Department of Informatics
King's College London
United Kingdom

7CCSMPRJ/7CCSMUIP Individual Project

DoS defence of IoT devices at the edge layer using P4

[redacted]
Course: MSc. Cyber Security

[redacted]

This dissertation is submitted for the degree of MSc in MSc. Cyber Security.

[redacted]

[redacted]

Abstract

Technological advancements led to a rapid increase in the number of IoT devices. These have different uses, from day-to-day wearable devices to sensors, with industrial use that regulates critical national infrastructure. Said devices use different protocols and ways of operating based on their use. Still, most have a lightweight and mobile architecture in mind, making them vulnerable to a wide range of malicious attacks due to the lack of resources and security controls. This leads to IoT devices being susceptible to malware that uses them as bots. The rise of botnets and their efficiency in conducting attacks can cause severe damage, the Mirai botnet being a valid example.

To prevent such attacks, researchers use the IoT ecosystem to their advantage. Enabling a programmable data plane at the edge switch in the fog layer that can inspect packets can prevent attacks by denying the edge layer devices from communicating with malicious IP addresses and securing the edge devices from various DoS attacks.

Denial of Service has become very popular and easy to conduct, with the number of attacks increasing daily and causing financial damage to both the private and the public sectors. Denial of Service attacks can also be used to attack IoT devices and render communication to them useless for some time or weaken them for an attack that can transform the said device into a botnet and possibly gain an entry point in a local network.

Virtual networks allow us to create a controlled and monitored environment where the traffic between devices is expected to behave in a certain way. This enables us to create a realistic environment, without the need for devices, with functional hosts and switches that simulate both IoT targets and botnets, with a programable switch inspecting packets travelling between them. This technology is used for setting up an ideal lab for monitoring network packets with little computational need.

This project uses a programmable P4 switch in a virtual network environment. The most common DoS attacks are carried out on virtual hosts, and the switch needs to mitigate these attacks. Besides mitigating these attacks, the switch can also prevent inward and outward communication with specific IP addresses that can be malicious. These addresses can represent specific C&C servers that want to communicate with already infected devices in the network or external botnets that conduct attacks. Both features are used together to create a lightweight intrusion detection framework capable of blocking ICMP flooding, TCP SYN flooding, and ping of death attacks.

Contents

List of Figures

1 Introduction

The Internet of Things is a novel paradigm shift in the IT arena[1]. It is a network containing lightweight, affordable and mobile devices capable of communication through different protocols. These devices are alluring to attack agents due to their computational restrictions. IoT devices can be a target of lots of different types of attacks, and malware can be injected into them following such an attack. These infected IoT devices can become botnets under the control of an attacker, which can further spread malware and attack devices. Such an example was the Mirai Botnet that managed to infect at its peak 600 thousand devices [2] and cause significant damage by launching DDoS attacks on specific organisations. It is estimated that botnets cause 110 billion us dollars in damage yearly[3]. Due to these facts, there is a remarkable arms race between malicious agents and researchers to find suitable techniques to mitigate the propagation of botnets in the booming market of IoT devices that could generate a market of up to 14.4 trillion us dollars[4]. The monetary values indicate the importance of the automated cyber defence of these numerous devices.

DDoS is the most common type of attack encountered to be effectuated by infected IoT botnets[5]. DDoS attacks are also carried out on IoT devices to gain access by infecting them by brute forcing or injecting malicious code. So a novel way of mitigating these attacks must be found to deter the rise of IoT botnets and their efficiency in carrying attacks and spreading.

Creating a safe environment where botnets and their attacks can be emulated and studied is a critical step in creating a viable mitigation strategy. Simulating such an environment with real-life equipment is costly, so using virtualisation instead is more feasible. Virtualisation is commonly used in the cyber security field to study malware and forensics further[6]. This project uses a tool called Mininet that allows us to create a virtual network with interactive components, allowing us to customise them at a basic level. Along with a packet sniffing tool WireShark, the interaction between hosts in the VN (virtual network) can be monitored and analysed more in-depth. These tools allow us to create a perfect environment to simulate IoT devices and the communication between them, how specific attacks affect them, and what mitigation techniques are efficient in preventing the spread of a botnet and its attacks.

Due to the architecture of the IoT ecosystem, a robust product able to defend from attacks and be placed at the sensing layer is unfeasible, and products at the cloud layer are not able to protect the device itself from being infected or attacked but can quarantine botnets. Thus the edge layer can be fitted with a mitigation mechanism using a programable data plane [7]. In this project, we will use P4, a programming language that is able to be deployed on a switch and filter packets and modify them based on the running program. A problem with P4 by itself is that it is hard to create a program that is aware of its state, so alongside it, a Thrift controller will be used to allow the P4 configured switch to become stateful and thus be wary of previous packets and rules added during the running of the environment.

1.1 Aims and Objectives

This project aims to create a lightweight product that can be deployed at the network edge and significantly affect the efficiency of Infrasture Layer Based DDoS attacks that target IoT devices and quarantine suspicious devices. This framework will negate Ping of Death attacks, ICMP flooding, TCP-SYN flooding attacks, as well as blacklist malicious IP addresses. P4 will be used to create a script capable of doing such, and Wireshark will be used to test if the implemented technique is a valid mitigation technique.

1.1.1 Deter communication with malicious IPs

For an effective way of stopping the spread of botnets, a certified way is by severing the communication between the botnet and the C&C server, basically isolating the tool from the malicious user. This is a crucial step in taking down a network of botnets[8]. This program implements a blacklist that can be populated with already-known malicious IPs; by doing this, possible infected devices communicating through the configured switch will not reach the C&C servers and vice versa, being quarantined to a certain point. The blacklist can also be updated at runtime to block agents that transfer malicious packages, possibly ending an ongoing attack.

1.1.2 Mitigate Ping of Death attacks

Ping of Death attacks can be severe on IoT devices. These attacks send a malformed ICMP packet that exceeds the expected size accepted by IPv4 standards. This malformed packet is classified as a DoS attack, but in some cases, it can also modify the memory of a poorly designed device. Due to the limitation of IoT devices, these are classified as a prime target for this type of attack[9]. The implemented framework uses the ability of P4 to inspect packets to its advantage and discards these malformed packages.

1.1.3 Mitigate ICMP flooding attacks

ICMP flooding is relatively easy to conduct. Even if it is not the most efficient type of DoS attack, by using amplification methods or using botnets to run it, it can still cripple the communication capability of a device, especially an IoT one, with bandwidth limitations and power limitations[10]. For detecting and mitigating such an attack, packets need to be inspected, analysed, and the malicious ones need to be dropped[11]. The framework is capable of doing this using a P4 bloom filter that performs packet inspection, and based on its outcome, packets are dropped or sent to their destination.

1.1.4 Mitigate TCP-SYN flooding attacks

TCP SYN flooding is the most common DoS attack conducted [12]. It exploits the TCP protocol to cripple the target. It is mainly used to power-starve a target and block its communication capabilities. Similar to ICMP flooding, packets need to be inspected and dropped. The packet inspection is more favourable due to the flags that can be found

in the packet that describe the stage of the handshake of the TCP protocol. P4 is very efficient in doing this due to its packet-sniffing properties.

1.1.5 Deployment and use

This product was developed to be deployed on a P4-capable switch. The design aims for a lightweight intrusion detection system. Alongside a controller, the software is capable of dropping malicious packets and also banning suspicious IPs. The current P4 program only allows for layer two traffic, which limits its intended purposes to just protecting IoT devices in a local network. Upon deployment, traffic between hosts is allowed and automatically monitored. Without the deployment of the controller, the mitigation would only be dropping malicious packets without the automatic ban of the attacker's IP.

2 Background and Literature Review

For protecting IoT devices and creating a viable framework for mitigating cyber attacks, we need certain information about the IoT ecosystem, as well as attacks and how they are carried out, and information about present cyber defence for these.

2.1 Internet of Things

There is no exact definition of the Internet of Things since the field is continuously expanding and developing. The first definition was categorising IoT devices as interoperable devices capable of communication with RFID technology by Kevin Ashton in 1999. [13].Nowadays, IoT devices are embedded devices capable of communication through internet protocols and are not directly operated by humans, having a certain autonomy[14]. One of the most significant drawbacks of this piece of technology is that the products have limitations in comparison to other internet-capable devices like personal computers or servers, limited battery life, limited bandwidth and limited processing power, needing special lightweight software for operating. The architecture of the IoT ecosystems consists of certain layers, each serving a purpose for interpreting and using data. The most common and widespread architecture consists of the device layer that represents a conglomeration of sensors and data collectors, the edge computing layer that provides a certain level of data analysis, the fog layer that can contain the edge layer and is also used for data processing, and the cloud layer, where data is stored, interpreted and further security implemented[15]. All these components work in unison, and the layer might differ based on the nature of the IoT and its purpose, as well as the use of the application taking information from the device.

For the purpose of this project, the focus is shifted to the edge layer that can be used to intercept communication between one or more IoT devices and their connection to the ethernet. There are a large number of devices that can be deployed on this layer, but the focus is on communication devices capable of monitoring or directing network traffic, such as switches, routers and controllers. A key factor to the project is programable switches, such as the ones that support the P4 language. This technology allows for greater security and improved functionality of IoT devices due to the programable aspect. As of the current factors, P4 has seen extended use and might soon become a staple for programmable switches due to its flexibility and its capacity to run on any device that supports it and protocol-agnostic goals[16].

