Sorting the Garbage: Filtering Out DRDoS Amplification Traffic in ISP Networks

Yury Zhauniarovich
Perfect Equanimity
Minsk, Belarus
yury@perfectequanimity.com

Priyanka Dodia
Qatar Computing Research Institute, HBKU
Doha, Qatar
pgdodia@hbku.edu.qa

***Abstract*—Distributed Reflected Denial of Service (DRDoS) attacks have been continuing to grow unprecedentedly in the recent years. Attackers abuse genuine services running some application protocols built over UDP to generate amplified traffic targeting victim network. An Internet Service Provider (ISP) may host hundreds or even thousands of hosts running these vulnerable protocols that could become amplifier nodes in DRDoS attacks. If abused, they can collectively cause large volumes of garbage amplification traffic flowing out of the ISP network. This wasteful bandwidth consumption costs the provider money and loss of Quality of Service (QoS) to its customers. Moreover, the owners of services vulnerable to amplification have to spend their resources to process illicit requests.**

In this paper, we propose a novel idea to filter out garbage traffic from an ISP network. We employ a special type of a honeypot that collects information about ongoing DRDoS attacks, and Software Defined Network (SDN) paradigm offering us a unified interface to deploy firewall rules on a large variety of network devices. The rules block incoming amplification requests from reaching amplifiers located within the provider network rescuing vulnerable services from being abused. This prevents garbage traffic from leaving the network enabling the provider to save money and improve QoS. Moreover, our solution also contributes to victim's liveliness because it reduces the attack traffic reaching the target network. In addition, it stimulates ISPs to implement ingress filtering best practices for all its network routers in order to minimize damage from an attacker located in the same network.

***Index Terms*—amplification attacks, garbage traffic filtering, honeypot, ISP networks**

I. Introduction

During the last several years, we observe an unprecedented growth of the number and the size of DRDoS attacks. In such attacks, crafted requests are sent to genuine machines, called amplifiers, which return amplified responses to a spoofed IP address of a victim, completely exhausting victim's bandwidth. These responses can be hundreds or even thousands [1] times larger in size than the corresponding originating requests. For instance, Github, a well-known platform for software development, has withstood such an attack experiencing whopping bandwidth of 1.35 Tb/sec [2]. Due to the low resource requirements from an attacker and a possibility to stay stealthy, these attacks have quickly gained popularity.

Y. Zhauniarovich was with Qatar Computing Research Institute, HBKU.

Amplification attacks are possible due to four factors: (1) there is no possibility for a genuine machine to check if a request comes from an original IP address (no sender verification); (2) some protocols return a response considerably larger in size than the corresponding request (amplification); (3) there are many amplifiers in the Internet; and (4) traffic with spoofed IP addresses is allowed to pass network perimeters. A number of initiatives have been proposed to eliminate or reduce the influence of these factors. First, some vulnerable protocols have been patched either to remove or to reduce the amplification factor [3]. Second, a number of services around the world have been launched to reveal vulnerable hosts and educate people how to fix the issue [4]. Third, tools to shutdown or reduce the power of an attack by sending special commands to vulnerable hosts have been also proposed [5].

However, the most effective way of dealing with DRDoS attacks is to prevent packets with forged source IP addresses to pass through networks. This goal can be achieved if routing entities apply filtering of ingress traffic, allowing only the packets with valid source IP addresses to pass. Such recommendations have been provided in RFC 2827 [6] better known as IETF's Best Current Practice document 38 (BCP 38). However, 18 years after this document appeared they are not yet applied everywhere [7] because ISPs do not directly benefit from their implementation. These recommendations require an ISP to spend its resources on the recommendation implementation and traffic filtering, while they do not protect itself from forged traffic coming from external networks. Not surprisingly, DRDoS attacks are still a real threat to the Internet.

ISPs suffer a lot from this unwanted traffic. First, it exhausts ISPs' and their customers' bandwidth affecting QoS. Nowadays, ISPs in some countries, e.g., in the UK [8], must report to their customers minimum guaranteed speed, and if they fail to deliver it the users have the right to break the contract without any penalty. Huge amounts of garbage DRDoS traffic may indirectly cause such an outcome. Second, ISPs are usually entitled to pay for the traffic, especially, if it is asymmetric [9]. That is why, they are not happy with the services entitled for its generation, like Netflix and YouTube, so that they want to take down or violate the net neutrality principle [10]. Clearly, the traffic generated by amplifiers hosted within the perimeter of an ISP network may reach substantial amounts. For instance, based on our experiments only a single host vulnerable to

NTP amplification could generate more than 2 TB of garbage traffic a day. An ISP network may host hundreds or even thousands of such amplifiers that if abused, could potentially waste significant amount of ISP's resources and money.

In this paper, we propose a novel idea of filtering out such spoofed traffic at the edge of an ISP network. This prevents garbage traffic being generated by amplifiers located within the network, saving ISP's resources. Unlike the majority of the existing DRDoS mitigating solutions [11]–[13] that are focused on protecting victim's network, we aim at shielding amplifiers from unwanted traffic. Contrary to BCP 38 [6], ISPs *directly benefit from our solution* because it saves their money and improves QoS for their customers. Moreover, it aids other Internet citizens indirectly. First, our method permits to save resources of ISP customers hosting services vulnerable to amplification. Second, our solution helps to reduce or completely eliminate (if deployed world-wide) attack traffic to victims. Third, it encourages an ISP to implement BCP 38.

Our approach relies on an amplification honeypot, e.g., AmpPot [14], that provides information about an ongoing attack. By default, an amplification honeypot is not used by benign clients because it does not advertise its services. Therefore, only malicious users, who have previously scanned the network and discovered this vulnerable host, employ it for an attack. Benignly participating in an attack, the honeypot collects the information, namely victim IP address and what service is abused, that allows us to block spoofed traffic entering ISP's network. As a result, it will not reach other vulnerable hosts in the network and will not be amplified.

