Filtering DDoS traffic using the P4 programming language

Jan-Jaap Korpershoek
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
j.j.korpershoek@student.utwente.nl

ABSTRACT

To counter the growing problem of DDoS attacks, the availability of prevention and mitigation tools is important. Currently most of those tools are expensive in software and/or hardware. Open source solutions are very limited in both availability and functionality. The P4 programming language may be used as a step in the right direction. The language is platform independent and can thus be used to run the same code on several hardware components. This makes it a good platform for wide-scale open-source development of DDoS protection tools. The usefulness of P4 for filtering DDoS packets is analysed by assessing how known filtering methods could be implemented in P4 and by implementing one of the methods, History-Based IP filtering (HIF), and documenting the problems and limitations. It turns out that P4's lack of loops, the reliance on control-plane for filling match-action tables and the limited storage facilities of registers are the main limitations. However, despite these limitations, simpler filtering mechanisms can be implemented or already have open source implementations. The P4 implementation of HIF is shown to drop 99.87% of DDoS packets. However, it also drops about 10.19% of legitimate traffic.

Keywords

DDoS, P4, filtering, BMV2, software switch, History-Based IP filtering, HIF

1. INTRODUCTION

With the steady incline in rate and severity of Distributed Denial of Service (DDoS) attacks [9], the need for defence mechanisms against those attacks grows. According to [20] there are two main types of defence mechanisms, firstly prevention and secondly detection and mitigation. Prevention uses techniques that are in place before any attack happens. Examples of this are *over provisioning* and *modifying scheduling algorithms* (e.g. ordering traffic according to the amount of suspicion). The detection and mitigation mechanism tries to detect when a DDoS attack is happening and reacts accordingly. The first part of this mechanism, detection is concerned with determining whether a system is being attacked. To mitigate such an attack multiple tactics can be used, such as rate-limiting and filtering. Rate limiting means that a fraction of the packets is dropped either at the victim server, or somewhere upstream, in order to reach manageable numbers. Filtering seeks to eliminate attack traffic, while keeping the normal traffic.

29th *Twente Student Conference on IT* July. 6th, 2018, Enschede, The Netherlands.

Even though companies dedicated to DDoS mitigation are capable to handle DDoS attacks, small companies often have neither the right knowledge for development of mitigation systems nor the funds to buy off the shelf products or services. Even though there is an abundance research available on this topic, there the set of open-source tools that can be deployed is very minimal [12].

The P4 language (Programming Protocol-independent Packet Processors) [11] could be used as a platform to facilitate the development of efficient open-source tools for this domain. This programming language is designed to be target independent (i.e. suitable for describing everything from high-performance forwarding ASICs (Application-Specific Integrated Circuits) to software switches [6]) and is therefore suitable for wide-scale open-source deployment.

P4 is already rapidly rising in popularity, according to [14] the P4 Language Consortium boasts over 12 university members and 44 industry members, including companies such as Microsoft, Intel, Cisco, and VMWare. SIGCOMM 2016 included five papers related to P4. Using P4, developers have created a variety of powerful new applications including advanced network diagnostics and telemetry and responsive traffic engineering.

1.1 Objective

The goal of this research is to evaluate the suitability of P4 for the implementation of filtering techniques. To this end several filtering techniques are explained and discussed to determine how they can, or cannot, be implemented in P4. In addition, a filtering technique is selected in section 3.2 based on the requirements listed in section 3.2.1. This filtering technique is implemented and verified to assess the correctness of the implementation.

This verification is both qualitative and quantitative. The qualitative part consists of determining if the filtering works as expected, i.e. DDoS related packets are filtered and normal packets are passed to some extent. The quantitative part consists of measurements of the rates of the filtering mechanism that are described in table 1.

The results of this research are a starting point for developers of state-of-the art open-source DDoS protection tools in P4. These tools could then be used by any company to make their services more resilient against these types of attacks and thereby increase the reliability of their systems.

Table 1: Performance measurement rates

Name	Description
True drop rate	The percentage of DDoS packets that is dropped during the test.
False drop rate	The percentage of normal packets that is dropped during the test.
False pass rate	The percentage of DDoS packets that is passed during the test.
True pass rate	The percentage of normal packets that is passed during the test.

