Exploitation of P4 Programmable Switch Networks

by

Mees Frensel
Supervisors: Fernando Kuipers, Chenxing Ji

A Paper
Submitted to EEMCS faculty
Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 19, 2022

Abstract

P4 programmable data-planes provide operators with a flexible method to set up data-plane forwarding logic. To deploy networks with confidence, a switch's forwarding logic should correspond with its intended behavior. Programs loaded onto programmable data-planes don't necessarily go through as much testing as traditional fixed-function devices from large manufacturers. Security is therefore of utmost importance.

The main question this research attempts to answer, is whether a single compromised P4 switch can corrupt the entire (P4) network. In this scenario the attacker already has access to the compromised switch, and the assumption is made that all devices blindly trust each other. Two load balancing schemes are investigated, Clove-ECN and HULA. The former performs load balancing on the hosts, and results show that switches can transparently influence traffic flow by manipulating the ECN bits. The latter is designed for implementation on the data-plane, e.g. using P4, and we can conclude that HULA is susceptible to attacks by spoofing probe packets with false data.

1 Introduction

Data-plane programmability allows operators to program the forwarding layer of network devices. Network switches conceptually operate on two layers: the data-plane and the control-plane. The data-plane is responsible for forwarding packets to the right port(s), often using match action tables. The control-plane fills the match action tables with forwarding rules. In high performance switches, the data-plane functionality (or 'forwarding logic') is traditionally handled by an Application-Specific Integrated Circuit (ASIC), while the control-plane operates on a separate generic CPU.

The switch ASIC as well as the control-plane logic is developed and manufactured by the switch manufacturer, who thus controls which protocols get implemented. It takes more time and requires more expensive hardware to use custom protocols. Software Defined Networking (SDN) overcomes these drawbacks. SDN physically separates the control-plane from the data-plane, which often are on the same chip in low to medium performance switches. The control plane functionality is given to the end user while the data-plane is still on a fixed-function ASIC [1].

SDN architectures like OpenFlow have to support all protocols end users may want on both the data-plane and the control-plane, which complicates configuration [2]. In the last decade, data-plane programmability has started to gain traction, which promises to decrease this complexity and increase flexibility and ease of use. Using the programming language P4 [3], network operators have the freedom to implement whatever protocol or other data-plane functionality they need on the device. This brings many advantages over the old restricted bottom-up approach, e.g. flexibility, but also using system resources and power only for tasks and protocols that are used in the network.

In closed networks like data centers, the network operators may not consider security to be of the highest priority. It can be assumed that the switches programmed by yourself will not attack the other switches. In this research and the accompanying experiments the assumption is therefore made that these devices blindly trust each other.

The main question this research attempts to answer, is whether a single compromised P4 switch can corrupt the entire (P4) network. Corruption can mean a disruption of data flow to and from the compromised switch or between two unrelated switches, modifying another switch's data tables, etc. A single compromised switch can influence all data being sent from and to it, because it processes packets to forward them to the correct location. Thus, the question that remains is to which extent data (flow) can be corrupted.

To aid in finding the answer, some subquestions are formulated.

1. Can the attacker obtain access to the rest of the switches in the network? This could

possibly mean that programmable switches are unsafe to use in a network that is not physically secured.

2. To what extent can the attacker corrupt the behavior of the network traffic? To this end, the attacker does not necessarily need access to other switches.
3. Lastly, what defense mechanism can be used against a single corrupted programmable switch? For end users of programmable data-plane devices, this sub-question is perhaps the most interesting.

Some research has already been done on security of P4 devices [4]. The capabilities of an exploited device are explored by Black & Scott-Hayward [5]. Their paper and most other previous research mostly focuses on the security of a switch device itself, while this research looks into how a compromised switch can influence the network it is on.

This research paper begins with the background and methodology in sections 2 and 3, where algorithms and tools are explained, and related work is discussed. In sections 4 and 5, we introduce ECN poisoning, an exploit of load balancing algorithms using ECN for congestion feedback, and an exploit of HULA, a load balancing architecture designed for programmable data-plane networks. From section 6 onwards, results, discussion and conclusions are discussed.