To filter out spoofed traffic, in this work we employ Software Defined Networking (SDN) paradigm. In particular, we developed an SDN Firewall application that, having information from the honeypot, automatically deploys firewall rules on edge switches that drop spoofed traffic. SDN provides us a generic interface to interact with network devices making our solution vendor-agnostic. It is also possible to use other interfaces to contact with network devices, e.g., Interface to Network Security Functions (I2NSF) [15]. Other technologies could be used with our approach to block amplification traffic as well. For instance, we can make use of software or hardware firewalls given an interface to interact with this devices. Similarly, it is also possible to use Border Gateway Protocol (BGP) Flow Specification Rules [16] to drop unwanted traffic. Although in the current work, we concentrate on attack traffic dropping, we can apply other traffic shaping actions, e.g., rate limit or redirect it for additional examination.

It should be noted that not all providers are created equal [17]. *Tier-1* ISP has access to the global Internet and does not buy traffic from other ISPs, operating only through peering agreements [17]. *Tier-2* providers connect Tier-1 with Tier-3 ISPs. They usually buy the traffic from Tier-1 providers, but could also have direct peering agreements. *Tier-3* are the last-mile ISPs providing connection to end users. In this work, we target Tier-2 and Tier-3 ISPs, because they are usually entitled to pay for the traffic. Moreover, these are the providers hosting services vulnerable for amplification.

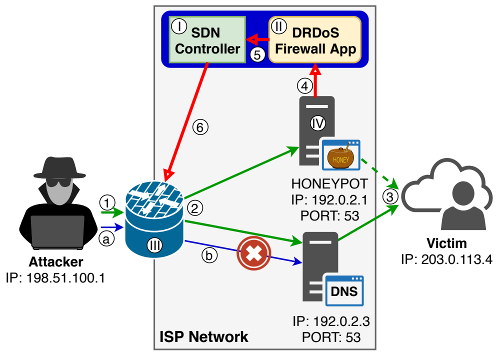


Fig. 1. System Overview

II. System Overview

Figure 1 gives a high-level overview of our system. There are four main components (marked with Roman numerals): (I) SDN Controller; (II) DRDoS Firewall Application; (III) SDN Forwarding Device; (IV) Amplification Honeypot. All incoming traffic to the ISP network passes through *SDN Forwarding Device*. This edge device plays the role of a firewall filtering out the traffic that matches defined flow rules. These rules are generated by *DRDoS Firewall Application* based on the data provided by *Amplification Honeypot*. DRDoS Firewall Application uses the functionality provided by *SDN Controller* to deploy the rules on SDN Forwarding Device. While currently we employ an ad-hoc interface between Amplification Honeypot and DRDoS Firewall Application, generally it is possible to rely on Distributed-Denial-of-Service Open Threat Signaling (DOTS) architecture [18].

In order to launch an attack, at first an adversary searches hosts vulnerable for amplification. Let's assume that during this scan s/he has discovered two hosts in ISP's network vulnerable for DNS amplification: an open resolver (IP `192.0.2.3`) and our honeypot (IP `192.0.2.1`). During the attack, the adversary sends requests with a spoofed victim IP address to these two hosts (Steps 1 and 2, green arrows) targeting a vulnerable protocol on a predefined UDP port (e.g., port 53 for DNS). At Step 3, the vulnerable server and the honeypot generate amplified replies in response to the received requests. However, the latter also starts monitoring the attack, and if its volume exceeds a predefined threshold (the amount of the requests received in a period of time for a combination of source IP address and destination UDP port) it sends an alert to DRDoS Firewall Application (Step 4). This application through the SDN Controller (Step 5) issues an OpenFlow firewall rule to the edge SDN Forwarding Device (Step 6) that blocks all incoming packets with the source IP address and the destination port matching to the victim IP address and DNS port correspondingly. Hence, all consecutive requests from the attacker (Step a, blue arrow) will be blocked by the edge device and will not reach the vulnerable servers (Step b).

At the beginning of the operation, a rule on SDN Forwarding Device is set that allows the traffic coming to Amplification Honeypot to pass (this rule has higher priority than the ones

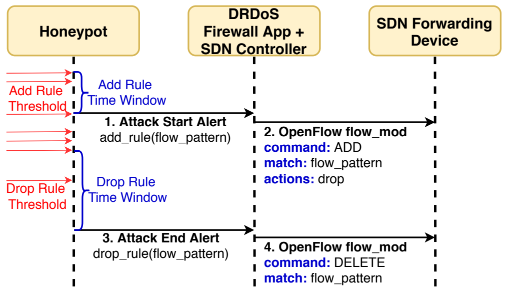


Fig. 2. System Workflow

issued by DRDoS Firewall Application). Such whitelisting enables the honeypot to continue monitoring the attack. Once the attack is over (no amplification requests are received within specified period of time), the honeypot notifies DRDoS Firewall Application to drop the corresponding firewall rule.

A. Work Flow

Figure 2 explains the workflow of our system. The honeypot monitors all incoming packets sent by attackers (represented with red arrows), be them scans or attack requests. This gives the honeypot real-time visibility on ongoing amplification attacks. Packets are split into *flows* according to a *key*, e.g., by *source IP address-destination port*. We use these fields because in order to attack a specific victim using a particular vulnerable service, it is required to provide real victim's source address and destination port of the vulnerable service.

