P4TrafficTool: Automated Code Generation for P4 Traffic Generators and Analyzers

Deepanshu Jindal
deepanshu.jindal.cs116@cse.iitd.ac.in
IIT Delhi

Raj Joshi
rajjoshi@comp.nus.edu.sg
National University of Singapore

Ben Leong
benleong@comp.nus.edu.sg
National University of Singapore

CCS CONCEPTS

• **Networks → Programmable networks**.

1 INTRODUCTION

The flexibility to design next generation networks faster with new protocols (e.g. Geneve [7]) and applications (e.g. in-network telemetry) is a key driver behind the rise of programmable dataplanes and the P4 programming language [14]. New protocols and applications invariably require new packet headers (or protocol layers). The P4 programming language [3] allows new packet headers to be expressed in the form of header definitions and their "binding" with other headers to be expressed as a parser specification. P4 programs with custom headers are then easy to compile and deploy on programmable hardware. However, once deployed, the testing of these P4 deployments, especially in production settings, requires traffic generators and traffic analyzers that support these new headers.

Commonly used tools such as WireShark [20] do not support these new headers *out of the box*. To address this issue, the current approach is to write new plugin code for extensible traffic generators and analyzers so that these tools can generate and analyze traffic with the new headers. However, writing the plugin code can be complex, time consuming and error-prone. For example, to write the plugin code for WireShark [20], a P4 developer would require to learn the Lua programming language [13] and the WireShark dissector plugin framework. Furthermore, the plugin code needs to be modified and re-written every time the header fields change during the development cycle of a protocol or application. Maintaining an up-to-date plugin code thus inflicts a significant overhead for a P4 developer as the headers evolve over time. While it is relatively simple to write plugin code for the Python-based Scapy [19], we found that Scapy cannot support testing fast 10+ Gbps hardware systems, e.g. testing a microburst monitoring system such as BurstRadar [11]. In such instances, a DPDK-based [5] tool such as MoonGen [6] or Pcap++ [18] will be required. These tools in turn require the developer to write relatively complex plugin code.

Also, while working with P4, we found that existing plugin code generators have limited functionality. For example, while the P4 WireShark Dissector Generator [17] can generate plugin code for WireShark, it supports only a single new header in the header stack and only at the end of the stack. P4pktgen [15] generates input packets to cover all paths of a P4 program, but it does not provide any plugin code for a traffic generator or analyzer.

In this paper, we present P4TrafficTool, a new plugin code generation tool that automatically generates the plugin code required to support new headers in P4 programs for a number of common traffic generators and analyzers. P4TrafficTool currently generates plugin code for WireShark, Scapy, MoonGen and Pcap++, and is easily extensible. P4TrafficTool allows these *target tools* to be used for testing P4 programs with new headers on software (bmv2 [2]) and hardware targets (Barefoot Tofino [1]) thereby accelerating the development of new protocols and applications.

2 HOW P4TRAFFICTOOL WORKS

To generate the plugin code for different target tools, P4TrafficTool needs the description of new headers and the information about how these new headers are *linked* to other headers in the protocol stack. To obtain this information from a P4 program in a structured format, we use the open source P4 reference compiler (p4c) [16] to obtain a High Level Intermediate Representation (HLIR). Using the HLIR representation, P4TrafficTool identifies the headers that are not metadata and extracts the header descriptions. It then generates the data structures for these headers in the format required by each target tool. Next, we determine the correct data types for the header fields. For example, for Pcap++, a `uint8_t` type would be used for a *padded* 4-bit field whereas `uint16_t` would be used for a 16-bit field. P4TrafficTool tries to determine if any of the identified headers are standard headers (Ethernet, IPv4, TCP, etc.) and provides an option to use in-built implementations of these headers instead of adding new plugin code. This avoids code redundancy and also improves performance if optimized implementation for such headers is available in the target tool(s). For example, the plugin code for WireShark is interpreted at run time and is therefore slower than the compiled in-built code. Once the header definitions are available, P4TrafficTool uses the

SOSR '19, April 3–4, 2019, San Jose, CA, USA
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6710-3/19/04.
https://doi.org/10.1145/3314148.3318047

Table 1: Lines of code generated by P4TrafficTool

P4 Program[1]	Scapy (Python)	WireShark (Lua)	MoonGen (Lua)	Pcap++ (C/C++)
basic_postcards	33	116	360	274
basic_tunnel	22	79	150	101
hula [12]	47	150	304	211
linear_road [8]	120	411	1805	1317
mri	47	144	450	303
netcache [10]	100	189	1691	1221
netchain [9]	42	195	340	239
p4paxos [4]	26	118	207	152
qmetadata	47	172	450	338
src_routing	29	83	151	101

parser specification to construct a *parser control graph* which is independent of the target tool. The parser control graph encodes the possible transitions between the headers based on the values of certain header fields. As an example, for the Ethernet header, a value of the `etherType` field equal to `0xFFFF` could denote a transition that the next header is a new header `foo`. P4TrafficTool then translates the parser control graph into header binding/mapping data structures that are specific to the target tool(s). The final output of P4TrafficTool is a set of files containing the generated code for each target tool. For Pcap++ and MoonGen, P4TrafficTool also provides the user with additional instructions and code to properly integrate the plugin code with the common library files of the target tool.