2 Background

Data-plane programmability is a relatively new concept. Security thereof is still in its infancy. Some techniques are used to try and break the security of network protocols on P4 switches. P4 is discussed in the introduction; the techniques, protocols, and related tools are introduced here.

ECN

Explicit Congestion Notification (ECN) is a mechanism to signal network congestion without resorting to dropping packets immediately. It was defined as a part of the IPv4 header in 2001 [6] and web server support is commonplace nowadays [7]. ECN is used to signal from end to end that the traffic route is congested, to which a host can react by lowering the transmission rate. For example, switches using active queue management may signal congestion when the queue size is over 20 packets, and start dropping packets when the queue is bigger than 50. The data sender might then respond by lowering the transmission rate.

ECN can also be used by switches to determine the best path from A to B. For example, if there are two paths between hosts, and the path currently in use is getting congested, the switch can decide to use the other, free path. This research aims to explore whether this functionality can be abused to force most or all traffic along one of multiple paths.

This is a security and specifically an availability issue, since this can be used to overload parts of the network without signaling congestion from end to end. Furthermore, if a network is vulnerable to such a simple attack, this probably means that the assumption that all devices in the network trust each other is too broad. The devices can in turn no longer assume other devices are trustworthy.

HULA

The Hop-by-hop Utilization-aware Load-balancing Architecture (HULA) is a data-plane load-balancing algorithm designed for switches with programmable data-planes [8]. The fact that it is designed for programmable data-plane networks makes it an attractive thing to study. HULA is aimed mostly at data center networks, but can be used in any topology that has a notion of upstream and downstream links.

Data center load balancing using HULA provides a number of advantages. It can quickly adapt to volatile workloads and is more effective and scalable than existing load balancing schemes [8]. Also, it doesn't require any changes to the end hosts. A downside is the requirement of a network of programmable switches.

Tools and systems

Using Mininet, it is easy to simulate a network of switches and hosts running real kernels and software [9]. Mininet is used for simulating networks, and is focused on experimenting with large Software Defined Networks. In our experiments, Mininet simulates the hosts and the P4 programmable switches.

Behavioral Model version 2 (bmv2) is the reference software implementation of a P4-programmable network switch [10], and is used as software switch within Mininet. Any valid P4 program can be loaded onto bmv2 switches. Their performance is lacking compared to hardware implementations like the Tofino chips [11]. However, its functionality is near-complete and bmv2 is the standard in research and learning communities. The switches and topologies used in this research are simulated in Mininet using bmv2.

3 Methodology

Influencing devices in a network without being noticed is an important aspect of cyber attacks. If an attack is in plain sight, e.g. a denial of service (DoS) attack from one single source, the target can simply block traffic from the attacker and the attack is mitigated. However, a Distributed DoS attack's source is harder to pin-point, since malicious traffic is coming from multiple sources and is difficult to distinguish from genuine network traffic. It is therefore a goal that the attack uses existing mechanisms in a way that is similar to their normal usage.

We conducted experiments based on two different network functions to investigate the security of networks using data-plane programmable devices. The first is an attempt to exploit load balancing algorithms that use Explicit Congestion Notification (ECN) to balance network traffic over multiple links. Clove-ECN is an example of such an algorithm, working at the host/server level [12]. It influences packet headers in such a way that a switch using Equal-Cost Multi-Path routing chooses the least congested path. The hosts determine this by inspecting ECN data, which a P4 programmable switch can change, and a corrupt switch could then use to influence traffic flow.

While Clove-ECN does not run on the data-plane, or even on the network switches, it is interesting to see what impact a corrupt P4 switch can have. It is not unthinkable that in the near future some new load balancing scheme is presented that runs on the data-plane and uses ECN for congestion control. Also, the attack can be performed by anyone with a way to modify packet headers.

Along with this exploit, HULA is investigated. Since it is designed for programmable data-planes, it is an especially interesting concept to explore. It is not possible to use ECN to influence the HULA load balancing, as it completely depends on its own protocol sending probe packets. Instead, to change the behavior of HULA, the attacker changes the contents of these probe packets, which signal link utilization levels.

