Keeping P4 switches fast and fault-free through automatic verification

Dániel Lukács, Gergely Pongrácz, Máté Tejfel

Abstract: The SDN dataplane is going through a paradigm shift, as softwarization of switches sees an increased pull from the market. Yet, software tooling to support development with these new technologies is still in its infancy.

In this work, we introduce a framework for verifying data plane protocols defined in the P4 language. Using symbolic execution, the framework checks *crash-freedom* and *bounded execution* properties of P4 protocols, and verifies performance requirements by estimating *lower and upper bounds* of packet processing time. This paper explains related terminology, and briefly describes the methodologies used to reach this goal.

Keywords: programmable switches, P4, SDN, performance modeling, symbolic, execution

Introduction

Currently in the networking industry, network devices are being commoditised fast and software gets more and more market share, as consumers want scalable and easily replaceable devices, while vendors want to keep development costs low. Software-defined networking (SDN) [1] technologies address this need by allowing network administrators to dynamically control network topology, configurations, and protocols.

New languages are emerging, aiming to assist network engineers to define the functioning of switches in the SDN forwarding plane. Among them, P4 intends to keep the best aspects of both hardware and software by enabling network engineers to communicate their intent in a general high-level language, while the task of compiling high-level protocol description to low-level target architectures is delegated to the backend software. The hybrid approach is highly effective, but burdens backends, static analyzers, and verification frameworks, as now these also have to take into account low-level targets.

The framework presented in this paper intends to verify *functional requirements* (e.g. execution is bounded and error-free) and *non-functional requirements* (e.g. performance goals are reached) of network protocols in P4, in order to support new users of the language, and to provide insights and dynamic feedback to developers of the language ecosystem.

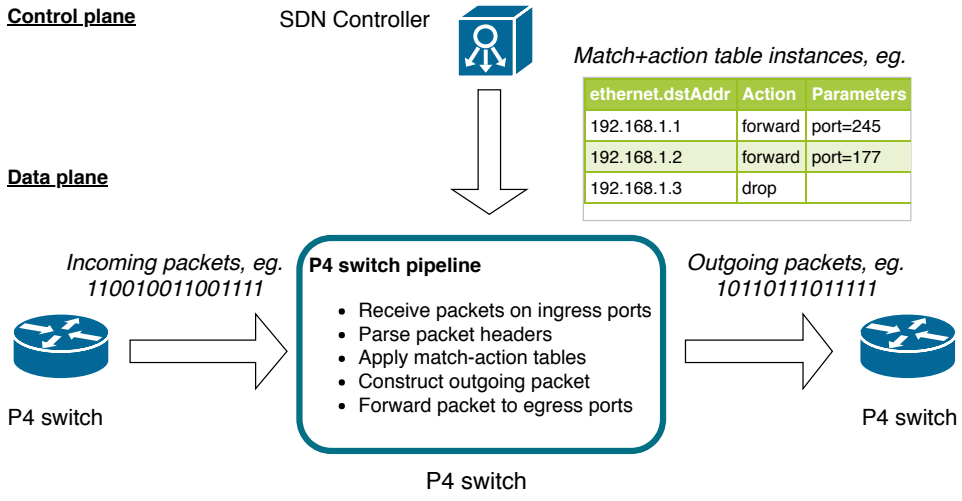

Figure 1: Application context of P4 pipelines

About P4

In Figure 1, the application context of pipelines defined in the P4 language is depicted. Being network protocols, P4 programs describe the packet filtering and forwarding activities to be performed by network devices. Unprocessed packets enter the pipeline on the *ingress-ports*, and processed packets exit on the *egress-ports*. In most pipelines, packets (bit-sequences) get *parsed* (i. e. headers and payload are identified) by a user-defined state machine, then packet headers are matched against one or more key-value stores (called *match+action tables*) as specified by the control flow definition. When a match is found, the corresponding *actions* are performed to modify the packet or the program state, or to trigger device-dependent side-effects (e.g. sending a message to the SDN controller). Match+action tables (successors of conventional routing tables) are received from the SDN controller at runtime and may change repeatedly during the operation of the switch.

