Detection and Mitigation Mechanisms for Attacks in Programmable Data Planes

Frank Broy
Supervisor(s): Chenxing Ji, Fernando Kuipers
EEMCS, Delft University of Technology, The Netherlands
22 June 2022

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

Detection and Mitigation Mechanisms for Attacks in Programmable Data Planes

Author: Frank Broy
Supervisor: Chenxing Ji
Responsible Professor: Fernando Kuipers
EEMCS, Delft University of Technology, The Netherlands

Abstract

DDoS attacks are becoming more common and sophisticated. Only recently, in 2017, Google claims they have mitigated an attack which sent 2.54 Tbps of traffic to their servers. In order to prevent these attacks, more and more robust defence mechanisms need to be put in place to withstand the malicious traffic and secure the networks.

Programmable data planes allow the users to specify which rules the headers of a packet need to follow and what happens if they are different. With this freedom, achieving more secure networks becomes possible. The use of the programming language P4 makes it easy to modify the functionality of the switches and limit the behaviour of the network in order to reduce the attack surface. This paper describes certain attacks and mitigation techniques for them, such as DoS attacks and SYN-flood attacks. The paper will list existing defence techniques and enumerate their advantages and drawbacks. There will be two proof of concept detection and mitigation techniques in P4, and these implementations will be compared to already existing ones. The P4 implementations will be provided as well as comparison and performance graphs.

1 Introduction

With the first routers emerging in the 1970s for public use, people nowadays are used to having constant access to a world of connected networks. During these 50 years, many new technologies have emerged, and the field of computer-networks has been growing constantly. With the massive growth on the usage of networks, there are also many security concerns that arise. In the last few years, the number of attacks, especially DDoS attacks, has been increasing [1]. The attacks continue to become more sophisticated and powerful [2], with the most powerful ones reaching more than 2.5 Tbps of traffic flow [3]. These attacks can drastically slow down networks to a point where parts of a network or even the whole network become completely unresponsive, which could have direct consequences on the real world. So in order to mitigate these problems, it is imperative to upgrade the detection and mitigation techniques for these types of attacks.

The current implementation of routers, modems, and switches consists of a black box implementation, meaning that only the vendor knows and can change how the device's inner algorithms work. The user can only change the settings of the control plane but cannot modify the logic or functionality of the data plane. However, with the latest emergence of programmable data planes and Software-Defined Networking (SDN), the data plane can be modified with a programming language called P4. P4 allows for defining new protocol headers and new processing algorithms, which give the user more control over their network [4]. While P4 is still a relatively new language, which came up in 2014, it is already standardised today in the P4 Language Consortium [5] and supported by various platforms used in the industry.

This report will answer the question "What mechanism(s) can the programmable data-plane adopt to prevent a specific attack (e.g., DDoS and SYN flooding)?", so which mechanisms can be implemented in the data plane using P4 to detect and mitigate DDoS and SYN-flood attacks up to a certain point. This paper contains implementations of these detection and mitigation techniques for general DoS and SYN flood attacks and compares the results to existing techniques and implementations such as [6][7][8][9]. Finally, there will be a conclusion of the various implementations as well as a discussion to develop a more secure way of networking in programmable data planes.

The paper will start with a small introduction to P4 and a more in-depth problem description in section 2. Afterwards, a general overview of existing attacks and mitigations will be listed together with their advantages and drawbacks in section 3. Section 4 will explain the experiments in more detail, contain the results as well as a comparison to other papers. Section 7 will be about the ethical aspects and the reproducibility of the research. Next, section 5 will discuss the results obtained by this research, possible improvements and other findings. Finally, a conclusion will be drawn, and possible improvements or issues will be discussed in section 6.

2 Methodology

In order to conduct this research, specific attacks need to be simulated and launched onto a network. This network will be simulated using Mininet, and the attacks will be generated using various tools and scripts. The detection and mitigation mechanisms will be implemented using the P4 programming language, and graphs and other results will be generated using Python.

2.1 P4

The Programming Protocol-independent Packet Processors (P4) programming language is specifically designed to specify how packet data is processed on network devices, specifically on the data plane. P4 allows the users to implement particular behaviour in their networks, which they can customise at their own will in no time. P4 has been around since 2014 and already has its own language consortium [5] and specification [10].

A P4 program is made up of 5 different parts:

- Parser, which checks if the incoming packets' headers correspond to the definitions
- Ingress, defines the tables and the processing algorithms in a Match-Action pipeline
- Verification, checks if the packets has the correct checksum and if there were no errors
- Egress, defines the tables and the processing algorithms in a Match-Action pipeline with egress specific information
- Deparser, which puts the packet back together how it will be send out later

Every P4 program is compiled into configuration JSON files by a P4 compiler such as p4c [11]. The program can now be run with the mininet switch and by injecting the runtime rules. The whole workflow of a P4 program is listed in Figure 1.

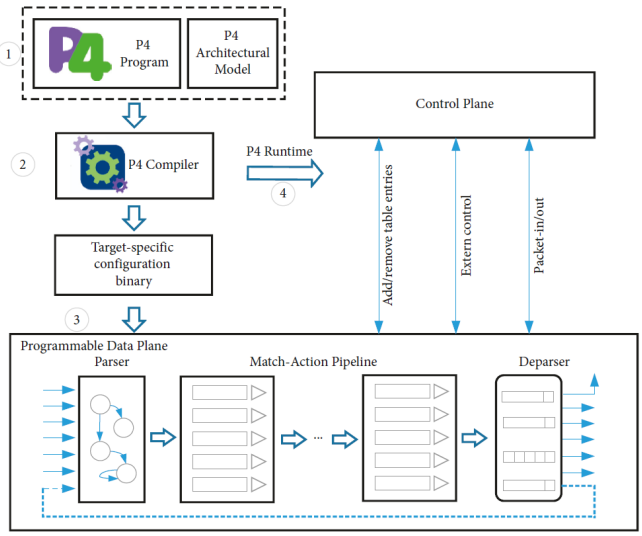

Figure 1: Workflow in the P4 programming language [12]

P4 can be used in a security aspect of networks since it allows for a complete definition of the packets and how they are processed. Limiting the correct packets makes the attack surface smaller, and it becomes easier to defend against the attacks since attackers will not have as many options. With P4, the detection and mitigation of attacks can happen closer to the actual source. Therefore, they will be more efficient compared to first treating the packets and then analysing them.

2.2 Types of Attacks

This paper focuses primarily on detecting and mitigating denial of service attacks. These attacks aim to slow down or even completely take down a network. They can come in various forms, and exploit and abuse different protocols. Some of these attacks will be described in the following sections, next to some detection and mitigation mechanisms.