In our case, every flow corresponds either to an attack or scan set of packets. Each unique flow is assigned with a counter that counts the amount of packets arrived within a predefined time interval (*Add Rule Time Window*). If the number of packets exceeds a threshold (*Add Rule Threshold*), meaning that incoming packets belong to an attack rather than to a scan, the honeypot generates an *Attack Start Alert*. It is sent to DRDoS Firewall Application together with the metainformation about the flow `flow_pattern`. DRDoS Firewall Application using SDN Controller issues an OpenFlow `flow_mod` instruction to SDN Forwarding Device, requesting to add a new rule (command `ADD`) to drop packets (actions `drop`) fitting the criteria (matching `flow_pattern`, e.g., with specific source IP address and destination port). Thus, once the attack is detected, SDN Forwarding Device will block all incoming packets matching the pattern. Once the rule is added, the honeypot starts to monitor when the attack is over. If the amount of packets monitored on the honeypot drops below *Drop Rule Threshold* during *Drop Rule Time Window*, the honeypot instructs DRDoS Firewall Application to remove the corresponding rule from SDN Forwarding Device.

Note, it is impossible to block all the traffic coming to amplifiers (or even only to their vulnerable services). If we do this, customers will not have access to these services. Despite being abused by attackers, they are still legitimate services and should be accessible by clients from other networks.

B. Implementation Details

a) Amplification Honeypot: In this work, as Amplification Honeypot we adapted AmpPot [14]. AmpPot is able to monitor more than 10 different UDP services vulnerable to amplification and is widely adopted by the research community for the analysis of DRDoS attacks [19]–[22]. We modified AmpPot in several aspects. First, we removed all rate limiting mechanisms for incoming traffic. Hence, we are able to receive all packets hitting the honeypot. Still, the honeypot mildly participates in the attack (and only in the beginning) due to its rate limiting for outgoing traffic that was not lifted. Second, we enabled the honeypot to monitor when an attack to a particular victim is started and when it is over. Third, we connected AmpPot to DRDoS Firewall Application.

b) SDN Components: In this work, we use POX [23] as an SDN Controller. POX is widely adopted by researchers in the SDN community [24]–[26]. It implements OpenFlow 1.0 specification [27] making our solution forward compatible with newer standards. We developed DRDoS Firewall Application that runs over POX SDN Controller and uses its API. Basically, DRDoS Firewall App is an application that uses the functionality provided by POX SDN Controller, which is another program running on the same host (blue rectangle in Figure 1). While a number of related DDoS SDN-based solutions (see Section V) are built using *reactive mode*, our solution is developed over *proactive mode*. Despite this, it is not completely proactive per se. Rules are derived at runtime using the honeypot data, and every new flow does not necessary result in adding a new rule. Hence, our system is rather *hybrid* than completely *proactive*. Even though such decision makes our solution dependent on the external system, it provides more flexibility for its extension. Moreover, such design is more controller friendly and less prone to DoS attacks [28] because controller and network apps are not required to analyze all previously unseen flows. It also provides low network latency by not spending time on forwarding new packets to the controller for analysis. The latter is a crucial property for ISP edge devices dealing with gigabits-per-second traffic, where even small delays are undesirable.

Upon receiving an attack start alert from Amplification Honeypot, DRDoS Firewall Application using POX OpenFlow-compatible Southbound API adds a rule to OpenFlow's Open vSwitch, which we use as SDN Forwarding Device. To add a rule, DRDoS Firewall Application creates a `ofp_flow_mod` message to execute an `OFPFC_ADD` command. The application fills message's `match` structure with the values obtained from the honeypot data: `nw_src` – source IP address; `tp_dst` – destination port.

All packets that match this criteria are assigned with a `drop` action. One particular case is the `nw_src` field. It should be mentioned that adversaries often attack a whole subnetwork not only on a single IP address. In order to cover this case, we added a feature to our DRDoS Firewall Application to block an entire subnetwork. This functionality relies on a partial match of IP addresses implemented in POX.

c) System Simulation: We validated our approach by simulating a simplified case using GNS3 network emulator [29]. GNS3 supports multiple emulators, which can be used in GNS3 projects, including Dynamips; Docker containers; Qemu, Virtualbox and VMWare virtual machines (VMs). In this work, we extensively rely on Docker. Docker containers run on the same host kernel, thus consuming considerably less system resources than traditional VMs, that allows researchers to increase considerably the number of emulated devices. GNS3 supports multiple containers that can be run as parts of a topology. In this work, we build separate Docker images to run SDN Controller, Amplification Honeypot, a vulnerable amplifier host, attacker and victim machines, and virtual Open vSwitch [30] appliance for SDN Forwarding Device.

III. Evaluation

A. Dataset

The main information supplier for our system is Amplification Honeypot. Therefore, to make close-to-reality judgements we have to use real data collected from a honeypot. To achieve this goal, we modified AmpPot [14] so it is able to record details about all incoming requests. As a dataset we use one month data collected in October 2017. At the time, the honeypot was already operational for several months. Although for the evaluation of our system we employ the information only about the arrival time, source IP address and destination UDP port, we share the whole dataset with the community[1]. It occupies more than 52Gb of disk space in a highly compressed columnar storage format called Parquet [31] widely used in Big Data systems for efficient data storage.

Figure 3 shows the amount of requests our honeypot received each day. We can see that on average it receives around 204 million packets a day (maximum number is around 335 million packets recorded on Oct. 31, while minimum value is about 95 million recorded on Oct. 1). The data on the figure are grouped by the destination port that identifies the vulnerable protocol used to attack victims. As it can be seen, in terms of the number of requests, the most heavily used protocols to launch attacks are: NTP (123) – 192.3 million requests per day on average, DNS (53) – 7.5 million, CharGen (19) – 2.3 million, and RIPv1 (520) – 2.2 million. Not surprisingly, due to its high amplification factor (around 550 [20]) NTP remains one of the most attractive protocol for attackers.

