A Programmable Framework for Validating Data Planes

Pietro Bressana
Universitá della Svizzera italiana
pietro.bressana@usi.ch

Noa Zilberman
University of Cambridge
noa.zilberman@cl.cam.ac.uk

Robert Soulé
Universitá della Svizzera italiana
Barefoot Networks
robert.soule@usi.ch

ABSTRACT

Due to the emerging trend of programmable network hardware, developers have begun to explore ways to accelerate various applications and services. As a result, there is a pressing need for new tools and techniques for debugging network devices. This paper presents NetDebug, a fully programmable hardware-software framework for validating and real-time debugging of programmable data planes. We describe validation use cases, compare our design to alternative solutions, and present a preliminary evaluation using a prototype implementation.

1 INTRODUCTION

The way in which we use network devices is changing. The emerging trend of programmable data planes [8] has lead to increased interest in *in-network computing*, a form of hardware acceleration in which applications and services traditionally running on servers are executed on network devices [9, 10]. This begs the question: *Given that in-network computing is asking network devices to do so much more work, how can we be sure that they behave correctly?*

Validating the correctness of software applications is widely regarded as a difficult task [5]. The challenges become more acute when moving programs to the network hardware, for several reasons. First, the separation of the control plane and data plane makes it hard to reason about the exact configuration in which a program is run. Second, debugging network devices often depends on network traffic. Once a switch stops sending packets, existing tools have no way to diagnose error. Third, network programming languages, such as P4 [11], are designed to be target-independent by abstracting away architecture-specific details. Although this increases code portability, it can lead to problems; code may have undefined behavior on some targets, and compiler may support only a subset of the language specification.

There is clearly a need for new tools and techniques to support network debugging. To address these challenges, we introduce NetDebug, a programmable hardware-software framework, which provides validation and real-time debugging of programmable data planes.

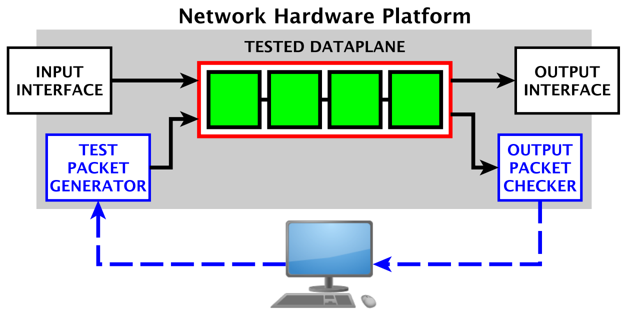

Figure 1: The proposed architecture

2 SYSTEM OVERVIEW

NetDebug provides a fully programmable test infrastructure inside network devices. It allows debugging in real time at full line rate. NetDebug is deployed in parallel to live traffic. Developers write test and validation code using P4. We chose P4 because it was designed specifically for network programming and has been widely adopted on a number of hardware platforms [1–3]. However, NetDebug is independent from the language of the application. It can validate data planes designed using a number of different workflows and languages, including high level synthesis, P4, C/C# and hardware description languages.

The NetDebug architecture, shown in Figure 1, is composed of two hardware modules implemented inside the target network platform: a test packet generator and an output packet checker. Both modules are managed by a software tool, running on a host computer, which uses a dedicated interface to configure the generation of test packets and to collect test results. The hardware infrastructure, which is fully programmable through P4 language, is internally connected to the data plane under test. This design allows NetDebug to validate the data plane while avoiding the surrounding hardware, including the network interfaces. Users can generate custom test packets and inject them directly into the data plane under test, while running verification on output packets at line rate, at real time. This is specifically beneficial for programmable data planes, where the same architecture is used for many different programs. If a bug prevents packets from being correctly forwarded to the output interfaces of the device, users can find where the fault occurred, even inside the data plane.

3 USE-CASES

NetDebug can be applied to a wide range of use cases, including but not limited to:

- **Functional testing**: finding/detecting functional bugs in the data plane and in the control plane
- **Performance testing**: performance metrics, such as throughput, packet rate and latency
- **Compiler check**: finding/detecting limitations in the compiler
- **Architecture check**: finding/detecting limitations in the architecture
- **Resources quantification**: evaluating the consumption of hardware resources
- **Status monitoring**: providing periodic internal status information
- **Comparison**: comparing alternative specifications of the same program

Use-case	**NetDebug**	**Software formal verifiers**	**External network testers**
Functional testing	✓	✓	partial
Performance testing	✓		partial
Compiler check	✓		partial
Architecture check	✓		partial
Resources quantification	✓		
Status monitoring	✓		
Comparison	✓	partial	partial

Table 1: Use-cases

Table 1 summarizes the capabilities of NetDebug compared to related work. Software formal verification tools [5] can check *functional* correctness. They can not be used to test a hardware implementation, since verification is performed only on the software specification of the programs. External network testers [6] are able to run only partial tests related to *functional*, *performance*, *compiler* and *architecture* use-cases, since they lack a "internal view" of the device under test, as they are limited by the external interfaces of a device, and have no visibility into internal data plane failures. For the same reason, they are unable to test either the *resources* or *status monitoring* use-cases. NetDebug can perform full *comparisons*, since it is able to run tests related to all the discussed use-cases. Software formal verification tools and external network testers can make comparisons based on the use-cases that they are able to test.

4 EVALUATION

We have implemented a prototype of NetDebug on NetFPGA SUME [7] using Xilinx SDNet [4], which translates P4 specifications into a hardware module. We have used NetDebug to debug several data plane programs.

Preliminary experiments have already provided some useful insights. For example, using NetDebug, we discovered that the *reject* parser state, an essential feature of P4 language, is not implemented by SDNet. This meant that any packet coming into the data plane was sent out to the next hop, even if it was supposed to be dropped. Our framework immediately detected this severe bug, that would not be noticed by applying software formal verification to the data plane program [5].

5 CONCLUSIONS

We have presented NetDebug, a programmable framework for validating data planes. NetDebug leverages both the P4 language and hardware design to provide flexibility and visibility into programmable network devices. We have built a prototype of NetDebug, and used it to detect a bug not visible through software formal verification tools. As in-network computing becomes increasingly popular, NetDebug addresses an urgent need for improved tools and techniques for data plane debugging and verification.

REFERENCES

[1] 2018. Barefoot Tofino. https://barefootnetworks.com/products/brief-tofino/. (2018).
[2] 2018. Netronome SmartNICs. https://www.netronome.com/. (2018).
[3] 2018. What is an FPGA? https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. (2018).
[4] 2018. Xilinx SDNet. https://www.xilinx.com/products/design-tools/software-zone/sdnet.html. (2018).
[5] C. Cascaval et al. 2018. p4v: Practical Verification for Programmable Data Planes *(SIGCOMM '18)*.
[6] G. Antichi et al. 2014. "OSNT: open source network tester". *IEEE Network* (2014).
[7] N. Zilberman et al. 2014. "NetFPGA SUME: Toward 100 Gbps as Research Commodity". (2014).
[8] P. Bosshart et al. 2013. "Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN" *(SIGCOMM '13)*.
[9] X. Jin et al. 2017. "NetCache: Balancing Key-Value Stores with Fast In-Network Caching" *(SOSP '17)*.
[10] X. Jin et al. 2018. NetChain: Scale-Free Sub-RTT Coordination.
[11] The P4 Language Consortium. 2017. "P4$_{16}$ Language specification (v1.0.0)". (May 2017).