Fingerprint-Based Automated Rule Generation for DDoS Mitigation using the Berkeley Packet Filter

Dirk Koelewijn
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
d.koelewijn@student.utwente.nl

ABSTRACT

Distributed Denial of Service (DDoS) attacks have become more and more present in our everyday society, both increasing significantly in numbers and intensity. Although more advanced methods for DDoS mitigation are emerging, there exists nearly no research on kernel level DDoS mitigation. Therefore, we designed a method to automatically generate extended Berkeley Packet Filter programs for DDoS mitigation, based on DDoS attack fingerprints from DDoSDB.org. We show that existing work only focuses on the performance of eBPF and that no research exist on DDoS mitigation using eBPF or similar techniques. Furthermore, we present a method to convert fingerprints to eBPF rules, as well as a method to reduce the size of fingerprints while maintaining as much precision as possible. Finally, we show that our method has an overall accuracy of over 95%, a true positive rate of at least 93% and a true negative rate for over 98% on more than 90% of the simulated attacks.

Keywords

Fingerprint-based DDoS mitigation, automated rule generation, extended Berkeley Packet Filter

1. INTRODUCTION

Distributed Denial of Service (DDoS) attacks have become more and more present in our everyday society. These attacks, in which targets are flooded by large amounts of internet traffic, have increased in numbers by 16% between the summer of 2017 and the summer of 2018 alone [2]. Besides that, the intensity in terms of maximum bandwidth has also increased: the largest observed attack had a strength of over 1.3 Tbps and attacks over 300 Gbps occur more frequently [2]. Content Delivery Network provider Akamai's latest report shows a maximum bandwidth growth of 9% per quarter, making the growth remarkably stable: the maximal bandwidth of DDoS attacks is now expected to double every two years [1].

To aid in the ongoing efforts to mitigate DDoS attacks, dr. Jair Santanna from the University of Twente launched *DDoSDB* [18]. This platform stores so-called *fingerprints* containing an analysis of many aspects of DDoS attacks. Furthermore, the platform allows users to download attack traffic that can be used to replay attacks. Using these fingerprints it is possible to automatically generate rules for DDoS mitigation systems.

A review from Osanaiye, Choo and Dlodlo shows that a lot of research on DDoS mitigation methods exists, all targeting different detection techniques, deployment locations and attack types [17]. In the past years, technologies like Border Gateway Protocol (BGP) Flowspec [8], several Intrusion Detection Systems [3, 15, 6] and Web Application Filters [20, 19] have emerged. However, these technologies only target routers or are functioning on top of operating systems, leaving low-level functionality of operating systems unused for DDoS mitigation.

To fill this gap, this research focuses on the use of the extended version of the Berkeley Packet Filter (eBPF), a technology that allows for filtering packets in the lowest layer of operating systems: the kernel [16]. Considering that filtering packets in the kernel could have significant performance advantages [5], the use of eBPF in addition to the current technologies could be highly valuable in the ongoing battle against DDoS attacks.

The goal of this paper is therefore to design a method for automated eBPF generation for DDoS mitigation based on attack fingerprints from DDoSDB and evaluate its accuracy. To achieve this goal, we defined the following research questions:

RQ1 What methods currently exist that use eBPF or similar techniques for DDoS mitigation or for packet filtering in general?

RQ2 What methods can be used to automatically generate eBPF rules for DDoS mitigation based on DDoSDB attack fingerprints?

RQ3 What is the accuracy of the designed generator of eBPF rules for DDoS mitigation based on DDoSDB attack fingerprints?

To answer the first question, we investigate current applications of eBPF for packet filtering, as well as existing research on similar DDoS mitigation techniques. Next, several possible methods are combined with the requirements and restrictions of eBPF, in order to design an automated eBPF generation method based on DDoSDB attack fingerprints. Finally, our design is be validated and evaluated by simulating corresponding attacks to answer the last question.

This paper starts with an elaboration on the existing methods for DDoS mitigation and discusses the added value of using eBPF in section 2. Next, we discuss the requirements for our designed method in section 3, combining

30th Twente Student Conference on IT Febr. 1st, 2019, Enschede, The Netherlands.

the requirements for DDoS mitigation systems and the requirements for eBPF. After this, we present the designed method in section 4, which is evaluated and validated in section 5. Finally, this is followed by the conclusions, future work and acknowledgements in sections 6 and 7.

2. EXISTING WORK

In order to answer the first research question, *What methods currently exist that use eBPF or similar techniques for DDoS mitigation or for packet filtering in general?*, this section will elaborate on existing work regarding eBPF and DDoS mitigation systems or packet filtering in general. First we will elaborate on the current usage of eBPF for packet filtering, after which we will elaborate on similar DDoS mitigation systems.

2.1 Packet filtering using eBPF

Although there exist many methods for fingerprint or signature based DDoS mitigation systems [14, 7, 13], there is little to no publicly available research on methods specially designed for DDoS mitigation using eBPF, nor on the accuracy of such methods. In this subsection we therefore discuss existing research on packet filtering using eBPF in general.

Bertin has written a paper [4] on behalf of Content Delivery Network provider *Cloudflare* on their solution for DDoS mitigation using eBPF. In the paper, Bertin mentions that using eBPF has performance advantages. However, the paper does not mention any numbers about the performance of eBPF, nor how rules are generated or the accuracy of their rules.

The paper of Høiland-Jørgensen et al [9] elaborates more on the performance of eBPF and especially its socket filtering function, eXpress Data Path (XDP). According to this paper, XDP is significantly more efficient in terms of CPU usage than popular packet processing tool *Data Plane Development Kit* (DPDK). The main reason for this is that XDP does not need a pulling mechanism to access the packets, as eBPF programs and with it XDP programs are invoked for each incoming packet. In the research, it is shown that XDP can reach the maximal capacity of the PCI bus at 115 Gbps. The same research also elaborates on DDoS performance: XDP could easily filter 10 Gbps of traffic on a single core, making XDP feasible for usage in DDoS mitigation [9].

Furthermore, Tumolo has showed in his paper that eBPF can be over 10 times faster at filtering packets for larger rule sets than the built-in firewall of Linux, *iptables* [22]. Furthermore, the latency of eBPF is always smaller than that of iptables and is nearly not increasing as the rule set grows.

