Building an Internet Router with P4Pi

Radostin Stoyanov
University of Oxford
radostin.stoyanov@eng.ox.ac.uk

Adam Wolnikowski*
Humatics
awolnikowski@humatics.com

Robert Soulé
Yale University
robert.soule@yale.edu

Sándor Laki
Eötvös Loránd University
lakis@inf.elte.hu

Noa Zilberman
University of Oxford
noa.zilberman@eng.ox.ac.uk

ABSTRACT

Building an Internet Router is a popular, hands-on project used to teach computer networks. However, there is currently no hardware target that allows students to develop the project in P4 without incurring significant cost or encountering FPGA knowledge barriers. This paper presents P4Pi as a target for the Building an Internet Router project. P4Pi is a platform for developing, testing, and evaluating P4 programs on a Raspberry Pi device. We describe the architecture of the router project on P4Pi, and discuss the practical aspects of running it as a class project. The P4Pi-based router project is low-cost and easy to adopt, enabling students to focus on their P4 programming skills and to evaluate their designs on a physical target through interoperability tests with their colleagues.

KEYWORDS

Network Education, P4, Programmable Switches

1 INTRODUCTION

Twenty years ago, computer networking courses were very popular. As the Internet gained widespread adoption by household users, students became increasingly curious to understand how it functioned. However, for a generation of students weaned on omnipresent network connectivity, the novelty of networking topics has worn off. Today, without the visual effects that make topics such as graphics or robotics popular, many students view networking courses as a blasé alphabet soup of protocol acronyms.

To make things worse, while the contents of computer networks courses have significantly evolved over the years, the practical experiences of students have not appreciably changed. Many computer networking courses focus on users of networks, with hands-on exercises based on socket programming. This is, of course, important—even necessary—for students to learn. But, we argue that such exercises fail to foster a level of excitement in students for *understanding how networks work.*

To address this problem, Nick McKeown at Stanford University began offering a course almost 15 years ago, called CS344 *Build an Internet Router* [3], in which students used the open hardware NetFPGA platform [9] to develop a complete IPv4 router over the duration of a semester. This project allowed students to gain hands-on experience with development on a real hardware target, and to evaluate their design by testing the interoperability between different students' design. Since the initial offering, variations of this course have been taught at several universities.

While this course and its project were successful at achieving the pedagogical goal of teaching students about networking hardware architecture and design, it is a difficult course to replicate. With a current cost of around $1500 USD per board, the NetFPGA hardware can be expensive. Moreover, early iterations of the course required more than computer networking knowledge from students (e.g., familiarity with Verilog or VHDL, the general process of designing RTL-based logic and the associated verification process). It can also be very time consuming for teaching assistants and faculty to support students using FPGA devices when those students have no prior FPGA experience.

The recently released P4Pi platform [7], developed by the P4 Education Working group, offers an attractive target for network education. With P4Pi, users can implement a network data plane in the P4 language, and deploy the data plane in a software switch running on a Raspberry Pi. Because Raspberry Pi boards are relatively inexpensive, with a price tag of less than many textbooks (under $100), it is feasible for every student in the class to have their own device. Moreover, because P4 is a small, high-level language, students can gain basic proficiency in P4 programming after only one or two lectures.

A prior publication [7] described the technical aspects of P4Pi, focusing on using P4 with T4P4S [16, 17] on the device. This paper extends the previous work with a detailed description of an implementation of the *Build an Internet Router* project [3] on P4Pi. We feel that sharing this description is valuable to the P4/networking community for several reasons. First, we aim to provide sufficient details of the project and implementation that networking instructors would feel

*Adam Wolnikowski worked on the project while at Yale University.

comfortable adopting the project for use in their courses. To further support this goal, we have made a skeleton of the project publicly available on GitHub [15]. Second, in porting the project to P4Pi, we implemented a number of control plane utilities that are worth describing, as they are useful for general users of the P4Pi platform, beyond the specifics of this particular project. Third, this paper describes a reasonably complex use case for P4Pi involving multiple hardware devices. This use-case is far more complicated than any of the projects described in the prior publication [7], demonstrating the feasibility of P4Pi as a platform for realistic networking projects. We hope that such an exemplar will help grow the open-source community around the platform.

