p4pktgen: Automated Test Case Generation for P4 Programs

ABSTRACT

With the rise of programmable network switches, network infrastructure is becoming more flexible and more capable than ever before. Programming languages such as P4 lower the barrier for changing the inner workings of network switches and offer a uniform experience across different devices. However, this programmability also brings the risk of introducing hard-to-catch bugs at a level that was previously covered by well-tested devices with a fixed set of capabilities. Subtle discrepancies between different implementations pose a risk of introducing bugs at a layer that is opaque to the user.

To reap the benefit of programmable hardware and keep—or improve upon—the reliability of traditional approaches, new tools are needed. We present `p4pktgen`, a tool for automatically generating test cases for P4 programs using symbolic execution. These test cases can be used to validate that P4 programs act as intended on a device.

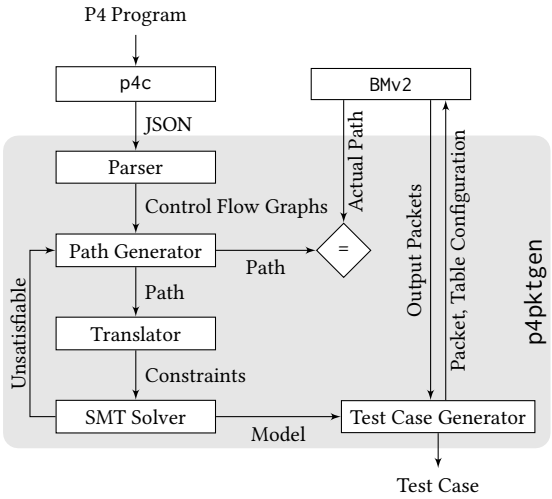

Figure 1: Overview of `p4pktgen`.

1 PRESENTER/PROJECT

Presenter: Andres Nötzli, *Stanford University*
Joint work with:

- Jehandad Khan, *Virginia Tech*
- Andy Fingerhut, *Cisco Systems*
- Clark Barrett, *Stanford University*
- Peter Athanas, *Virginia Tech*

URL: https://github.com/p4pktgen/p4pktgen

2 ADDITIONAL INFORMATION

In our demo/poster, we present `p4pktgen`, an open source tool for automatically generating test cases for P4 programs. Test cases generated by `p4pktgen` can be employed in a wide range of scenarios. One of the primary uses that we envision is to validate that P4 programs act as intended on their target devices, by running the same test packets through a software reference implementation and the target hardware and comparing the output of the two implementations. Alternatively, P4 can be used to express a specification of existing, non-programmable hardware, which can then be tested against a software reference implementation.

At a high level, `p4pktgen` takes a P4 program and produces test cases in the form of packets and table configurations. The user can choose between generating test cases for all paths or generating test cases that prioritize branch coverage. Figure 1 provides an overview of `p4pktgen`'s design.

`p4pktgen` generates its test cases using symbolic execution along concrete paths. Symbolic execution is an analysis technique for programs that translates a given program into logical formulas to examine its behavior across all possible inputs. After parsing, `p4pktgen` generates (partial) paths through the control flow graph of the given P4 program. Given a path through the program, `p4pktgen` tries to generate a packet that exercises that path. To do this, it must craft a packet that triggers the correct parser transitions, the correct conditional branches, and the correct table actions. These requirements can be expressed as a set of constraints over the inputs, creating a formula that expresses the conditions for taking the target path through the program. We use a solver for satisfiability modulo theories (SMT) to find a concrete assignment for the variables to satisfy the constraints. If the solver returns an assignment, `p4pktgen` uses it to craft a test packet for the path and the table entries. If the solver determines that no such assignment exists, `p4pktgen` uses that information to not further explore paths with the same infeasible prefix. Optionally, `p4pktgen` validates the test cases with `BMv2`, a software reference implementation of a P4 switch, by sending the test packet to the switch, parsing its output, and comparing it against the expected output.

In our experience, `p4pktgen` is capable of quickly achieving good coverage of paths or branches in a program. In under 22 minutes, it covers 95% of the branches in `switch.p4`,[1] one of the largest open-source P4 programs. We also found that `p4pktgen` can cover all paths in programs with thousands of paths in a matter of minutes. Finally, `p4pktgen` helped us discover five bugs in the open source P4 tools.

[1] In our tests, we modified `switch.p4` slightly to remove features not currently supported by `p4pktgen`, such as P4 hashes and action profiles.