A Programmable, Hardware-Assisted Network Protocol Fuzzer

Yuxin Tang Ang Chen
Rice University

Fuzzing is an important approach to software testing, particularly when the source code is not available. At a high level, a fuzzer generates random inputs to the program under test, and identifies the (sequences of) inputs that trigger faulty behaviors, such as a program crash. In the context of networking software, a network protocol fuzzer produces sequences of packets based on a protocol specification as test cases. Since many network protocols are stateful, the fuzzer also needs to closely follow specified state transitions when deciding on the next packets to send [3]. One important metric for a fuzzing test is *time to first fault* (TTFF) [1], which quantifies the performance and effectiveness of a fuzzer by measuring the time it takes to trigger the first fault in the tested software. Since modern software tends to be large and complex, fuzzers with higher performance can cover more test cases and provide higher assurance.

Our system: `p4f`. We describe `p4f`, a programmable, high-performance network protocol fuzzer. What sets `p4f` apart from existing fuzzing tools is its ability to a) accept high-level protocol specifications in a declarative language, and b) automatically synthesize fuzzer implementations that are targeted to run directly on switch hardware. The `p4f` system leverages a recent trend in the networking community that has developed *programmable data planes* on modern switches. Devices such as Intel FlexPipe and Barefoot Tofino are integrated with a reconfigurable match/action architecture that enables custom definition of network protocols, programmable parsers, and linespeed on-chip packet generation. The initial motivation behind this line of work is the quest for more network programmability both in the control plane (as found in software-defined networks) and also in the data plane (which used to be fixed-function blackboxes). Our key observation is that such *new capabilities on programmable data planes match nicely with the design requirements for a network protocol fuzzer*—namely, the ability to generate packets for custom protocols at high speed.

Specifying network protocols. Users of `p4f` can specify network protocols in a high-level language, analogous to using a data format parser [2]. This declarative interface relieves users from the burden of reasoning about low-level implementations of the fuzzer, and allows fast prototyping of new protocol fuzzers by redefining the specification. Our language has two basic constructs: packets and sequences. A *packet* simply describes the data format of a particular type of packets in a protocol (e.g., HTTP GET). The fuzzer can generate a *sequence* of packets by randomly mutating certain protocol fields, by following state transitions, or both. Consider the following example, which defines an HTTP GET request for `http 1.1`, with random strings as the URI and random data as payload; string lengths and data sizes can be further parameterized.

```
http-get = ip|tcp|op|uri|ver|crlf|pload
op       = "GET"
uri      = rand[10]
ver      = "HTTP/1.1"
crlf     = "\r\n"
pload    = rand[100]
```

Based on this, a fuzzer could further define a sequence $(\texttt{http-get})^k$ to generate k random HTTP GET requests, or a sequence $(\texttt{http-post})^t$ to generate t HTTP POST requests, or an interleaving sequence of GET and POST requests using $(\texttt{http-get|http-post})^*$.

Compilation strategies. Our compiler takes a protocol specification, and generates a protocol fuzzer written in P4, which is a language for programmable data planes. A packet in the specification is directly compiled as header and parser definitions, and packet assembly pipelines. A sequence is compiled by instantiating read/write registers to keep track of the protocol progress, and by protocol mutations using pre-coded randomness.

Initial validation. Our current `p4f` prototype is handwritten in P4 for a subset of the HTTP protocol. It runs in an emulated environment in Mininet, and generates test packets to emulated network servers. Our initial results show that the current prototype can generate 19 packets per second as an HTTP fuzzer. We note, however, that this experiment is meant as a feasibility study rather than performance benchmarks, because the final `p4f` system would run directly on switch hardware. As ongoing work, we are building a fully functional compiler that can generate P4-based fuzzers based on user-defined specifications.

References

[1] State of Fuzzing. `https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/state-of-fuzzing-2017.pdf`.

[2] J. Bangert and N. Zeldovich. Nail: A practical tool for parsing and generating data formats. In *Proc. OSDI*, 2014.

[3] G. Banks, M. Cova, V. Felmetsger, K. Almeroth, R. Kemmerer, and G. Vigna. SNOOZE: Toward a stateful network protocol fuzzer. In *Proc. ISC*, 2006.

Yuxin Tang: Student author