The rest of this paper is organized as follows: §2 provides the background to the *Build an Internet Router* project, explains its components, and limitations to current implementations. §3 provides background on P4Pi and its architecture. §4 provides high level details of the P4Pi-based project, while sections §5 and §6 cover the data and control plane implementation. §7 explains how the project is run on P4Pi. §8 focuses on the evaluation by students. We discuss some insights and limitations in §9 and conclude in §10.

2 BUILDING AN INTERNET ROUTER

Build an Internet Router is intended for a project based course. Students are assigned a set of deliverables, functioning as milestones toward the final goal of the course: building an Internet router. Typically, students don't start from scratch, but are provided with some infrastructure code, e.g., the code of the NetFPGA interfaces in the original project. In addition, students are required to write a design document containing diagrams, pseudocode, flow charts, or whatever else might be helpful to explain their key design decisions.

The project requires developing both the data plane and the control plane components of the switch. The original versions of the project used Verilog for the data plane implementation [3, 10], while more recent incarnations used P4-NetFPGA [4] or bmv2 [5, 6, 13]. The control plane was originally written in C, and later versions used Python on top of the Scapy packet processing library [1, 2].

As part of the project, four protocols need to be supported: IP (typically IPv4), ARP, ICMP and PW-OSPF. PW-OSPF (Pee-Wee OSPF) [14] is a simplified link state routing protocol based on OSPFv2 (RFC 1247). The students need to familiarize themselves with the specifications as part of the project.

While the original project had great pedagogical benefits, it was a difficult course to manage. The early incarnations of the project (e.g., CS344 at Stanford, P33 at Cambridge) required that students work in teams of two, one with an expertise in Verilog and FPGA design for the data plane implementation, and the other with knowledge in C (or later Python) for the control plane. More recent versions of the project (e.g., CS344 at Stanford) substituted Verilog for P4 using P4-NetFPGA [5], but still required as a knowledge component of FPGA design. Having FPGA design as a prerequisite raised the bar for course entry. Consequently, and for the lack of other alternatives, some versions of the project moved to a software-switch model (e.g., Yale CPSC 435/535), losing the realistic component of the hands-on experience.

While NetFPGA has been a key target for the prototyping of network devices in research for over a decade, running a lab for only 20 students, working in pairs, requires ten servers, each equipped with the platform and high performance NIC. The cost of such a setup is over $10K, does not scale to larger classes, and requires maintenance of the platforms.

3 P4PI

P4Pi [7] is a low-cost, open-source platform for teaching and research, running on top of Raspberry Pi devices[1]. The cost of Raspberry Pi boards (under $100) means that every student, or pair of students, in class can use their own device. It was developed by the P4 Education Working Group in order to make P4 knowledge more widely accessible. The release includes the tools required to use P4Pi in class and at home, such as tutorials, sample code, tools and community forums.

Because P4Pi is based on Raspberry Pi, it lends itself to different compilers and P4 architectures. Laki et al. [7] describe an architecture using the T4P4S [16, 17] compiler and a DPDK-based software switch, with a separation of data plane and control plane. In this paper, we describe a somewhat different implementation (§4 to §6), that uses the bmv2 *simple_switch_grpc* target.

4 P4PI-BASED ROUTER ARCHITECTURE

Like previous versions of the *Build an Internet Router* project, the P4Pi-based version is designed as a semester-long project. Students are required to design the data plane in P4, and the control plane in Python, including support for PW-OSPF [14]. Students' projects must pass interoperability tests, where routers designed by different teams are connected together. The students receive starter code, similar to previous incarnations of this project [3, 5, 10, 13], including a minimal P4 program for the data plane, and a minimal Python program for the control plane.