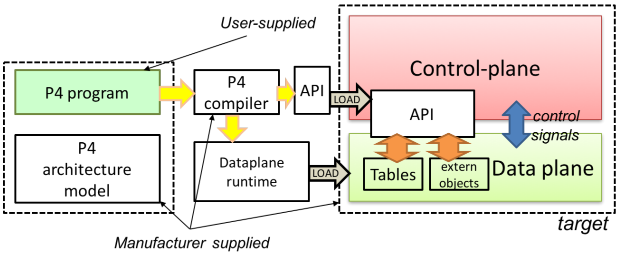

Figure 1: Programming a target with P4 [8]

2. BACKGROUND

2.1 The P4 language

P4 is set in the context of Software defined Networking (SDN). SDN aims to give operators more control by separating the control plane (i.e. high-level software to control the behaviour of a switch) from the data plane (i.e. software or hardware that directly controls the forwarding of packets, also known as the forwarding-plane). P4 is designed to program the data plane of forwarding devices in a way that is target-independent, so that the same code can run on multiple hardware devices; and protocol-independent so that switches programmed in this language are not tied to specific protocols [11]. However, P4 programs do depend on the architecture of a device. This architecture has to be specified by the manufacturer in the form of a P4 architecture model [8]. The data planes can use common interfaces like OpenFlow so that they can be configured by existing control plane software [11].

The main benefit of P4 is that it provides more flexibility for the expression of forwarding policies. This in contrast to traditional switches which provide fixed-function forwarding engines [8].

2.1.1 *Features*

The most recent version of P4 is $P4_{16}$. This section describes a relevant subset of the current feature-set of P4 and how those relate to filtering techniques. For the full specification, see [8]. As can be seen in fig. 1, a P4 program builds on a manufacturer supplied P4 architecture model. When it is compiled, a data plane runtime and API is generated. The data plane runtime is used to control the data plane of the switch, while the API is used by the control plane to communicate data with the data plane.

The basic structure of a P4 program includes the following elements.

Parser Extract the necessary headers from packets.

Match-Action pipeline Match keys, which are derived from the packets or metadata, to actions using tables and execute the actions. This matching can be done exactly, based on a pattern (ternary) or based on the longest prefix.

Deparser Insert (adapted) headers back into the packet.

A program may include multiple elements of these types. The elements are always specified and executed in this order, first the parsers, then the Match-Action pipelines finally the deparsers.

Like other programming languages P4 also has statements and expressions, these can be used in control blocks (i.e. initiating code of pipelines and deparsers), action blocks and parser states. These statements include variable declarations and assignments, calls to actions. Actions in P4 can be compared to functions in other languages, they have input and output arguments and can perform statements. The expressions include the usual comparisons and arithmetic operations.

A limitation of P4 is that it does not include floating point numbers. Another limitation is P4's lack of loop constructs, this is to keep a bound on the computation time for each packet.

A P4 program follows the flow of a packet through the application, therefore if no packet is being processed, no tasks can be done from the data plane.

2.1.1.1 *Data storage and retrieval.*

There are several ways in which a P4 program can store data. Some of these storage elements are only available for the current packet, such as user-defined metadata, packet headers and local variables. Other elements are available across packets, such as registers, counters and meters, these objects are provided by the architecture using the *extern* construct. Counters can count a number of packets or bytes. Meters can measure the rate at which a specific event happens (i.e. arrival of a packet). Registers are similar to arrays in other programming languages. They have a predefined number of slots which can be used to store one of the base types, match_kind, bool, integer [8].

The match-action tables are a special type of storage elements since P4 programs cannot store data in them, however, they can read the data that is stored in the tables by the control plane. These tables can be used to store data necessary for filtering, however, if that data needs to be collected from packets, a message should be sent to the control plane for every packet so that the control plane can fill the match-action table. This generates a considerable overhead which denies the performance benefits of using P4 for this task instead of higher level filtering devices.

For some types of filtering mechanisms timestamps are necessary, those can be obtained from the *standard_metadata* fields *ingress_global_timestamp* and *egress_global_timestamp*. This *standard_metadata* also includes other useful fields like ingress and egress ports for routing, a drop flag and packet length.

2.2 BMV2 software switch

The P4 capable device that is used for testing during this research is the BMV2 software switch [1]. A software switch is a switch that is simulated in software, this in contrast to a normal switch that is implemented in hardware.

2.2.1 *Communication from control plane to data plane*

To communicate with the data plane of the BMV2 switch there are two main methods, namely P4 runtime [3], which uses a GRPC [3] interface, and the BMV2 CLI. P4 runtime

Table 2: Categorisation of techniques according to the location in the network and whether they need external detection or not

Technique	Location	Ext. Det.
Ingress/egress filtering	Source	No
Martian address filtering	Source	No
source address validation	Source	No
Route-based packet filtering	Path	No
Path identifier	Victim	No
Signature-based filtering	Victim	No
Hop-count filtering	Victim	Yes
History-Based IP filtering	Victim	Yes
PacketScore	Victim	Yes
Confidence-Based filtering	Victim	Yes

has the benefit that it is more scriptable, however, it lacks some features, like writing registers, that the CLI does have. Because register writing is a feature that is necessary for the implementation described in section 4, the CLI is used during this research.