Figure 2 depicts a small L2 routing protocol in P4. Here, packets get parsed, then they are matched based on their destination address and a suitable egress port is selected according to the match+action table instance. After the processing, the now modified packet gets forwarded to its destination.

```
#include <core.p4>
#include <v1model.p4>

header ethernet_t{
    bit<48> dstAddr;
    bit<48> srcAddr;
    bit<16> etherType;
}
ethernet_t ethernet;

parser ParserImpl(packet_in packet,
                  out headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t smeta){
    state start {
        packet.extract(hdr.ethernet);
        transition accept;
    }
}

control Ingress(inout headers hdr,
                inout metadata meta,
                inout standard_metadata_t smeta){
    action forward(bit<9> port) {
        standard_metadata.egress_spec = port;
    }

    table dmac {
        actions = { forward; }
        key = { hdr.ethernet.dstAddr: exact; }
        size = 512;
    }
    apply { dmac.apply(); }
}

control DeparserImpl(packet_out packet,
                     in headers hdr){
    apply { packet.emit(hdr.ethernet); }
}

SwitchImpl(ParserImpl() Ingress(),DeparserImpl())
                                              main;
```

Figure 2: A simple L2 routing protocol in P4 that forwards packets based on their destination address according to the current content of the routing table.

Motivation

While software gives increased power for engineers to express their intent, the price of rising architectural complexity is the prevalence of faults and performance anomalies in shipped products. For example, the survey of Kim et. al [2] concludes that testing correctness, and guarantees of service quality (e.g. performance) are the most important problems in configuration automatisation. Efficient development requires improved tooling for developers in order to develop, troubleshoot and fine-tune efficient network setups proactively in development-time instead of deployment-time. On the other hand, automatised troubleshooting can also be utilised by the SDN controller automatically in deployment-time lessening the need for human intervention. Apart from providing tooling support for developers, performance estimations enable backend developers (e.g. T4P4S compiler project [3] at ELTE University, Hungary) to compare efficiency of the compilation output against a theoretical optimum. By getting a clear picture on the challenges of developing optimal backends for P4, developer experience can then be used in a feedback loop to improve and extend the language itself.

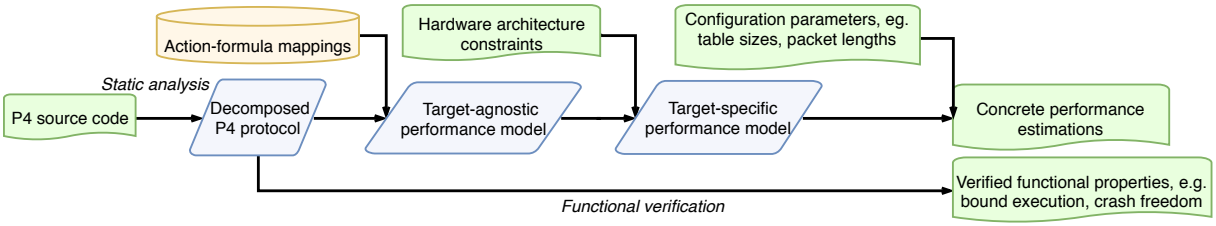

Figure 2: Dataflow diagram depicting inputs, outputs and intermediate results of the P4 verification framework in development

Related work

Our current work is only concerned with verification of switch behaviour in isolation. Before programmable switches, device-level verification meant model checking hardware implementations of protocols, and this was feasible as the protocols to verify were relatively simple. With arbitrarily complex protocols in the SDN forwarding plane appeared new research challenges too. Examples of interesting works on verification of software protocols are VigNat [4] implemented in C; verification of a data center protocol in P4 [5]; and verification of P4 programs by first compiling them to Datalog [6]. We are yet unaware of any approach that is able to automatically verify performance requirements on P4 protocols.