All together, it can be concluded that research on the performance of eBPF has very promising results, while no research exists on methods specially designed for eBPF. Given the high performance, research into methods for DDoS mitigation using eBPF can be of great value in the fight against DDoS attacks.

2.2 Similar DDoS mitigation systems

The earlier mentioned review of Osanaiye, Choo and Dlodlo [17] shows that a lot of research has been done into DDoS mitigation systems, with many deployment locations and classification method types. The most popular deployment location is the access point, due to the ease of deployment [17]. Deploying at the access point has as drawback that the bandwidth may already be saturated, but does allow for application layer filtering of which research of Karnwal [12] is an example. Likewise, deploying at the source end or in the intermediate network allows to save bandwidth early on [17], but access to the application layer is impossible if the connection is encrypted.

In addition to the deployment type, two classification types exist. The first is signature or fingerprint based detection, which uses a known description of the attack to block it. The advantages of this type of classification is the accuracy in detecting known attacks, where disadvantages are maintaining the database of known attacks and the inability to detect unknown attacks [17]. The other classification type is anomaly based detection, that uses machine learning to detect any abnormal traffic. These methods are better in detecting known attacks, but are difficult to configure properly for accurate classification in general and do not perform as well as signature based attacks [17].

Of all the investigated articles in the review of Osanaiye, Choo and Dlodlo, there are none that do DDoS mitigation in the kernel or a similar location in operating systems. Instead, most access point mitigation systems are deployed in the virtual machine [6, 15, 3, 8, 7, 14] that contains the server. Except for the research mentioned before, no other research into kernel level DDoS mitigation could be found outside of the review as well.

All together, it can be concluded that all these researches solely focus on the performance of eBPF, which is very promising, and do not mention the accuracy of their methods. In addition, it can be concluded that, except for the method of Tumolo [22], little to no details are given on the used method itself. At last, it can be concluded that no similar techniques to eBPF are currently being used for DDoS mitigation or packet filtering in general. All together, it can therefore be concluded research into the accuracy of eBPF would be of added value.

3. REQUIREMENTS

DDoS mitigation systems aim to minimize the results of a DDoS attack. The main requirement of a mitigation system is therefore to maximize the amount of normal traffic and minimize the amount of attack traffic that reaches the destination. To achieve this, the design should not only accurately separate attack traffic from normal traffic, but also do this fast enough to prevent it from getting congested itself. Additionally, as our designed mitigation solution is meant to be used next to other solutions and not as a replacement, not filtering normal traffic could be considered extra important.

For DDoS mitigation based on DDoSDB fingerprints, we therefore define the following requirements for the design:

- The generated eBPF rules should be capable of filtering traffic on a normal computer in real-time for speeds up to 900 Mbps, the maximum capacity of our network setup;
- The method should only use DDoSDB fingerprints as resources for generating the eBPF rules.

Next to the requirements for DDoS mitigation systems in general, the usage of eBPF imposes additional requirements. For security reasons, eBPF rules have a maximum length after being compiled from C to assembly of 4096 instructions [23]. This imposes an extra challenge, limiting the maximum size for generated rules.

For eBPF, we therefore add the following requirement:

- The generated eBPF rules are together less than 4096 assembly instructions when compiled.

4. METHOD DESIGN

To answer the second research question, *What methods can be used to automatically generate eBPF rules for DDoS mitigation based on DDoSDB attack fingerprints?*, this section elaborates on the different possibilities and final design choices for a method to automatically generate eBPF rules out of DDoSDB attack fingerprints for DDoS mitigation. The method design can be split into three main parts:

Subsection 4.1 discusses the methods to convert fingerprints to rules. How fingerprints can be reduced to produce smaller rule sets in order to fit into eBPF rules will be discussed in subsection 4.2, after which the conversion of rules into eBPF rules will be discussed in subsection 4.3.

4.1 General rule generation

For every DDoS attack in DDoSDB, a *JSON* fingerprint stores general properties of an attack, including the protocol, source Internet Protocol (IP) addresses, source ports and destination ports. In addition to that, fingerprints can also store protocol specific information, like the Transmission Control Protocol (TCP) flags or the value of a Domain Name System (DNS) query. Listing 1 shows an example of a small User Datagram Protocol (UDP) fingerprint.

Listing 1. Example UDP attack fingerprint

```
{
  "start_timestamp": 1429087320.977101,
  "protocol": "UDP",
  "file_type": "pcap",
  "start_time": "2015-04-15 08:42:00",
  "dst_ports": [
    46608.0,
    50515.0,
    2579.0,
    37808.0,
    3587.0,
    ...
  ],
  "duration_sec": 31.86268186569214,
  "src_ips": [
    "14.134.128.104",
    "14.134.172.145"
  ],
  "src_ports": [
    32769.0
  ]
}
```

For each property, a fingerprint includes one or more values that where common for attack packets to have. For example, if we would use the fingerprint of listing 1, a UDP packet from IP `14.134.128.104` with port `32769` would be highly suspicious as both the source IP and port are in the list, whereas a packet from `1.2.3.4` with port `5467` would not be suspicious at all.

As can be seen from the listing, DDoSDB fingerprints do not include any probability weights or ratios. The only rate of suspicion that can be calculated is the number of properties in the packet that match with values in the fingerprint. Although this does not allow for a statistical approach, this does ease the decision making for rule generation: the only decision to make is the numeric value of this threshold, which can only be a small natural number due to the limited amount of properties that can occur in a fingerprint.

Listing 2 shows the relationship between a fingerprint and the rules in a Python example. For each property that the packet and the fingerprint share, it increases a counter if the value in the packet is in the list of values of the fingerprint. In the end, the threshold determines how many matching properties a packet needs to be dropped.

Listing 2. Fingerprint to rule conversion

```
matched = 0

# prop = property (keyword)
for prop in fingerprint:
    if prop in packet:
        if packet[prop] in fingerprint[prop]:
            matched += 1

if matched >= threshold:
    # Drop packet
else:
    # Pass packet
```

This threshold can influence the accuracy in two ways. Decreasing the threshold increases the chance of a random packet matching the rule, because less properties have to be matched. This results in an at least the same and possibly higher drop rate for both attack and normal packets. Likewise, increasing the threshold decreases the chance that a random packet is matched and will therefore result in an at most the same and possibly lower drop rate for both attack and normal packets.