Now with an overview of the IoT ecosystem, how it works, and edge layer capabilities, we need to analyse possible threats both to the data and to the device.

2.2 Attacks and risks

One of the biggest risks to IoT devices is botnets. Botnets are infected devices, sometimes even personal computers or servers. The malware affecting these are engineered in such a way that it conducts reconnaissance and infects other devices discovered through numerous means of attacks. These are primarily a big problem for IoT devices due to their

lightweight design. Constraints like limited processing mean that IoT devices can not use extensive cryptography used by standard devices for security purposes. This leads to lots of types of attacks that can compromise communication between the targeted device and other ones connected, and the packets be extracted through an eavesdropping attack. The lightweight cryptography can be broken, and the data interpreted from the packets. One more weak point of IoT infrastructure is that products come with a factory setting, which is usually widely available information[17]. This means that attacks can easily be carried out on new devices before proper configuration, and some credentials be used for security or through different communication techniques that are device dependent; the product can be reset to factory state, becoming vulnerable. Gaining access to credentials means that the product can be easily infected with malware transforming it into a botnet. The limitations regarding battery life mean that some IoT devices are susceptible to power starving attacks[18].Through these, certain key components of infrastructure can be disabled, such as cameras or critical sensors. The most known attacks for doing such are Denial of Service or Distributed Denial of Service attacks. This is not the only use of such attacks, and they have become widely used against IoT devices to weaken them for a much more severe attack that assumes control of the targeted device. DoS and DDoS attacks can also be used to brute force credentials; in the end, the attacker assumes control over the device. Due to these circumstances, the project will focus on combating these attacks. There are lots of types of denial of service attacks, but the focus is on ones that exploit the design of most used communication protocols.

The information on attacks and risks stated in this paragraph is limited to the scope of the project, and numerous attack vectors can be used against IoT infrastructure. Attacks on radio enabled devices and other protocol specific devices are not mitigated and can not be mitigated by the proposed framework capable of only tackling Ethernet traffic using the IPv4 protocol. There are lots of device-specific attacks and vulnerabilities that as well can not be mitigated by the framework.

2.3 Available defence mechanism

Due to the layered infrastructure, there are lots of defence techniques that can be deployed, each with its own benefits and disadvantages. For adequate protection, defence mechanisms should be deployed at all layers. The cloud layer is the most easily protected due to the fact that it does not have limitations like the perception layer. The perception layer is hard to protect due to the mentioned constraints. Most of the commercially available products used to mitigate and deter attacks use a technology that synchronizes data from all the layers for optimum detection. In some cases, that might not be possible due to all of the different components or can be too costly, so an edge layer approach should be more feasible.

The main inspiration of this project draws from two research papers Febro, Aldo, et al. "Synchronizing DDoS defense at network edge with P4, SDN, and Blockchain.",(2022) [19] and Febro, Aldo, et al. "Edge security for SIP-enabled IoT devices with P4.",(2022) [7]. The first one uses a distributed method of denying DDoS attacks by stopping communication with certain IPs; this also inspired the functionality of a blacklist in this

project that acts as a mitigation technique. The second research paper uses a bloom filter that uses the architecture of the SIP protocol as an advantage. This mechanism stops brute force attacks carried on devices. Something similar in regard to the bloom filter is also implemented in this project for the detection of TCP SYN and ICMP flooding attacks. Both are robust mechanisms and highly capable of mitigating attacks. This project aims to further extend the use of the bloom filter for detecting attacks carried through other protocols while still focusing on DoS and DDoS attacks.

There are some other frameworks created with the use of Software-defined networks that are used to mitigate DDoS and DoS. These are similar to P4 approaches due to the deployment at the edge layer and control over the data plane.

Some approaches are stateless and use statistics about the number of packets sent to classify them as malicious or real users[20] [21]. These approaches used OpenFlow and proved helpful in detecting DoS and DDoS attacks, but they are stateless and have no notion of time and only rely on information stored in packets for detection. The inability to change state detects the attack but cannot enter in a defensive mode used for mitigation. Stateless approaches can be useful when detecting flooding attacks; mitigating the incoming attack is tricky and complicated, raising the need for a stateful framework capable of both detection and mitigation[22]. Some projects already capable of doing that, besides the first two research papers, mentioned da Silveira Ilha, Alexandre, et al. "Euclid: A fully in-network, P4-based approach for real-time DDoS attack detection and mitigation.",(2020), which creates an entropy-based detection method reinforced with a stateful P4 switch capable of going into a defensive stance[23]. This approach also detects and mitigates the attack by changing states, making it more feasible in a real-world scenario.

There are also combinations of machine learning and P4 frameworks [24] for detection. Machine Learning is a very common and efficient way of detecting malicious traffic [25] [26], used in numerous projects for detection purposes. The problem with machine learning arises from the data used to train the model that can be biased, plus depending on the application; it might be quite computationally heavy.

2.4 Project specific information

This proposed framework focuses on some DDoS attack techniques, not all of them, based on the number of attacks and their simplicity to conduct. Some surveys [12] state that the most common DDoS attack is TCP SYN flooding, so a framework must mitigate this attack for it to be considered efficient in the case of deployment on a device capable of communication through such protocols. Ping of death is also a very severe threat [27] that can significantly affect servers and cloud infrastructure, so the effects on an IoT device would be much worse, in some cases even causing buffer overflows, allowing for malicious code injection. ICMP flooding attacks are fairly easy to conduct due to the simplicity of the protocol, and they can be amplified by exploiting the protocol[28], and many amplification services are available for malicious attackers. Thus a mitigation technique needs to also be created for this type of attack as well to ensure a well-balanced defence mechanism.

For deterring Ping of Death attacks, an approach where the size of the packets was limited was selected [9] due to its extensive use in academia. This approach limits most of these intrusion attempts and works for all protocols by inspecting IPv4 packets. ICMP flooding is more difficult to mitigate, and some algorithm similar to the leaky bucket or token bucket is widely used as mitigation due to the fact that they are network congestion mechanisms[29]. The problem is that it does not detect that an attack is happening. The achievement of a detection mechanism can be done similarly to mitigating TCP SYN flooding.

For the mitigation of TCP SYN flooding attacks, the idea of a bloom filter implemented in P4 just as in Febro, Aldo, et al. "Edge security for SIP-enabled IoT devices with P4.",(2022) [7] was chosen. A bloom filter is a probabilistic data structure, so based on the data chosen for making a probabilistic choice, it can be chosen such that it also works for TCP and ICMP traffic, and the choice acts as a detection mechanism.

Open-source content available about P4 and P4utils was heavily used for the creation of the project and the framework.[30] Already present examples of code and functionality greatly showed the capacity of the P4 programming language and what can be achieved through it. Packet forwarding was used based on the available open-source information in the example code as a starting point for the P4 program. It is also used in the results section to demonstrate the capacity of the implemented project against a switch capable only of packet forwarding without any detection and mitigation mechanism implemented.

3 Objectives, Specification and Design

3.1 Objectives

The objectives of this framework are to detect and mitigate specific cyber-attacks.To achieve these goals, there is a conglomeration of tools used together for the best outcome, some of the most prominent being the P4 program that uses its capacity to inspect packets and the flexible Python programming language for creating a controller. The approaches of detection are different based on the attack, but upon detection, malicious packets will be dropped, and the controller will populate a table with blacklisted IPs. These blacklisted IPs have the packets transmitted through the switch automatically blocked. These are the primary objectives of the implemented framework, but besides building a viable product, there is also the need for testing. So another goal is necessary, the one of testing the capacity of the project in an environment as close as possible to a real-world one. To achieve it, we need the use other tools besides the ones already mentioned. These objectives cover both the implementation and also a certain quality assurance needed for any valid intrusion detection system.

3.2 Specification and Design

In order to achieve a lightweight framework capable of mitigating Ping of death, ICMP flooding, and TCP SYN flooding, first, the design needs to start with an environment where we can test attacks, detection and mitigation. Then the environment needs to simulate as close as possible a real-world scenario. The design and specification of how this was achieved are discussed in the next subsections. Implementing software capable of detecting and mitigating said attacks is discussed in the implementation section since the design of it is highly dependable on P4. After that, testing the implemented version is also a crucial step for checking the performance and how its deployment would affect a network in normal circumstances, as well as attack circumstances. The design of how normal traffic is achieved and how it is tough out is discussed in a subsection, while further testing strategies are discussed in the evaluation section.

3.2.1 Tools used

To create a viable experimental environment, we needed some open-source tools that were easily accessible. The most important is Mininet which allows us to simulate hosts, switches and controllers and the links between them. This is a very powerful virtualisation tool that is necessary for creating a switch capable of communication with hosts. One of the most critical parts of Mininet is the topology that can be customised, and it needs to be designed for our purpose. Due to its importance, the topology will be discussed in a separate subsection. Another crucial tool is P4, a programming language used to control the data plane of a switch. This is the main focus of the design of the project. It should be at the centre of the project due to the fact that the primary goal of detecting and mitigating is achieved through it. To combine Mininet and P4, we use a Python library called P4 utils. Its purpose is to simplify the deployment of

a virtual Mininet network that includes the deployment of P4 capable switches in the topology. This is achieved through the use of APIs. This dramatically simplifies the task of creating a viable laboratory environment. There are other tools needed for the installation and running of the two needed tools, but their use is mostly used for under the hood functionality that is not relevant to the design.