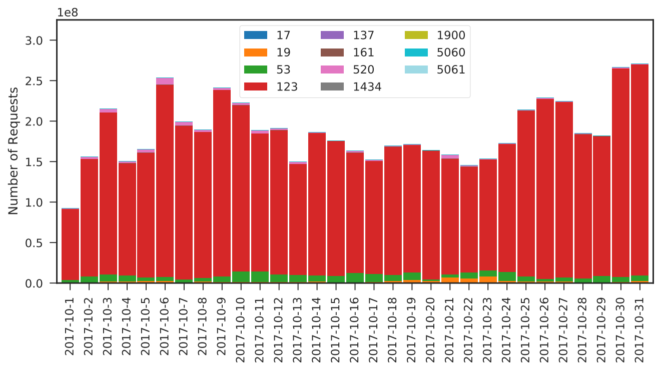


Fig. 3. Amount of Requests by Destination Port

B. System Parameter Evaluation

1) Honeypot Parameters: To identify the start and the end of attacks, we rely on four parameters introduced in Section II-A: *Add Rule Time Window* (`ar_tw`), *Add Rule Threshold* (`ar_th`); and *Drop Rule Time Window* (`dr_tw`), *Drop Rule Threshold* (`dr_th`). We mark a particular flow as an attack if we have received at least `ar_th` packets matching flow pattern in less than `ar_tw` seconds. Correspondingly, an attack is finished if we have received less than `dr_th` packets matching flow pattern during `dr_tw` seconds. In order to detect attack start, we maintain a hashmap that for each flow (identified by its flow pattern, e.g., `src_ip`: 203.0.113.4 - `dst_port`: 53) creates a queue, size of which is equal to `ar_th`. When the honeypot receives a packet matching this flow pattern, it appends its timestamp in the beginning of the queue. After adding it, we compute the difference between the first timestamp in the queue and the last. If the difference is less than `ar_tw` and the queue is full, we issue *Attack Start Alert*. Once the attack is started, we create a similar structure for drop candidates.

Our approach allows us to reduce memory consumption and improve performance. Indeed, the amount of the consumed memory depends only on the amount of flows. In our dataset, the amount of flows does not exceed 11,000 entities per day (see Figure 4). Therefore, total memory overhead for *one day data* is maximum 9MB (11,000 flows, 100 timestamps in each queue, 8 bytes to store one timestamp, 6 bytes to identify a queue). Moreover, the computational overheads are also negligible in this case. The computational complexity is *O(1)*, that corresponds to adding an entry into a queue.

2) SDN Components Parameters: Flow rules are stored in the network device memory in flow tables. Obviously, these network devices have a finite memory capacity limiting the amount of rules that can be stored. Typically, modern equipment can store roughly 8,000 entries; more advanced hardware samples are able to increase this number up to 500,000 [32]. Another important constraint is the amount of table modification calls a controller can handle per second. Currently, the throughput of these devices varies from 38 to 1000 `flow_mod` operations per second [32]. Thus, the practicality of our approach depends if we fit these constraints. In order to evaluate this, we developed a simulator of our system. It is fed with the data from our dataset, however, instead of issuing alerts to the controller it records the timestamp of every `flow_mod` operation and the current amount of rules. We add a filtering rule if we have received at least 100 packets (`ar_th`) with the same key in the last 1 hour (`ar_tw`). Such conservative values allow us to whitelist safely the scanners[2]

[1]https://quiltdata.com/package/zyrikby/amppot_full_data

[2]Please refer to the original paper [14] for the methodology and the results of the scanning activity analysis.

while still blocking the amplification attacks. A rule is dropped if we have received less than 2 packets (`dr_th`) with the same key in the last 1 hour (`dr_tw`). Figure 4 shows the amount of flows related to scanning and attacking activities.

Figure 5a shows maximum number of rules simultaneously active on the switch in each day. On average, this value is equal to 422. However, on Oct. 11 it reaches 2,239. In general, this value is at least three times lower than the capacity of the modern network devices [32]. Still, we can use whole /24 subnetwork as a part of the key instead of particular IP address in order to reduce this number. It is quite common that attackers launch DDoS attacks at a whole /24 subnetwork rather than a single IP address. There are two rationales for such behavior. First, often the target of attackers is not a single IP but rather the whole organization, which usually has several public IP addresses from the same subnetwork. Second, such approach allows adversaries to bypass protection mechanisms detecting an attack if the amount of packets to a particular IP exceeds a predefined threshold. Figure 5b shows the amount of rules if we use whole /24 subnetwork instead of single IP address. In this case, the maximum amount of simultaneously active rules is equal to 395 and the average is 288.

In Figure 7, blue (key: `src_ip-dst_port`) and orange (key: `src_subnet-dst_port`) lines show how the amount of rules changes during Oct. 11. As it can be seen, two spikes of the blue line at 7:00-9:00 and 13:00-15:00 are smoothed if subnetwork is used. It is clear, in these periods of time adversaries targeted whole subnetworks rather than a single IP. At the same time, on Oct. 21 we do not observe such behavior: the green and red lines are almost fused into one.

In Figure 6, we visualize the maximum frequency of `flow_mod` calls (in Figure 6a key is `src_ip-dst_port`, while in Figure 6b it is `src_subnet-dst_port`). The maximum observed frequency across all the month is 78 `flow_mod` calls per second (Hz) for `src_ip-dst_port`. This number is twice higher than the lower bound of 38 operations per second [32]. Unfortunately, the usage of /24 subnetwork instead of specific IP address does not help as before. Indeed, even with the `src_subnet-dst_port`) key we still reach the maximum frequency of 76 `flow_mod` calls per second, which can be clearly seen in Figure 8.