The architecture of the P4Pi-based router is illustrated in Figure 1. The data plane, implemented as a software switch over bmv2, is marked in green. It connects to the gRPC control plane (orange), which connects to the P4Runtime gRPC socket and sends/receives packets through the CPU

[1] Currently tested on Raspberry Pi 4 model B.

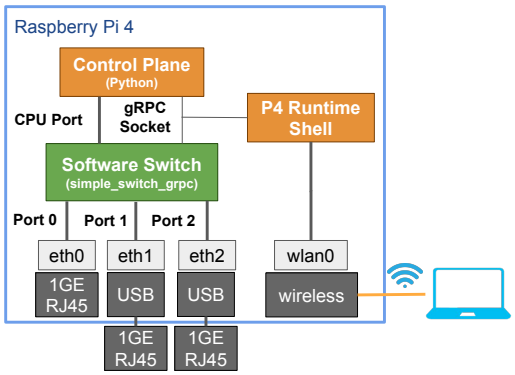

Figure 1: P4Pi Router Architecture

port of the switch. The P4Runtime Shell can be used to access, monitor, and query the data plane.

The data plane has multiple ports: the physical gigabit Ethernet port of the Raspberry Pi, and one or more USB ports converted into Ethernet ports with USB-to-Ethernet adapters. In order to test a network of 4 P4Pi platforms connected in a mesh topology, 3 physical ports (1×Ethernet+2×USB-to-Ethernet) are required.

Students can connect to P4Pi using the wireless network, and remotely log into the platform using *ssh*. The wireless port is not used by the P4 program, and works with the operating system, as in a standard Raspberry Pi deployment. Once logged in, students can access the P4Runtime Shell. Students can write their code on P4Pi, but it is typically more convenient to first develop on one's laptop or server, and then upload the code to P4Pi.

5 DATA PLANE IMPLEMENTATION

The P4Pi-based router is implemented using *simple_switch_grpc*, and can run over bmv2. Therefore, students can first develop and test in an emulation environment (e.g., Mininet [8]) before testing on a harder-to-debug hardware target. The project's logic is implemented in the ingress pipeline of the data plane, where the output port needs to be chosen.

The implementation of the parser must support multiple protocols: Ethernet (provided in the starter code), ARP, and IP. Our reference design supports both IPv4 and IPv6. While ICMP must be implemented as part of the project, ICMP processing is done in the control plane and only the IP header is processed in the data plane. We note that other implementations are possible.

The match-action pipeline consists of six match-action tables:

(1) A routing table using longest prefix match on destination IP address to associate an output port and next-hop IP address (a table for IPv4 and a table for IPv6).
(2) A local IP table using exact match on destination IP address to identify if the packet should be sent to the control plane, or associate an output port and next-hop IP address if it is already available in the table (a table for IPv4 and a table for IPv6).
(3) An ARP table that performs an exact match on the next-hop IP address and, if found, associates a corresponding MAC address.
(4) Layer 2 forwarding table that performs an exact match on destination MAC address and sends the packet to a specific port, a number of ports in a multicast group, or drops the packet.

All incoming ARP packets are sent to the CPU port, and all packets arriving from the CPU with a valid destination port (i.e., routed in the control plane) are sent to the designated destination port. The TTL of IP packets is decremented and packets are dropped if TTL=0. The next table is used to look up Local IP addresses. It is used to identify IP addresses that should be forwarded to the CPU, and if there is no match, the destination is looked up in the routing table. For outgoing packets, the next hop MAC address will be looked up in the ARP table, and updated in the Ethernet header.

If a packet is sent to the CPU, a header is added with metadata such as the source port. Incoming packets from the CPU need to be decapsulated, and information from the header (e.g., EtherType, destination port) is saved to the metadata bus. It is then used to set the destination port without matching on the tables.