Denial of Service

Denial of Service attacks are a collection of malicious traffic that disrupts the normal network flow by sending loads of requests quickly. Due to the high load of traffic, the network will need more time to respond to the requests or might even become completely unresponsive if the traffic is too much to handle. Denial of service attacks can be conducted from a single machine, but they can also be executed from multiple devices at once, making it a Distributed Denial of Service (DDoS) attack. These DDoS attacks are most often executed through botnets, where previously infected machines are used to send malicious traffic to the target. Defending against DDoS attacks is hard because of various reasons:

- the attacks can contain a huge amount of traffic that is sent to the server
- the attacks come from various sources, so it is hard to block them all
- it is hard to distinguish legitimate traffic from malicious traffic and therefore very complicated to drop the malicious traffic without affecting the legitimate one

There are many different types of DoS attacks, and the size can be measured in 3 kinds of units, packets per second, bytes per second and requests per second. DoS attacks can target various parts of network infrastructure, ranging from the application layer to the protocol layer. The attacks can either focus on the number of requests sent or on the size of the requests. In both cases, a complete denial of service can be achieved.

SYN-flood

Another sort of attack which can bring down networks is protocol abuse, where flaws in a specific protocol are exploited. One of those abuses is the so-called SYN flood attack.

SYN floods are an exploitation of the three-way handshake of the TCP protocol. In a typical TCP handshake, the user sends a SYN (synchronise) packet to the server to ask it to open a connection. When the server allows the connection, it opens a port and replies with a SYN-ACK (synchronise acknowledge) packet. The connection is established when the user replies with an ACK (acknowledge) packet, and communication can happen. However, if the user sends a SYN packet but never replies to the SYN-ACK sent by the server, the server keeps the selected port open for a certain amount of time. If the user floods the server with SYN requests, then the server will run out of ports that it can open since they are all occupied, and this leads to a denial of service since the server cannot respond to any new incoming requests. An illustration of the three-way handshake of the TCP protocol as well as of a SYN flood attack can be seen in Figure 2. Some potential mitigation techniques are described in section 4.

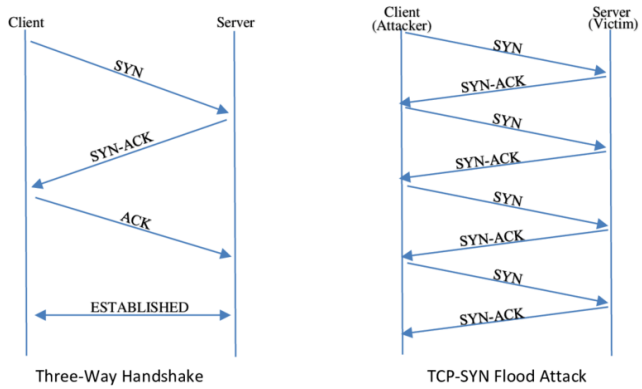

Figure 2: Normal TCP handshake (left) and SYN-flood attack (right) [13]

Other attacks

Various attacks can take down or slow down a network. Next to the before-mentioned attacks, there is also, for example, the HTTP flood attack. In this attack, the attacker sends a lot of HTTP GET requests to the server so that the server cannot handle the traffic load anymore and slows down or becomes completely unresponsive. Since an HTTP GET request asks for a particular response, the attacker can choose the request so that the response is massive. In this case, the attacker might not need to send as many different requests but can send only a few which want a bigger response and therefore put more load on the network when retrieving the results. So, in this case, the attacker looks more for the size of the response rather than the number of requests. These attacks are also called amplification attacks.

Another attack is a so-called Slowloris attack [14]. This attack tries to take down a web server by keeping as many connections open to the target. The connections are opened, and only a partial request is sent. The server then waits for the rest of the request, which is sent at a later point in time. However, the response will only come in parts and will never be complete. This method forces the target to keep the connections open, which at some point, if all connections are used, will lead to unresponsiveness since no further connections can be accepted.

An attack, which exploits reassembling packets, is the so-called Ping of Death [15]. During this attack, the attacker sends multiple malformed pings to the target. If an IP packet is too big, it is split into several smaller fragments, which are later on reassembled. The maximum size of an IP packet is 65,535 bytes, and if that limit is surpassed, memory overflows might occur, which is what this attack exploits. In the Ping of Death attack scenario, when the target reassembles the IP packets, it ends up with a packet larger than 65,535 bytes, which will overflow the memory buffers, leading to a denial of service on the target.

2.3 Virtual Network Simulation

To simulate the network that is being attacked, Mininet [16] is being used. The network is being simulated since it gives easier access to a network and since the network will be attacked, which is not ethical and even illegal. Mininet allows quickly simulating a virtual network and creating different topologies suited for different needs. In the topology that was used in the experiments, three hosts and one switch were used, as can be seen in Figure 3.

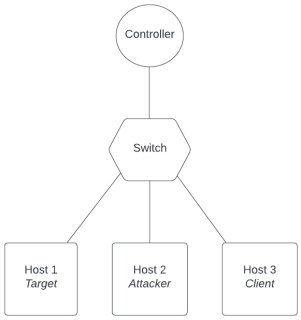

Figure 3: Network topology used for the experiments

The way the topology was used was that host one was the target, host two was the attacker, and host three checked the target's response times during an attack. All the hosts were connected to the same switch, which was itself connected to a controller.

2.4 Tools and Scripts

To simulate certain attacks, such as the SYN-request flooding, Hping3 [17] was used. Hping3 allows to send and receive packets which a single command. It has many kinds of attacks built in, so it is quite simple to simulate an attack, such as SYN floods or a DoS attacks, which were used for the experiments in this research.

For all other scripts that were used in this project, Python was used to automate these functionalities. One script for example was a latency tester script, which kept sending pings to the target host and measure the response times. Another script was automating a TCP handshake between host one and host three, which was used to measure the response time during a SYN flood attack.

3 Attack Detection and Mitigation Techniques

General attacks on networks that try to slow them down or even take them out completely come in various forms and can exploit various vulnerabilities of the target. The most crucial part of blocking malicious traffic is to prevent regular non-malicious traffic from being processed on the network or dropped beforehand. Hence it is imperative to use a good mitigation technique to distinguish between malicious and regular traffic. Over the past few years, many techniques have been discovered and worked on [18] [19] [20], such as machine learning and even blockchain. It has become harder to successfully detect malicious traffic since the attackers use spoofed information and other hiding techniques to hide their real identity and make it look natural. Another reason why these attacks become harder to mitigate is the sheer amount of traffic attackers can generate. With various botnets around

and more IoT devices that can be infected, the amount of traffic attackers can generate in these attackers is ever-growing.