However, in Figure 8 we can see that this frequency is not constant during the whole day. There is just one big spike at 11:25:57. In the rest of the day, the frequency barely exceeds 20 operations per second during several very short periods. Therefore, the more sophisticated solution to the problem is to add to our system the awareness about SDN network constraints and smooth the spikes according to the given limitations. Despite the fact our system will allow to pass slightly more attack packets in this case, this amount will be still negligible because 90% of the days we observe maximum frequency of 27 Hz or less.

One could expect the maximum frequency of the `flow_mod` calls is achieved in the same day when the maximum number of rules is observed. However, as we see from Figures 5 and 6 this is not the case. Maximum number of simultaneously active rules is observed on Oct. 11, while maximum frequency is spotted on Oct. 21. That is why we selected these two days for closer inspection.

In general, the experiments show the practicality of our approach. With the current rapid development of both hardware and software, it will consume minor resources bringing considerable benefits.

3) Delays: Figure 9 shows the attack phases. On this figure, the dots represent packets belonging to one flow. The period between the add rule (*ar*) and drop rule events (*dr*) is the *protection time*, because during this interval our system blocks amplification requests. Note, if an attacker starts and stops to abuse amplifiers instantly, we might **not** need to wait additional time and could drop the rule when the last packet (*dr_lp*) arrives. We call this interval *effective protection time*. We call the time between the *0*'th (the *ar_fp* event; *fp* means first packet) and *ar_th*'th (the *ar* event) packets which is less or equal to *ar_tw*, as the *attack detection* phase. The time before the *ar_fp* event we call as *boot* phase.

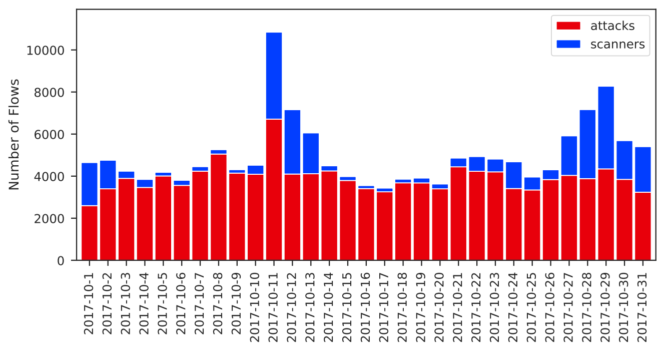
Figure 10 shows experimental cumulative distribution functions (ECDFs) for boot, attack detection and effective protection times for the `src_ip-dst_port` key limited to two minutes interval[3]. In order to plot this graph, we selected the corresponding durations for all attack-related flows in the month. In general, we registered 121,648 (79%) attack and 33,255 (21%) scanner related flows for the `src_ip-dst_port` key; 113,232 (94%) and 6,706 (6%) correspondingly for the `src_subnet-dst_port` key. First, Figure 10 shows that more than 99% of all flows have boot time equal to 0 meaning that an attack usually starts almost instantly, our selected values of *ar_th* and *ar_tw* allow us effectively capture attack related behavior. Second, for more than 90% of attack flows, attack detection time is less than 1 minute. This means that the attacks are usually quite intensive and once started they lead to the add rule generation event quite fast. Third, Figure 10 confirms that for more than 95% of attacks the effective protection time is more than 15 seconds. This gives enough time to deploy a rule to block an attack and get benefits from our solution.

In order to assess the amount of attack packets our system could miss, we have plotted attack packet speed ECDF (Figure 11). This graph shows 90% of all attacks generate less than 250 packets per second. If a cumulative delay for rule deployment is less than 1 second (a very conservative assumption [33]) the volume of such attacks will be less than 250 packets for 90% of the flows.

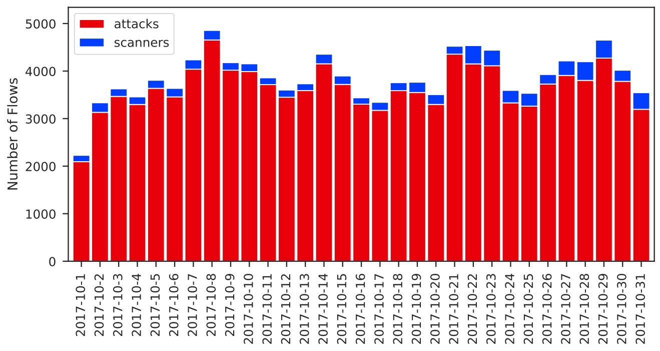
C. Advantages Evaluation

Based on the parameters described in the previous sections, we evaluate the benefits in terms of the saved bandwidth, which our system can bring to ISPs. In order to achieve this goal, we make the following assumption. We evaluate gains caused by the honeypot, i.e., we assume that our AmpPot is the only vulnerable service in an ISP network. However, a

[3]The ECDF for `src_subnet-dst_port` is similar.