2.3 DDoS attacks

DDoS attacks can be divided into two categories, bandwidth depletion and resource depletion. Both categories aim to prevent legitimate communication with the server. They do this in different ways. Bandwidth depletion attacks achieve this by getting packets dropped because there is not enough bandwidth. Resource depletion attacks achieve this by sending requests that require some processing by the server, thereby they keep the server busy with the attack packets. The effect of this is that it does not respond to legitimate packets. A frequently used technique in DDoS attacks is IP spoofing [19], (i.e. packets are sent with a different source address than the actual source). Out of the 18 types of attacks analysed by [17] 11 use some form of IP spoofing.

3. FILTERING

Filtering is one of the methods used to mitigate DDoS attacks. The techniques described in section 3.1 can be categorised on their location in the network and on whether they need a separate detection mechanism to switch between normal and attack phase. The location can be at the source (source-initiated), along the path (path-based) or at the victim (victim-initiated) [16]. This categorisation is shown in table 2.

3.1 Techniques

This section describes commonly used filtering techniques and their suitability for implementation in this research according to the requirements set in section 3.2.1.

3.1.1 Ingress/egress filtering

Ingress filtering drops packets with spoofed IP addresses by only allowing a predefined range of IP addresses into the network. Egress filtering drops packets that are leaving a subnet but do not have a valid source IP from that network.

According to [17] the success of these techniques depends on if the range of valid IP addresses is known, which is not always the case. Also, attackers can spoof IP addresses in such a way that they are still within the valid range. These types of filtering are easily implemented in P4 because they require only table lookups and the table entries do not depend on the received packets, so no communication is needed from data plane to control plane. They have been implemented in [21].

3.1.2 Martian address filtering and source address validation

Martian address filtering drops packets with reserved or invalid source IPs. This prevents attackers from randomly spoofing packets.

Source address validation drops packets that arrive on a different port than where a packet would be routed when returned to the sender. In other words, if an address is never reached over a specific port, no packets should be received from that address over that port. A drawback of this technique is that it yields many false positives for asymmetric routes. Another drawback is that not all routers in the internet implement this, so it is currently not effective [17]. These filtering techniques are defined in RFC 1812 [10]. They have been implemented in [21].

3.1.3 Route-based packet filtering (RPF)

This filtering technique filters based on the principle that each link of a core router accepts traffic from only a limited number of source addresses. This way the network topology can be used to drop packets that have an incorrect source for their destination. Significant success of this technique will be achieved if 18% of the autonomous systems implement this filtering technique. This amount is impractical to realise considering how the internet is structured currently. To include the source address in a BGP message, the format has to be adapted. This results in bigger BGP message size and longer processing times for these messages. Attackers can bypass this filtering technique by choosing spoofed IP addresses in such a way that they are not detected [17]. For an implementation in P4, the source IP address and link of the core router can be used as keys to a match-action table, in which the accepted combinations can be saved by the control plane. An implementation on a single device is hard to test since the amount of packets that is filtered per device is very low, several routers are necessary to achieve a good accuracy.

3.1.4 Hop-count filtering (HCF)

This filtering technique uses the TTL value to determine how many hops a packet has travelled. It calculates this hop count by subtracting known initial TTL values (30, 32, 60, 64, 128, 255) from the received TTL. It uses the hop count to determine if an source IP address is likely to have been spoofed by comparing the values of received packets with the saved values.

It has two states, *alert* and *action*. The alert state is the normal state, during this state no packets are dropped even if they are detected to be spoofed, this prevents collateral damage. The action state is entered when a DDoS attack is detected, during this state all packets with an incorrect hop count are dropped. This technique can recognise close to 90% of spoofed IP packets [15]. This can drop packets with spoofed IPs because those packets often also have invalid hop counts. However, this technique has the drawback that the number of false positives is high [17]. For implementation in P4 registers can be used to save the hop count values for IP addresses by using hashed IP addresses as register indexes. This technique is described in [15].

3.1.5 History-Based IP filtering (HIF)

For this filtering technique a database of valid sources is maintained based on a history of normal traffic from those IP addresses. During the training phase the number of packets that have been received from a source and the number of days a source has been seen are stored. During the attack phase thresholds are used on the number of packets, the number of days, or both, so that all packets that have a value in the database below the threshold, or do not exist are seen as attack traffic and are consequently dropped. To ensure that only recently active sources are kept in the database a sliding window is used. This means that IPs that have not been seen for a predefined time are removed from the database. This technique fails if the attacker can first simulate normal traffic, and then attack from the same source [17]. For implementation in P4 registers can be used as hashtables to store the necessary data. The sliding window can be implemented by checking the last seen timestamp while updating a packet or applying the filter. This technique is described in [18].