A P4 implementation of HULA is available on GitHub [13]. It contains all features and requirements presented in the paper, except for some probe optimizations in multicast groups. These optimizations do not add to HULA's security, nor are they necessary to run it. Some basic networking features are missing as well, and this makes it impossible, for example, to do `iperf` tests. First, we verified HULA's basic load balancing functionality, which works as expected. Results are shown and discussed in section 6. A 16-host fat-tree topology is used, which is shown in Figure 2.

The experiment setup is as follows: network traffic is created between two hosts that do not belong to the same pod (hosts 1–4, 5–8, etc.). In our specific setting, `h1` repeatedly sends TCP packets to `h7`. Then, we investigate the behavior at `s100`. Its job is to forward packets to either `s202` or `s203`, depending on path utilization. By sniffing packets on its upstream interfaces, we can determine how many packets are leaving towards `h7`, and what the distribution over the aggregation switches is.

The essence of the exploit presented in this paper is discussed in section 5. For our experiments, `s202` is the compromised switch that attempts to manipulate HULA to send more packets along paths that contain it. This is achieved by loading a different version of the P4 implementation, that halves the actual path utilization.

Specifically, the ingress stage begins with updating the utilization statistics of the currently used ingress port. This happens for every packet. Utilization is based on an exponential moving weight average using the incoming packet's size as input. The malicious implementation does not simply use the packet's size but divides it by 2. Over time, this lead to an 'average utilization‘ value that is precisely 50% of the actual port utilization. This should lead to a higher-than-expected usage of the paths across the corrupt switch.

4 ECN poisoning

A novel exploit coined ECN poisoning is presented. The exploit abuses the Explicit Congestion Notification (ECN) bits of the IPv4 header to steer more packets towards itself instead of other switches. The primary goal of the exploit is to influence network traffic in any way, while remaining unseen, but especially forcing more traffic to pass through our compromised switch. An attacker could use this to overload the network, inspect more packets than otherwise would pass through the switch, etc.

The basic load balancer uses an algorithm similar to Clove-ECN. A combination of Equal-Cost Multi-Path (ECMP) and ECN is used. When none of the equal-cost paths are congested, the load balancer uses ECMP to make decisions on which path to take, i.e. the 5-tuple hash is used [14]. When one or more equal-cost paths are congested (based on ECN), they are no longer equal: see Figure 2. The number of hops is the same, but latency on the congested path will increase. Therefore, the ECMP algorithm is adjusted to give congested paths a lower priority than free paths. This basic algorithm is different from Clove-ECN in that it runs on the switches instead of the hosts.

Clove-ECN can be used in networks with traditional non-programmable switches that support ECN and ECMP, with the load balancing algorithm running in the hypervisor on end hosts. Therefore, it is not necessary that a data center using Clove-ECN for load balancing has switches with programmable data-planes. Nevertheless, simulating this network with P4 programmable switches serves as a good proof-of-concept.

Exploiting an ECN-based load balancing algorithm leans on the simple idea of resetting ECN bits. Consider the simple version of the fat tree topology shown in Figure 1, which is actually a single *pod*. There are two paths from h1 to h2, one through each core switch. ECN-based load balancing algorithms like Clove-ECN will use ECN information to better distribute traffic across the network. This mostly entails throttling traffic across congested paths, and preferring less congested paths.

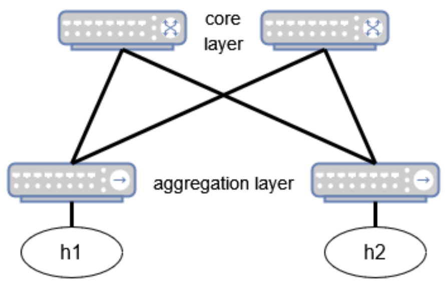

Figure 1: A very simple tree topology with only two hosts. If one of two core switches would discard ECN information, an ECN-based load balancing algorithm would prefer paths via that switch.

When one core switch is compromised, the switch could be programmed in such a way that it resets the ECN bits to `00`, effectively discarding the ECN information. The load balancing algorithm interprets this as a congestion-free path and prefers it when other path are congested. Results show that when preferably uncongested links are used, discarding ECN information does lead to a higher usage of the path along the compromised switch. Further research will have to explore the security impact of such an attack.

5 HULA exploitation

Exploiting HULA to influence network traffic has not been written about before. We propose an idea that disrupts HULA's functioning by manipulating path utilization values. The exploit has been tested on a 16-host fat-tree topology (see Figure 2) and successfully routes more traffic across paths that contain the compromised switch.