To easily ensure the functionality and testing of the implemented framework, we employ the use of another open source tool called Wireshark. This allows for packet sniffing and inspection of headers. Wireshark is a standalone tool and does not need any dependencies like the previous tools. It is capable of sniffing packets of a virtual network like Mininet.

The last tool used is hping3. This is a tool capable of sending packets from one source to another. It has great flexibility in terms of what type of traffic it can send and the type of communication protocol used. Thus it becomes the main tool for simulating the cyber attacks proposed by running them on virtual hosts. Its name suggests that it is a more advanced tool than ping, allowing for the transmission of TCP and UDP packets besides ICMP ones supported by the ping tool. Due to its flexibility, it is also used to create realistic normal traffic, which we will discuss later in a subsection. Its installation is easy and capable of running inside the emulated virtual network.

3.2.2 Suitable machine

To run all the above tools concurrently, we need a certain operating system where they are supported, and their compatibility is guaranteed. The decision of virtual machine was chosen because sometimes Mininet might be quite aggressive on the installed operating system. A Linux Ubuntu 18.04 operating system was installed on a virtual machine. The image of the machine was a prebuilt version by the creators of P4 utils that ensures the installation of the P4 utils library, the Mininet tool, alongside the BMv2 tool, which is the behavioural model used by P4, and the final component, the necessary compiler of P4, besides other under the hood required dependencies. The machine has access to 10 GB of random accessed memory and 16 processor cores. These requirements are necessary because emulating a virtual network and attacks while monitoring them is quite computationally draining. These resources ensure smooth operation. The mentioned setup is critical for running the project due to the inexistence of an express installation possibility. The design of the necessary machine was done through iteration processes to ensure the best possible environment for production and testing.

3.2.3 Virtual network and topology

The topology created inside of Mininet should resemble a real-world network topology in which IoT devices are interconnected through a switch. The design of the topology is the same as the one used in one of the supporting research papers. The topology consists of ten hosts that are meant to emulate IoT devices, and some of them can also be interpreted as infected devices or attackers for testing purposes. To allow communication between the hosts, a P4 capable switch is placed in the middle of the network. The only

way nodes can communicate with each other or to the open web is through the P4 switch and its forwarding abilities. This design feature allows for all the communication to be inspectable and monitored by the switch, enforcing the implemented security mechanisms. For convenience, a controller is attached to the switch for enforced protection and complete functionality of the program. The design resembles a local network. This also implies certain IP assignment rules that are further discussed in the implementation section. This design of multiple hosts was chosen because most IoT devices are deployed alongside one another. The local network design was chosen because, in most cases, IoT devices are not accessible through the open web. They are usually deployed on a local network or subnetwork, and they can be accessed from the open web through a router that also enforces some security rules and possibly a switch. The figure below illustrates the topology and how it is connected.

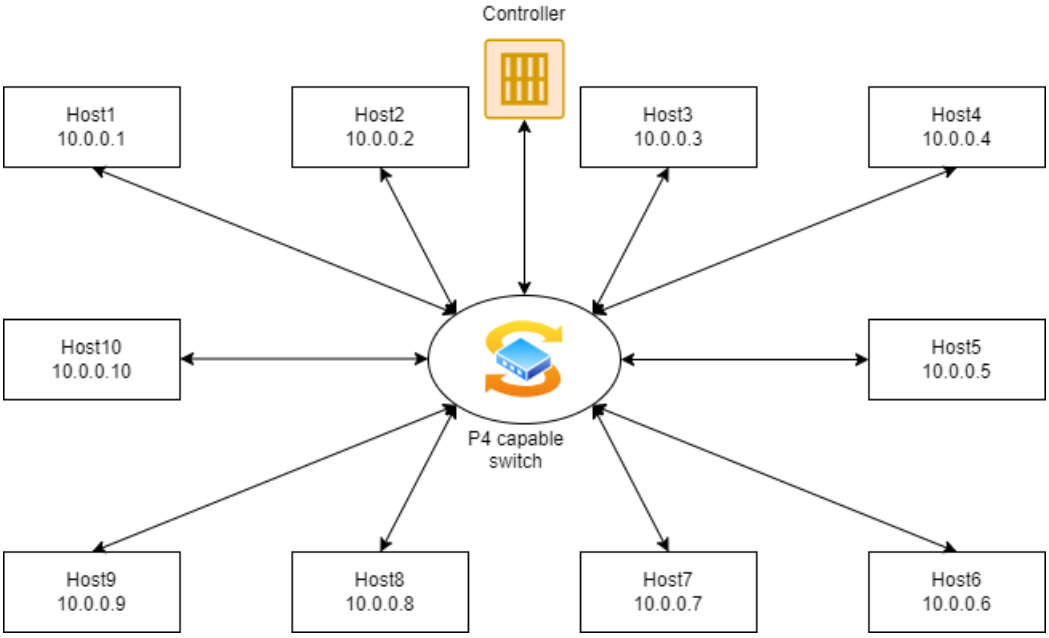

Figure 1: Topology

3.2.4 Traffic

Traffic is a critical part of networks, and for our framework, we simulated ICMP and TCP traffic between the hosts. This was achieved by using a task scheduler implemented by P4 utils. The traffic set by the scheduler is done through a text file that has as variables the source host, the duration of the communication in seconds, the start time of the communication in seconds after the start of the network, and a command such as a ping or a python script. For the last variable, the hping3 command with different destination hosts and different communication protocols was used. Regarding the variables that

take into consideration time, the maximum communication time that was decided was 3 minutes. Both variables are randomly assigned. The text file can be created using a provided Python script. An automated technique for random traffic generation was needed due to the high amount of hosts. The attack traffic is generated with the hping3 tool, but it is not included in the file read by the task scheduler. The attacks are carried through the Mininet client such that monitoring through Wireshark is easier since the start of the simulated attack is manual. The randomization factor ensures that traffic is similar to a real-life scenario and that attacks can be clearly identified by the switch due to their distinct footprint.

4 Methodology and Implementation

To achieve all the goals proposed in the previous section and use all the tools to their maximum capacity, the implementation was done in steps, each one of them covering a milestone crucial towards the full implementation of the framework. Each of these milestones is covered below in the adequate subsection.

4.1 Network

The implementation of the virtual Mininet network was done using P4utils. It is achieved with a Python script. It allows for custom topologies that can be coded in the script with the help of easy-to-use functions. The custom topology has a P4-capable switch and ten other hosts meant to represent IoT devices. Next, the connection between these nodes was done by creating links between them and the switch. The links needed open ports. Each node has port one open for connections with the switch. The switch has ten open ports, port number one for node one, port number two for node two and so on. P4utils also allows for the limiting of bandwidth, but we decided against it, even if it would simulate a real life scenario better. Now the topology is ready to be run without any problems, but packets can not be forwarded yet due to the fact there is no P4 script assigned to the switch.

The IPs of the devices are permanently assigned the same every time the network is started, and automatic layer two IP assignment is used. There is no need for using a layer three assignment strategy or a custom one because the aim of the project is detecting attacks without discriminating certain subnets. In some other custom topologies, another assignment strategy might be needed, and for a real scenario, it depends on the deployment of the switch in the network and its use. Based on the assignment strategy the implementation of the P4 script might be different.

Another reason for the IPs is that the aim of the framework is to protect devices on a local network and be a second layer of protection to a router without the need to access other subnets. In a real world scenario, in the topology of the network, a router would be placed before a switch when accessing a local network from the internet. Even if the local network had different subnets, it would make no difference in how the attack would take place and its efficiency; it only modifies the way the switch needs to forward

packets.

To ensure communication between the hosts, after the assignment of a P4 script capable of layer2 forwarding, we need populated ARP tables. The network automatically makes the ARP calls at the start of the network, so the ARP table does not need to be manually populated. This is done through the use of P4utils. The manual population is also available by modifying the script, but that is not necessary in our case, and automation of it is more convenient.

4.2 Forwarding

To allow forwarding, we need a working P4 program capable of doing that for layer 2 IP assignment. The standard format of a P4 program is headers, parsers, checksum verification, ingress processing, egress processing, checksum computation, deparser and switch. In the headers part, we need to declare the headers for our communication protocols. Here we set the sizes for each field of a header. We declare headers for ethernet, IPv4, ICMP and TCP. ICMP and TCP headers are not crucial for packet forwarding, but they come in handy at later steps of implementation, where we need to inspect packets with these headers. The declaration is made by allocating space for all the needed components of the headers. In the headers section, metadata that is used by the program is also declared; for now, no metadata is necessary.

Checksum Verification and Computation are used to check the integrity of the arriving and outgoing packets through a built-in P4 function checksum. At both steps, all of the components of the IPv4 header are used to check if the packet is not corrupted by the network transmitted through or the P4 program. They are crucial for ensuring adequate delivery of packets. If the packets fail these steps, they will not be transmitted.

The parser extract and interprets the declared headers. It identifies each protocol header and stores it in metadata. In our case, it first checks the Ethernet header, then the IP header and based on the protocol TCP, ICMP, it branches off to different states. The deparser takes the headers from the metadata and puts them back together in the correct order so the packet can be sent out, rebuilding the headers with the information saved in the metadata.

