



Figure 2: Parser merging in ShadowP4. **Figure 3: Packet life-cycle for two testing operators.**

(SPM), runs on the switch, intercepts messages between the control plane (controllers) and the merged PDP Program, and uses the ShadowConfiguration to determine how to appropriately modify the messages for each programs.

ShadowP4 Compiler. SPC merges PDP Programs in three stages. **First**, SPC merges multiple control flows of PDP Programs into single one with maximum resource overlap. Central to the SPC’s design is the identification of the maximum data plane resources within all PDP Programs that can be “safely” merged while maintaining the semantics and dependencies of each PDP Program. The left part of Figure 1 gives an example merge (*Merged.P4*) of two control flows (*Production.P4* and *Testing.P4*), where the tables (B) (C) can be “safely” merged; however, tables (A) (B) can not because merging them will introduce a loop to the control flow. Given this information, SPC merges the PDP Programs by: (1) rewriting the table IDs to avoid conflicting IDs, (2) rewriting “GoTo” statements for all tables except the merged tables to reflect the new Table IDs, (3) for merged tables, SPC will the add appropriate branches, each as an independent “GoTo”, to keep the dependency consistent with those before merging. **Second**, SPC merges the parsers of PDP Programs. As each parser is a finite state machine (FSM), we align the parsers’ FSM and merge identical states. Figure 2 presents an example of two parsers being merged, where the merged states are in pink. We introduce the SFlag state which specifies that packets with the SFlag, *i.e.*, test packets, should parse the IPv6 header type, where as only packets without the SFlag, *i.e.*, non-test packets, should be able to parse the VLAN header type. **Third**, SPC will add corresponding tables to support testing operators as shown below.

ShadowP4 operators. To support **A-B Testing** composition, SPC adds an extra table STC, or Shadow Traffic Control in Figure 1, which identifies a packet as either “test” or “production” traffic and adds a bit (ShadowBit) to the packet’s metadata. The STC is the first table that all packets encounter and affixes the ShadowBit and also guides the packets along the appropriate pipeline. By populating STC tables, the network operators can specify which traffic or which percent of

traffic goes through the testing “pipeline”. The **Differential Testing** operator is similar to the A-B Testing, using STC to management testing traffic. The key difference is that: the packets in A-B Testing either go to the production or the test instances. Whereas in the Differential Testing, the test packets must be copied and send through both programs. To do this, we leverage the recirculate primitive, which recirculates a packet for processing in the pipeline. The packet life-cycle is shown in Figure 3. To support **comparator**, SPC adds (1) one output record table (ORT) to the end of each pipeline to copy the outputs of each version to the pre-defined metadata, and (2) comparator table (CTB) to the end of the merged pipeline to compare the outputs. The fields to be recorded and compared can be configured to various types by operators through SPI, for example, either 16-bits or multiple 32-bits of packet header or metadata. CTB will report a message along with the packet to controller if the outputs are not equal. To tackle overheads of recirculating packets, the operators can reduce the number of packets sampled to achieve an acceptable threshold.

3 IMPLEMENTATION AND EVALUATION

Implementation: We developed ShadowP4 based on the high-level intermediate representations (HLIR) of P4 programs and it merges them into one program. Merging the HLIR allows us to operate at a platform independent level while maintaining the complete semantics of the P4 language.

Testing Demo: We demonstrate the use of several ShadowP4 operators work for network testing on the Bmv2 [1] switch. For A-B Testing, we first provide two P4 programs and indicate the testing version to SPI. Then, we evaluate the merged program from SPC on a single P4 switch Bmv2. At runtime, we will configure the proportion for testing. We will, also, demo how to leverage Differential Testing to test the routing behavior of two different routing applications. To demo this, we will provide two identical P4 router programs to the SPI and set the comparator to record and compare the next-hop (32 bits) fields of the testing packets. At runtime, we control the routing tables of the two programs with two different routing applications and demonstrate differences as captured and exposed to the end user by the SPA.

Performance Overheads: We evaluated the performance of ShadowP4 on Bmv2 and hardware ONetSwitch [4], using various real P4 programs including L2 Switch, Router, VLAN [2]. Due to space limitation, next we summarize our findings: we observed that in the software switch (Bmv2), the throughput decreased by less than 1.5% and the delay penalty was less than 3%. For the hardware switch (ONetSwitch), we observed that both throughput and delay were degraded by less than 1%.

REFERENCES

- [1] P4 Language Consortium. 2018. Behavioral Model. (2018). <https://github.com/p4lang/behavioral-model>
- [2] P4 Language Consortium. 2018. P4 Example Programs. (2018). <https://github.com/p4lang/tutorials>
- [3] Ramesh Govindan et al. 2016. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *Proc. of SIGCOMM'16*. 58–72.
- [4] Chengchen Hu et al. 2014. Design of All Programmable Innovation Platform for Software Defined Networking. In *ONS'14*. USENIX, Santa Clara, CA.
- [5] Eric Keller, Minlan Yu, Matthew Caesar, and Jennifer Rexford. 2009. Virtually Eliminating Router Bugs. In *Proc. of CoNEXT'09*. ACM, New York, NY, USA, 13–24.
- [6] Twitter. 2015. Diffy: Testing Services Without Writing Tests. (2015).
- [7] Kaushik Veeraraghavan et al. 2016. Kraken: Leveraging Live Traffic Tests to Identify and Resolve Resource Utilization Bottlenecks in Large Scale Web Services. In *Proc. of OSDI'16*. USENIX, Savannah, GA, 635–651.
- [8] Danyang Zhuo, Qiao Zhang, Xin Yang, and Vincent Liu. 2016. Canaries in the Network. In *Proc. of HotNets'16*. ACM, New York, NY, USA, 36–42.