HULA works on a hop-by-hop basis, where every switch in the network keeps track of the best next hop to each top of rack (ToR) switch in a match action table. This *bestHop* table is updated frequently with data about the best next hop to a ToR switch from probe packets sent by the ToR switches. Probe packets are

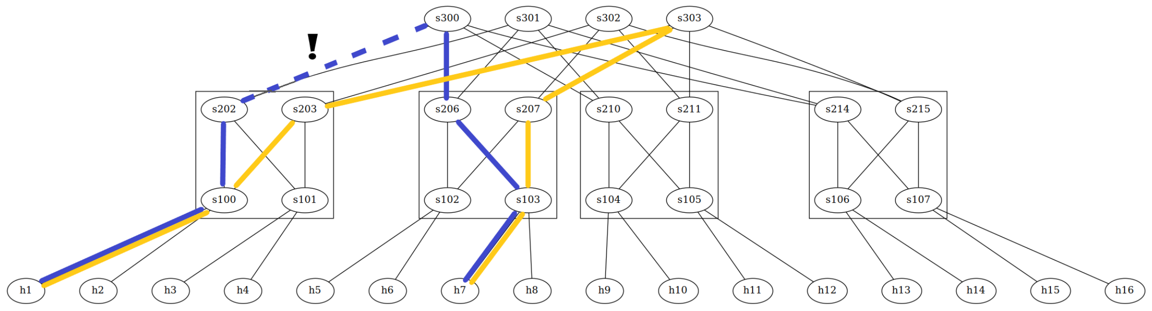

Figure 2: Fat tree topology [13]. The top row contains the core switches, below that the aggregation switches, below that the edge switches, connecting to the hosts at the bottom. Two equal-cost paths from host h1 to host h7 in a fat tree topology. When one of the paths gets congested, the load balancing algorithm will adjust to send more traffic over the other path(s).

sent with an interval which is in the order of magnitude of the ToR to ToR round-trip-time.

The main idea of HULA exploitation is to modify the probe packet header data to present our compromised switch as a very good 'next hop' option to other switches in the network. The HULA header contains two fields: *torID* (24 bits), the identifier of the ToR switch that generated the packet, and *minUtil* (8 bits) which is the utilization of the path from the sender to the ToR switch. When a switch receives a HULA packet from another switch, it can update the *bestHop* table with the switch's mac-address, port, or other identifier.

To influence network traffic without being noticed, it is important to not make the minimum path utilization values so low they are unrealistic. A very robust implementation of HULA can put measures in place to detect suspicious values from other switches. An adversarial can decrease the utilization values by a certain percentage. On the other hand, switches can't make any assumptions about the maximum possible utilization of a path because HULA supports asymmetric networks. Therefore, as long as actual utilization does not approach 100%, the compromised switch can pretend to have a lower utilization than it actually has, without impacting performance. This results in the desired effect of sending more data past the compromised switch.

There are several ways to achieve the same general result. The first method is to simply report lower utilization values right at the start of packet processing, while otherwise running the same HULA implementation. It is this approach that was experimented with and produced promising results. Another method is to do honest bookkeeping within the compromised switch, but under-report the minimum path utilization when forwarding probe packets.

6 Results

Due to difficulties with running P4 code using bmv2 and Mininet, there was no time to test the proposed exploits in a realistic scenario. These were not issues with P4; the switches did not get the correct ARP and other routing information via Mininet. It appears this was specific to the author's laptop. Nonetheless, some results were achieved, which are presented here.

Using ECN to signal congestion works reliably, and combining it with a naive ECMP implementation does in fact balance traffic across multiple links. Since connections are not necessarily routed across the same path in the naive implementation, packet reordering may occur at the receiving end. This is undesirable but it works for a proof of concept. Dropping ECN data on one of the switches in the network did however not successfully influence load balancing.