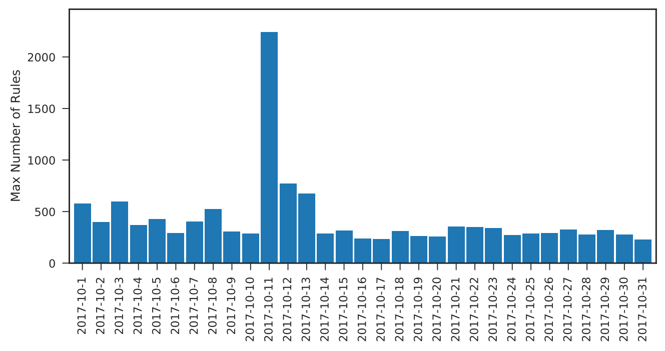


(a) Key: `src_ip-dst_port`

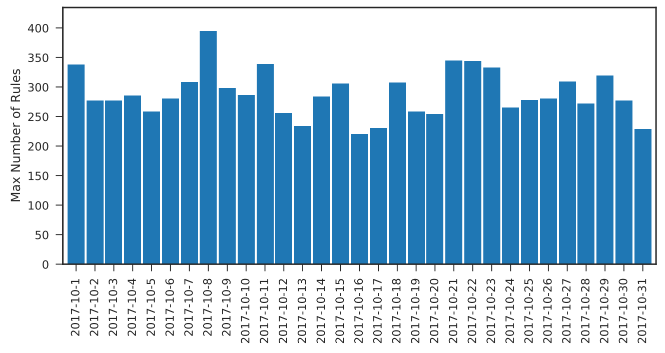


(b) Key: `src_subnet-dst_port`

Fig. 4. Number of Flows

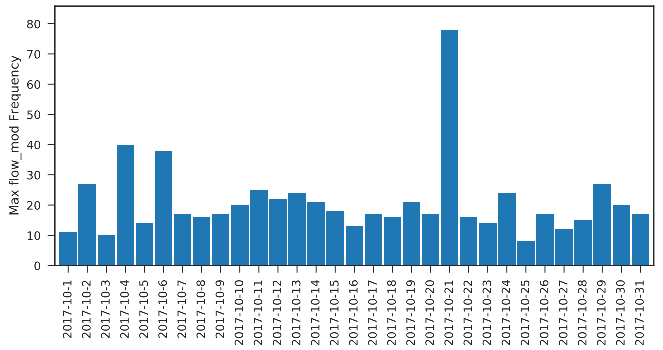


(a) Key: `src_ip-dst_port`

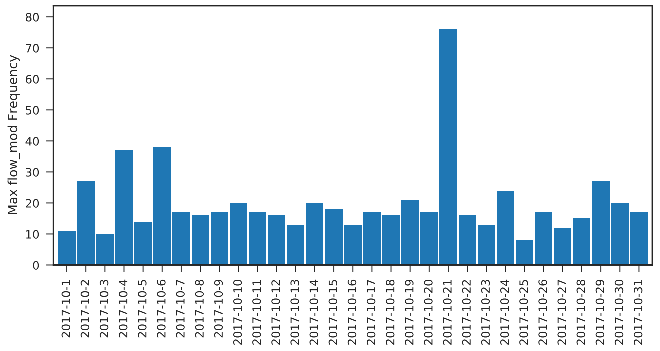


(b) Key: `src_subnet-dst_port`

Fig. 5. Maximum Number of Rules



(a) Key: `src_ip-dst_port`



(b) Key: `src_subnet-dst_port`

Fig. 6. Maximum Frequency of `flow_mod` Calls

real ISP network may contain hundreds or thousands of such hosts [7]. The gains can be calculated based on the formula:

$$G = \sum_{\forall p \in P} (T_p - M_p) * (UPS_p * BAF_p + 52) \qquad (1)$$

where P is a set of all vulnerable protocols, T_p – total number of packets received by our honeypot for protocol p, M_p – number of packets missed before a rule is deployed, UPS_p – average UDP payload size for protocol p, BAF_p is an average amplification factor, and 52 is the minimum size of a UDP packet (24 bytes for Ethernet + 20 for IP + 8 for UDP headers). For UPS_p and BAF_p, we used the values reported in [20].

Figure 12 shows the gains (the red part of each bar) obtained for every day using Formula 1 if the filtering is done by source IP and destination port. The graph shows similar behavior and numbers when filtering is done by /24 subnetwork and

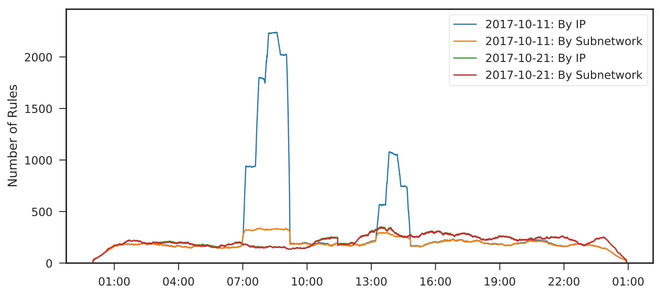


Fig. 7. Amount of Rules

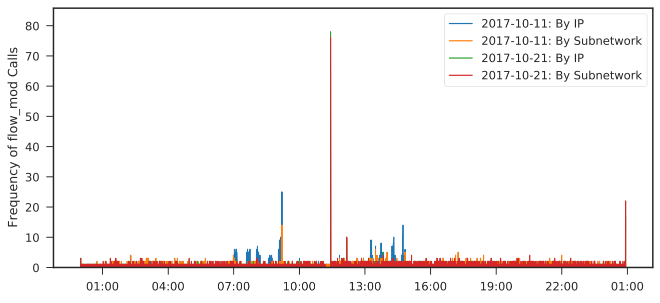


Fig. 8. Frequency of `flow_mod` Calls

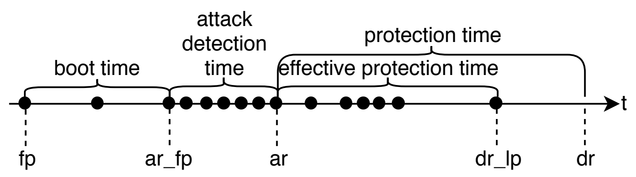


Fig. 9. Attack Phases and Delays

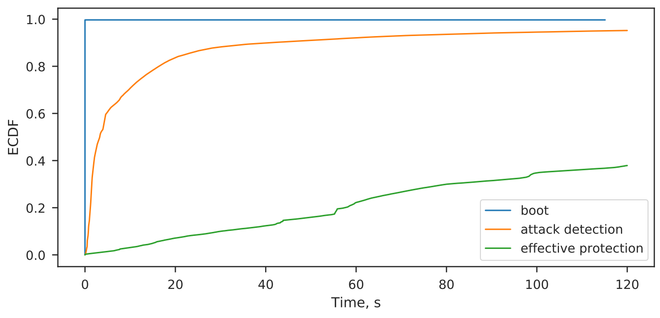


Fig. 10. Phases Duration ECDF (key: `src_ip-dst_port`)