The design holds several counters that can be queried, such as the number of ARP packets, IP packets and packets sent to the CPU.

The deparser implementation is straight forward, emitting all valid headers. On top of the Ethernet, IP and ARP headers, a bespoke header is added to packets sent to the CPU, as described above.

6 CONTROL PLANE IMPLEMENTATION

All the routing functionality is implemented in the control plane, following the classic separation of roles between the data and the control plane. The control plane is implemented in Python using the P4Runtime API [11] and Scapy [2]. Using P4Runtime, the controller can send/receive messages to/from the software switch over a gRPC connection. P4Runtime allows the controller to access the entities defined in a P4 program, such as tables and counters. Scapy is used to send/receive packets over the interface used as a CPU port. The control plane could be extended by sending and receiving packets over P4Runtime StreamChannel, which is a bidirectional stream that can be used for packet I/O, among other things.

Students are required to implement PW-OSPF [14], a simplified version of OSPFv2. The routing table is populated with

information received in HELLO packets, which are broadcasted periodically by the routers to all neighbors. In addition to HELLO packets, a router generates and processes link state update messages (LSU). These updates contain a router's view of the network and are triggered by changes to link status (e.g., addition or deletion of a router) or a timeout of HELLO packets from a neighbor router. Each router independently calculates the shortest path to every other router, using Djikstra's algorithm, and updates the data plane's forwarding table with the next hop accordingly.

The control plane handles more than just PW-OSPF packets. First, it handles ARP messages, and updates the ARP table in the data plane. The control plane can send ARP requests, and remove entries in the ARP table that have timed out. In addition, it queues packets that are pending ARP replies. Second, the control plane handles ICMP messages, which appear on the data plane as IP packets with a local IP address, and are therefore sent for processing to the control plane. It responds to ICMP echo requests, and generates ICMP host unreachable packets. Finally, the control plane handles all corrupted or otherwise incorrect IP packets, as well as any other packets addressed directly to the router.

7 RUNNING THE ROUTER ON P4PI

For a P4Pi-based router project, each student (or a pair of students) is assigned a P4Pi platform. Unlike the NetFPGA-based project, there is no need to install any toolchains; one can simply download the most recent P4Pi image.

The P4 and control plane code can be developed directly on the platform. However, we expect that most students would prefer to develop on their own laptop, with their favorite text editor, and transfer the files to P4Pi using, e.g., *scp*. Once the code is ready, students connect to P4Pi using *ssh*. Next, the network interfaces need to be set up. This means both physically connecting the Ethernet ports (USB-to-Ethernet) and Ethernet cables, as well as setting static IP addresses (using *ifconfig*). Note that no special setup is required for USB-to-Ethernet adaptors. Once connected, the operating system should automatically identify the Ethernet interfaces.

With the code deployed to the platform, and the hardware connected, it is time to run the router. While we expect future versions of the project to support different targets, we describe here the process with bmv2 *simple_switch_grpc* and P4Runtime on P4Pi[2].

To start the data plane, *simple_switch_grpc* is called with the compiled P4 program and the interface to port attachments as parameters. The control plane is started by calling the Python program written by the student, which also populates the tables. To view and monitor changes in the contents

[2]Note that we use bmv2 with a physical network, not an emulation environment.