Manipulating path utilization data in the HULA architecture did successfully route more traffic towards the compromised switch. Throughput during normal behavior is shown in Figure 3 and shows that switch `s100` evenly balances traffic to `s202` and `s203`. When the switch is compromised, the reported bandwidth is 50% less than the actual throughput. The consequence is that throughput through both switches is balanced until equal. This results in a distribution of a third through the normal switch, and two thirds of the traffic through the compromised switch. This is graphed in Figure 4. We can conclude that it is possible and quite easy to influence HULA load balancing.

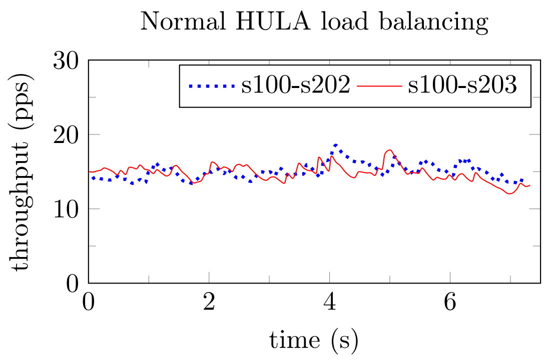

Figure 3: Traffic from h1 to h7 passing through s100. HULA load balances packets across the two next available hops, s202 and s203.

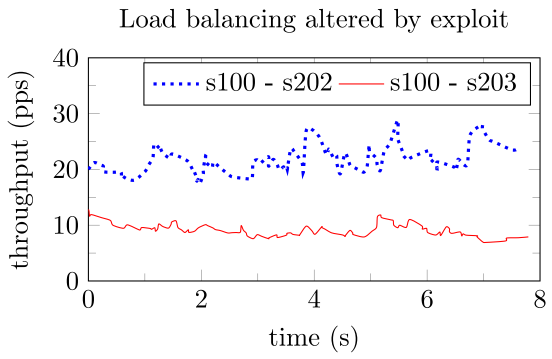

Figure 4: Traffic from h1 to h7 passing through s100. HULA load balances packets across the two next available hops, but gets wrong path utilization values and thus routes more traffic through the compromised switch s202.

7 Responsible Research

When doing research on security, inevitably the question arises whether a person could use your results for an evil purpose. While the experiments presented in this paper can be used in a real-life scenario, it is unlikely that they are truly applicable to such a situation. Especially the combination of having to gain access to a switch, and the assumption that devices completely trust each other, is unrealistic. Any system administrator worth their salt will have security policies in place to prevent unauthorized access. Therefore, this research does not raise ethical issues.

When reading through related work, one thing that caught attention was the possibility to reproduce methods, or lack thereof. Some papers explain for example a new algorithm in detail, but do not provide directions for a P4 implementation. In this paper a middle way was sought, providing the general steps to reproduce results, while noting some specific relevant things.

8 Discussion

Previous works ([5; 15] among others) investigate the impact of exploits in the P4 ecosystem, from P4 code to the interaction between the data-plane and the control-plane. These works explore the security on single network devices, but leave networks of programmable data-plane switches out of the picture. A lot of research on P4 networks and security involve securing a network using P4 [16]. This research focuses on the security of a network of (P4) programmable switches, specifically data center networks using different forms of load balancing. It is incomparable to previous works known to the author.

In the introduction, one core assumption was mentioned that the experiments and results presented here rely on: the hosts and switches trust all other hosts and switches in the network. In a data center environment, this is a realistic assumption to make as data centers often have very strong (physical) security measures in place. The scenario with a compromised switch is therefore sensitive to the exploits presented in this research. The security measures data centers have in place do mean that it is impossible or at least very hard to get access to and compromise such a switch.

A less secure environment (e.g. offices) would be susceptible to the attacks presented in this paper. However, it is unlikely that such an environment does any load balancing in an exposed part of the network, let alone use HULA for that purpose. It raises the question of how useful the attacks are. Nevertheless, they serve as a proof-of-concept and can be used as a starting point for further research into the security of programmable data-plane networks. Suggestions for further research are to let go of the core assumption and focus on realistic and practical setups.

9 Conclusions and Future Work

In this paper, the security of a P4 programmable network is researched. Specifically, whether a compromised P4 switch can corrupt the entire network. The first research question is about an attacker obtaining access to the rest of the switches in the network, if one switch is compromised. From this paper no such conclusion can be drawn yet, and more research has to be done to confirm or deny this.