Programmable network devices have the advantage, that the attack surface can be limited by only allowing what is necessary on the servers. If a protocol is not implemented, it also cannot be exploited and even if it is present, small modifications can be included to make it more secure. Programmable devices also save resources and are therefore faster than the normal devices that are mostly used today. The following paragraphs will discuss various techniques that exist to detect and mitigate these attacks and some drawbacks and advantages of them.

3.1 Heavy-hitter detection for general DDoS attacks

As one of some naive solutions to withstand denial of service attacks, one could upgrade the network's bandwidth to make it ready to accept more traffic at once. However, an attacker could quickly increase the amount of attack traffic and make the upgrade pointless. Hence, more elaborate defence strategies are needed. One of those strategies could be a firewall, which controls a network's incoming and outgoing flow by monitoring the packets that flow within and blocking malicious traffic from entering the network due to predefined rules and policies. However, a firewall can only block the malicious traffic up to a certain point defined by its rules. If there is both malicious and regular traffic coming in from a specific port, it is hard not to block any regular traffic next to the malicious one. The main problem is how to distinguish between legitimate and malicious traffic, since the traffic coming from the attack might look very similar to the normal one. Additionally, the firewall needs to be placed at the correct place in the network to be effective. If it is too far into the network, the malicious traffic could already take down the network before it even gets to the firewall.

Another solution against DDoS attacks is upstream filtering. In this solution, all the traffic first passes through an external filter system. This filter changes, for example, the target's IP address in the DNS system, tunnels the requests or uses other mechanics to filter out the bad traffic before it can get to the actual target. Since these filters need to be adopted on more extensive networks to function since they mainly redirect traffic to keep the flow going, control over the bigger network is needed to incorporate these systems. So passing the traffic around within the network and distributing the load is effective if the network is big enough and robust enough to support the load until it can be filtered.

Rate-limiting can be an effective strategy for reducing the traffic sent from a source if that source can be accurately tracked down. By imposing a limit on how many messages a single source can send within a specific timeframe, the attacker cannot exceed the threshold, and all messages sent above the threshold will be automatically dropped. When the threshold is exceeded multiple times, a ban for the whole source can also be put in place, which will stop the source from sending any other messages in the future permanently. However, this technique is only applicable when the source can be accurately identified. A rate limit cannot be imposed as soon as a source uses spoofed information such as source IP address or source port. So within a network where every host is well identified, this can be a valid mechanism, but as soon as sources from the outside can send traffic, rate-limiting becomes powerless.

By applying machine learning to the security aspect of networks, a new technique has been developed recently that could learn to identify attacks based on previous ones that have been mitigated. By training an algorithm with data from previous attacks, a machine learning algorithm might be able to detect incoming attacks faster than any other built-in mechanism in the network [13] [18]. These papers have shown that up to 97% of network anomalies could be detected using a trained machine learning algorithm, which proves that machine learning is indeed a valid approach to network security.

There are also other mechanics, which have been studied in more detail as well, such as the algorithm proposed in [6], which uses entropy to detect attacks. During an attack, certain IP addresses might get more traffic than others, which increases the entropy of those destinations. Once the entropy exceeds a certain threshold, the traffic is flagged as malicious, and actions may be taken. Suppose a single IP address receives only a few requests. In that case, the entropy might also already be bigger than those compared to others. This is why dynamic thresholds are put in place, such that minor deviations of legitimate traffic are not flagged as malicious. In P4, the implementation works by first parsing the incoming packets, then using match-action pipelines to determine egress ports, and performing the entropy estimation. The entropy estimation is performed by applying custom hash functions to the packet information and obtaining estimates for the various frequency estimations. The traffic is then analysed based on observation windows, and if there is a threshold that is exceeded, an alarm is set off and the mitigation sets in.

There is also an extension to the previous mitigation technique, described in [7], which does not only take into account source addresses for anomaly detection but also destination addresses. So ddosm-p4 also uses entropy to identify anomalies within the traffic, but it uses more fields of the packets. So it first parses the incoming packets and then applies the entropy calculation during match-action pipelines. As in the previous algorithm, when the entropy exceeds a certain threshold, the traffic is flagged, and mitigation mechanisms are put in place. Other than ddosd-p4, this algorithm works with fixed-point arithmetic since P4 does not support floating-point arithmetic. The other implementation uses bit shifts and integers only, whereas here, integers and decimal parts are stored as bits, which allows for higher precision.

3.2 Heavy-hitter detection for SYN-flood attacks

Part of some naive mitigation strategies for SYN-flood attacks is also just increasing the number of connections the server can make. In this case, more connections can be opened, and the backlog will not be full as quickly. However, since it is also easy for an attacker to also scale up the amount of TCP SYN packets sent, this method proves to be not very efficient in mitigating the attacks. Another method which sounds helpful, but can be easily beaten by the attacker by scaling up the attack volume, is decreasing the amount of time a connection is kept alive between sending the SYN-ACK and waiting for

the ACK. If the connection is not open for as long, a new connection can be made again more quickly. However, this might have a drawback for users with a slower connection speed since their connection might sometimes be dropped even though it is legitimate. A last naive strategy is replacing the oldest alive TCP connection with the newest incoming one. This mechanism has the advantage that every request coming in will be served with a connection. However, it also means that users who are currently connected might have their connection revoked if the backlog is full and their connection is the oldest. Hence these methods might be helpful on small scales and might be able to help against minimal attacks. However, as soon as the attacker can also scale up the attacks, which is almost always the case, these defence mechanisms become useless.

However, some more advanced mitigation techniques exist that are more effective against SYN flood attacks; one of them is SYN cookies. Instead of opening a port immediately when a SYN request is received, a cookie is created using a hash function, and a secret key is stored on the server. This cookie is then sent to the user with the SYN-ACK request, and the connection is only opened when the final ACK is received with the valid cookie. If the cookie corresponds to the one created during the SYN request, the connection is opened, and the TCP handshake was successful. A normal SYN-flood attack does not work anymore since the complete handshake needs to be completed to use the network's resources. However, these cookies need to have some extra security features to prevent being exploited. First of all, they need to expire after a certain time, or else the attacker might be able to use them repeatedly. Another aspect they need to follow is that they should only be valid for the machine/user that sends the SYN request. Otherwise, the attacker could exploit it by sending ACK requests with the same cookie from multiple machines.