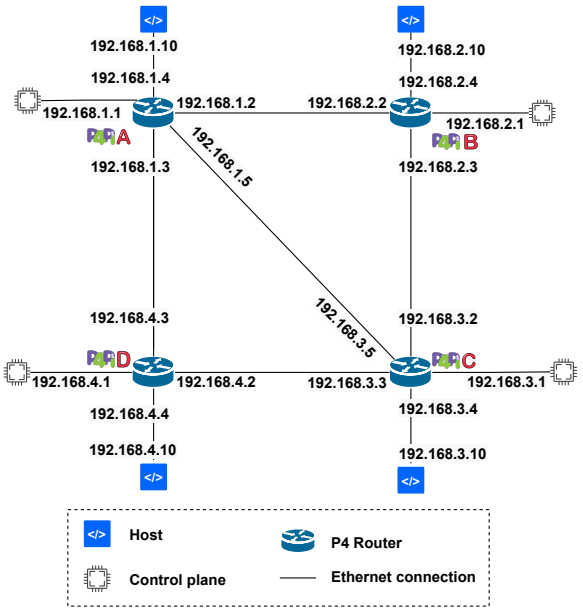

Figure 2: An example topology used to test the operation of multiple P4Pi Routers.

of tables and counters, a P4Runtime client such as P4Runtime Shell [12] can be used.

8 EVALUATION AND INTEROPERABILITY

Students can test their router in two modes: standalone or connected to a group of routers. Note that in standalone mode, the router is still part of a network, as the P4Pi platform is connected to one or more student laptops. In this mode, it is possible to test that a range of functions are acting correctly, including ARP, ICMP (e.g., using ping), and IPv4/IPv6. Additionally, one can use the following approach to test a router. First, create virtual Ethernet (veth) interface pairs, moving one end of each pair to another network namespace. Then, attach the interface that is visible in the root namespace to the switch and send traffic to it. Moreover, one can test that entries are added and deleted from the routing table using the P4Runtime Shell. Finally, a basic performance test can be done by running *iperf* between two laptops with P4Pi in the middle. By comparing the throughput using a direct connection or with P4Pi in the middle, it is possible to assess the limitations of the implementation.

To perform tests with multiple P4Pi devices, one can construct a network topology such as the one illustrated in Figure 2 using the 1GbE RJ-45 (Ethernet) port and USB-to-Ethernet adaptors. Each Ethernet interface is configured with a static IP address. Note that the same setup can be first developed and tested on Mininet, using the same compiler, target, P4 code, and control plane. This setup allows students to

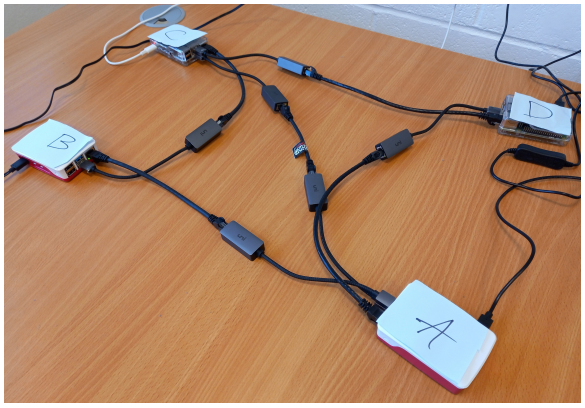

Figure 3: P4Pi Interoperability Setup.

```
P4Pi-A [192.168.1.1] Adjacency List:
[192.168.3.1]:
  -  ('192.168.2.1', '192.168.2.3', '255.255.255.0')
  -  ('192.168.4.1', '192.168.4.2', '255.255.255.0')

[192.168.1.1]:
  -  ('192.168.2.1', '192.168.2.2', '255.255.255.0')
  -  ('192.168.4.1', '192.168.4.3', '255.255.255.0')

[192.168.4.1]:
  -  ('192.168.3.1', '192.168.4.2', '255.255.255.0')
  -  ('192.168.1.1', '192.168.4.3', '255.255.255.0')

[192.168.2.1]:
  -  ('192.168.1.1', '192.168.2.2', '255.255.255.0')
  -  ('192.168.3.1', '192.168.2.3', '255.255.255.0')
```

Figure 4: P4Pi-A control plane adjacency list.

check interoperability between different designs and test the operation of PW-OSPF. Figure 3 shows an example, similar to Figure 2, using actual hardware.