Please note that setting the threshold is not included in the method. Multiple thresholds will be tested in the verification in section 5.

4.2 Fingerprint reduction

The fingerprint to rule generation method described in the previous section works for some fingerprints, but not for all: due to the limited size of eBPF rules, as mentioned in section 3, not all fingerprints produce eBPF rules small enough to be loaded. As a result of this, the size of a significant amount of fingerprints has to be reduced in order to fit into the maximum of 4096 instructions for eBPF.

This reducing can only be done by removing or replacing values of properties, or even entire properties, until the amount of values is below a maximum amount, P_{max}. This maximum amount, P_{max}, is the maximum size that a fingerprint can have in order to be loaded into eBPF and is dependent on the efficiency of the implementation. In practice, only the amount of values for the IP address, source port and destination port have to be reduced, because these are the only properties that can have large amounts of values.

In order to meet our requirements defined in section 3, a reduction method is needed that minimally impacts the accuracy of a fingerprint. Considering that the fingerprints do not contain the likelihood for a value to be present in an attack, the statistically best reduction can be simplified to satisfying the following two requirements:

- The chance that a *random* packet matches the reduced fingerprint should be as small as possible;
- The chance that a DDoS-related packet that *originally* matched the fingerprint matches the reduced fingerprint should as high as possible;

However, fingerprints sizes vary from less than 10 values to over 500,000 values for only the IP address. The choice for the statistically best reduction is therefore challenging: With 2^{16} ports, 2^{32} IP (version 4) addresses and many more aggregated groups to choose from and up to P_{max} choices to make, the number of possible reductions is huge. This can make it very hard to calculate or guess the best possible reduction in short period of time. Therefore, a reduction method should have an amount of possible configurations small enough to find the best possible configuration in a reasonable amount of time.

In section 4.2.1, we will elaborate on a suitable reduction method. After that, we will elaborate on how the statistically best configuration can be found in section 4.2.2.

4.2.1 Reduction method

Considering that there is no likelihood given for a value of a property, there would be no statistical basis to determine which values can be deleted best: all values should therefore be treated as equally likely to occur. Furthermore, deleting values in a fingerprint with many times as values as allowed for P_{max} can drastically decrease the amount of attack traffic being dropped in various cases, although this depends on the value for the threshold discussed in section 4.1.

The only other way to reduce the amount of values is to *aggregate* values into groups. This guarantees that all packets that would have matched the fingerprint originally still match. The disadvantage of aggregation is that it can also increase the amount of normal packets being matched, but unlike with deleting values, it is possible to determine a configuration that is statistically best. This will be discussed in section 4.2.2. Therefore, the reduction method will be based on aggregation.

Aggregating can be done in two main ways: by *distance* and by *bit shift*. In the first case, all values that are less than the specified distance apart will be aggregated in the a (min, max) group, which will match all values for which $value \in [min, max]$. Table 1 shows an example distance aggregation. In practice, this means that a group still needs two values, namely the minimum and maximum value, making it only reduce the amount of values for groups larger than two.

Distance	Values
0	{3, 4, 6, 9}
1	{(3, 4), 6, 9}
2	{(3 ,6), 9}
3	{(3, 9)}

Table 1. Example distance aggregation

Next to distance, aggregation can also be done by bit shift. In this case, the last n bits are chopped from the value, after which values are aggregated. Groups that only differ by the last bit can recursively be merged as well by chopping the last bit off. An advantage of this method is that a group of values now only needs one value, namely the remaining bits, meaning that less aggregation can be required. In addition, bit shifting has only a small possible amount of configurations, because a property only has a limited amount of bits. The disadvantage is that this reduction can include relatively more value in groups. Table 2 shows an example aggregation by bit shift.

Considering that bit shifting requires less values and has less possible configurations allowing for an easier choice, the method we use is bit shifting. Each property p in the set of properties P now has to be aggregated in a way that for the value count v_p and the maximum amount of values P_{max} holds that:

Shift	(Bits, shift)
Normal	{(1000, 0), (1010, 0), (1011, 0), (1100, 0)}
0	{(1000, 0), (101, 1), (1100, 0)}
1	{(10, 2), (110, 1)}
2	{(1, 3)}

Table 2. Example bit shift aggregation

$$\sum_{p}^{P} v_p \leq P_{max} \tag{1}$$

Given that both the source and destination port have 16 bits and the IP address has 32 bits and are the only properties that will be reduced, this gives an amount $16*16*32 = 8192$ possible configurations. In section 4.2.2, we will discuss how we can efficiently choose the most optimal configuration.

4.2.2 Configuration

Considering that aggregation guarantees that all originally matched packets still match, we can define our *precision* as the chance that a random packet not matches the fingerprint.

If P is the set of properties, n_p is the number of values that match property p and N_p is the total number of possibilities for p:

$$precision = 1 - \prod_{p}^{P} \frac{n_p}{N_p} \tag{2}$$

The best reduction can now be defined as the reduction that leaves the highest precision. Or, when expressed in the amount of distinct packets M that could match the fingerprint:

$$\begin{aligned} M &= \prod_{p}^{P} n_p \\ precision &= 1 - (\prod_{p}^{P} \frac{1}{N_p} * M) \end{aligned} \tag{3}$$

As N_p is constant for each fingerprint instance as long as no properties are deleted, using the reduction with the lowest M will also have the highest *precision*. As the IP address *ip*, source port *src* and destination port *dst* are the only properties that can contain multiple values, M can be rewritten as:

$$M = n_{ip} * n_{src} * n_{dst} \tag{4}$$

In order to find the optimal reduction, a bit shift should be found for each property so that equation (1) holds and that the value of M and with it the *precision* are maximal. First, we calculate the minimal bit shift for each property so that the amount of values for that property are below P_{max} with a binary search like algorithm. After that, all larger shifts will be tried for each individual property, calculating both the amount of values as the amount of values that would possibly match.