Ingress and egress processing are the main stages of a P4 program. Ingress processing processes packets after parsing, and this is where the main logic of intrusion detection is written. Egress processing is used mainly for modifying headers based on routing decisions or applying quality-of-service policies. In egress processing, the logic of P4 can be used to detect outgoing attacks, but that is out of the scope of the current project.

Forwarding is achieved in ingress processing. First, we need two basic methods, drop and forwarding IPv4 packets. Dropping is achieved by removing the metadata; basically, the packet and the headers are deleted, so the packet is destroyed and not transmitted anymore. Forwarding IPv4 stores the source and destination Mac address, as well as decrement the time to live by one. Forwarding is achieved with the help of a table that stores a correlation between the IP address and MAC address. The command to add said correlation is "table_add ipv4_lpm ipv4_forward 10.0.0.1/32 =>00:00:0a:00:00:01 1", where we make a connection between host 1 with the Mac address 00:00:0a:00:00:01 to

IP 10.0.0.1. The default action of the table is to drop if no matching entry is found. The table is declared in the ingress processing section, and it is a staple functionality of the P4 programming language. If the IPv4 header is correct, the packet is forwarded only if the table is populated with the IP address contained in the packet header. The table is populated at the start of the network by the Python script creating the network using a text file where the commands are written. More sophisticated P4 programs that are used for commercial purposes allow the automatic population of the table at runtime with the use of a controller.

The implementation mentioned above allows for all IPv4 traffic to be forwarded to the right destination, and now the implementation of the security mechanisms can begin, having achieved the functionality of a simple switch.

4.3 Ping of Death Detection and Mitigation

The Ping of Death attack is characterized by sending a large malformed packet to a target device. Due to the architecture of the IPv4 header, we can deduct the size of the packet. In the figure below, we can see the segment's total length using the header, which tells us the size of the packet. Using that information, we can now detect oversized packets that consist a Ping of Death attack. If a packet is over a certain threshold, it is going to be dropped, and the source IP address will be added to the blacklist as a mitigation strategy. The implementation of this detection technique is done in ingress processing. An if statement checks if the length is over a certain threshold for normal IPv4 packets, and if it is an ICMP packet, it is checked against another threshold. This is a fairly common detection technique, but it has some weak points if the attack uses fragmented packets.

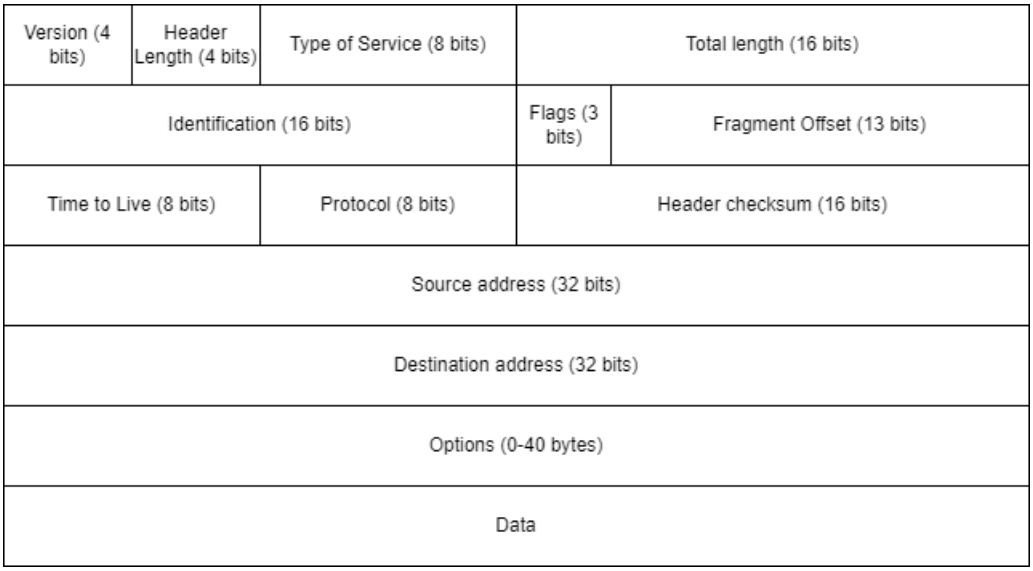

Figure 2: IPv4 Header

4.4 TCP SYN flooding Detection and Mitigation

TCP SYN flooding abuses the protocol's handshake mechanism by sending SYN flags over and over again without ending the connection between the two. To detect this type of attack, standard intrusion detection systems use time associated techniques with the arrival of the packet as a piece of information to make decisions if an attack is happening or not. P4 is not capable of being aware of the time of the arrival of the packet. Instead, we use a bloom filter that only takes into consideration the type of packets received and their number. A bloom filter is a probabilistic data structure that classifies elements based on a set. It uses hashes to represent the packets and decides if they fit in a set or not, in our case, a packet causing TCP SYN flooding or standard traffic. As with any probabilistic data structure, it can have false positives while classifying, but it is extremely memory efficient compared to other methods.

The implementation of this detection mechanism starts with three variables. The threshold, which decides how many packets are needed to determine if it is a cyber-attack or not, and the bloom filter width and entries are used to fine-tune the filter. Then we need four more variables stored in the metadata, two counters and two hash values. These last four are used when updating the bloom filter. The bloom filter is updated every time a TCP packet with only the SYN flag is sent. The bloom filter takes its decision based on four header elements used to calculate the hashes. The source and destination IP addresses, available in the IPv4 header, the SYN flag and the protocol header, available in the IPv4 header, are used for this calculation. Using the source and destination port from the TCP header doesn't prove helpful in detecting the attack because, in a flooding attack, the source port and destination port vary at every packet, making the information redundant. Other information available in the headers is not useful in detecting this type of threat. As mitigation, the source IP address will be banned as well as the packet dropped, and so will future ones from the same IP.

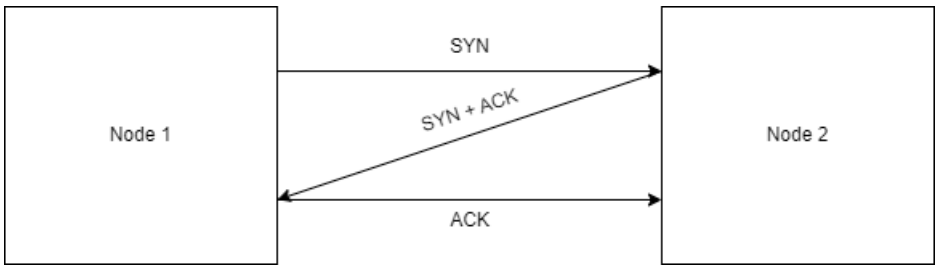

Figure 3: TCP Handshake

4.5 ICMP flooding Detection and Mitigation

An ICMP flooding attack sends lots of ICMP packets, and the receiver needs to send a response. This attack abuses the protocol's mechanism as well. There are multiple ways of detecting this attack, some being standard techniques to mitigate heavy flow. Different ways of detection and implementation are discussed in the next subsections.