One can confirm that PW-OSPF is implemented correctly by printing the adjacency list (computed using Djikstra's algorithm) for each router to the screen. The adjacency list contains all routers in the network as well as the subnets associated with each link. When a failure happens or a new switch is added, it is not sufficient for the router to find any possible route; it must be the shortest path. Figure 4 shows a screenshot of the adjacency list on P4Pi-A router, using the topology in Figure 2.

By using a hardware setup, students can test for unexpected behaviors which can lead to incorrect router operation. For example: *What happens when a link goes up and down within short time periods (i.e., within the duration of the periodic messages)?* or *What happens when a link that was connected to port 1 is moved to port 2?* These, as well as other "real world" problems are much easier to create and explore using P4Pi than in any software-based environment.

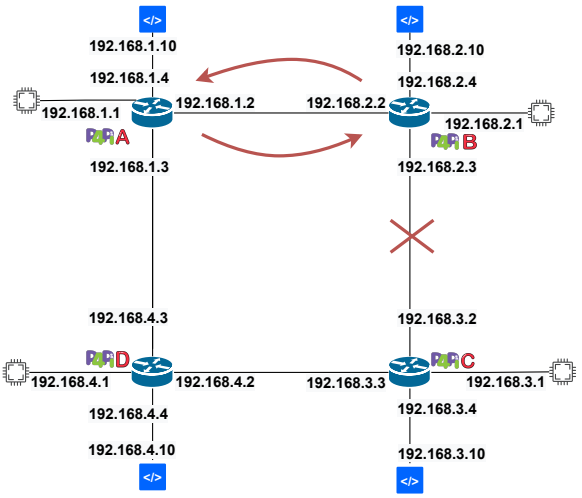

Figure 5: An example routing loop problem in operation of multiple P4Pi Routers connected in ring topology.

A common implementation pitfall that can be tested is the handling of "routing loops". An example of this problem is illustrated in Figure 5. Initially, P4Pi-A routes to P4Pi-C through P4Pi-B. When the link between P4Pi-B and P4Pi-C fails, P4Pi-B finds that the shortest path to P4Pi-C is through P4Pi-A. However, P4Pi-A still routes through P4Pi-B. Thus, the traffic for P4Pi-C arrives at either P4Pi-A or P4Pi-B and then loops between these two routers, at least for the periodic updates' duration, unless the implementation prevents the behavior. A routing loop can also be created as a destination becomes unreachable, and a good implementation would attend to that as well.

9 DISCUSSION

Engagement. We believe that using a P4Pi-based router will increase student engagement. Adding a hardware component makes the project feel "real" and enhances the experience. Furthermore, when a student takes the Raspberry Pi with them, the student feels a stronger connection to the platform, and they can experiment with using their switches at home.

Target. Future versions of this project may support different targets. These may be hardware targets (different versions of Raspberry Pi that run the same operating system), or data plane targets (different P4 compilers or architectures).

Realism. While we refer in this paper to P4Pi as a hardware target, the data plane is in fact implemented as a software switch. However, the goal of the router project is functionality rather than performance. Importantly, the hardware experience of students, connecting and disconnecting cables, resembles a switch ASIC.

Flexibility. Different academic programs run courses of varying length, from one condensed week to a 4 months long semester. It is possible to adapt the project to the length of the course by varying the project's scope. For example, implementing just the data plane or only the control plane. Educators may also choose to decrease or increase the complexity of the project, for example, by including more contents in the starter code, or by requiring more functionality in the data plane (e.g., ARP response).

10 CONCLUSION

Build an Internet Router is a powerful exercise for students learning computer networks. By implementing *Build an Internet Router* on P4Pi, students gain hands-on experience with a hardware target at low cost. P4Pi allows students to develop the project without knowledge of FPGA design, and without the overhead of complex hardware maintenance. Moreover, P4Pi allows students to create different interoperability scenarios easily and flexibly. P4Pi is an open source project, and all the materials described in the paper are available on GitHub [15].