Finally, the method will search for the combination with the highest precision for which equation (1) also holds. This combination can easily be found using brute force considering that equation (1) and (4) have to be calculated at most 8192 times, a calculation of only milliseconds. More efficient ways of finding this combination may be possible, but this is sufficient to answer the second research question.

4.3 Converting rules to eBPF

To convert a set of rules into eBPF, two things are needed: A method to generate a C program out of rules and a method to convert the C program to eBPF. For the last part we used IOVisor's Berkeley Compiler Collection (BCC) [11], that allows to compile and load eBPF rules into the kernel from Python.

Generating a C program is slightly more complicated, because packets have to be decomposed manually. In addition, the generated program should use as few code as possible and conform to eBPF's strict security checks for actions that could crash the kernel. In order to achieve this, a tool was implemented in Python that inserts code modules for decomposition based on the protocols it depends on and converts the rules itself to C code. In the current implementation, the value of P_{max} is around one thousand.

Considering that efficiency is only a minor concern in this research, this tool will not be further elaborated on in this paper. The tool will be freely available after the publication of this paper on `github.com/DirkKoelewijn/research-project`. Due to the limited time scope of this research and the fact that packages have to be decomposed manually, the support for attack protocols is currently limited to UDP and TCP.

To answer our second research question, *What methods can be used to automatically generate eBPF rules for DDoS mitigation based on DDoSDB attack fingerprints?*, we showed that DDoS attack fingerprints contain no information on value likelihood, making the amount of matching properties the only, simple indicator of whether a packet belongs to the attack that can be easily compared to a threshold with a limited amount of options. Furthermore, we showed that fingerprints may have to be reduced and that aggregation by bit shifting is favorable due to the big reduction of values and relatively low amount of possible configurations. In addition, we showed that for this reduction, the most optimal configuration can be found using basic probability theory and in a reasonable period of time. Last, we discussed to conversion of rules into eBPF. We discussed various design options and conclude that the presented design is expected to have the best results.

5. VERIFICATION

To verify whether the requirements defined in section 3 are met and to answer the last research question, *What is the accuracy of the design for automatically generated eBPF for DDoS mitigation based on DDoSDB attack fingerprints?*, this section elaborates on both the verification method and the results.

5.1 Verification method

The most obvious method for verification is to simulate attacks to a computer that runs the correspondingly generated eBPF rules. In addition to just manually reading attack files, simulations allow to check test real-time filtering and to check whether generated rules can be compiled to the limited amount of instructions. Therefore, simulations will be used to verify the designed method.

The next subsections elaborate on the setup for the simulations, which attacks will be simulated and what will be used as normal traffic. In addition, the method to label traffic packets and how attacks will be replayed will be discussed. Finally, the verification metrics will be discussed.

5.1.1 Setup

The simulations will be performed on a closed network in which only two computers, an attacker and a defender, are connected via a router. Both computers are normal consumer computers that are connected to the router via Ethernet cables. All components in the network have a theoretical maximum capacity of at least 1 Gbps, but in practice the maximum speed that could be reached when replaying the attacks was often between 900 and 950 Mbps.

5.1.2 Attacks

Simulations are done for 129 UDP attacks and 156 TCP attacks, 285 attacks in total. As mentioned before in section 4.3, UDP and TCP are the only attack protocols that are currently supported. The value count of the fingerprints corresponding to these attacks ranges from less than ten to nearly 500,000 values, of which about half requires reduction. It should be noted that this are not all UDP and TCP attacks from DDoSDB: slightly more than 50 TCP attack fingerprints where removed after it was concluded that the specified TCP flags were a clear mismatch with the corresponding attack, probably due to a bug in the fingerprint generation.

5.1.3 Normal traffic

To test the influence of the method on normal internet traffic, normal traffic has to be sent with the attack. The normal traffic comes from TCPReplay's `bigFlows.pcap` [21], containing a total of nearly 800,000 packets (368 MB) from 132 applications. This traffic was captured on "a busy private network's access point to the Internet" [21] and is expected to be a good representation of how normal internet traffic looks like.

5.1.4 Labeling

To be able to determine later on whether a packet was attack or normal traffic, all sent packets have to be labeled. Considering that the traffic is sent over a local network that only requires the Ethernet protocol to send packets, we chose to modify an IP version 4 property that is not present in a fingerprint: the Time To Live (TTL). This property is normally used to stop packets from circling around the internet, but is not modified or used in our local network. The TTL is therefore very useful to label packets.

Attack traffic is in our case labeled with a TTL of 10, where normal traffic is marked with a TTL of 20. These low values for the TTL, that is recommended to start at 64 by the Internet Assigned Numbers Authority (IANA) [10], are not expected to be reached for other packets inside our network of only two computers and a router.

5.1.5 Configuration of simulations

To be able to test the real-time packet filtering requirement defined in section 3, the combined bandwidth of attack and normal traffic should be at least 900 Mbps. To allow both the attack as the normal traffic to send all their packets multiple times, the bandwidth is equally split between the two: both are allocated 450 Mbps using `tcpreplay`'s `-M` option. A higher total bandwidth may congest the setup, as mentioned in subsection 5.1.1 and is therefore

not used.

The simulation process is fully automated for all attacks, in which the attacker and defender communicate via a direct TCP stream. An attack simulation consists of the following steps for each attack:

1. The defending computer asks the attacking computer to check if the attack traffic is present at the attacker. If not, the attack should be downloaded: this does however require to connect the network to the internet. All attack traffic was therefore downloaded before the final tests were done;
2. The attacker sends that it is ready to attack;
3. If the defender is done with optionally reducing the fingerprint, it requests to start the attack;
4. The attacker sends a confirmation of the attack, after which the defender immediately loads the eBPF rules for 30 seconds. Three seconds after the confirmation was sent, the attack is started for 25 seconds;
5. The defender keeps track of packets are dropped or passed correctly and saves the result.

This method allows to test large amounts of attacks without any human intervention. Furthermore, 25 seconds of sending both normal and attack traffic is sufficient to send all packets multiple times. All attacks will be simulated with three thresholds for the amount of rules to be matched: match all, match all but one and match all but two.