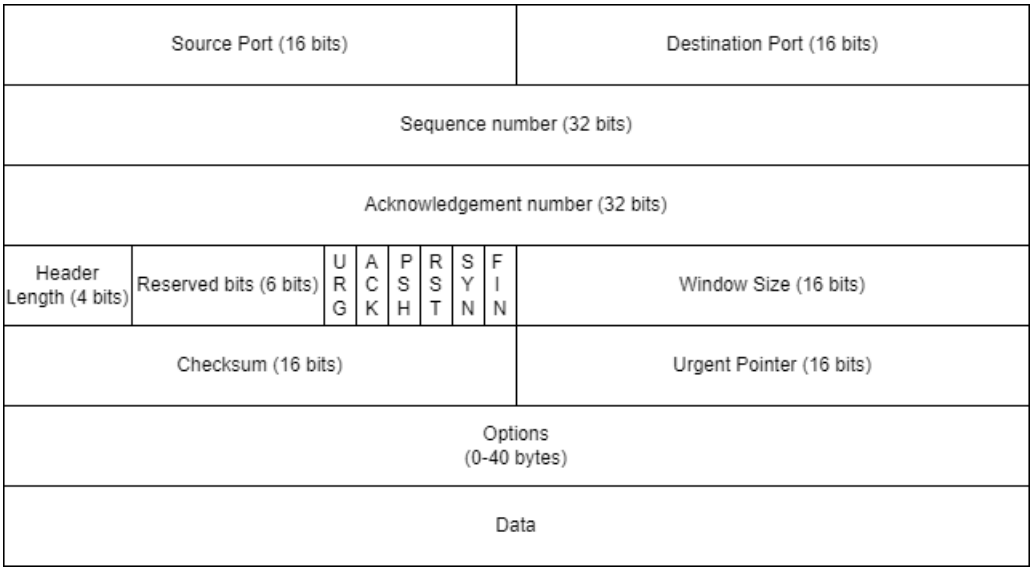

Figure 4: TCP Header

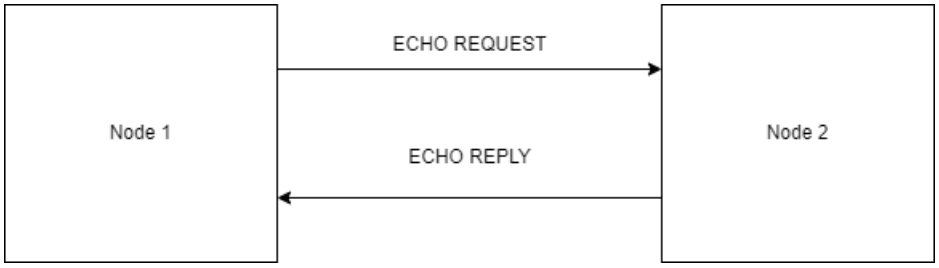

Figure 5: ICMP Handshake

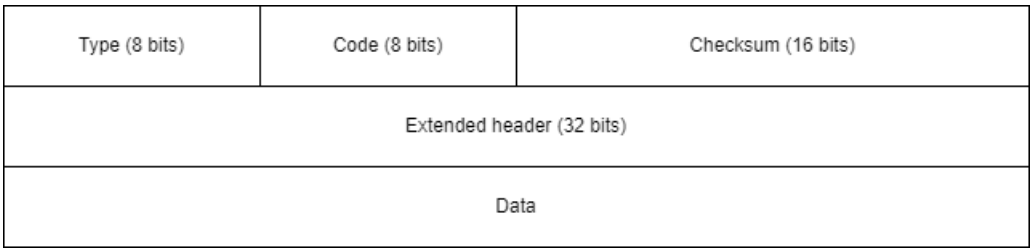

Figure 6: ICMP Header

4.5.1 ICMP flooding Detection and Mitigation based on probability

The most common way of mitigating this attack using ICMP is by using probability to either drop or let through a packet. After implementing a simple probabilistic function to drop ICMP packets based on a percentage, it was proved to be more of a rate limiter

than an intrusion detection system. This method can not be used for banning IPs as mitigation since it is heavily volatile and unreliable. For a better probabilistic result, the probabilistic function needs to take into account flow or time, which is very memory heavy for a P4 program. For the program to take into consideration time, it is impossible for P4 without the use of the controller plan, this meaning that the logic needs to be shifted to the controller to achieve this. Implementing this would change the scope of the project, so other methods are required.

4.5.2 ICMP flooding Detection and Mitigation based on a threshold

Another well-known method of dealing with heavy traffic and flooding is the leaky bucket algorithm or the token bucket algorithm. The leaky bucket transform burst of traffic into uniform traffic, and the so-called bucket filters the flow burst based on the clock at a constant rate. The token bucket uses tokens for each packet, and tokens are generated in a finite amount at a certain amount of time. Both of these approaches are difficult to implement without a controller capable of keeping track of the time in p4, and specific workarounds need to be found. Both of these approaches based on a temporal threshold are more rate limiters and do not give us a confident answer if an ICMP flooding attack occurs, so the mitigation used for these would be dropping the back, which does not stop the attack.

4.5.3 ICMP flooding implemented Detection and Mitigation

The chosen approach for mitigating this attack is a bloom filter. It is very similar in implementation to the one used for mitigating TCP SYN flooding, but the hashes are calculated based on the source and destination IP address, the protocol element, all three available in the IPv4 header. The type and code available in the ICMP header are also used for the calculation of the hashes. With this information, the bloom filter is able to classify the ICMP traffic as an attack or standard traffic. As mitigation, the source IP address is banned and the packet is dropped, as well as future ones coming from said IP address.

4.6 Blacklist

The blacklist is the feature that enables the mitigation of the attacks for this project. This is achieved by creating a table that can be populated with IPs. IPs can be added before the start of the network. In a real-world scenario and not a simulated lab, these IPs added before the use of the switch can be already known malicious addresses used for infected devices or command and control servers. The table can be populated using "table_add blacklist_table drop 10.0.0.7/32 =>", adding host 7 as a blocked address. Before sending a packet, a boolean value that represents if the source IP address in the header is banned, is checked. If so, the packet is dropped. A problem might arise from a large number of entries in the table since looking up an IP might take some time.

4.6.1 Blacklist at runtime

To achieve a robust mitigation technique, the table needs to be populated at runtime. To achieve this, we need a controller capable of sending commands and thus populating the table. The most memory efficient and computationally efficient approach is to send a digest message from the P4 program. The controller takes this digest message and automatically sends the populate table command. The implementation is done by creating a structure in the P4 program containing an IP that can be digested. After every time a detection mechanism is triggered, the digest message is sent, and the packet is dropped. The controller is using Thrift API and is digesting every message received. Since it is only expected to receive IPs to ban, it only has that functionality. It also informs the user when it is connected and also shows errors if any arise. Just running the controller alongside the network Python script is enough to mitigate potential attacks. The controller is a modified version of an open-source variant available on GitHub and provided by the creators of P4[31].

4.7 Setting thresholds and variables

The most critical component of this lightweight framework is the thresholds that can indicate if a packet is considered malicious or not. For each type of cyber attack, a different threshold is needed.

For the Ping of Death attack, the most common threshold is 65,515 bytes[9] [32], considering the ICMP header beside the IPv4 one. For simplicity, we are using a threshold of 1,468 bytes. This is calculated by using the maximum transmission unit of 1500 bytes[33]. From that value, we subtract the ethernet header 14 bytes, the IP header 20 bytes and the ICMP header 8 bytes, leaving us with a 1468 bytes threshold for detecting a ping of Death attack.

For TCP SYN flooding, a threshold is hard to set. In most cases, it is set based on an algorithm that takes into consideration traffic flow, and it usually is variable[34]. In our environment, it is hard to achieve that, so we need a general value. In some research papers, the analysis of the number of packets affecting a target based on the number of packets sent is shown[35]. The effect of the attack on the target seems to ramp up significantly after 100 packets are sent, so this is a viable threshold. The mentioned research paper also uses IoT devices, but a detection and mitigation technique is done using SDN.

Setting a threshold for ICMP is also very similar to setting one for TCP SYN flooding. It is very dependent on the traffic flow, and it is usually set through an algorithm[12]. Using previous experiments using SDN[36], we can set a static threshold for our program. A number of 1000 packets would be enough to make the difference between regular ICMP traffic and malicious one.

The threshold values are thought for IoT traffic, taking into consideration the limited bandwidth and limited computation power of these devices. A server or a more efficient machine might need looser thresholds, but in our case, considering ten interconnected devices, it should suffice.

5 Results, Analysis and Evaluation

5.1 Results and analysis

A crucial step for the implemented project is testing and analysing. Since the P4 program does not have any direct way of outputting the user info about the packet, that limits our usual ways of debugging and testing. The best approach is to use a packet sniffer like Wireshark such that we get an idea of how the program behaves and what the communication and packets look like.

5.1.1 Testing functionality

Testing the detection and mitigating techniques was done by sending the malicious traffic between two hosts and seeing how one host sends the data and what is received by the other host. We start the network and the controller and two instances of Wireshark. We capture the packets that are received and sent by the two hosts by sniffing them. To make the testing more manageable, we are testing the flooding attacks first, and we set a lower threshold since we are not going to run the attacks in flood mode. The threshold is going to be set for both at 20 malicious packets. This instance is also done without any background traffic such that the result is more evident. Now running the attack traffic, we can see that for the first 20 packets, the hosts are communicating normally, and we receive the appropriate response to the packet for ICMP Echo Reply and for TCP SYN ACK. Now after the threshold is reached, we can see that the attacking host keeps sending packets, but the second host does not receive any more messages. This means that the program works as intended, and malicious packets are dropped accordingly. For detecting and mitigating a Ping Of Death attack, we can simply send an oversized packet, and we can see that it is dropped and does not reach its target, proving that all the detection techniques work. In the figures, we can see the packets and their count, captured on hosts 1 and 2 after running ”h1 hping3 10.0.0.2 –icmp -c 25” and ”h1 hping3 10.0.0.2 -S -c 25”. Now for testing the mitigation techniques besides the basic drop packet one. To see that the blacklist functionality works, we need to test it at the beginning of the network, where the command is added through the test file read by the network Python script and at runtime. For the first step, we add host 7 to the blacklist. Now the host is incapable of communication, and we can observe it through Wireshark. Any communication toward it is also blocked. At runtime, we need to test it in the same way, but we need an attack first to result in a banned IP. Besides the information given by the controller that the IP was added, testing that outgoing and incoming packets are blocked is a success. This concludes the functionality of the detection and mitigation techniques, but the way they are achieved still leads to some false positives, further discussed in the next section.

