDN programming systems, which ignore switch diversity or at most simply receive reports of switch features (in newer version of OpenFlow), Jive introduces a novel, proactive probing engine that measures the performance of each switch according to a well-structured set of *Jive patterns*, where each Jive pattern consists of a sequence of standard OpenFlow flow modification commands and a corresponding data traffic pattern. Utilizing the measurement results from the Jive patterns, Jive derives switch capabilities as well as the costs of a set of equivalent operations that can be utilized, through *expression rewriting*, to optimize networks with diverse capabilities and behaviors. We emphasize that despite the progress made by Jive, its scope is still limited, focusing mainly on performance. Additional switch diversities, such as ability to provide different functionalities, remain to be explored.

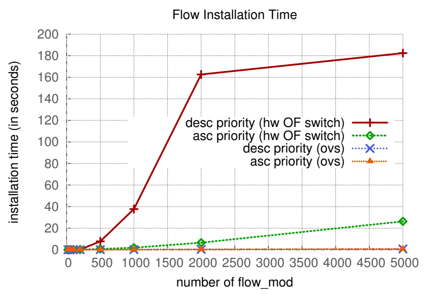

Figure 1: Behaviors of two real switches under two Jive patterns: ascending and descending.

Example Jive Patterns. We illustrate some basic ideas of Jive with two Jive patterns: (1) inserting a sequence of rules in order of increasing priority (ascending); and (2) in order of decreasing priority (descending). Figure 1 shows insertion latency for 5,000 rules on two switches: an OVS software switch and a hardware switch. Despite the simplicity of the patterns, the probing results reveal quite a few interesting properties in the behaviors of the two switches. Jive utilizes these properties to compute optimal strategies to satisfy future application requests. First, for the hardware switch, Jive records that the descending pattern has a substantially longer latency than that of the ascending pattern. The difference is about 6x for the experiments. Therefore, if Jive has 5,000 rules to install on the hardware switch, it will write the sequence in ascending to achieve optimal rule installation latency. Second, comparing across switches, Jive records that insertion into the flow table of the hardware switch is substantially slower than into that of the software switch. Hence, when Jive needs to install a low-bandwidth flow where start up latency is more important, Jive will put the flow at the software switch, instead of the hardware switch. Third, since the performance of applying a Jive pattern on a switch depends on the switch's capacity parameters, Jive can infer these parameters. In particular, when applying the ascending pattern, Jive also sends data packets matching the inserted rules. If Jive detects larger delay of the packets for newly installed rules, Jive has detected the TCAM size. In Figure 1, Jive detects that the insertion delay has switched to linear, verifying insertion into a software table.

Architecture. Jive uses an extensible architecture to measure switch behaviors according to Jive patterns and utilize the measurement results to optimize network behaviors. New Jive patterns can be continuously added and utilized. Figure 2 below shows the major architecture components of Jive.

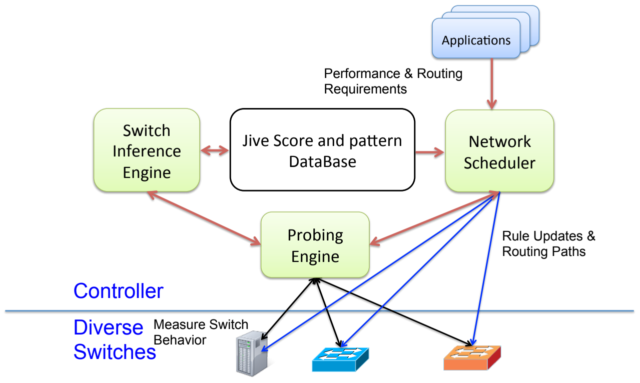

Figure 2: Jive architecture components.

Specifically, the Jive framework generates a list of Jive patterns and stores measurement results into a Jive Score DataBase (JSDB). The Switch Inference Engine utilizes JSDB to compute standard switch capabilities and behaviors (e.g., TCAM size, cache replacement algorithm) with the help of the Probing Engine. Given a Jive pattern and a switch, the the Probing Engine applies the pattern to the switch and collects measurements results that are stored in JSDB. Jive App API exposes a simple API for applications to express their flow requirements on path setup completion time, data plane latency and throughput, waypoint (switches/devices that must be traversed), and avoidance (nodes to avoid). Network Scheduler utilizes the application requirements, the JSDB and potentially the probing engine, to compute optimal network routing and switch installation plans.

Initial Implementation and Evaluation Results. We have conducted a number of initial evaluations on the effectiveness of Jive. In particular, we designed minimal Jive patterns to measure basic switch capabilities such as TCAM sizes. We have also introduced an initial set of Jive patterns to allow effective expression rewriting. Figure 3 shows one expression rewriting result, where Jive automatically uses dummy insertion + later modification to replace insertion, obtaining a 6x flow-mod latency reduction.

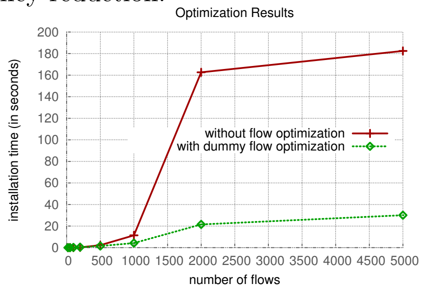

Figure 3: Initial evaluations: reduction of installation latency.

References

[1] *OpenFlow Switch Specification*, Version 1.4.0, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf