SDLoad: An Extensible Framework for SDN Workload Generation

Nicolas Laurent Stefano Vissicchio* Marco Canini
Université catholique de Louvain
<name.surname>@uclouvain.be

ABSTRACT

We propose a unified approach for workload generation to ease evaluation and comparison of SDN control-plane proposals. Our approach is based on SDLoad, an extensible framework capable of generating custom workloads satisfying input constraints, along user-defined evaluation axes.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; D.2.5 [**Software Engineering**]: Testing tools

Keywords

SDN; workload generation; evaluation; benchmarking

1. INTRODUCTION

To be practically usable in real-world networks, the SDN control plane needs to provide a highly available, correct and performant service. Recent research work has proposed several SDN control-plane designs, including support for elastic behavior [2], middleware layers [1] and network operating systems [3, 5]. As such, the SDN control plane is quickly becoming a sophisticated software stack whose behavior is determined by many interacting components.

Unfortunately, an accurate evaluation and a comparative analysis of different proposals and configurations of the SDN control plane is hard to achieve. A major limiting factor in this context is the absence of common baselines and easy-to-use workloads that mimic a range of heterogeneous network conditions. This is partially due to the limitations of the few existing SDN control-plane testing tools, like Cbench [6], or commercial solutions (*e.g.*, Ixia's IxNetwork or Spirent's TestCenter). Indeed, those tools can only produce workloads of predefined types (*e.g.*, OpenFlow rules), which hampers the testing of arbitrary SDN control-plane components.

9*Stefano Vissicchio is a postdoctoral researcher of the Belgian fund for scientific research (F.R.S.-FNRS).

HotSDN'14, August 22, 2014, Chicago, IL, USA.
ACM 978-1-4503-2989-7/14/08.
http://dx.doi.org/10.1145/2620728.2620771 .

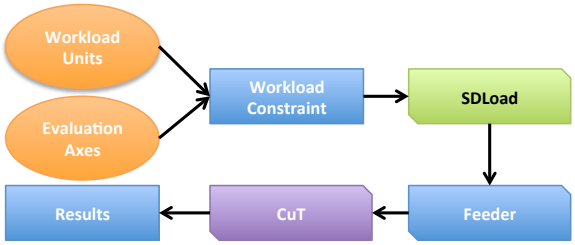

Figure 1: Overview of our approach for workload generation for an arbitrary Component under Test (CuT). Once the user has identified his workload units and the axes along which they may vary, she only has to write an SDLoad constraint to obtain a working parameterized workload generator; whereas previously she would have had to code ad-hoc generators from scratch.

In this work, we propose an approach to enable a common ground for comparison and consequently improve reproducibility of research by creating a suite of useful benchmarks. The key component of our approach is SDLoad, a novel software framework supporting workload generation for a wide range of SDN control-plane components. Beyond that, SDLoad is designed to be extensible, hence enabling the user to finely craft complex workloads for new or future SDN components. We believe that our contribution represents a significant step towards a rigorous measurement-based evaluation of SDN control-plane architectures.

2. TOWARDS A UNIFIED APPROACH TO SDN WORKLOAD GENERATION

At a high level, an experimental evaluation of SDN control-plane components should run the Component under Test (CuT) against a wide range of workloads. Those workloads should represent relevant control-plane events, ranging from high-level changes (*e.g.*, policy updates) to new network conditions (*e.g.*, device failures) and time-related events (*e.g.*, timer expirations). This way, properties of both the control-plane computation (*e.g.*, time consumed or communication overhead) and its outcome (*e.g.*, number of produced rules, time needed to install them on the switches, and so on) can be experimentally assessed.

In this work, we make a first step towards a unified approach for rapid, reliable evaluation of SDN control-plane components. The goal is to establish a uniform approach to the assessment of SDN control-plane proposals and their comparison in a range of different networking settings.

An overview of our approach is presented in Fig. 1. The key component is the artificial generation of workloads for arbitrary CuTs. This is implemented by our SDLoad tool, which performs all the input generation operations that are common across the evaluation of different CuTs.

To be as generic as possible, we define *an SDN workload as a sequence of temporally-related groups of workload units*. The workload units are application-specific. Depending on the CuT, they can indeed represent concepts at different levels of abstraction, from OpenFlow rules to network policies. Their definition is left to the user, who has to identify (*i*) *workload constraints*, expressing structure and consistency properties of inputs to the CuT, and (*ii*) *evaluation axes* along which inputs differ to model a given networking scenario (potentially including statistical distributions). If neither workload constraints nor evaluation axes are provided, SDLoad can also be used to systematically explore the input space (*e.g.*, to expose the behavior of the system with unexpected input).

