# Testbed Evaluation of an Attestation-Capable, Programmable Software Switch

Alexander Wolosewicz Nishanth Shyamkumar Nik Sultana
Illinois Institute of Technology
Chicago, IL, USA

## Abstract

This paper describes a testbed evaluation of the first implementation of an *attestation-capable*, programmable network switch. This evaluation was carried out on two testbeds: a local university testbed, and on the FABRIC testbed.

Through its *programmability*, the switch supports flexible network management, since the switch can execute a program on each packet that is forwarded by this switch. And through *attestation*, this switch protects against bugs in the program, misconfiguration, and subversion by an adversary who can control the program on the switch. Attestation provides an important primitive on which to build more reliable and secure networked systems. The evaluation of this switch in a physical testbed checks for correct behavior and measures the throughput of the switch implementation.

## 1 Introduction

The P4 language [3] is used to write simple but effective packet-processors on smart NICs and programmable switches. Programmability provides flexibility to network management through the customization of NIC and switch behavior. In addition to flexibility, however, programmability also carries risks: bugs in the program might undermine the reliability or security of the network, and the privacy of users.

Recently, the use of *Remote Attestation* (RA) was suggested to mitigate these risks [4]. An *attestation-capable* network element produces evidence about the program that processes each packet. This allows the operator and users of a programmable network to check the integrity of the network.

This paper describes the testbed evaluation of the first implementation of an attestation-capable, programmable switch. Our switch is open-sourced online.[1] This implementation extends BMv2—the reference P4 software switch [1]—to generate evidence of its trustworthiness and to expose that evidence to be remotely appraised. This switch is designed to preserve *integrity* of this evidence: it prevents the P4 program from manipulating this evidence.

The evaluation of this switch in a physical testbed checks for correct behavior and measures the throughput of the switch implementation. This evaluation was carried out on a local university testbed and on the FABRIC testbed.

[1] https://github.com/awolosewicz/bmv2-remote-attestation

## 2 Switch Architecture

The attestation-capable switch modifies the reference P4 software switch [1] to generate digests about three pieces of information: (1) the dataplane registers (which hold persistent state between packets), (2) look-up tables, and (3) the P4 program. Whenever any of these is updated, a dedicated, read-only register in the switch is updated with the digest.

The contents of these read-only registers are then embedded into packets that are processed by the switch. Thus each packet carries evidence of how it was processed.

Fig. 1 shows how the RA data is embedded into packets that traverse the packet-processing pipeline. The RA data is generated outside this pipeline—a new checksum is generated whenever the relevant state is changed. In this prototype, we use a custom hop-by-hop (HBH) IPv6 extension to carry the evidence. We use HPH because of its flexibility and the ability of hosts to easily parse the packet even if they do not recognize the structure. This functionality could be extended to other protocols with some small adjustments, such as smaller checksums to fit the IPv4 options field. Critically, all RA functions take place outside the scope of the program running on the switch. The P4 program cannot overwrite these registers; and trying to find suitable hash collision will be challenging for an adversary.

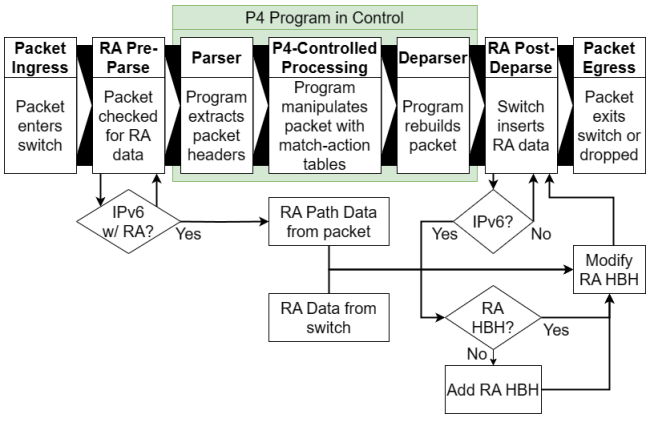


**Figure 1: Packet Processing in an Attesting Switch.**

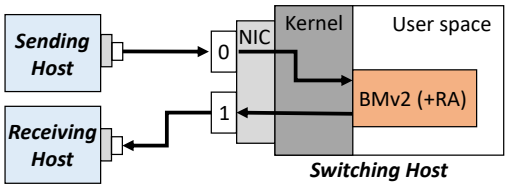


**Figure 2: Testbed setup used to evaluate correctness and throughput. The Switching Host uses a 2-port NIC—its ports are labelled "0" and "1" in this figure.**

## 3 Prototype Validation

The initial testing of the prototype focused on ensuring that the checksums were ① generated when their respective state elements were updated, ② consistent with unique states, ③ readable by the control plane and program, ④ correctly inserted into transiting packets, that ⑤ any attempts by a P4 program to place false data in the RA header are replaced with authentic data, and that ⑥ the switch's insertion of RA data did not corrupt the underlying payload.

## 4 Local Testbed Setup and Evaluation

We use a physical testbed setup as shown in Fig. 2. The attestation-capable BMv2 switch (System Under Test, or SUT) is programmable and it is configured to execute a simple router P4 program that handles IPv6 and IPv4 routing of packets across subnets. The receiver and sender are given IPv6 addresses on different subnets and are directly connected to the SUT on 2 different ports. The sender's and receiver's routing table and neighbor table are configured to be consistent with the above setup.

### 4.1 Evaluation and Workloads

The topology shown in Fig. 2 was used to check for both correctness and throughput. To check *correctness*, the traffic workload consists of ICMPv6 echo requests. To check *throughput*, the traffic workload consists of UDP packets.