3.1.6 Path identifier (Pi)

This filtering technique uses a path identifier that is saved in a packet to drop all packets that have the same path identifier as a detected malicious packet. Drawbacks of this technique are that multiple routers (20-50% of the internet) have to work together to achieve a reasonable amount of DDoS protection. Another drawback is that since the identifier has a small size, there will be overlap in different paths. Thus, it increases the chance of false positives [17]. A P4 implementation could use match-action tables in which the key is the path identifier of a packet. If another device detects an attack packet with a specific path-identifier, this identifier can be stored in the match-action table, with the drop action attached. This technique is not suitable for implementation on a single device, since it needs several routers to mark packets. This technique is described in [22].

3.1.7 PacketScore

This filtering technique uses Bayes' Theorem to calculate the probability that a packet is legitimate based on baseline values that are calculated using a statistical analysis of normal traffic. This technique is described in [16]. A drawback of this technique is that is uses a lot of storage for the attributes for normal traffic [17]. Generation of the nominal profile occurs in different periods, the results of which are combined into a final value based on the occurrence rates of different measured attribute values. These rates are calculated by dividing the number of packets with a certain attribute by the total amount of packets. To combine these values usually loop constructs are used to calculate a resulting value for each attribute. Since P4 has no loop constructs (see section 2.1.1) another way should be found to do this. One way to solve this problem would be to store the total amount of packets for each period separate from the attribute counts and doing the division while comparing. However, for combining the results of multiple periods into one result, again loop constructs are necessary. To work around that the results of each period should be kept in memory until an attack happens. This is not feasible due to memory constraints.

3.1.8 Confidence-Based filtering (CBF)

This filtering technique follows the same idea as PacketScore in that it uses a profile of normal traffic to calculate a threshold below which a packet should be dropped. However, it can handle larger attacks because scoring and discarding packets is not related to attack intensity. Another benefit is that the calculations for CBF are less complex than those for PacketScore [13]. For the calculation of confidence values, the occurrences of attribute values are counted during a time-window. At the end of each time-window the confidence values are calculated based on those stored values. For the generation of the nominal profile the same implementation problems arise as with PacketScore. Implementation in P4 follows a similar approach to an implementation for PacketScore.

3.1.9 Signature-based filtering

In this filtering technique packets are compared against known signatures of DDoS attacks that have been created based on previous attacks. An example of signatures that could be used can be found in [2]. This technique can very accurately drop packets belonging to a DDoS attack if the signature is known. A drawback of this technique is that at least one attack has to succeed before it can be blocked, because the signature is not known in advance. Implementation in P4 using the signature structure of [2] should be possible since the match-action tables could be used to look up IP addresses, ports and protocols. There is a multitude of different Signature-based filtering techniques, so there is no clearly defined algorithm.

3.2 Filtering technique selection

3.2.1 Requirements

To evaluate the suitability for implementing a filtering technique in P4, one must be selected that will yield a useful result. Therefore this technique should be a popular technique, since that means that an implementation will probably be used. Another benefit of popular techniques is that the amount of documentation and examples that are available is bigger, which facilitates the implementation. Furthermore, for testing purposes it is convenient that the algorithm can be tested on a single device. To advance the state of the art it is necessary for this research to yield new results, therefore there should not already exist a P4 implementation of the selected filtering technique.

As a result of the above reasoning, the selected filtering technique should have the following characteristics.

- It must be commonly used in real-world applications
- It must be implementable on a single device
- It must not already have an open-source implementation in P4
- It must have potential for an implementation in P4 (i.e. it must not be known to be impossible beforehand)

3.2.2 Selection

Based on the survey of the filtering techniques in section 3.1, there are three that meet the requirements set in section 3.2.1, they are listed below.

1. History-Based IP filtering (HIF)
2. Hop-count filtering (HCF)
3. Signature-Based filtering

For Signature-Based filtering there is no single algorithm to implement, therefore this technique is not suitable for this research because then choosing and designing a good Signature-Based filtering technique would deduct too much time from the objective of this research.

HIF and HCF both use a history of normal traffic to set a standard which new packets must adhere to. However in HIF this matching is less strict than in HCF since HIF uses a threshold which packets must be above, while HCF uses an exact value. This makes HIF more flexible because with different thresholds the algorithm can be adapted to the kind of traffic that a server is receiving. In addition, the different filtering criteria of HIF make it suitable for a step by step implementation in which each step can be tested independently.

Because of these reasons HIF is selected as the algorithm that is implemented in P4.

4. IMPLEMENTATION

This section describes the P4 implementation of History-Based IP filtering (HIF), the technique selected in section 3.2. As described in section 3.1.5 the technique is divided into two phases, the training phase and the attack phase. The implementation of these phases is discussed below. Where necessary the technique is explained in more detail.