Once workload units are defined, SDLoad implements a generic generation process that ensures the output workload complies with the given constraints while randomizing its elements along the specified evaluation axes. Since some CuTs may need a custom generation process (*e.g.*, to comply with specific workload constraints), we design SDLoad with extensibility goals in mind.

3. SDLOAD

SDLoad is implemented as a Java library of about 6,000 lines of code, and it is publicly available at `https://bitbucket.org/norswap/sdload`. We now describe its main features that support our workload generation approach.

Domain Specific Language (DSL). To define workload constraints and evaluation axes, SDLoad provides a small DSL, based on Java methods. The SDLoad DSL enables users to specify generator and constraint objects. The core of the DSL is formed by a set of *core methods* that return a new constraint. Core methods can take parameters, some of which may themselves be constraints. *Secondary methods* can then be invoked on constraints in order to customize them further. Secondary methods always return the constraint on which they were called, so that they can be chained freely (*e.g.*, `core().secondary().secondary()`). Some secondary methods can only be called on certain kinds of constraints (*e.g.*, constraints representing a list), while others can be called on all constraints.

Fig. 2 shows an example of a simple constraint definition. In the figure, `list`, `dict`, `choice` and `anyOf` are core methods, while `key` is a secondary method specific to the `dict` core method, and `or` is specific to `choice`. SDLoad always generates a hierarchy of lists and dictionaries with key strings, so the `list` and `dict` methods are especially important. There are secondary methods to constraint the size of the list, and secondary methods to add constraints on the dictionary's keys and values. To express workload unit distributions (*e.g.*, timings), relative probabilities of alternative constraints can be expressed through the `choice` construct; *e.g.*, `choice().or(0.1, value("a")).or(0.9, value("b"))` constrains the input to be either "a" or "b" with 10% and 90% probability, respectively.

Workflow. The user is required to provide, through the DSL interface, a top-level constraint characterizing the workload to be generated, as well as an associated generator. This constraint can reference sub-constraints and, in the same fashion, the generator can reference sub-generators.

```
Constraint networkEvents =
list(choice()
  .or(0.1, dict()
    .key("type", anyOf(
                        value("switch_up"),
                        value("switch_down")))
    .key("switch",SwitchGenerator.constraint))
  .or(0.9, dict()
    .key("type", value("packet_in"))
    .key("from", SwitchGenerator.constraint)
    .key("header", sdnModel.headerFields))
)).size(nbEventsToGenerate);
```

Figure 2: A simple constraint definition example illustrating the SDLoad domain specific language. This example models a set of network events to which a controller may wish to react. There are events representing switches going up or down, as well as events representing "packet-ins": packets transferred to the controller by a switch.

To support SDN workload generation, SDLoad bundles pre-defined constraints, generators and other utilities. They include a topology layer, which can load network topologies from `.graphml` files. The topology layer manages the switch IDs and port numbers, and offers functions to compute both shortest and random paths between pairs of switches. SDLoad also includes a set of constraints and generators that model OpenFlow messages and their constitutive parts (matching criteria, forwarding actions, *etc.*).

SDLoad supplies the generated input in a format that is agnostic to any binary format or language's object model, namely as a hierarchy of lists and dictionaries with key strings, which can be serialized in JSON. The conversion of SDLoad output to input for the CuT is left to data format translators (represented by the Feeder block in Fig. 1), which is external to SDLoad.

Experiences. We successfully used SDLoad to experiment with recently proposed control-plane components, namely (*i*) ESPRES, a scheduler taking as input groups of interdependent OpenFlow commands [4], and (*ii*) an algorithm to order forwarding updates while preserving network-wide connectivity [7]. The corresponding constraints used in our experiments are available in the SDLoad source code.

Acknowledgments. This work was (partially) supported by the ARC grant 13/18-054 from Communauté française de Belgique.

4. REFERENCES

[1] M. Canini et al. Software Transactional Networking: Concurrent and Consistent Policy Composition. In *HotSDN*, 2013.

[2] A. Dixit et al. Towards an Elastic Distributed SDN Controller. In *HotSDN*, 2013.

[3] ON.LAB. ONOS: Open Network Operating System. In *ONS*, 2014.

[4] P. Perešíni et al. ESPRES: Easy Scheduling and Prioritization for SDN. In *ONS*, 2014.

[5] T. Koponen et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, 2010.

[6] A. Tootoonchian et al. On Controller Performance in Software-defined Networks. In *Hot-ICE*, 2012.

[7] S. Vissicchio et al. Safe Update of Hybrid SDN Networks. Technical report, UCLouvain, 2013.