Second, we explore to what extent the attacker can corrupt the behavior of the network. Some ideas of potential exploits of load balancing schemes are proposed using a corrupt P4 switch. The first approach looks into congestion-aware load balancers that use Explicit Congestion Notification (ECN) information for congestion feedback. The main idea is to abuse this fact by ignoring ECN in the corrupted switch, and pretending the path along it is free from congestion.

The Clove-ECN load balancing scheme is investigated, which runs on a standard network using ECMP, and runs load balancing on the hosts. ECN is used to get information on the state of the network, and helps choose the best path from host to host. While Clove-ECN can run on a standard network, we consider the case where one of the switches in the network is a corrupted P4 switch that the attacker controls. It is then possible to manipulate ECN data, after which hosts will prefer the path along the corrupt switch.

A network with switches running a basic version of ECN+ECMP load balancing can be influenced by a single compromised switch. It is easy to discard ECN information, which other elements in the network use to properly balance traffic. Even though it is possible to influence load balancing, it is hard to make any statements on the security impact because of the simplicity of the setup. Future work is to thoroughly test this idea on a network running Clove-ECN or another load balancing architecture.

HULA, a load balancing scheme designed for programmable data-plane data center networks, gets congestion information directly from switches by regularly sending probe packets across the network. HULA is susceptible to probe packet spoofing. The corrupt switch can alter utilization data in the probe packets, after which top-of-rack switches will prefer the 'less utilized' path along the corrupt switch. While this is an availability and thus a security problem, HULA is applied almost solely in data center networks which use extensive security measures to prevent access to the network in the first place.

Third, mitigation methods for both load balancing schemes are probably best sought in securing communications. For example, IPsec may be used to prevent switches from changing ECN header data. While preventing switches from changing data would help secure Clove-ECN, this is not an option for HULA. Since the switches have to report their utilization and best next hop, they are free to modify probe packets. While HULA is robust against link failures, it is probable that there is no good solution to secure the scheme as a whole. Again, this will have to be confirmed by further work.

Securing networks has been a challenge since the inception of the internet. Programmable data-plane networks are no exception to this, and can even prove to be even harder to secure. Programmable devices performing non-standard algorithms and schemes have to rely on the security of those algorithms, as they cannot trust other devices in the network. A secure network proves to be a difficult feat to achieve.

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] H. Song, "Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-Proof Forwarding Plane," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pp. 127––132, Association for Computing Machinery, 2013.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[4] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, vol. 9, pp. 87094–87155, 2021.

[5] C. Black and S. Scott-Hayward, "Adversarial Exploitation of P4 Data Planes," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 508–514, 2021.

[6] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, RFC Editor, September 2001.

[7] D. Murray, T. Koziniec, S. Zander, M. Dixon, and P. Koutsakis, "An analysis of changing enterprise network traffic characteristics," in *The 23rd Asia-Pacific Conference on Communications*, 2017.

[8] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, SOSR '16, Association for Computing Machinery, 2016.

[9] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM Workshop on Hot Topics in Networks, Hotnets-9*, 2010.

[10] A. Bas, A. Fingerhut, *et al.*, "Behavioral model v2: the reference p4 software switch." https://github.com/p4lang/behavioral-model, 2022. Accessed: 2 June 2022.

[11] "Intel, Kaloom create P4-programmable network solutions." https://networkbuilders.intel.com/solutionslibrary/intel-kaloom-create-p4-programmable-network-solutions, August 2020. Accessed: 9 June 2022.

[12] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford, "Clove: Congestion-aware load balancing at the virtual edge," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, pp. 323—-335, Association for Computing Machinery, 2017.

[13] R. Nigam, "Implementation of the HULA data-plane load balancing protocol." https://github.com/rachitnigam/Hula-hoop, 2022. Accessed: 16 June 2022.

[14] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 2992, RFC Editor, November 2000.

[15] M. V. Dumitru, D. Dumitrescu, and C. Raiciu, "Can we exploit buggy p4 programs?," in *Proceedings of the Symposium on SDN Research*, SOSR '20, p. 62–68, Association for Computing Machinery, 2020.

[16] Y. Gao and Z. Wang, "A review of p4 programmable data planes for network security," *Mobile Information Systems*, vol. 2021, 2021.