The implementation only filters IPv4 packets because the most suitable test data and DDoS data available only contains IPv4 addresses. Besides that, the only notable difference of using IPv6 for the implementation would be a deviation in the probability of hash collisions due to fact that IPv6 has more addresses, which would not yield results about the suitability of P4 in the area of this research.

The code of the implementation is open-source and available including results of the tests executed in this research. [1]

4.1 Data storage

The data that is needed for the filtering during the attack phase is the number of days that the IP has been seen and the number of packets that have been received. In addition, to implement the sliding window and to count the number of days a source IP has been seen, a timestamp of the previous packet is needed, therefore, this timestamp should also be stored.

So the fields that need to be stored per source IP are

- Number of packets
- Number of days
- Timestamp of previous packet

Match-action tables were considered as a storage method for this data because of the big amount of memory that is available to them. The action would then be the same for all entries, namely, the filter action. The data would then be stored as action parameters. Despite those benefits, they were not used because of the performance limitations described in paragraph 2.1.1.1. A problem that was observed with an implementation using tables for storage was that the number of packets that were observed was significantly lower than the number that was sent. This is presumably due to the fact that the delay in connection between data plane and control plane caused the processing of packets to use outdated data.

The remaining option for the data storage was to use registers. They are used as hash tables, the IP address is hashed, and the resulting number is used as the index of the register. The main drawback of using registers is that there is a limited amount of memory available (see paragraph 2.1.1.1). The result of this is that the hashing algorithm (CRC32) has to be limited to a certain target space, the size of the register. The biggest register size that could be used as the hash map in the BMV2 switch was empirically determined to be 2^{22}. For higher values than that, the program would occasionally crash during initialisation. For hardware switches this boundary for the size of the registers is predefined and can usually be found in the device specification.

Due to the fact that the source space is bigger than the target space, there are collisions in the hashes, i.e. multiple source IPs will have the same hash. Consequently, there are cases in which multiple sources that would not reach the thresholds independently, may reach them together because of a collision, or a source would pass during the filtering phase, because a colliding source has reached the thresholds. Another, more severe, problem is if attack sources have collisions with valid sources so that their packets are not filtered.

[1] `https://github.com/JJK96/P4-filtering`

4.2 Training phase

During this phase, the data discussed in section 4.1 is collected. Saving the timestamp of each packet and counting the number of packets is trivial. However, in determining whether a source IP has been seen before in the current day some complications were encountered. Normally, a loop would be used to repeatedly add the length of a day to the current start of the day until the difference between the current time and the start of the day is less than a day. Thus, a next day in which packets are received always starts an exact number of days after the previous day. Naturally, days in which no packets are received do not have to be considered. Since P4 has no loop constructs, this kind of algorithm is not possible, so there are two possibilities left. The first is to update the *day start* from the control plane by updating a register. However, this was not done in this implementation because this requires the register to be updated every so often, which means that a scriptable API like P4 runtime [7] is needed. On the other hand, to change the switch from training to attack phase (as described in section 4.4) a register write is needed, which is not possible with P4 runtime (see section 2.2.1). The second possibility is to use an approximate calculation that does not include loops. In the implementation of this research, the next day is always the arrival time of a packet, which means that if no packet arrives exactly one day after the previous day start, that previous day will be slightly longer. The impact of this algorithmic flaw is shown in section 6.2.

The HIF paper [18] states that only packets of valid TCP connections should be used to update the database. An implementation of that part of the filtering technique would consist of a lookup for the source IP of each packet to see in what state the TCP connection is and it would and a correct update of the TCP state. This kind of lookups and conditional updates are also shown in other parts of the P4 implementation and thus have no additional value in showing the capabilities and limitations of P4 and in thus not incorporated. However it should be noted that the additional register for saving the TCP status for each source IP will increase the overall required memory, which might present difficulties depending on the amount of memory available

4.3 Attack phase

During this phase, packets are filtered based on the values that have been stored in the training phase. There are two rules based on which HIF can filter packets. The first is based on the number of packets that has been received from an IP, the packet threshold. The second is based on the number of days a source IP has been seen. In this research both of these rules are used, after the example of [18].

4.4 Phase transitioning

The phase of the switch is toggled between training and attack phase using a one-bit register which is set using the BMV2 CLI. In a production implementation this would be set as a result of an external attack detection mechanism.

5. VERIFICATION

This section evaluates the correctness of the implementation by analysing the processed output of the BMV2

Table 3: Datasets of legitimate data with relevant test parameters: the number of algorithmic days and the replay speed

Name	Description	Alg. days	Replay speed
ftp-small	1 day of ftp data from [4]	50	400x
ftp-big	10 days of ftp data from [4]	10	200x

switch and comparing it to expected values that are calculated by a Python program that performs the same action as the switch on the same data. Differences in the output are discussed.