P4TrafficTool is currently implemented with about 1400 lines of Python code and is available at https://git.io/fhnVe (and open-sourced under MIT License). For Pcap++ and MoonGen, P4TrafficTool also adds additional extensions to support header fields with non-standard bit widths such as 24, 40, or 48. Since there are no data types available in C/C++ (for Pcap++) or Lua (for MoonGen) to support fields with such bit widths, P4TrafficTool defines its own data structures and associated methods to handle such non-standard bit-width fields. It also handles variable length header fields. Further, P4TrafficTool transparently performs the endianness conversion between the network and the host by providing users with simple getter and setter functions in the plugin code.

3 EVALUATION

To evaluate how P4TrafficTool can save time for a P4 developer, we obtained[2] 10 different publicly available P4 programs that use new headers other than the standard TCP/IP protocol stack, and generated the plugin code for the 4 supported target tools. The amount of plugin code generated by P4TrafficTool for each target tool is shown in Table 1. These numbers suggest that P4TrafficTool would allow a P4 developer to avoid writing hundreds of lines of plugin code. The savings are especially significant for MoonGen and Pcap++. The automated generation of these headers will likely avoid human error. We also found that P4TrafficTool works well for P4 programs written for proprietary hardware such as Barefoot Tofino [1].

[1]Some P4 programs are slightly adapted due to the limitations (§4)
[2]https://git.io/fhnaX

4 LIMITATIONS AND FUTURE WORK

A packet starting with a header other than Ethernet and a transition to the next header based on multiple header field values is currently not supported. We are working on the patches required to support such use cases. We are also planning to incorporate automatic code generation for checksum calculation and verification. For now, the user can define appropriate functions to compute the checksum header fields. P4TrafficTool currently requires individual header fields to be byte-aligned with appropriate padding to generate code for Pcap++ and MoonGen. We plan to add data structure support for non-byte-aligned fields.

REFERENCES

[1] Barefoot Tofino 2019. https://goo.gl/cdEK1E.
[2] Behavioral Model 2019. https://git.io/fhnni.
[3] Pat Bosshart, Dan Daly, Glen Gibb, et al. 2014. P4: Programming protocol-independent packet processors. *SIGCOMM CCR* (2014).
[4] Huynh Tu Dang, Marco Canini, Fernando Pedone, et al. 2016. Paxos made switch-y. *SIGCOMM CCR* (2016).
[5] DPDK 2019. https://dpdk.org
[6] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, et al. 2015. MoonGen: A scriptable high-speed packet generator. In *Proceedings of IMC*.
[7] Geneve 2019. https://goo.gl/4tuEby.
[8] Theo Jepsen, Masoud Moshref, Antonio Carzaniga, et al. 2018. Life in the fast lane: A line-rate linear road. In *Proceedings of SOSR*.
[9] Xin Jin, Xiaozhou Li, et al. 2018. NetChain: Scale-Free Sub-RTT Coordination. In *Proceedings of NSDI*.
[10] Xin Jin, Xiaozhou Li, Haoyu Zhang, et al. 2017. NetCache: Balancing key-value stores with fast in-network caching. In *Proceedings of SOSP*.
[11] Raj Joshi, Ting Qu, Mun Choon Chan, et al. 2018. BurstRadar: Practical real-time microburst monitoring for datacenter networks. In *Proceedings of APSys*.
[12] Naga Katta, Mukesh Hira, Changhoon Kim, et al. 2016. Hula: Scalable load balancing using programmable data planes. In *Proceedings of SOSR*.
[13] Lua Programming Language 2019. https://lua.org.
[14] Nick McKeown. 2017. Why Does the Internet Need a Programmable Forwarding Plane? https://youtu.be/zR88Nlg3n3g.
[15] Andres Nötzli, Jehandad Khan, Andy Fingerhut, et al. 2018. P4pktgen: Automated test case generation for P4 programs. In *Proceedings of SOSR*.
[16] P4 Reference Compiler 2019. https://git.io/fhnnX.
[17] P4 WireShark Dissector 2019. https://git.io/fhnXh.
[18] PcapPlusPlus 2019. https://git.io/fhnZL.
[19] Scapy 2019. https://scapy.net.
[20] Wireshark 2019. https://wireshark.org.