Methodology

The framework presented in this paper analyses P4 protocols accompanied by sufficient HW architecture descriptions to verify satisfaction of functional requirements (execution is bounded and crash-free) and non-functional requirements (performance goals are reached). In this paper, we intend to examine, synthesise, and build on two compositional methodologies in order to reach both goals.

In [7], the authors manually analyse the Ethernet protocol and a specific hardware architecture, which are then synthesised into a sequence of primitive packet processing actions called *elementary operations (EOs)* in order to quantify performance. We intend to automatise and apply the methodology in this work to P4 pipelines, as these can also be naturally decomposed into *primitive actions* and *extern objects*, which are dependent on the target network device.

To guarantee well-foundedness of the performance models, we need to show that the verified protocols always halt in an accepting program state. In [8], the authors show that *crash-freedom*, *bounded execution* and certain filtering properties can be proved by symbolically executing protocols of the Click network programming language. The authors utilise the composability of Click to avoid state-space explosion and to effectively linearise the cost of the search. They perform the search for problematic memory states in small pipeline components instead of the whole pipeline. As P4 pipelines are also compositional, the aforementioned idea applies to P4 similarly: to reduce the search space, we can search each parser blocks, control blocks, and user-defined actions in isolation, and only perform checks on the whole pipeline.

Figure 2 depicts a high-level view on the dataflow of the proposed method. The inputs of our method (to be supplied by its users) are the P4 protocol source code, the hardware architecture description, and specific network configuration parameters. The framework verifies correctness of the source and other functional requirements, followed by performance analysis. The performance estimates resulting from this analysis can then be used to verify that non-functional requirements are also satisfied. Building on the aforementioned research also allows us to validate our framework against the results in those works.

Conclusions

Scalability and vendor-independence gets more and more valuable for stakeholders in networking technologies, and at the same time vendors are required by the market to keep development costs low by moving from hardware to software. As a result, new technologies are emerging aiming to make the SDN data plane more programmable. Among them is P4, a language for describing switching pipelines. Our ongoing research effort aims to develop a framework to verify functional and non-functional requirements of network protocols defined in P4. In this paper, we presented a methodology based on [7] and [8] to check *crash-freedom* and *bounded execution* properties of P4 protocols, and estimate *lower and upper bounds* of packet processing time in order to verify that performance goals were reached. We believe that future iterations of this technology will bring user experience and efficiency of P4 users to new levels, and also provide deeper insights to developers of the language ecosystem.

Acknowledgements

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013).

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[2] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. Kinetic: Verifiable Dynamic Network Control. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 59–72, Berkeley, CA, USA, 2015. USENIX Association.

[3] Sándor Laki, Dániel Horpácsi, Péter Vörös, Róbert Kitlei, Dániel Leskó, and Máté Tejfel. High speed packet forwarding compiled from protocol independent data plane specifications. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 629–630, New York, NY, USA, 2016. ACM.

[4] Arseniy Zaostrovnykh, Solal Pirelli, Luis Pedrosa, Katerina Argyraki, and George Candea. A Formally Verified NAT. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 141–154, New York, NY, USA, 2017. ACM.

[5] Lucas Freire, Miguel Neves, Alberto Schaeffer-Filho, and Marinho Barcellos. Poster: Finding vulnerabilities in p4 programs with assertion-based verification, 10 2017.

[6] Nick McKeown, Dan Talayco, George Varghese, Nuno Lopes, Nikolaj Bjorner, and Andrey Rybalchenko. Automatically verifying reachability and well-formedness in p4 networks. Technical report, September 2016.

[7] A. Sapio, M. Baldi, and G. Pongrácz. Cross-Platform Estimation of Network Function Performance. In *2015 Fourth European Workshop on Software Defined Networks*, pages 73–78, Sept 2015.

[8] Mihai Dobrescu and Katerina Argyraki. Software dataplane verification. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 101–114, Berkeley, CA, USA, 2014. USENIX Association.