Another mitigation technique against SYN-floods is a proxy or firewall, which filters out malicious traffic before getting to the server. The initial TCP handshake is performed on the proxy, which means that the server does not know about the attempt of a user to connect until the proxy or firewall decides that the traffic is legitimate. If the traffic is considered non-malicious, the proxy forwards the request to the actual server, establishing the connection. This technique has the advantage that the malicious traffic might never reach the actual server, and hence the server will not be slowed down or even taken down by an attack. However, the firewall might let some malicious traffic pass through if it is not configured correctly. An attack might also take down the firewall, and in that case, the firewall or proxy will not be able to communicate with the server. Hence this mechanism might only prove helpful if the proxy and firewall are configured correctly.

Some of these mechanisms have been implemented already by [8] and [9] and have proven to be effective. The mechanisms implemented in these papers are all based on a proxy, which filters the traffic and only sends non-malicious traffic to the actual destination. They created four different implementations, all based on the same system, namely a whitelist and forwarding. The first implementation is the SYN cookie, which creates a cookie on the SYN request of the user and stores it on the proxy. The user and the proxy then complete the handshake, and as soon as this is done and the cookie is valid, the IP is whitelisted, and the proxy does the handshake with the server and completes the connection with the user. The second implementation is the SYN authentication reset bloom filter method, which sends an RST packet as a response to the first SYN of the user. The IP is then whitelisted, and when the user tries to reconnect, the request is accepted, and the proxy forwards the request to the server and thus creates the connection. The third implementation is the SYN authentication cookie, which creates a cookie if the IP is not whitelisted yet. If the cookie is valid in the ACK request, the proxy sends an RST request to the user and terminates the connection, but the IP is whitelisted. If the user reconnects with the same IP, since the IP is already whitelisted, no cookie is sent, and the connection is established. The fourth and final implementation is the SYN authentication reset digest method, which is similar to method two. However, a digest is used instead of a bloom filter to whitelist the IP.

4 Experiments and Results

To test out certain strategies to overcome attacks, the main idea of this research was to verify a proof of concept and later discuss certain improvements based on the results and already existing research. A proof of concept was decided upon before the actual implementation started, based on previously known facts and weaknesses of the attacks. The following sections describe the implementations and also compare them to more sophisticated techniques from other research.

4.1 DoS attacks

Main idea

Since flood attacks usually consist of the same requests repeated many times, the main idea for this proof of concept implementation was to count the incoming requests and as soon as the amount exceeded a certain threshold, the next packets were dropped. This idea was tried before [21] [22] [23], however, these were mainly implemented in other SDN and without P4, or were focused on different kind of attacks, like an amplification attack. To make it possible to compare the results to other research, the main assumption for this proof of concept was that the traffic was coming from a single source and that the origin information, such as source IP address and source IP port, were not spoofed. This is of course not a realistic attack, since today most DoS attacks as distributed over various machines coming from botnets and also very often use spoofed information in the packet headers. However, if this proof of concept proves to be effective for this simple attack scenario, then it might be viable to scale it up to the distributed and spoofed attacks by modifying certain settings within the detection mechanism itself and using other fields than the source information. These modifications could range from simply changing the thresholds to using formulas and other counting procedures, such as a sliding window protocol to make the detection even more efficient.

Implementation details

To implement the proof of concept, a hash-map was used to count the occurrences of the packets. Each entry in the

hash-map contains the occurrences of the packet information hashed and once it exceeds a certain threshold, the future packets which get the same hash-index are dropped. In this simple proof of concept, the source IP address, destination IP address, source port and destination port were hashed as an object and that index was used to count the occurrences. So for each incoming packet, the hash-index is calculated, then the value in the hash-map is read and incremented and then the packet is either dropped if the threshold is exceeded or processed if not. The main difference between this proof of concept implementation and the other papers that are compared, is that the proof of concept uses a simple counter which is incremented per packet, whereas the other implementations keep track of the entropy of the whole topology at a time. So the proof of concept is therefore more focused on a single host rather than a whole network.

The implementation was run on a network topology as describes in Figure 3, where one host was the target, one host the attacker and one host which measured responsiveness of the target during an attack. During an attack, the third host was sending ping requests to the target in order to check its responsiveness. These pings were sent from a Python script and later on graphed to compare them. Next to the response times, the throughput was also tested and compared. The following scenarios were considered and compared:

- Response time with no attack
- Response time during an attack with no mitigation
- Response time during an attack with proof of concept mitigation
- Response time during an attack with ddosd-p4 mitigation [24]
- Response time during an attack with ddosm-p4 mitigation [25]

To simulate the attack, the hping3 tool [17] was used. Since the attack was simulated on a virtual machine, the attack quantity was capped at 100 requests per second, using the following request: 'sudo hping3 –faster -p <destination port> -s <source port> –keep <target IP>'.

Results

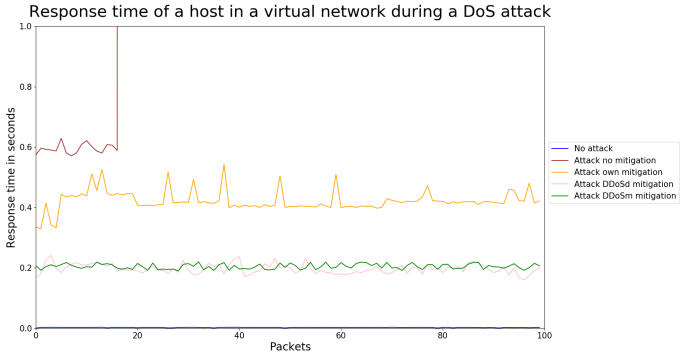

Figure 4: Graph showing the response times of the network during a DoS attack with various mitigations in place

The graph Figure 4 shows the response time of a packet on the Y-axis and the packet numbers on the X-axis. Each run contains 100 packets and for every packet a response time is registered. The results can be found in Table 1. These results were obtained after running every test 10 times and averaging the results of all the runs. The different sub-graphs can be found in Appendix A for a more detailed view.

During an attack with no mitigation in place, the response time is way higher than without an attack and a complete denial of service occurs almost every single time. In this case it is hard to give an average response time, since only a few pings make it through before the network becomes completely unresponsive, but it would be around 0.6 - 0.8 seconds. The average bandwidth is also affected by the attack, and the average throughput is only 2.67 Mbits/sec.

The proof of concept mitigation improves the response time of the network during an attack to 0.43 seconds. This is already faster than without a mitigation in place, but the main improvement is that the network never becomes completely unresponsive. The response time is constant and over various runs, the network always responded to every ping and no regular packets were dropped. The throughput is only slightly higher than without a mitigation but the constant availability of the network even during an attack makes up for it.

The DDoSd-P4 mitigation [6] has an average response time of 0.19 seconds and also never becomes completely unresponsive. No non-malicious packets are dropped and the response time is pretty constant. The improvement over the proof of concept implementation could be that the actual calculation of the entropy is more efficient than calculating the hash-index for every packet. So this mechanism proves to be effective against these kinds of attacks with good response times and no down time at all. This also shows in the average throughput, which is 8.31 Mbits/sec during and attack.