5.1 Test data

In the absence of a perfect dataset as described in section 6.3, the datasets that are described in table 3 are used. Some of the tests are executed on the smaller dataset because processing and analysing that data takes considerably less time than for the big dataset. For most of the tests the big dataset is used, because it is closer to a real-world situation, and will thus yield more accurate results. In addition to these sets of legitimate data, also DDoS data is necessary. This is obtained from DDoSDB [2] by contacting its institutor. Since for HIF only the source IPs are important, a custom dataset is generated with one packet per IP found in the DDoSDB data. The amount of packets per IP address does not matter for HIF because either all or none of the packets of an IP will be dropped. For the tests, the datasets are split in two parts with a ratio of 9:1. The bigger part is used as training data for the training phase. The smaller part is mixed with the DDoS traffic and used as test data for the attack phase.

5.2 Setup

The testing setup is a mininet [5] network in which two hosts are connected to each other via a BMV2 switch. The packets are replayed from a pcap file by the first host, they are sent to the switch, which forwards them to the second host. The speed of the replay depends on the average speed at which packets arrive. For the small dataset the time between packets was small enough, so the replay speed could be higher than for the big dataset. Meanwhile, the input and output traffic of the switch is saved in pcap format to be analysed by the Python program later. The results of this analysis are shown in fig. 2 and fig. 3. They show the result of the training phase, and the filtering results of P4 in relation to Python.

5.3 Results

5.3.1 Training

The results of the training phase are shown in fig. 2. It shows how the sources relate to the packet and day thresholds. In the figure, two threshold combinations are shown, the initial, in which the day threshold is 5 and the final in which the day threshold is 1. Sources that lie in the upper right quadrant of the utilised threshold combination exceed both thresholds. Packets from those sources will thus be admitted during an attack, while packets from all other quadrants are dropped. From fig. 2 it is clear that a significant portion of the addresses fall below the initial day threshold of 5. This naturally results in an unacceptable false positive rate. In addition, it is evident that several sources of legitimate traffic are seen only one day, therefore a day threshold of 1 is used.

5.3.2 Filtering

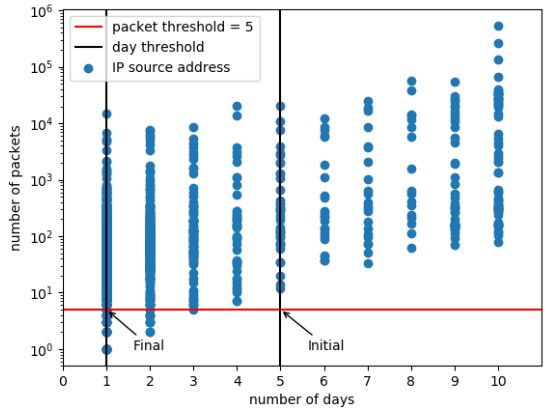

Figure 2: Result of the training phase on ftp-big, showing how sources related to the day and packet thresholds

Table 4: The filtering rates for P4 and Python for ftp-big

Rate	P4 (%)	Python (%)
True drop rate	99.87	100.00
False drop rate	10.19	10.19
False pass rate	0.13	0.00
True pass rate	89.81	89.81

.

In fig. 3, the output after the filtering by P4 is shown. The input is not shown because the enormous amount of input traffic would obscure the differences between Python and P4. The sources marked in red are DDoS sources that were passed by P4 because they had values in the database that were higher than the threshold, however, they were not seen before by the Python implementation (i.e. they had no value in the database that was constructed during training). The Python implementation did not pass any DDoS packets. There can be two reasons that the packets were seen by the switch and not by Python.

The first reason is that they were not captured in the pcap files that were saved by mininet. During the tests on ftp-big a difference in the number of packets reported by P4 and by Python was observed. P4 reported 4655 (0.18%) more packets than Python. So there is a difference between the number of packets that passed the switch and the number of packets saved in the pcap files. However, it is unlikely that this is the cause of the difference in filtered output because the switch reports that from some of these differing IP addresses it has received over 8000 packets, which is far more than can be accounted for by the difference in the pcaps.

The second, more likely reason is that this is the result of the hash collisions discussed in section 4.1. The IP addresses would have the same hash in the switch as an IP address that was seen, but in Python the hashes would be different. In table 4 it can be seen that the complete difference between P4 and Python is in the DDoS packets. So the way in which the filter passes legitimate traffic is exactly the same for both P4 and Python. The results of filtering for ftp-big are shown in table 4.