5.1.2 False positives

Due to the way bloom filters work, it is a probabilistic approach; thus, false positives might arise. One example is that since the program is packet and not time dependent, it

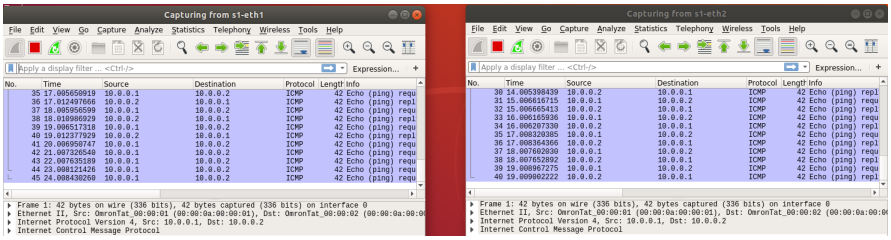

Figure 7: ICMP packet capture

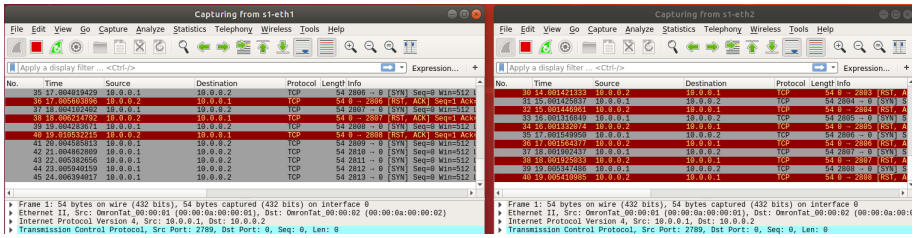

Figure 8: TCP packet capture

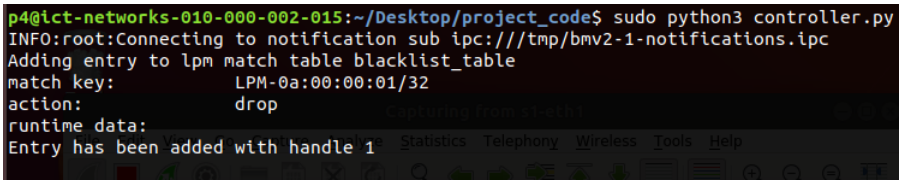

Figure 9: Controller populating table

might falsely accuse a host of flooding after it sends a number of packets over the threshold over a specific long time, like a day. This happens if that only host communicates through the switch, with no other traffic that can be assigned to sets. Some different types of false positives might arise from the data fed to the filter, like source and destination IP, that are crucial for the decision-making process. This can cause certain subnets to be banned due to their similarity to an attacker. Based on the testing done with attacks during simulated normal traffic, no false positives were found, but that does not exclude their possibility. The chance of false positives, of course, rises with the amount of time that the network is run. In our case, it is run only for three minutes due to the fact that emulating a network is quite computationally costly, and all the Pcap files are saved, draining the space available on the virtual machine. For finding false positives, there might be the need for a particular testing technique to be applied for more robust statistical analysis, but this might be out of the scope of the project since it requires a more powerful machine with more storage capacity plus a way to monitor all the traffic and the controller to observe when a new entry is added to the table.

5.1.3 Testing efficiency versus legacy switch

For further testing, we decided to use the throughput and the processing power consumed by the switch as metrics. The throughput should clearly state which version of a switch is better against attacks, the legacy or the implemented one. The processing power should confirm the fact that it is a lightweight framework and that the attack clearly is mitigated. Firstly we have both switch variants running with regular traffic; then, we ran them with a TCP SYN flooding attack. It does not matter what DoS attack it is since the metrics should be similar for all of them. We can clearly see that they are similar in normal traffic, but when faced with a DoS attack, the normal one consumes much more resources. This concludes that our framework mitigates the attacks, and it is also a lightweight security mechanism since it is comparable in normal circumstances to a standard switch.

Looking at the throughput, we can see the regular traffic running accordingly for both switches. For our defence mechanism, we can clearly see the peak where the attack started ramping up, and that it was mitigated, and that the traffic returned to normal. Unfortunately, we could not have a significant capture of the throughput for the standard switch under DoS circumstances since the attack starved the computer of memory, and Wireshark was constantly crashing. This proves the advantages it has of securing a network over a standard switch. Our framework not only stops the attack but also maintains the same quality of communication for the devices sending information through the switch.

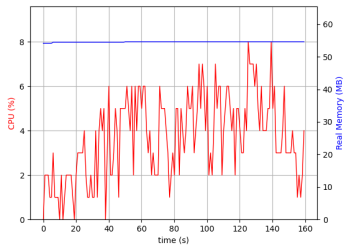

Figure 10: Implemented switch under normal traffic (CPU)

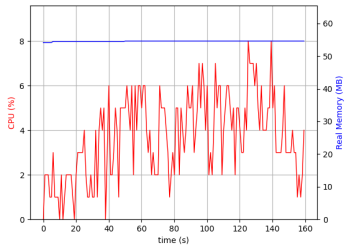

Figure 11: Legacy switch under normal traffic (CPU)

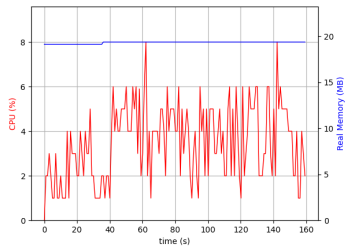

Figure 12: Implemented switch under DoS traffic (CPU)

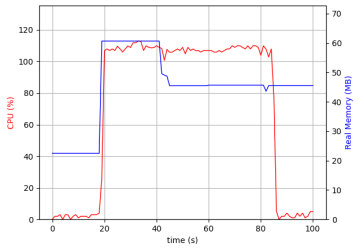

Figure 13: Legacy switch under DoS traffic (CPU)

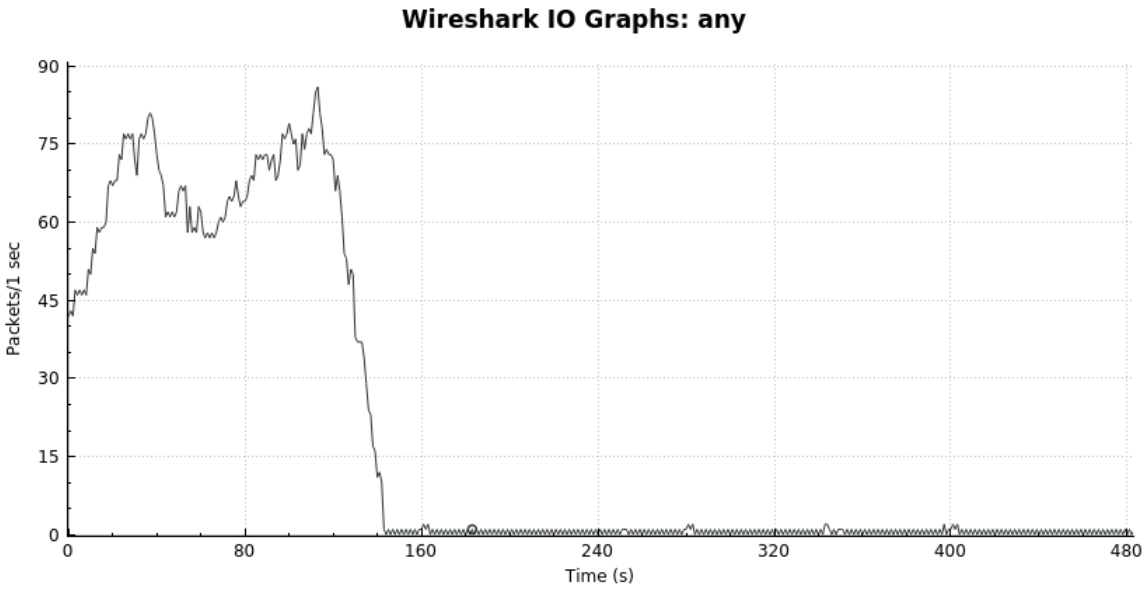

Figure 14: Normal traffic (Wireshark)

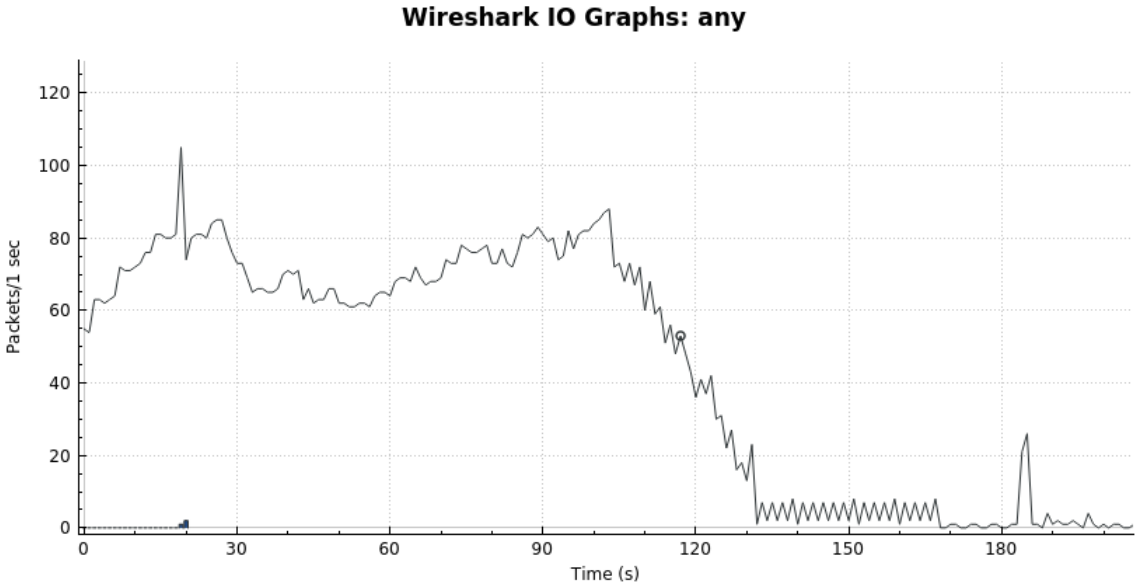

Figure 15: Implemented switch under DoS (Wireshark)

5.2 Comparison

The efficiency of the switch is guaranteed against a legacy switch, but what about other techniques and mechanisms? Unfortunately, there is no point of comparison against the supporting research papers because the implementation and mode of testing are different. Even if that project mitigates DoS attacks, it mitigates ones based on SIP, which is a different protocol, and the attack is carried out in a different way, plus it is a more robust framework. Using our throughput information, we can still compare it to some other available research papers like Goksel, Nail, and Mehmet Demirci. "DoS attack detection using packet statistics in SDN." 2019[21]. This research paper uses packet statistics in an edge environment that can be associated with ours. It also has metrics on throughput but uses SDN instead of P4. Comparing our results, we can see that the framework implemented in P4 performs slightly better than the one using SDN, but no conclusive decisions can be made since their topology is different, and it surely impacts the throughput.