For the DDoSm-P4 implementation [7], the average response time is 0.205 seconds and the average throughput is 9.43 Mbits/sec. These numbers are similar to the DDoSd mitigation, which was to be expected, since DDoSm is an extension of DDoSd. The main difference in the response time might come from the extra checks and table lookups that DDoSm does compared to DDoSd. With this simple attack example it is quite hard to say which of the two implementations would be better in a real-life scenario, but given that the DDoSm implementation takes more packet parameters into account and also uses fixed-point arithmetic, the first guess would be that DDoSm would perform better than DDoSd. To confirm this, more tests would need to be conducted but that is outside of the scope of this project.

So overall the proof of concept mitigation technique works in detecting and mitigating a very basic denial of service attack which floods the network with requests. There exist more efficient algorithms, such as DDoSd and DDoSm, which are also more suited for real-life applications, however the proof of concept could be adapted to fit more onto real-life applications and could also be optimised. A discussion about this can be found in section 5.

Table 1: Results from the different mitigation techniques against DoS attacks

	Average response time	Average throughput
No attack	2-3 milliseconds	53 Mbits/sec
Attack no mitigation	up to 1 second before DoS	2.67 Mbits/sec
Attack own mitigation	0.43 seconds	2.92 Mbits/sec
Attack DDoSd mitigation	0.19 seconds	8.31 Mbits/sec
Attack DDoSm mitigation	0.205 seconds	9.43 Mbits/sec

4.2 SYN-flood attacks

Main idea

The proof of concept mitigation technique against SYN-flood attacks builds upon the same idea as the previous one. Namely a hash-map implementation, which counts the amount of SYN packets from a certain source and if they exceed a certain threshold, they are dropped. The difference in the hashing mechanism is that also the type of packet is taken into account, such that if a client's SYN packets are dropped because the threshold was exceeded, an ACK packet would still be accepted and a connection could be opened.

Implementation details

Similarly to the DoS proof of concept implementation, this one also uses a hash-map to count the occurrences of packets. From each incoming packet, the source and destination IP address and port will be used, as well as the TCP SYN and ACK flags, to generate a hash-index. Once the occurrences exceed a certain threshold, the packets will be dropped and not processed any further. Since the TCP flags are used in this scenario, a SYN packet might be dropped, whereas an ACK packet with the same source and destination information might still be processed.

The implementation was run on a network topology as describes in Figure 3, where one host was the target, one host the attacker and one host which measured responsiveness of the target during an attack. During an attack, the third host was trying to complete a TCP handshake with the target and send a request over. This procedure was automated using a Python script and the duration of it was measured and graphed to be compared with other techniques. The following scenarios were considered and compared:

- Duration with no attack
- Duration during an attack with no mitigation
- Duration during an attack with proof of concept mitigation
- Duration during an attack with SYN-proxy mitigation [8] [9]
  - SYN-cookies
  - SYN-auth-cookies
  - SYN-auth-reset-bloomfilter
  - SYN-auth-reset-digest

To simulate the attack, the hping3 tool [17] was used. Since the attack was simulated on a virtual machine, the attack quantity was capped at 100 SYN requests per second, using the following request: 'sudo hping3 -S –faster -p <destination port> -s <source port> –keep <target IP>'.

Results

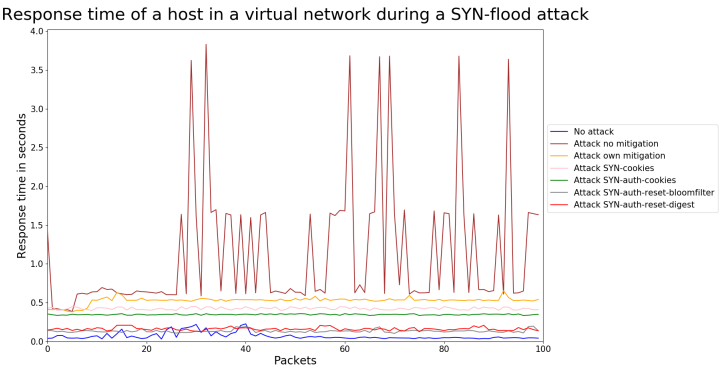

Figure 5: Graph showing the duration to complete a TCP handshake during an attack with various mitigations in place

The graph Figure 5 shows how long it took to get a SYN-ACK response from the server by sending a SYN-packet. For each run, 100 SYN-requests were send to the server and the duration to get a SYN-ACK request back is graphed. The Y-axis shows the duration and the X-axis the packet number. The results can be found in Table 1. These results were obtained after running every test 10 times and averaging the results of all the runs. The different sub-graphs can be found in Appendix A for a more detailed view.

If a SYN-flood attack is happening on this network and there is no mitigation in place, then the average duration is takes to get a SYN-ACK response to an ACK request is around 1.07 seconds. This is due to the fact that all the ports are occupied and the packet needs to wait until one gets freed before a response gets send. The bandwidth also suffers from the attack and drops from 57.03 to 2.47 Mbits/sec on average. The main difference compared to the DoS attack is that the network never became completely unresponsive, even though the response times were slower in this case.

With the proof of concept mitigation in place, the SYN-flood attack does not have as big of an effect on the network as without a mitigation. With the mitigation in place, the average response time rises from 56 milliseconds without an attack to 0.52 seconds during an ongoing SYN-flood attack. This is twice as fast as without a mitigation in place, so it is a working solution. The bandwidth is not much higher with 3.18 Mbits/sec with mitigation compared to 2.47 Mbits/sec without mitigation.