5.1.6 Metrics

Filtering packets is a form of binary classification: packets have to be classified as attack or not. Therefore, the following common binary classification metrics will be used:

- *Accuracy.* The fraction of packets that have been classified correctly into attack or normal traffic;
- *True Positive Rate (TPR).* The fraction of attack packets that has been dropped and therefore has been classified correctly;
- *True Negatve Rate (TNR).* The fraction of normal packets that has *not* been dropped and therefore has been classified correctly.

Considering that the accuracy is very dependent on the chosen bandwidth division, the TPR and TNR will be considered the most important.

5.2 Verification results

The results of the simulations described in the previous section are divided into five subsections. Int the first three subsections, we will discuss the results for the three threshold values as mentioned in section 5.1.5. For each threshold, the accuracy metrics as mentioned in section 5.1.6 will be discussed. The next subsection discusses the effect of the reduction and the results for the other, more general requirements will be discussed in the last subsection.

5.2.1 Strict: match all

The most strict threshold, that requires all fingerprint properties to be matched before it drops a packet, is expected to drop both the least attack as the least normal packets as mentioned in section 4.1. Figure 1 shows the

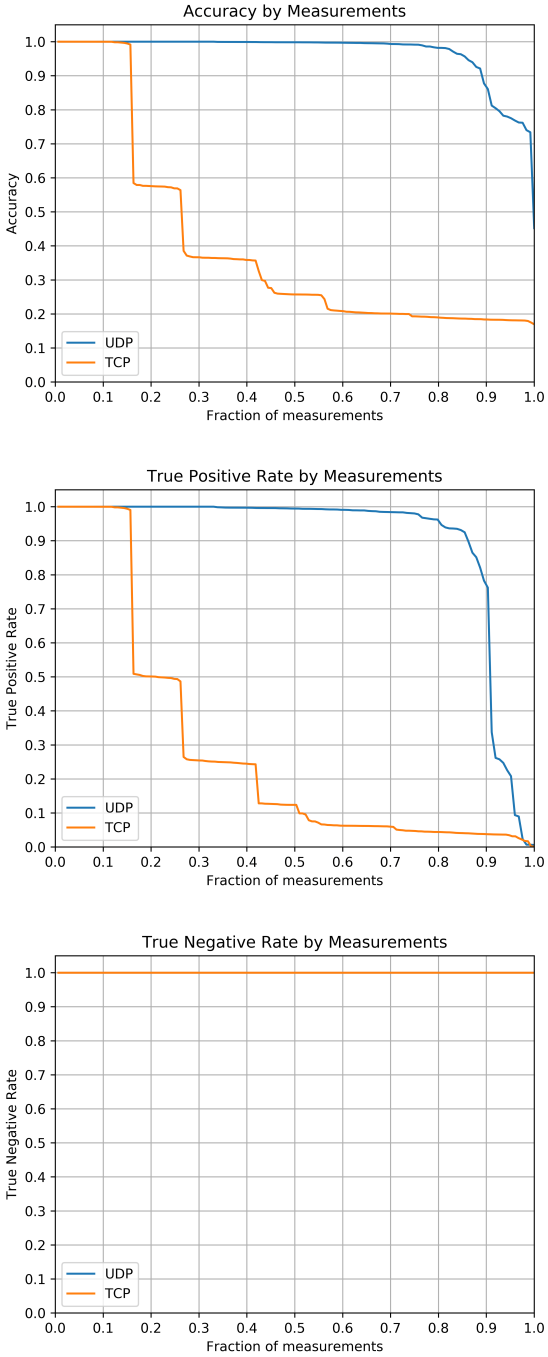

Figure 1. Accuracy metrics by fraction of measurements for strict threshold

accuracy metrics by the fraction of measurements for the most strict threshold.

Both for UDP and TCP, using the most strict threshold results in a near perfect TNR: in all simulations for both protocols, over 99% of the normal packets were not dropped. For TCP, most attack packets were however also not dropped, with 84% of the measurements having a TPR of approximately 50% or lower. This clearly indicates that the most strict threshold is too strict for the packets of TCP attacks.

The UDP attacks have a better TPR and with it better accuracy, considering that approximately 86% of the simulated attacks drop over 90% of the attack packets, indicating that much UDP fingerprints contain all values for most packets. Starting from 80% of the measurements for UDP, the TPR drops to a 0% accuracy. For the last 15% of the measurements, the threshold is too strict.

5.2.2 *Less strict: Match all but one*

For the slightly strict threshold, match all but one it can be expected that more packets will be dropped, possibly increasing the TPR but possibly decreasing the TNR. Figure 2 shows the accuracy metrics by the fraction of measurements for threshold that matches all but one.

The graphs show that both the accuracy as the TPR have increased for both protocols, but most significantly for TCP: Where over 80% had an accuracy under 60% for the most strict threshold, now approximately 93% of the TCP attacks can be mitigated with an accuracy of at least 98%. This indicates that most TCP attack packets have all but one of the IP address, ports or TCP flag and are therefore hard to match with the strict threshold. At the end of the graph, a steep drop in TPR however exists. Unlike the impact on the TPR, the impact on the TNR is very small for TCP: in less than 1% of the measurements, more than 10% of the normal traffic was blocked.

Although the chance for a normal packet to match the eBPF rules is higher for a less strict threshold, the UDP results do barely reflect any impact on the TNR: all measurements have a TNR of over 99%. The TPR drop in the end has been pushed to the right of the graph, reaching a TPR of above 90% for approximately 92% of the measurements. The drop in TPR is remarkably similar to that of TCP.

All together, lowering the threshold to all but one has a significant positive effect on the TPR, especially for TCP, while having only a very minor impact on the TNR. Therefore, the accuracy for this less strict threshold has significantly increased.

5.2.3 *Least strict: Match all but two*

For the least strict threshold that was tested, match all but two, the TPR is expected to further increase whereas the TNR is expected to finally decrease significantly with respect to the other thresholds. Figure 3 shows the accuracy metrics for the least strict threshold.