5.3 Evaluation and future improvement

The project achieves its intended purpose of being a lightweight framework for mitigating specific cyber attacks, but it still has some weak points and can be further improved. It can be further enhanced by adding mitigation for UDP flooding, securing against a large variable of attacks that use protocols.

Some weak points of the project are the fact that ping of death using fragmented

packets is still achievable. To mitigate this, the program needs to have knowledge of the previous packet. This is quite difficult to be achieved in P4, and a controller with lots of logic needs to be implemented to achieve such. For better flooding mitigation, the logic needs to be shifted from the switch to the controller, such that a better knowledge of time and state is possible since P4 is quite limited. Another big problem is IP spoofing, which is a common technique known for conducting cyber attacks[37]. In our framework, this can lead to banning innocent IPs. Even if the attack is stopped, the attacker is not. Combining this with amplification servers can be a serious threat to the implemented project. For better detection again, the controller needs to be improved. Having information about previous packets and timestamping received packets can identify attacks not only based on the contents of the packet but this would transform it into a computationally heavy task that might be harder to deploy than our current project and might not prove cost-efficient.

A significant improvement can be made by synchronizing switches with each other and sharing blocked IPs[19]. By doing such, we have a common list of banned IPs, and further attacks against other devices managed by other switches can be mitigated.

6 Legal, Social, Ethical and Professional Issues

Legal, social, ethical and professional issues are important issues, especially for a cyber security project that takes into consideration cyber-attacks that are punishable by crime under the 1990 computer misuse act[38] and it is built for the protection of the people or their goods. Its use for protection suggests that legal and professional issues should take priority to ensure top-quality security while also helping the cyber defence field. Social issues need to be mostly taken care of after the deployment of the project, while ethical ones need to be tough of both before and after. Discussing all of them together ensures that the Code of Conduct & Code of Good Practice issued by the British Computer Society and the Rule of Conduct issued by The Institution of Engineering and Technology are followed. As a technology developer located in the UK, following such standards is crucial for developing qualitative and sustainable products that can shape the world into a better place with fewer worries and better security. The next subsections state some viable issues for the developed project, and another subsection would state how these issues can be removed or, in the worst case, a workaround be found. Some issues are viable at the current stage, and some after deployment, but both would be taken into consideration.

6.1 Legal Issues

If we aim to deploy this project on any scale, then we need to take into consideration the critical tools used and licensing issues that might appear. The main tools used are P4 and Python, Mininet should not be taken into consideration in this case because it is only used for testing by emulating a real network inside a virtual one. P4 is an open-source project, but we might need particular authorization or licensing for capitalizing

on the framework implemented, making sure we are not violating any copyright law. Python is an open source software as well so it should not be a problem.

Another legal issue might arise from the packets that are inspected and the data stored in the tables. We need to make sure it does not violate the GDPR of the European Union, CCPA in California, and PIPEDA in Canada, regarding data collection and processing. Our project does not store personal data in a central database, so the issue would only be the data available in the tables and accessible through a controller. The client should be informed about this fact and what risks the data faces to be accessed. Their consent should also be inquired before installing the software like most modern applications.

Another issue is making sure it is not considered wiretapping, which is prosecuted in many jurisdictions. Wiretapping is an illegal interception of communication. This is an issue that takes form after deployment and depends on the area it is deployed to. The client of course should be made aware of the capabilities of the program to avoid future liabilities.

There is also the legal enforcement of regional network policies, and the program might be needed to be modified based on the region such that it enforces these policies and does not violate them. Another similar problem is the export regulations that can also dictate if the program needs to be further modified. Banning IPs can also lead to banning innocent IPs. This is probably due to the fact that false positives can happen since the detection mechanism is probabilistic. If such an event occurs loss of communication can occur and liability problems arise. This might incentivize the client to search for legal action against the producers of the software for monetary reparations for the damage. Mitigation needs to be found for this issue which can be considered critical for the success of the product on the market.

6.2 Social Issues

To ensure social fairness, we need to take into consideration the following issues and mitigate them.

Invasion of Privacy can be a severe social issue since some users might not be comfortable with their data being analyzed or monitored, so informing customers is crucial from a social standpoint. They need to know where all information is stored and that the product does not send any data to other devices. The information of data that is processed needs to be made known to the client as well. The single information collected is the one stored in the tables that might hint us to who the client is communicating, so it needs to be further protected.

Another problem might arise from the false positives, labelling innocent users as attackers. This is a severe issue since it labels people as malicious agents. The user needs to be made aware that such a case is possible. The affected party can ask for compensation for defamation damage, leading to further legal issues if this issue escalates.

Another problem is accountability. A person or group of persons needs to take the responsibility to update and maintain the system to ensure quality, as well as mitigate other issues that might appear and possible future discovered bugs. This action is neces-

sary for the continuation of the deployment of the product and also puts the customer's minds at ease and increases their trust since the product is further supported.

The digital divide can also be an issue. Deploying the product only on specific devices can lead to a particular population being more vulnerable to cyber-attacks and also creating a bias against unprotected devices, thus violating the BCS code of conduct[39]. Fair deployment and commercialization of this product need to be fully available. For further social implications, we need to make sure the program does not discriminate against particular IP subnets that represent a specific country or region. This can happen due to the probabilistic nature of the framework that takes into consideration source IP and destination IPs. Since these are tied to locations or internet providers, we do not want the creation of any bias. This issue can further develop and lead to severe social and possible racial bias due to a digital divide or a probabilistic error.

6.3 Ethical Issues

Ethics are necessary for every cyber security software. Protection needs to be enforced while also respecting ethics and not being too aggressive in defending the intended device or devices. Ethical issues might arise in our case, again from privacy. We need to ensure the data is used ethically for the scope of the project. Transparency as well can be viewed as an ethical issue too, so informing the client of all the necessary information and how the program runs is a critical component. The digital divide is both a social and ethical problem. For ethical purposes, fair and equal access needs to be ensured. Algorithmic bias and accountability can also be viewed as ethical problems. Bias can transform the program into an unethical tool of discrimination. Accountability is necessary if the program fails and cant protect or wrongly protect. This needs a responsible party to ensure customers and be fair in an ethical nature.

Some ethical issues can arise from misusing the program. Attackers can reverse engineer and launch spoofing attacks such that innocent IPs are banned. This means that the program was unable to protect efficiently and thus caused a loss to the owner. Permitting the use of the program as an attack tool is highly unethical, and a particular failsafe needs to be provided.

Some ethical issues need to be thought of before the start of the design of the project, while some need to be taken into consideration while deploying and modifying the project for customer use.

6.4 Professional Issues

Some professional issues might arise from competency. If the program detects malicious activity and results in an IP being banned, just based on the program's use the owner of the said IP address should not be prosecuted by the individual owning the deployed program, and even the owner of the program is not qualified to take legal action against the fake evildoer. This can be a false positive and an innocent person can be prosecuted. A special forensic investigation needs to be held by accredited persons or the corresponding authorities. Furthermore, since it is a cybercrime, catching and prosecuting the culprit

falls under the jurisdiction of the country he is in, so this is a serious matter that should not be done by anyone using or owning the program due to competency issues. The program is built as a probabilistic detection and mitigation technique and not as a foolproof entry detection system capable of carrying out forensics investigations. Besides this issue, before the deployment of the program, legal compliance should be checked.

Those responsible for maintaining and updating the program need to be both competent and possess a strong sense of professional integrity. This way, we ensure the quality assurance of the product as well as be more certain that the product is used for its intended scope of protection and is efficient in doing such. These persons are also needed to develop from a professional point of view to achieve the goals mentioned above.

Another professional issue and also legal is making sure that legal compliance is in place.

6.5 Tackling these Issues

In order to tackle the legal, social, ethical, and professional issues, we need to make specific changes to the project such that deployment on a large market is feasible. For the mitigation of most legal issues, a legal team is necessary to write a user agreement contract. This should state the information used by the program, how it is used in processing, and what information is stored and how it is stored. The legal team should also inform the developers what changes in term of functionality need to be done such that it is allowed in certain countries and fit all the local legislations. The developers should make those changes to ensure that the product is available anywhere, and thus a technical divide is avoidable.

For privacy reasons, the information stored in the table might need to be encrypted, and not any person with access to the switch and the controller can modify its contents. This way, data is secured, and the user can freely modify it; plus, this data is only stored locally, so it falls under the responsibility of the person that owns and operates the switch.