For the implementations with the SYN-proxy, the average response times range from 0.13 seconds for the SYN-Auth-Reset-Bloomfilter mitigation to 0.42 seconds for the SYN-

Table 2: Results from the different mitigation techniques against SYN-flood attacks

	Average response time	Average throughput
No attack	0.056 seconds	57.03 Mbits/sec
Attack no mitigation	1.07 seconds	2.47 Mbits/sec
Attack own mitigation	0.52 seconds	3.18 Mbits/sec
Attack SYN-Cookies mitigation	0.42 seconds	2.75 Mbits/sec
Attack SYN-Auth-Cookies mitigation	0.35 seconds	2.25 Mbits/sec
Attack SYN-Auth-Reset-Bloomfilter mitigation	0.13 seconds	4.98 Mbits/sec
Attack SYN-Auth-Reset-Digest mitigation	0.15 seconds	4.25 Mbits/sec

cookies mitigation. The average throughput is 2.25 Mbits/sec for the SYN-Auth-Cookies mitigation and 4.98 Mbits/sec for the SYN-Auth-Reset-Bloomfilter mitigation. The other techniques are in between these values. The similar results for the SYN-cookies and SYN-auth-cookies mitigation can be explained by the similar behaviour the 2 techniques have. In both cases, a cookie is generated on the first SYN-ACK packet, and if the final ACK contains the correct cookie, the connection is established. The only difference is the whitelist, which is only present in the SYN-auth-cookies mitigation, but not in the other. Similar results can also be observed for the last 2 SYN-proxy implementations. Since the only difference is how the presence in the whitelist is checked, so by using a bloomfilter or digest, the results are very close to each other with the bloomfilter performing a bit better in both duration of the TCP handshake and throughput. Overall, the SYN-reset implementations perform better than the SYN-cookie implementations, which might be the case since the cookie is calculated for each SYN packet, whereas if the IP is whitelisted on for the SYN-reset implementations, no further overhead is produced.

4.3 Availability

All the proof of concept implementations mentioned before, as well as the measuring scripts and topologies can be found on [26].

5 Discussion

Since the proof of concept implementations have proven effective against some simple attacks, the hypothesis that these mechanisms can be scaled to more sophisticated attacks remains valid. By tweaking and changing how the detection and mitigation algorithms work, these techniques might remain resistant to other attacks. The main idea of counting packets and comparing the number to a certain threshold is simple but effective on paper. By adding some sort of timer to decrement the counts after a while, this mechanism could already be improved since a malicious actor might just use an IP since the machine was infected. This would prevent normal users from having their machines locked out of the network forever if they were hacked and their machine abused for an attack. These ideas could however not be further tested during this project, due to a lack of time caused by problems that arose during other parts of the project.

The packet data allows the server to figure out the request and where it is coming from. With P4, it is pretty easy to define custom packets, which means that the complete network protocol can be customised. With this customisation, the security can also be improved since if an unknown packet arrives at the network, it can be dropped and not processed further. The known packets need to follow strict guidelines and can be adapted so that they cannot harm the server and can be easily analysed for attack detection.

In general, a combination of multiple detection and mitigation techniques might lead to better defence. However, the detection time should not be too long because otherwise, the actual mechanism might be too slow, and the attack might take down the network before it can react. So there needs to be a link between reactivity and security.

6 Conclusion and Future Work

With the growing rate of attacks and them becoming more and more sophisticated, the defence mechanisms also need to become stronger. Programmable data planes allow developers to customise their network protocol by defining custom headers for the packets. They let them decide what to do with packets that are not recognised or do not follow the rules. With this freedom, the security of networks can be improved since the users can decide themselves what their network should do and how.

P4 is a suitable programming language to implement these mitigation and detection algorithms, with the proof of concepts being proven effective. A significant amount of research in this field has already been done, and the results are very promising. The comparisons of the different implementations have shown that some algorithms are better than others. Their various functions can be extracted and used for other possible algorithms by analysing them in more detail.

Some of the algorithms share certain features, such as counting packets. This brings up whether it would be possible to train a machine-learning algorithm to detect these attacks. There has already been some research done in this field. However, it remains an open question if these machine learning algorithms will become more effective in mitigating the attacks than the current implementations in our networks.

Either way, with the recent developments of new attacks emerging, the defensive mechanisms also need to evolve and become more robust to withstand future attacks. Programmable data planes and the programming language P4 allow for these changes to be possible.

7 Responsible Research

7.1 Reproducibility

The various sections include all the steps to reproduce the before-mentioned results. To summarise everything, the VM is the one from [27], so it is an Ubuntu 20.04 OS, running with eight cores and 8192 MB of RAM. The Python version for the scripts is 3.8.10. Mininet version 2.3.0 is used, and the behavioural model and the P4 compiler were the latest builds from the mentioned repositories. Lastly, hping3 version 3.0.0-alpha-2 was used.

To run the code, a mininet topology should be run with one switch and three hosts, as described in Figure 3, and then the runtime rules defined by the P4 programs should be added to the topology. Inside mininet, a session for the attacker and the pinging host should be started, and from the attacker's console, the attack should be started. The pinging host can run the Python script to ping the target or establish a TCP connection when the attack is running. Once the 100 pings or handshakes are complete, the values will be written to a file and can be graphed and analysed later on.

7.2 Ethical aspects

The attacks and other tools used in this research could be used to harm networks or other machines. Everything described in this paper has been used solely to perform academic research, get insights into the mechanisms, and use those results to draw a conclusion. No attacks on existing networks were performed, and everything was kept within a sandboxed environment in a virtual machine.

A Sub-graphs

For each scenario, around 10 runs were made to get an average result of each technique. In the following graphs, 2 out of those runs were chosen per scenario to visualise them.

A.1 DoS attacks

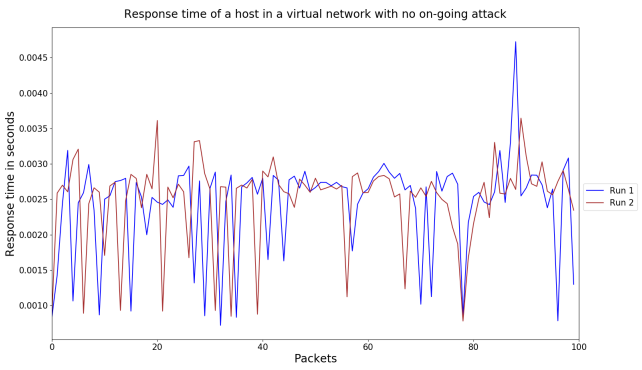

Figure 6: Average response time in a virtual network with no on-going attack

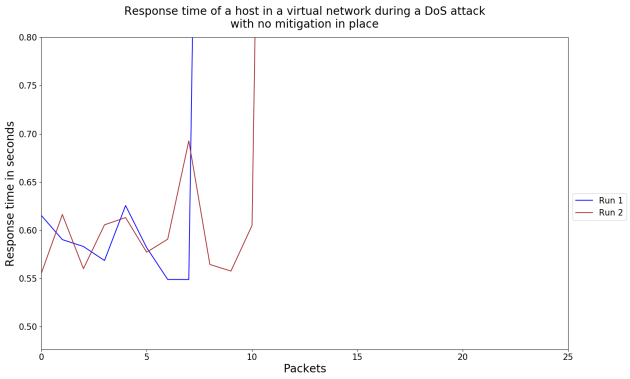

Figure 7: Average response time in a virtual network with an on-going attack and no mitigation in place