For the UDP measurements, the TPR has further increased as now approximately 95% has a TPR of over 90% and all measurements have a TPR of over 77%. For TCP, the TPR has also increased even further: approximately 99% of the measurements match over 99% of the attack traffic. It even turned out that the attack causing the drop has an incorrect fingerprint, but this was not noticed before like with the incorrect fingerprints mentioned in section 5.1.2. This indicates that for all tested TCP attacks, 99% of the attack traffic matches at least two out of the IP address,

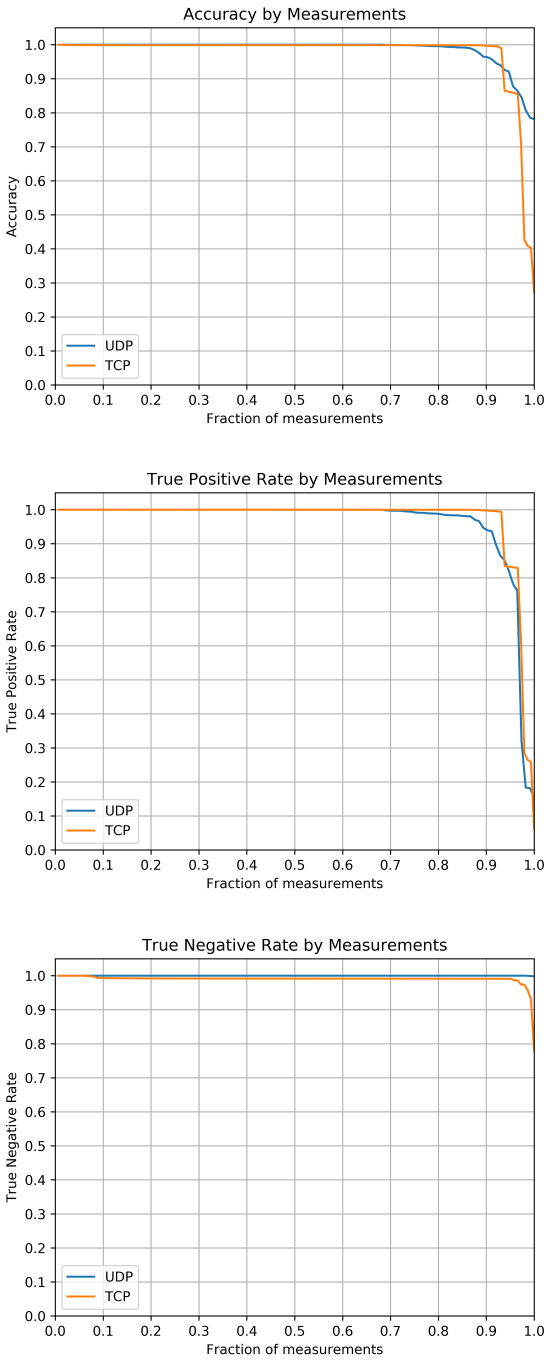

Figure 2. Accuracy metrics by fraction of measurements for less strict threshold

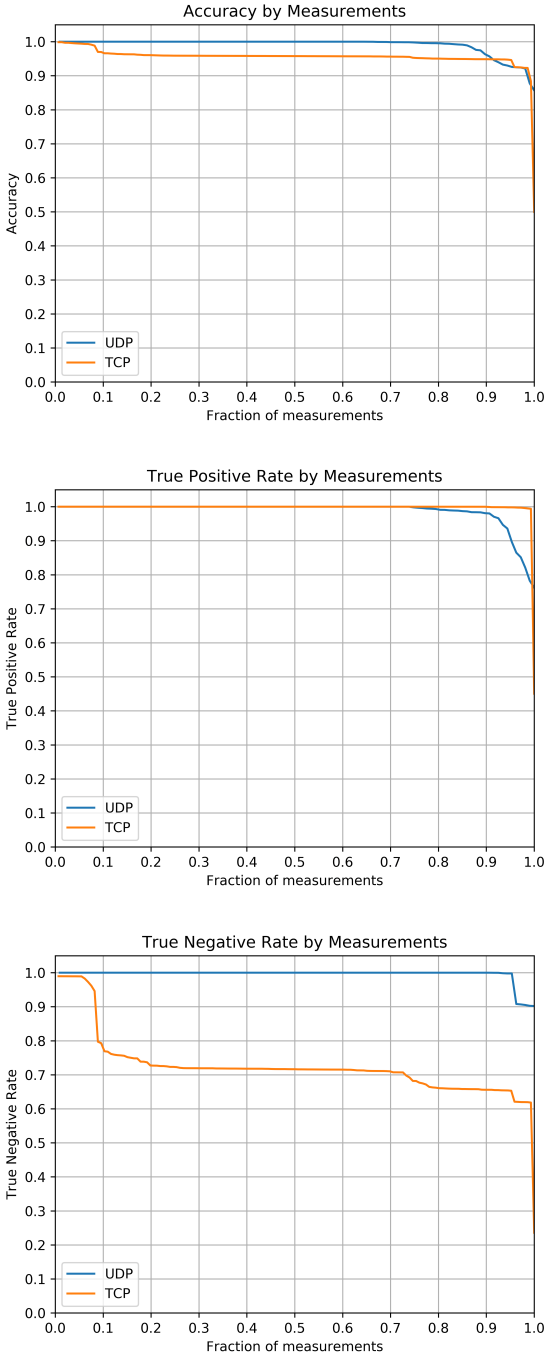

Figure 3. Accuracy metrics by fraction of measurements for least strict threshold

ports and TCP flag.

For UDP, the decreased threshold does not have a large impact on the TNR of most measurements: approximately 95% of the attacks has a TNR of over 99%. For all attacks the minimal TNR is just 90%, making UDP fingerprints still classifying normal packets with a higher accuracy than it classifies attack packets. This can partially be explained because of the normal traffic: it only consists for 19.4% out of UDP packets, while 80.4% are TCP packets. Nevertheless, the TNR for UDP is remarkably high.

The results largely match the expectations all together, further increasing the TPR and decreasing the TNR. All together, accuracy has gone up for UDP as the accuracy for normal traffic did not decrease much, whereas the accuracy has lowered the accuracy for TCP as a result of significant decrease in accuracy for normal traffic.

5.2.4 Impact of reduction

Another factor than the threshold that could significantly affect accuracy is the reduction of certain fingerprints. The main impact of reduction is expected to be classifying more traffic as attack traffic, as more packets potentially match the fingerprint. To investigate this, this section will elaborate on the accuracy related to the original amount of values in the fingerprint.