6. DISCUSSION

This section discusses different design choices, in addition

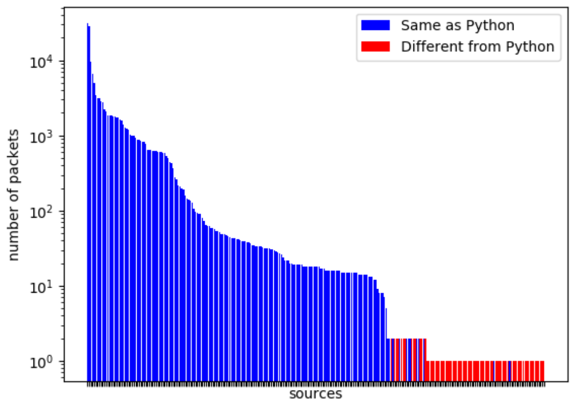

Figure 3: The filtered output of the switch for dataset ftp-big

it analyses and discusses their influence on the results.

6.1 Parameters

This section discusses the selection of values for the different parameters of HIF.

6.1.1 Day

The length of an algorithmic day (i.e. how long a day is in the code) has to be defined. For the big dataset this can just be the actual length of a day divided by the replay speed multiplier. However, for the small dataset, it should be chosen sufficiently small so that multiple algorithm days happen within the one day of data. Therefore, the length of a is decided to be 20000 times as small as a normal day, so that at a replay speed 400 times the normal speed, 50 algorithmic days happen. This amount is enough to give sources a chance to appear on multiple days while keeping the days long enough to distinguish between regularly connecting clients and incidental connections.

6.1.2 Timeout

The timeout parameter defines the size of the sliding window, i.e. the time after which an entry in the database is deleted if no packet with the same source IP is found. According to the example of [18] the timeout was initially set to two weeks.

On ftp-small the sliding window had a negative impact on the performance, because due to the nature of ftp, clients typically send packets in bursts, in between which they do not send packets for some time. This behaviour would result in sources timing out even though they do have legitimate behaviour. Besides that, the sliding window is used to get a higher *true drop rate* and a lower *false pass rate*, which, as can be seen in table 4, is not possible for our dataset. For these reasons the sliding window was disabled.

6.1.3 Packet threshold

The packet threshold determines the number of packets that have to be received from a source IP for packets from that address to be passed. It is set to 5 according to the example in [18]. It has been compared to a threshold of 1 for ftp-big. This resulted in very small differences in the rates (order of magnitude 0.01%). Therefore the threshold is kept at the value of 5.

6.2 Days calculation algorithm

Because the arrival time of the first packet after a day was used as the start of the next day, days are too long if a packet does not arrive at exactly the right time. The influence of this algorithmic flaw is analysed on ftp-small because it has more algorithmic days than ftp-big. The results in the number of days an IP has seen differ for 24 of the 622 IP addresses (3.85%). The differences are not higher than 3, while the highest number of days in the dataset is 27. The influence on the filtered output depends on the threshold for the number of days. When using the same method for calculating the next day there is no difference in the number of days.

In this research the day threshold is set to 1 (see section 5.3.1), therefore there is no influence of this flaw on the results.

6.3 Test Data

The perfect dataset for testing an HIF implementation would be at least a week of data and a spoofed DDoS attack on the same server. This is because if the attack and the data are from the same server, the most realistic performance measurements are obtained. Otherwise, if the servers differ, there will probably be less overlap in the IP addresses than expected in a real scenario. This lack of overlap is seen in table 4, the true drop rate for Python is 100% because none of the DDoS IP addresses appeared in the training set.

7. CONCLUSION

As seen in section 3.1, out of the 9 techniques, 2 can definitely not be implemented in P4. Namely, *PacketScore* and *Confidence-Based filtering (CBF)*, this is due to the lack of loop constructs in P4. *Ingress/egress filtering* and *Martian address filtering and source address validation*, already have implementations in P4. The other techniques have varying degrees of potential. *Route-based packet filtering (RPF)* and *Path identifier (Pi)* can be implemented in P4 with relative ease because they require only matching on rules that can be independently stored in the match-action tables by the control plane. Whether *Signature-based filtering* can be implemented depends on the exact algorithm. If the algorithm matches signatures that can be stored independently by the control plane, as is the case for RPF and Pi, it can be implemented, however if data needs to be gathered from the processed packets and stored by the data plane, an implementation is cumbersome. *Hop-count filtering (HCF)* is similar to *History-Based IP filtering (HIF)* in implementation, it will therefore have the same limitations.

The case of HIF has been discussed in more detail. Through an implementation of this algorithm several limitations of P4 have been encountered and circumvented where possible. Firstly, the lack of loop constructs can make it difficult to implement known algorithms correctly and can sometimes make an implementation completely impossible because data cannot be manipulated in the way that it should, as seen for CBF and *PacketScore*. Secondly, the fact that the big memory in match-action tables cannot be used by the data plane for storage makes that either the control plane has to be used for the action of storing data that is usually gathered in the data plane, or that other, smaller, storage mechanisms have to be used. Finally, registers, even though not as big as the match-action memory, can be used to store significant amounts of data, however, this is not always enough for a correct implementation, as is seen in the case of HIF. Another drawback of registers is that only base types can be stored, and no user-defined structures. This may make some complex implementations more difficult or less able to efficiently use the available space.