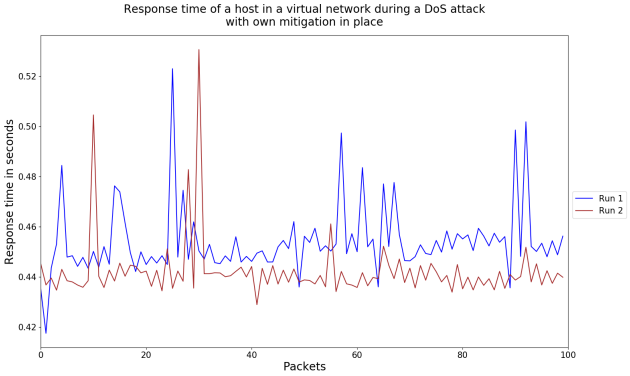

Figure 8: Average response time in a virtual network with an on-going attack and the proof of concept mitigation in place

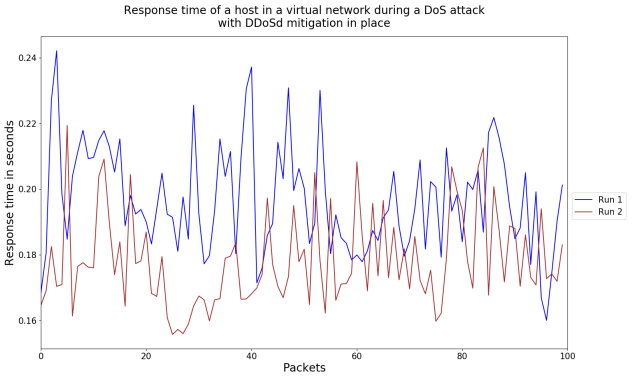

Figure 9: Average response time in a virtual network with an on-going attack and the DDoSd [6] mitigation in place

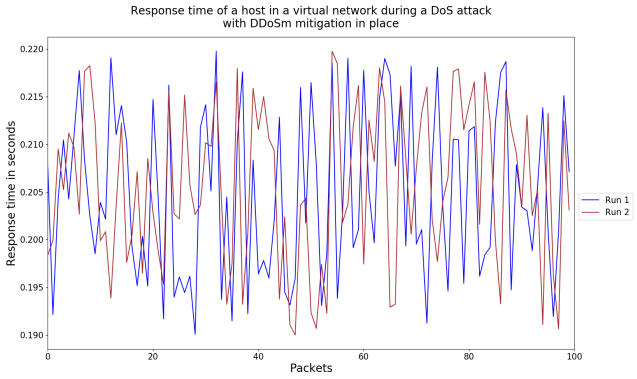

Figure 10: Average response time in a virtual network with an on-going attack and the DDoSm [7] mitigation in place

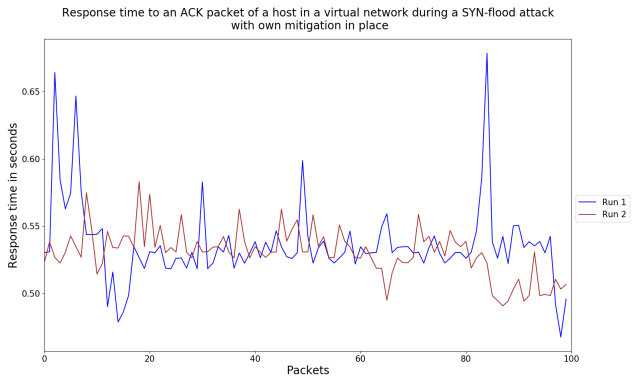

Figure 13: Average response time in a virtual network with an on-going attack and the proof of concept mitigation in place

A.2 SYN-floods

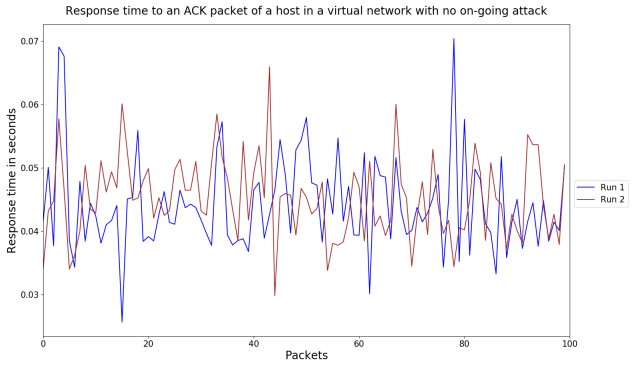

Figure 11: Average response time in a virtual network with no on-going attack

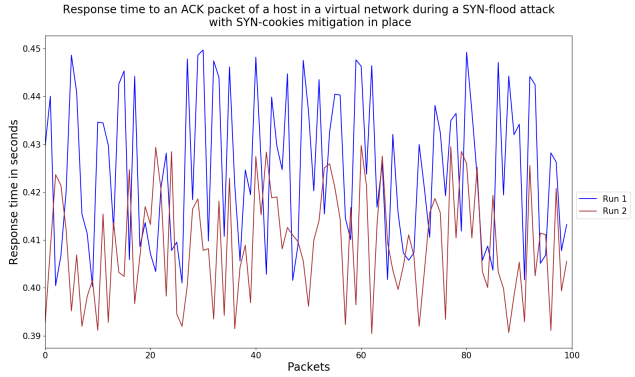

Figure 14: Average response time in a virtual network with an on-going SYN-flood attack and SYN-cookie [8] [9] mitigation in place

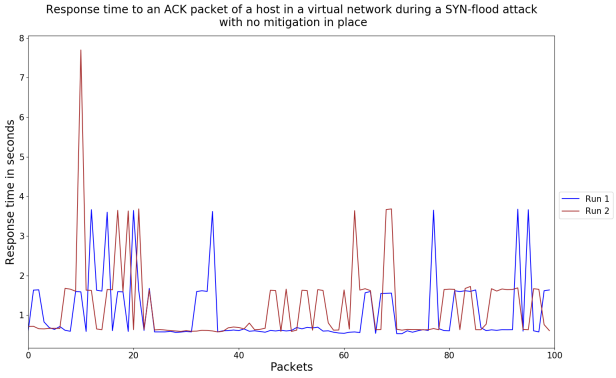

Figure 12: Average response time in a virtual network with an on-going attack and no mitigation in place