*Validation:* We tested the aspects as mentioned in Section 3. Aspects ①–③ were tested with the control plane CLI and tracking state change when adding rules to the P4 program table. Aspects ④ and ⑥ were tested by using an ICMPv6 echo request-response challenge from sender to receiver via the BMv2+RA switch. Aspect ⑤ was tested by running a P4 program that corrupts the attestation. We confirmed that, on packet exit from the switch, the corrupted hashes are overwritten with the correct hash values.

*Performance:* For measuring throughput we rely on iperf3 statistics. The sender generates IPv6/UDP packets of a single flow and payload size of 1362 bytes. The packets are sent at bitrates beginning at 100 Mbps and then increased in multiples of 2 until loss is detected, then gradually decreased until no loss is detected. We then run the experiment at this bitrate for 10 seconds. In these test cases, we have enabled NIC hardware offloads, such as Rx/Tx checksums. With offloading disabled, throughput reaches 560 Mbps.

For the SUT, we use a tc-ebpf program hooked to a clsact qdisc on the ingress and egress ports. The tc layer is at the bottom of the generic network stack and has visibility to all packets. The test is run 8 times.

We measure the performance difference between BMv2+RA and base BMv2 when sending IPv6 UDP packets, as the RA switch is actively involved in creating/modifying a HBH extension header for these payloads.

We observe no drop in performance for zero-packet loss from our test runs, even when the attestation evidence is embedded into each packet that exits the switch. Both switches perform at 270 Mbps when offloading is enabled. There is, however, a bandwidth overhead since the HBH extensions add another 104 bytes to the packet length. So in our test case, the overhead is 104/1362 i.e, 7.6% per IPv6 packet.

## 5 FABRIC Testbed Setup and Evaluation

The setup on FABRIC was similar to that described in Section 4 for the local testbed. On FABRIC we allocated three nodes—one each for the sending host, receiving host, and SUT. These nodes were configured to have IPv6 addresses, consistent with the configuration used in the local testbed.

### 5.1 Evaluation and Workloads

The workloads used on FABRIC were similar to those used in the local testbed (§4.1). The only difference is that the sender and receiver are now Ubuntu VMs that are connected to the VM that runs the switch, through L2 Site-to-Site links, which is an abstraction provided by FABRIC. The sender and receiver VMs have 2 cores for computation, and the switch VM is assigned 8 cores. We successfully tested the validity aspects as mentioned in §3. For performance, we relied entirely on iperf3 (version 3.7) for throughput measurements. We sent UDP packets of size 1362 bytes each, for a duration of 10 seconds between the two hosts via the RA switch. We measured 300Mbps throughput with no packet loss, after disabling all network card offloads, as recommended in a FABRIC reference lab notebook [2]. We are continuing to work on performance tuning on the FABRIC testbed.

*Acknowledgements.* We thank Mohammad Firas Sada and Sean Cummings for their help with the initial testbed setup. We thank Hyunsuk Bang for feedback on the switch design. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-19-C-0106. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funders.

## References


[1] 2022. The BMv2 Simple Switch target. https://github.com/p4lang/behavioral-model/blob/d52ac6257bb3a58606383d03b31ed89671504791/docs/simple_switch.md. Behavioral Model v2.

[2] 2023. Jupyter complex recipes from FABRIC. https://github.com/fabric-testbed/jupyter-examples/blob/main/fabric_examples/complex_recipes/p4_labs_bmv2/lab1_creating_a_slice_with_a_P4_switch.ipynb.

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (jul 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[4] Nik Sultana, Deborah Shands, and Vinod Yegneswaran. 2022. A Case for Remote Attestation in Programmable Dataplanes. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks* (Austin, Texas) *(HotNets '22)*. Association for Computing Machinery, New York, NY, USA, 122–129. https://doi.org/10.1145/3563766.3564100


## A Reproducibility

*Following the instructions in the Call For Papers, we are including an appendix on the reproducibility of this research.*

The BMv2+RA source code can be retrieved from the online repository[2] and the included README describes the necessary dependencies to install and how to compile the code. Once compiled, the included `ra-to-base.sh` script can be modified and run to install the target over an existing BMv2 installation. The script must be modified to include the paths that an existing BMv2 install has placed its files in. The script contains comments detailing its use.

Once installed, BMv2+RA replaces the normal BMv2 Simple Switch target, so any environments built to use that target like Hangar will function identically, using BMv2+RA instead of Simple Switch. The sole exception is that BMv2+RA assumes a modified v1model architecture, found in the repository as v1model.p4. By Default, ra-to-base.sh replaces the base v1model.p4 but some may choose to instead keep their existing file and copy the modified file as a new name. Whichever method is used, P4 files must be compiled using the modified v1model.p4 to run on BMv2+RA.

Once setup, aspects ①–③ are tested by initiating the switch with a Thrift port and in a separate console accessing the control plane by running `sswitch_CLI`. With the control plane open, standard BMv2 commands can be used to write to registers, modify tables, or swap programs, and the new `get_ra_data` command will show how such changes alter the switch's RA checksums. Using packet capture, aspect ④ and ⑥ can be verified by passing IPv6 traffic, such as ICMPv6 echo requests, through the switch instance and noting the correct addition of the RA header to the ICMPv6 packet structure. Finally, aspect ⑤ is demonstrated in a similar manner but with a P4 program that attempts to create an RA header with false data.

[2] https://github.com/awolosewicz/bmv2-remote-attestation