To summarise this conclusion, P4 targets a use case where actions are performed based on match-action tables that are filled from the control plane. Because of this, using it for storing data from the data-plane has its limitations. However, by choosing the right algorithms and workarounds it is possible to achieve satisfactory results, as is seen in table 4.

8. FUTURE WORK

The probability of the hash collisions described in paragraph 2.1.1.1 is currently not known in full, it is unknown how much the probability of a collision for IPv6 differs from IPv4. In addition, research should be done on the spreading of the hashes over the possible hash values. There may be room for improvement regarding the current hashing algorithm. Furthermore, some of the other filtering methods discussed in section 3.1 could be implemented to have a more complete evaluation of the suitability of P4 for implementing those techniques.

To evaluate the performance of P4 in this area in terms of speed an implementation could be run on P4-capable hardware. The performance of this application could be compared to the performance of other DDoS filtering tools.

9. REFERENCES

[1] Behavioral Model Repository. `https://github.com/p4lang/behavioral-model`. Accessed: 2018-05-03.

[2] DDoSDB - Collecting, Transforming, Applying and Disseminating DDoS Attack Knowledge! `https://github.com/jjsantanna/ddosdb`. Accessed: 2018-05-16.

[3] GRPC - A high performance, open-source universal RPC framework. `https://grpc.io/`. Accessed: 2018-05-12.

[4] Internet Traffic Archive. `https://ee.lbl.gov/anonymized-traces.html`. Accessed: 2018-05-24.

[5] Mininet - An Instant Virtual Network on your Laptop (or other PC). `http://mininet.org/`. Accessed: 2018-06-18.

[6] P4 language. `https://p4.org/`. Accessed: 2018-05-03.

[7] P4 Runtime - Putting the Control Plane in Charge of the Forwarding Plane. `https://p4.org/api/p4-runtime-putting-the-control-plane-in-charge-of-the-forwarding-plane.html`. Accessed: 2018-05-12.

[8] P4_16 Language Specification. `https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html`. Accessed: 2018-05-24.

[9] Q4 2017 State of the Internet / Security Report. `https://content.akamai.com/us-en-PG10413-q4-17-soti-security-report.html`. Accessed: 2018-05-12.

[10] F. Baker. Requirements for IP Version 4 Routers. RFC 1812, RFC Editor, June 1995.

[11] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[12] A. Cardigliano, L. Deri, and T. Lundstrom. Commoditising DDoS mitigation. In *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 523–528, Sept 2016.

[13] Q. Chen, W. Lin, W. Dou, and S. Yu. CBF: A Packet Filtering Method for DDoS Attack Defense in Cloud Environment. In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 427–434, Dec 2011.

[14] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon. Whippersnapper: A P4 Language Benchmark Suite. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 95–101, New York, NY, USA, 2017. ACM.

[15] C. Jin, H. Wang, and K. G. Shin. Hop-count Filtering: An Effective Defense Against Spoofed DDoS Traffic. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 30–41, New York, NY, USA, 2003. ACM.

[16] Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao. PacketScore: a statistics-based packet filtering scheme against distributed denial-of-service attacks. *IEEE Transactions on Dependable and Secure Computing*, 3(2):141–155, April 2006.

[17] T. Mahjabin, Y. Xiao, G. Sun, and W. Jiang. A survey of distributed denial-of-service attack, prevention, and mitigation techniques. *International Journal of Distributed Sensor Networks*, 13(12):1550147717741463, 2017.

[18] T. Peng, C. Leckie, and K. Ramamohanarao. Protection from distributed denial of service attacks using history-based IP filtering. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 1, pages 482–486. IEEE, 2003.

[19] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Comput. Surv.*, 39(1), Apr. 2007.

[20] A. Shameli-Sendi, M. Pourzandi, M. Fekih-Ahmed, and M. Cheriet. Taxonomy of distributed denial of service mitigation approaches for cloud computing. *Journal of Network and Computer Applications*, 58:165 – 179, 2015.

[21] A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, and M. Budiu. DC.P4: Programming the Forwarding Plane of a Data-center Switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 2:1–2:8, New York, NY, USA, 2015. ACM.

[22] A. Yaar, A. Perrig, and D. Song. Pi: a path identification mechanism to defend against DDoS attacks. In *2003 Symposium on Security and Privacy, 2003.*, pages 93–107, May 2003.