destination port. According to the evaluation on our dataset, our system allows an ISP to filter out approximately 1,505 GB of garbage traffic a day, with minimum value of 750 GB and maximum reaching whopping 2,214 GB. Here we underline that these estimations are done for the case when there is only one amplifier per each vulnerable protocol. The tiny blue parts of the bars in Figure 12 represent the approximate amount of amplified data leaving an ISP network due to the missed packets. Our calculations show that with the current settings

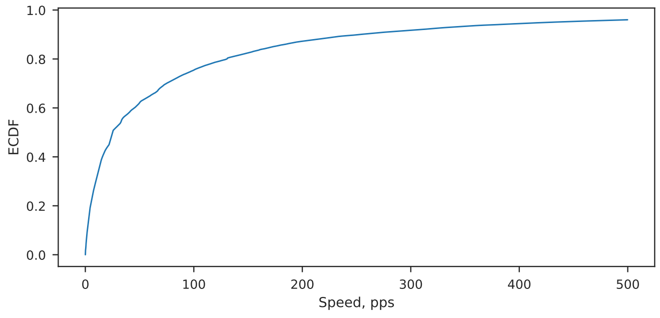


Fig. 11. Attack Packet Speed ECDF (key: `src_ip-dst_port`)

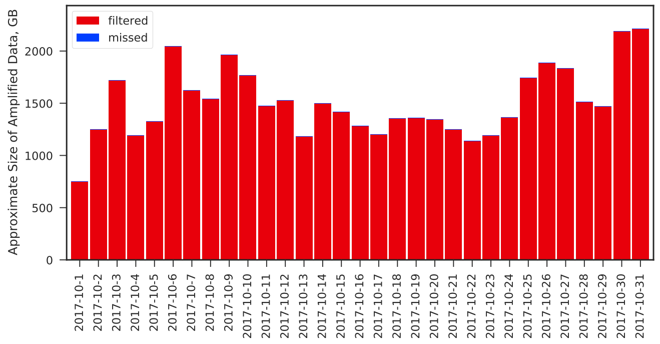


Fig. 12. Approximate Gains

we allow less than 0.2% of the amplified traffic to pass.

IV. Discussion

The usage of our system brings considerable advantages to ISPs. Still, there are also potential drawbacks. In this section, we discuss the limitations and future extensions of our approach.

a) Honeypot Discoverability: The ability to filter out garbage traffic in our system relies on the detection of attacks by our honeypot. If an attacker is aware about this fact, s/he may try to discover the IP address of the honeypot.As a result, the adversary may avoid to use the honeypot as an amplifier making our system unaware about the ongoing attacks. One obvious solution to this is to improve the mimicking of the vulnerable protocols, ideally by deploying real vulnerable services on different IP addresses and proxing the traffic through AmpPot [14].

b) Victim Network Blocking: If the honeypot is discovered, the adversary may use it to launch DoS attacks on legitimate clients. Indeed, the attacker may send several attack packets with spoofed client's IP address to the honeypot. This will result in legitimate requests from the client not reaching the corresponding service within the ISP network. A countermeasure is to filter packets by pair of IP address and destination port (as we have been doing in this work). Hence, for the client only a subset of services vulnerable for amplification will be unavailable. Another potential solution is to perform throttling rather than complete blocking. It is also possible to analyze additional properties of the packets e.g., TTL values [34], to support the decision.

c) Collateral Damage: Obviously, during an attack a legitimate victim would not be able to use exploited services of the ISP implementing our approach. Indeed, our system will block all the packets coming from victim's IP address to the service under attack. This may cause service unavailability for legitimate customers. The situation can be even more severe if a subnetwork is blocked. Note, such circumstances are unlikely so as (1) no legitimate clients should use honeypot; (2) it is difficult for an attacker to pinpoint a honeypot (as we discussed before) and abuse it. However, one potential remediation is to use deep packet inspection and block only specific protocol commands that can be abused, e.g., `monlist` for NTP. Additionally, in order to reduce collateral damage it is also possible to employ ISP network topology. Current ISPs usually have several entry points. If amplification requests come through one of these points, it is possible to deploy filtering rules on the corresponding edge device still allowing the traffic through other routers to pass.

d) Overloading of Switch Rule Table: If the attacker knows where the honeypot is, s/he may flood it with lots of different flows. Thus, the honeypot may create huge amount of rules that may overwhelm a switch. However, this type of attack is not unique to our system, rather it is typical for SDN enabled networks [35]. There are several solutions proposed that can also be applied to our case [28]. A more specific approach for our system is to increase *Add Rule Threshold* value. In this case, it will require an attacker to spend more resources to launch this attack.

e) Moving Target: Sometimes an attacker exploits amplifiers sequentially, i.e., at one point of time one subset of amplifiers is used and after a while the adversary switches to another set. In this case, the attack may be discovered late if the honeypot is abused in the end. One potential solution is to deploy several honeypots across the network and build the intelligence on top of the cumulative data. In a more extreme case, honeypots can be deployed world-wide providing necessary information to all interested ISPs. In a simple case, the information about the attacks from multiple honeypots is just combined together. That is to say, a source address - destination port tuple is blocked if it has been spotted in the attack to at least one honeypot. The work done by Krämer et al. [14] confirms that the amount of newly observed flows decreases as the number of honeypots grows: the tenth honeypot observes less than 10% of new attacks with respect to the information from nine others. Therefore, having decent number of honeypots distributed world-wide would allow to observe almost all attacks. In a more advanced case, to reduce the amount of false positives an ISP may require an attack to be spotted by several honeypots before installing a rule. Krupp et al. [19] showed that in their setup out of 48 deployed honeypots over 95% of all attacks used at least 4 of them.