ACKNOWLEDGMENTS

P4Pi is developed by the P4 Education Working Group, one of five working groups within the Open Networking Foundation (ONF) P4 Infra Project. We wish to thank the people who contributed to the prior versions of "Build an Internet Router", particularly Stephen Ibanez, Theo Jepsen, Andrew W. Moore, and Nick McKeown. We thank the Network Programming Initiative (NPI) for their generous support, which allows us to purchase and distribute Raspberry Pi boards to early adopters. This work was supported by the EPSRC Doctoral Training Partnership (DTP) [grant number EP/N509711/1].

REFERENCES

[1] Philippe Biondi. 2005. Scapy: explore the net with new eyes. *Technical report, Technical report, EADS Corporate Research Center* (2005).

[2] Philippe Biondi and Scapy community. 2008. Scapy. https://scapy.net/, Accessed: 2021-10-17.

[3] Glen Gibb, John W Lockwood, Jad Naous, Paul Hartke, and Nick McKeown. 2008. NetFPGA—an open platform for teaching how to build gigabit-rate network switches and routers. *IEEE Transactions on Education* 51, 3 (2008), 364–369.

[4] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. 2019. The P4->NetFPGA Workflow for Line-Rate Packet Processing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 1–9.

[5] Stephen Ibanez and Changhoon Kim. 2019. CS344 Stanford, Build an Internet Router.

[6] Changhoon Kim, Theo Jepsen, and Alex Mallery. 2021. CS344 Stanford, Build an Internet Router.

[7] Sándor Laki, Radostin Stoyanov, Dávid Kis, Robert Soulé, Péter Vörös, and Noa Zilberman. 2021. P4Pi: P4 on Raspberry Pi for Networking Education. *SIGCOMM Comput. Commun. Rev.* 51, 3 (2021). https://doi.org/10.1145/3477482.3477486

[8] Bob Lantz and Mininet Contributors. 2010. Mininet. http://mininet.org/, Accessed: 2021-10-17.

[9] John W Lockwood, Nick McKeown, Greg Watson, Glen Gibb, Paul Hartke, Jad Naous, Ramanan Raghuraman, and Jianying Luo. 2007. NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing. In *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. IEEE, 160–161.

[10] Andrew W. Moore. 2009. P33: Building an Internet Router. https://www.cl.cam.ac.uk/teaching/0910/P33/, Accessed: 2021-10-20.

[11] P4.org. 2018. P4Runtime control-plane API. url-https://github.com/p4lang/p4runtime/, Accessed: 2021-10-20.

[12] P4.org. 2019. An interactive Python shell for P4Runtime. https://github.com/p4lang/p4runtime-shell/, Accessed: 2021-10-21.

[13] Robert Soulé. 2021. CPSC 435/535, Building an Internet Router.

[14] Stanford University. 2008. Pee-Wee OSPF Protocol Details. https://www.cl.cam.ac.uk/teaching/0910/P33/documentation/pwospf/index.html. Originally at http://yuba.stanford.edu/cs344_public/docs/pwospf_ref.txt, Accessed: 2021-10-20.

[15] Radostin Stoyanov, Sándor Laki, Dávid Kis, Robert Soulé, Péter Vörös, and Noa Zilberman. 2021. P4Pi Repository. https://github.com/p4lang/p4pi.

[16] Péter Vörös, Dániel Horpácsi, Róbert Kitlei, Dániel Leskó, Máté Tejfel, and Sándor Laki. 2018. T4P4S: A Target-Independent Compiler for Protocol-Independent Packet Processors. In *IEEE International Conference on High Performance Switching and Routing*. IEEE, 1–8.

[17] Péter Vörös, Dániel Horpácsi, Róbert Kitlei, Dániel Leskó, Máté Tejfel, and Sándor Laki. 2018. T4P4S source. https://github.com/P4ELTE/t4p4s, Accessed: 2021-10-17.