Reduction has two possible side effects: it can increase the amount of normal traffic being classified as attack traffic, but it can also do this for the attack traffic itself. The reduction can therefore both decrease the TNR as increase the TPR. First, we will discuss the consequences for the TNR.

The TNR does not decrease for the most strict threshold, as figure 1 has shown. Therefore, we will only investigate the relation between TNR and reduction for the two less strict thresholds. Figure 4 shows the spread of the TNR by the amount of values in the original fingerprint.

Given that only fingerprints larger than a 1000 values were reduced, it is clearly visible that reduction significantly impacts the TNR for the least strict threshold. In this case, the combination of both reduction as using a loose threshold causes too much normal traffic to be classified as attack traffic.

For the more strict threshold however, match all but one, reduction shows to have little to no impact. The accuracy is still near 99% for nearly all fingerprints that are reduced. The fingerprint with the only TNR below 80% appeared to be incorrect, as mentioned in the previous subsection.

For the TPR, reduction also shows to have little to no impact. The graph shown in figure 5 shows this most clearly, as the most and least strict respectively drop a very low and very high percentage of the attack traffic anyway.

Although the lowest TPR's are corresponding to fingerprints that had to be reduced, the main part of reduced fingerprints performs good and the well performing fingerprints are bigger in size. Given that the larger fingerprints have more added loss of precision with reduction, it is considered unlikely that reduction structurally improves the TPR. This may only be the case for the two largest UDP fingerprints, but the amount of large UDP fingerprints is so small that it is hard to find a clear cause.

All together, the results show that reduction has little to no provable impact on both the TPR as the TNR. A systematic analysis of the attack and normal traffic files would however result in a decisive answer on this question. Due

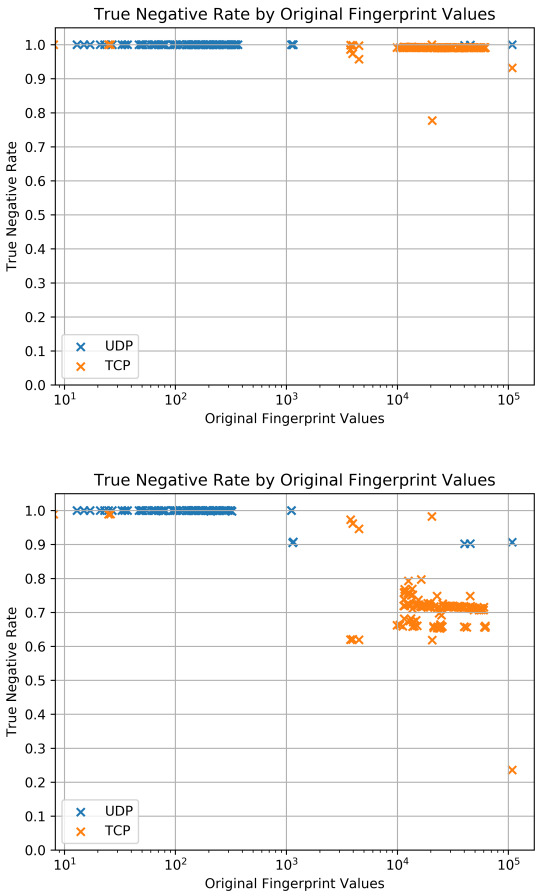

Figure 4. True Negative Rate (TNR) by amount of values in original fingerprint. Top: match all but one, bottom: match all but two.

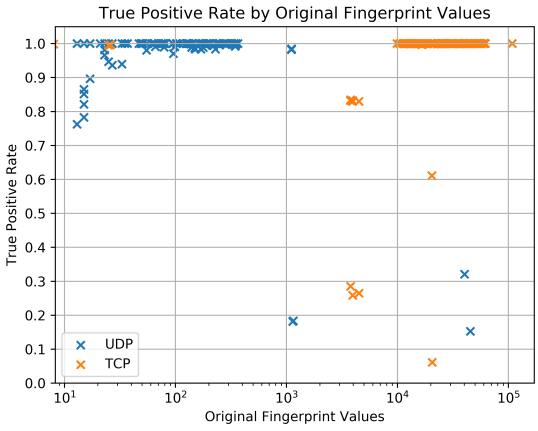

Figure 5. True Positive Rate (TNR) by amount of values in original fingerprint for threshold match all but one

to the limited time scope of this research, this has not been investigated.

5.2.5 *General requirements*

Next to the accuracy related requirements, section 3 defined three other requirements: real-time filtering for at least 900 Mbps, exclusive use of DDoSDB to generate eBPF rules and the generated eBPF rules should fit into the limited amount of assembly instructions.

All $285 * 3 = 855$ simulations were tested in less than 9 hours, making an average simulation cost less than 38 seconds. This includes, next to the 25 seconds of attack time, five seconds of waiting, the time to reduce and compile the fingerprint and the time to determine which fingerprints had the right protocol. In addition, manual checking of a set of randomly chosen attacks showed that all sent packets had been classified in real-time. Together, this shows that the DDoS mitigation method can be deployed and ran in short notice.

In addition, the entire method only uses the knowledge in the fingerprint itself and basic probability theory to generate eBPF rules. Although some exceptions exist, practically all reduced fingerprints have been successfully compiled into eBPF. For the exceptions, it was often unclear what caused the compiler to fail.

All together, we can now make our conclusions on the last research question, *What is the accuracy of the design for automatically generated eBPF for DDoS mitigation based on DDoSDB attack fingerprints?*

We have shown that the accuracy of the design is significantly impacted by the chosen threshold by evaluating three possible thresholds for both supported protocols. Next, we have shown that the maximal threshold significantly decreases the true positive rate, whereas the least strict threshold significantly decreases the true negative rate. It can therefore be concluded that the method only satisfies the requirements with the threshold set to all but one. We have shown that this threshold has a true positive rate of over 99 and 94% for more than 90% of the measurements of respectively TCP and UDP, while having a true negative rate of over 98% for over 96% of the measurements.