Unfortunately, the bias can not be removed due to its probabilistic nature, but the legal team can state that bias is possible, and the producers can not be held accountable for that. This does not mean that bias should be allowed. Future versions should search to fix any bias if any is found such that ethical and social issues are avoided.

For future iterations, quality and accountability can not be accounted for yet, but this version of the framework is fully functional and capable of being deployed in a virtual environment for testing and educational purposes. The project is not ready to be deployed as a valid intrusion detection system on the open net due to its lightweight design that does not mitigate complicated attacks. At this stage, professional software for a router connecting the open web to the switch should accompany it for proper protection. At this stage, it is ethically correct due to the fact that it is not invasive, and restarting the program would reset the banned IPs, which does not make it aggressive at banning IPs. From a social point of view, it is openly available, and since it is

computationally cheap and easy to use, it should not discriminate based on the device used.

The present code was done with the utmost competency and accountability, and the Code of Conduct & Code of Good Practice issued by the British Computer Society and the Rule of Conduct published by The Institution of Engineering and Technology[40] are followed to ensure the best development practices and the best outcome.

7 Conclusion

In conclusion, the framework is a valid detection and mitigation software against DoS attacks. It is built with deployment simplicity in mind; its lightweight design is perfect for deployment at the edge layer and protecting IoT devices. With the rise in the number of Internet of Things devices and the constantly rising threat of more sophisticated cyber-attacks and botnets, the number of such frameworks with surely increase, accelerated by a continuous arms race between cyber security engineers and malicious hackers that capitalise on the growth of the IoT field. P4 is a very capable programing language for developing defence mechanisms, and alongside a powerful controller, it can become an autonomous robust intrusion detection mechanism capable of blocking most cyber attacks aimed at defenceless IoT devices. With the continuous development of P4 and new functionalities added to support the control plane manipulation, it will surely become a staple of the networking community and will be deployed on most switches available, a fact supported by the actual way that P4 is received by the programming community. Its use in a simulated lab environment, alongside other networking tools, is crucial for developing and improving other defence tools, just like the implemented one.

References

[1] S. Madakam, V. Lake, V. Lake, V. Lake, *et al.*, "Internet of things (iot): A literature review," *Journal of Computer and Communications*, vol. 3, no. 05, p. 164, 2015.

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," in *26th USENIX security symposium (USENIX Security 17)*, pp. 1093–1110, 2017.

[3] "Fbi 2014," Jul 2014.

[4] A. Thierer and A. Castillo, "Projecting the growth and economic impact of the internet of things," *George Mason University, Mercatus Center, June*, vol. 15, 2015.

[5] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[6] J. Poore, J. C. Flores, and T. Atkison, "Evolution of digital forensics in virtualization by using virtual machine introspection," in *Proceedings of the 51st ACM Southeast Conference*, pp. 1–6, 2013.

[7] A. Febro, H. Xiao, J. Spring, and B. Christianson, "Edge security for sip-enabled iot devices with p4," *Computer Networks*, vol. 203, p. 108698, 2022.

[8] Y. Nadji, M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee, "Beheading hydras: Performing effective botnet takedowns," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, (New York, NY, USA), p. 121–132, Association for Computing Machinery, 2013.

[9] A. Abdollahi and M. Fathi, "An intrusion detection system on ping of death attacks in iot networks," *Wireless Personal Communications*, vol. 112, pp. 2057–2070, 2020.

[10] B. Kepçeoğlu, A. Murzaeva, and S. Demirci, "Performing energy consuming attacks on iot devices," in *2019 27th Telecommunications Forum (TELFOR)*, pp. 1–4, IEEE, 2019.

[11] H. S. Abdulkarem and A. Dawod, "Ddos attack detection and mitigation at sdn data plane layer," in *2020 2nd Global Power, Energy and Communication Conference (GPECOM)*, pp. 322–326, IEEE, 2020.

[12] R. Vishwakarma and A. K. Jain, "A survey of ddos attacking techniques and defence mechanisms in the iot network," *Telecommunication systems*, vol. 73, no. 1, pp. 3–25, 2020.

[13] S. Li, L. D. Xu, and S. Zhao, "The internet of things: a survey," *Information systems frontiers*, vol. 17, pp. 243–259, 2015.

[14] K. Rose, S. Eldridge, and L. Chapin, "The internet of things: An overview," *The internet society (ISOC)*, vol. 80, pp. 1–50, 2015.

[15] L. Tawalbeh, F. Muheidat, M. Tawalbeh, and M. Quwaider, "Iot privacy and security: Challenges and solutions," *Applied Sciences*, vol. 10, no. 12, p. 4102, 2020.

[16] D. Carrascal, E. Rojas, J. Alvarez-Horcajo, D. Lopez-Pajares, and I. Martínez-Yelmo, "Analysis of p4 and xdp for iot programmability in 6g and beyond," *IoT*, vol. 1, no. 2, pp. 605–622, 2020.

[17] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019.

[18] İ. Özçelik and R. R. Brooks, "Cusum-entropy: an efficient method for ddos attack detection," in *2016 4th International Istanbul Smart Grid Congress and Fair (ICSG)*, pp. 1–5, Ieee, 2016.

[19] A. Febro, H. Xiao, J. Spring, and B. Christianson, "Synchronizing ddos defense at network edge with p4, sdn, and blockchain," *Computer Networks*, vol. 216, p. 109267, 2022.

[20] R. N. Carvalho, J. L. Bordim, and E. A. P. Alchieri, "Entropy-based dos attack identification in sdn," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 627–634, IEEE, 2019.

[21] N. Goksel and M. Demirci, "Dos attack detection using packet statistics in sdn," in *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6, IEEE, 2019.

[22] J. Galeano-Brajones, J. Carmona-Murillo, J. F. Valenzuela-Valdés, and F. Luna-Valero, "Detection and mitigation of dos and ddos attacks in iot-based stateful sdn: An experimental approach," *Sensors*, vol. 20, no. 3, p. 816, 2020.

[23] 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.

[24] 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, pp. 1–27, 2022.

[25] R. Santos, D. Souza, W. Santo, A. Ribeiro, and E. Moreno, "Machine learning algorithms to detect ddos attacks in sdn," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 16, p. e5402, 2020.

[26] N. Bindra and M. Sood, "Detecting ddos attacks using machine learning techniques and contemporary intrusion detection dataset," *Automatic Control and Computer Sciences*, vol. 53, pp. 419–428, 2019.

[27] 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.

[28] P. Zaroo, "A survey of ddos attacks and some ddos defense mechanisms," *Advanced Information Assurance (CS 626)*, 2002.

[29] A. K. Pandey and C. P. Rangan, "Mitigating denial of service attack using proof of work and token bucket algorithm," in *IEEE Technology Students' Symposium*, pp. 43–47, IEEE, 2011.

[30] "P4 language."

[31] Nsg-Ethz, "Nsg-ethz/p4-learning: Compilation of p4 exercises, examples, documentation, slides for learning or teaching."

[32] J. Postel, "Internet protocol—darpa internet program protocol specification, rfc 791," *(No Title)*, 1981.

[33] D. Murray, T. Koziniec, K. Lee, and M. Dixon, "Large mtus and internet performance," in *2012 IEEE 13th International Conference on High Performance Switching and Routing*, pp. 82–87, IEEE, 2012.

[34] M. Bogdanoski, T. Suminoski, and A. Risteski, "Analysis of the syn flood dos attack," *International Journal of Computer Network and Information Security (IJCNIS)*, vol. 5, no. 8, pp. 1–11, 2013.

[35] P. Kumar, M. Tripathi, A. Nehra, M. Conti, and C. Lal, "Safety: Early detection and mitigation of tcp syn flood utilizing entropy in sdn," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1545–1559, 2018.

[36] A. Hamza, H. H. Gharakheili, T. A. Benson, and V. Sivaraman, "Detecting volumetric attacks on lot devices via sdn-based monitoring of mud activity," in *Proceedings of the 2019 ACM Symposium on SDN Research*, pp. 36–48, 2019.

[37] S. Rashid and S. P. Paul, "Proposed methods of ip spoofing detection & prevention," *International Journal of Science and Research*, vol. 2, no. 8, pp. 438–444, 2013.

[38] E. Participation, "Computer misuse act 1990," Jun 1990.

[39] "Bcs."

[40] "Iet."

A Appendix

A.1 Resources provided alongside the report

Attached is the source code and an already-built virtual machine containing the code. It needs Qemu, a Ubuntu virtualization tool. The .qcow2 file can be run immediately as Qemu is installed.

A.2 Running the code and experiments

There are 2 networks network.py and network_firewall.py, one only has forwarding and the other has the implemented product. Both can be run with "sudo python3 network_firewall.py", and starts a mininet client. The controller can be run with "sudo python3 controller.py" Feel free to comment the line of code containing the traffic in the network script based on your experimentation. Feel free to modify the thresholds to your liking at the start of the P4 file "firewall_pod_syn_icmp_ban.p4" Some Python code might be similar to `https://github.com/nsg-ethz/p4-utils` or `https://github.com/nsg-ethz/p4-learning`.

A.3 Virtual machine

The virtual machine is prebuilt and seems to have a problem with the normal terminal so it is advised to use Guake instead.