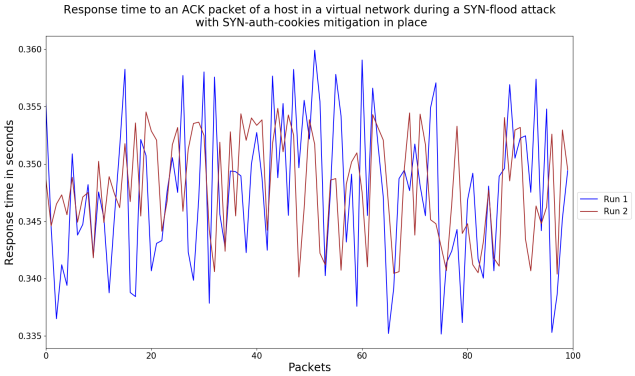

Figure 15: Average response time in a virtual network with an on-going attack and the SYN-auth-cookie [8][9] mitigation in place

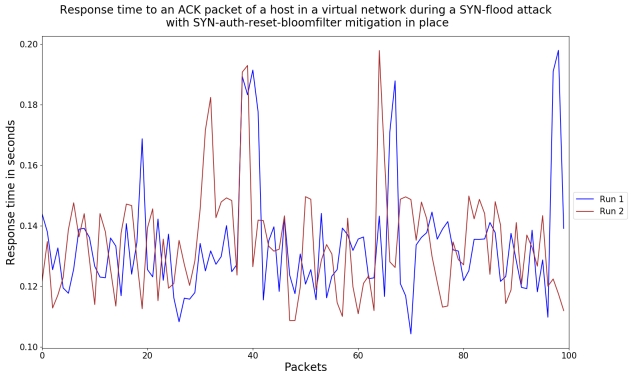

Figure 16: Average response time in a virtual network with an ongoing attack and the SYN-Auth-Reset-Bloomfilter [8][9] mitigation in place

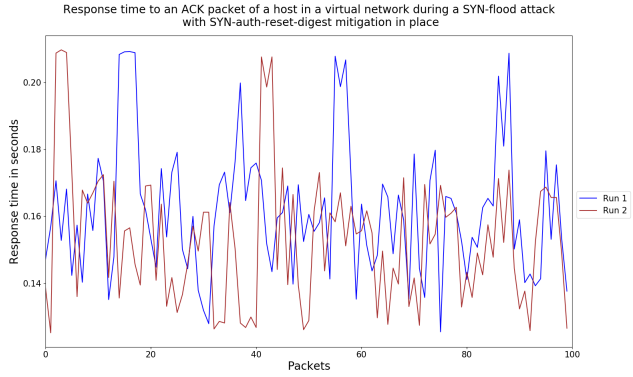

Figure 17: Average response time in a virtual network with an ongoing attack and the SYN-Auth-Reset-Digest [8][9] mitigation in place

References

[1] A. Wang, W. Chang, S. Chen, and A. Mohaisen, "Delving into internet ddos attacks by botnets: characterization and analysis," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2843–2855, 2018.

[2] S. Behal and K. Kumar, "Characterization and comparison of ddos attack tools and traffic generators: A review.," *Int. J. Netw. Secur.*, vol. 19, no. 3, pp. 383–393, 2017.

[3] C. Cimpanu, "Google says it mitigated a 2.54 tbps ddos attack in 2017." https://www.zdnet.com/article/google-says-it-mitigated-a-2-54-tbps-ddos-attack-in-2017-largest-known-to-date, Accessed on 19-05-2020, Oct 2020.

[4] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with p4: Fundamentals, advances, and applied research," *arXiv preprint arXiv:2101.10632*, 2021.

[5] "P4 open source programming language." https://p4.org/, Accessed on 19-05-2022, 2022.

[6] Â. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 19–27, IEEE, 2019.

[7] A. da Silveira Ilha, Â. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Euclid: A fully in-network, p4-based approach for real-time ddos attack detection and mitigation," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3121–3139, 2020.

[8] D. Scholz, S. Gallenmüller, H. Stubbe, and G. Carle, "Syn flood defense in programmable data planes," in *Proceedings of the 3rd P4 Workshop in Europe*, pp. 13–20, 2020.

[9] D. Scholz, S. Gallenmüller, H. Stubbe, B. Jaber, M. Rouhi, and G. Carle, "Me love (syn-) cookies: Syn flood mitigation in programmable data planes," *arXiv preprint arXiv:2003.03221*, 2020.

[10] "P4_16 language specification." https://p4.org/p4-spec/docs/P4-16-v1.2.2.html, Accessed on 19-05-2022, May 2021.

[11] "P4lang/p4c: P4_16 reference compiler." https://github.com/p4lang/p4c, Accessed on 19-05-2022, 2022.

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

[13] K. Kostas, "Anomaly Detection in Networks Using Machine Learning," Master's thesis, University of Essex, Colchester, UK, 2018.

[14] K. K. N. Tiwari and R. Kumar, "Denial of service attack using slowloris," *ISO 9001:2008*, vol. 07, July 2020.

[15] F. Yihunie, E. Abdelfattah, and A. Odeh, "Analysis of ping of death dos and ddos attacks," in *2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pp. 1–4, 2018.

[16] "Mininet: An instant virtual network on your laptop (or other pc)." http://mininet.org/, Accessed on 19-05-2022, 2022.

[17] "Hping security tool." http://www.hping.org/hping3.html, Accessed on 19-05-2022.

[18] F. Musumeci, A. C. Fidanci, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-enabled ddos attacks detection in p4 programmable networks," *Journal of Network and Systems Management*, vol. 30, no. 1, pp. 1–27, 2022.

[19] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, and K.-K. R. Choo, "P4-to-blockchain: A secure blockchain-enabled packet parser for software defined networking," *Computers & Security*, vol. 88, p. 101629, 2020.

[20] R. Poddar and H. Babu, "Decision tree based iot attack detection in programmable data plane using p4 language," in *International Conference on Advanced Information Networking and Applications*, pp. 671–683, Springer, 2022.

[21] X. Z. Khooi, L. Csikor, D. M. Divakaran, and M. S. Kang, "Dida: Distributed in-network defense architecture against amplified reflection ddos attacks," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pp. 277–281, IEEE, 2020.

[22] J. Boite, P.-A. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "Statesec: Stateful monitoring for ddos protection in software defined networks," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–9, IEEE, 2017.

[23] F. Rebecchi, J. Boite, P.-A. Nardin, M. Bouet, and V. Conan, "Ddos protection with stateful software-defined networking," *International Journal of Network Management*, vol. 29, no. 1, p. e2042, 2019.

[24] "Ddosd-p4 repository." https://github.com/aclapolli/ddosd-p4, Accessed on 29-05-2022, 2022.

[25] "Ddosm-p4 repository." https://github.com/asilha/ddosm-p4, Accessed on 29-05-2022, 2022.

[26] "Paper github repository." https://github.com/FBroy/DMMA_PDP, 2022.

[27] "P4 tutorial." https://github.com/p4lang/tutorials, Accessed on 29-05-2022, 2022.