Furthermore, we have shown that our reduction method has little to no provable positive or negative impact on the accuracy. Additionally, the additional requirements for real-time packet filtering, basing eBPF rules only on DDoSDB fingerprints and compiled size have been satisfied.

All together, it can be concluded that our method successfully satisfies the requirements with an overall accuracy of over 95% , a true positive rate of at least 93% and a true negative rate for over 98% for more than 90% of the measurements.

6. CONCLUSIONS

In this research, we designed and evaluated a method to automatically generate eBPF programs for DDoS mitigation based on attack fingerprints of *DDoSDB*. We have identified the most important requirements, discussed related existing work and presented a design for such a DDoS mitigation system. Additionally, we have analyzed and verified the designed method for multiple configurations and design choices.

First, we conclude that related existing work only focuses on the performance of eBPF in packet filtering instead of

on the accuracy. We also showed that the performance of eBPF is very promising for DDoS mitigation and showed that little to no research exists on DDoS mitigation methods that use techniques similar to eBPF.

Next, we concluded that DDoSDB attack fingerprints contain no information on value probability and showed that this simplifies the conversion of fingerprints to rules to only one configurable threshold. Additionally, we discussed various design choices, we showed that some fingerprints have to be reduced and that we can find a most optimal configuration for this reduction with simple probability theory.

Finally, we have shown the method configuration significantly impacts the accuracy of the method. We have shown that for the optimal configuration, our method has an overall accuracy of over 95% , a true positive rate of at least 93% and a true negative rate for over 98% for over 90% of the measurements. All together, we therefore conclude that our method for automated eBPF generation based on DDoSDB fingerprints successfully satisfies all specified requirements.

However, future work could further elaborate on the possibilities of eBPF for DDoS mitigation. The usage of eBPF for more attack protocols has to be investigated, as well as more advanced fingerprint reduction methods that could further improve the accuracy. Additionally, extending the fingerprints with value probabilities could potentially result in improved accuracy. Finally, the actual performance of the method has to be investigated as well.

7. ACKNOWLEDGEMENTS

At the end of this research, I would like to take a moment to thank my supervisor, dr. Jair Santanna, for his advice, support and unstoppable enthusiasm. Without his efforts, this research would not have been possible. Furthermore, I would like to thank my other reviewers for the useful feedback and anyone who is or was involved in the development of the tools used in this research that greatly benefited this research.

8. REFERENCES

[1] Akamai Technologies. State of the Internet: A Year in Review, December 2018.

[2] Akamai Technologies. State of the Internet: Summer 2018, June 2018.

[3] A. Bakshi and B. Yogesh. Securing cloud from DDOS attacks using intrusion detection system in virtual machine. *2nd International Conference on Communication Software and Networks, ICCSN 2010*, pages 260–264, 2010.

[4] G. Bertin. XDP in practice: integrating XDP into our DDoS mitigation pipeline. 2017.

[5] Cloudflare. Advanced DDoS Protection and Mitigation. https://www.cloudflare.com/ddos/, 2018. Accessed: 2018-11-24.

[6] I. Gul and M. Hussain. Distributed cloud intrusion detection model. *International Journal of Advanced Science and Engineering Technology*, 34:71–82, 2011.

[7] S. Gupta and P. Kumar. VM Profile Based Optimized Network Attack Pattern Detection Scheme for DDOS Attacks in Cloud. *Communications in Computer and Information Science*, 377 CCIS:255–261, 2013.

[8] N. Hinze, M. Nawrocki, M. Jonker, A. Dainotti, T. C. Schmidt, and M. Wählisch. On the Potential of BGP Flowspec for DDoS Mitigation at Two Sources: ISP and IXP. In *ACM SIGCOMM 2018 Conference Posters and Demos, August 20-25, 2018, Budapest, Hungary*, pages 57–59, 2018.

[9] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller. The eXpress data path. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies - CoNEXT '18*, 2018.

[10] Internet Assigned Numbers Authority. Internet Protocol Version 4 (IPv4) Parameters. http://www.iana.org/assignments/ip-parameters/ip-parameters.xml, 2018.

[11] IOVisor. BCC - Tools for BPF-based Linux IO analysis, networking, monitoring, and more. https://github.com/iovisor/bcc, 2019.

[12] T. Karnwal, T. Sivakumar, and G. Aghila. A comber approach to protect cloud computing against xml ddos and http ddos attack. In *2012 IEEE Students' Conference on Electrical, Electronics and Computer Science*, pages 1–5, March 2012.

[13] T. Karnwal, S. Thandapanii, and A. Gnanasekaran. A filter tree approach to protect cloud computing against XML DDoS and HTTP DDoS attack. *Advances in Intelligent Systems and Computing*, 182 AISC:459–469, 2013.

[14] C. C. Lo, C. C. Huang, and J. Ku. A cooperative intrusion detection system framework for cloud computing networks. *Proceedings of the International Conference on Parallel Processing Workshops*, pages 280–284, 2010.

[15] A. M. Lonea, D. E. Popescu, and H. Tianfield. Detecting DDoS attacks in cloud computing environment. *International Journal of Computers, Communications and Control*, 8(1):70–78, 2013.

[16] S. Mccanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. Technical report, 1992.

[17] O. Osanaiye, K. K. R. Choo, and M. Dlodlo. Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework. *Journal of Network and Computer Applications*, 67:147–165, 2016.

[18] J. Santanna. DDoS DB. https://ddosdb.org/, 2018.

[19] S. Sivabalan and P. J. Radcliffe. A novel framework to detect and block DDoS attack at the application layer. *IEEE 2013 Tencon - Spring, TENCONSpring 2013 - Conference Proceedings*, pages 578–582, 2013.

[20] D. Stevanovic and N. Vlajic. Application-layer DDoS in dynamic Web-domains: Building defenses against next-generation attack behavior. *2014 IEEE Conference on Communications and Network Security, CNS 2014*, pages 490–491, 2014.

[21] TCPReplay. Sample captures. http://tcpreplay.appneta.com/wiki/captures.html, 2019.

[22] M. Tumolo. Towards a faster Iptables in eBPF. 2018.

[23] E. Zannoni. New (and Exciting!) Developments in Linux Tracing, 2015.