f) Attacker Resides in the Same ISP Network: In order to block incoming amplification requests, an attacker should reside outside the protected ISP network. If the adversary is located within the network then the attack requests are not passing through the edge device and are not blocked. In this case, the requests will reach vulnerable services, will be amplified and generate garbage traffic. In order to make the nettwork proof to this kind of scenario, the ISP should also implement the BCP 38 recommendations for ingress traffic filtering [6] on each routing device. Then, amplification requests will not be able to reach amplifiers in other network segments. Hence, the deployment of our system may indirectly influence on the adoption of this best practice.

V. Related Work

There have been several DDoS attack detection and mitigation solutions proposed in the past for ISPs using honeypots as one of the components. For instance, Sardana et al. [11] proposed a system that was designed to detect DDoS attack traffic using packet entropy variations calculated in small time windows. Suspicious flows are tagged and redirected to a honeypot server which monitors and responds to the attack protecting victim's host. Even though the system shares similar components with our approach, the goals are very different. Sardana's system aims at protecting end hosts using a honeypot which substitutes the victim, while our method uses the data from honeypot to block unwanted traffic to come in.

Li et al. [12] propose an architecture based on SDNs called DrawBridge to help ISPs to filter unwanted DDoS/DRDoS traffic. The system tries to bridge the communication gap between ISPs and victims of DDoS attacks. Currently, ISPs independently engineer the traffic not taking into account the fact if the data is welcomed by end users. DrawBridge allows ISP users (victims) to communicate filtering rules to the controllers that deploy them at ISP switches blocking the unwanted traffic. Despite sharing a similar goal as ours, i.e, filtering garbage traffic at ISPs, their approach heavily relies on end user's capability on detecting unwanted traffic that is a hard task in case if end user hosts are used as DRDoS amplifiers. The victim side traffic detection, rule formation criteria, and details of overall system implementation are not discussed. Another SDN based "DDoS Mitigation as a Service" framework is proposed by Sahay et al. [13]. The framework uses ISP customer side intelligence to mitigate DDoS attacks. Upon detection of an attack an ISP tags unwanted traffic and redirects it to security middleboxes. This SDN based framework is limited to flooding based attacks (such as TCP SYN, UDP, and ICMP flooding) and still aims at the protection of end clients from DDoS attack.

Few DRDoS detection and mitigation solutions using SDNs have emerged lately. Yet, the primary focus of these approaches lies in identifying amplification traffic targeting the victim side contrary to our focus, i.e., blocking amplification requests coming into an ISP network. Aizuddin et al. [36] propose a near-real time flow based DNS amplification detection and mitigation at the target victim network. DNS replies are filtered if no matching DNS requests are found. Another recent SDN based solution addressing DNS amplification problem was proposed by Xing et al. [37]. Their system aims at detecting a DNS amplification attack on a victim through the analysis of packet entropy variation. Once the attack is detected the system is able to pinpoint amplifiers within the same network and isolate them. More recently, Chen et al. [38] offered a machine learning based solution to detect DRDoS attacks. A system categorizes DNS/NTP packets using an SVM classifier and blocks the flows belonging to an amplification attack using an SDN controller. The system is among a few able to detect amplification request packets and thus, to prevent silent amplification by vulnerable hosts in the network. However, the solution is limited to DNS and NTP protocols. Moreover, it requires quite substantial time to make a decision (average response time for DNS is 11.8s), while our system could react almost immediately.

VI. Conclusion

In this paper, we propose a novel approach to filter out amplification traffic from an ISP network. It relies on data collected from an amplification honeypot to derive filtering rules. We implemented the prototype of our method employing SDN, and the results of our evaluation based on the real honeypot data show the practicality of our approach. Indeed, in terms of SDN component constraints the system can be deployed even on low-end SDN hardware. Our calculations confirm that the method helps in filtering out substantial amounts of garbage traffic saving ISP's money and improving QoS for its customers. Moreover, by restricting vulnerable hosts in an attack participation, it helps victims making the ongoing "storm" lighter. Additionally, the deployment of our system encourages ISPs to implement ingress filtering best practices. Being implemented worldwide, these recommendations will make spoofing attacks impossible.

References

[1] K. Team. How to Generate 2TB/s Reflection DDoS Data Flow via a Family Network. [Online]. Available: http://powerofcommunity.net/poc2017/shengbao.pdf

[2] S. Kottler. (2018, March) February 28th DDoS Incident Report. [Online]. Available: https://githubengineering.com/ddos-incident-report/

[3] (2014) NTP Amplification Attacks Using CVE-2013-5211. [Online]. Available: https://www.us-cert.gov/ncas/alerts/TA14-013A

[4] Open Resolver Project. [Online]. Available: http://openresolverproject.org/

[5] https://twitter.com/037. MEMFIXED DDoS Mitigation Tool. [Online]. Available: https://github.com/649/Memfixed-Mitigation-Tool

[6] D. S. P. Ferguson, "Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing," RFC, Tech. Rep., May 2000. [Online]. Available: https://tools.ietf.org/html/rfc2827

[7] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," in *Proc. of NDSS*, February 2014.

[8] M. Jackson. Ofcom Enhances UK Code of Practice for Broadband ISP Speeds. [Online]. Available: https://goo.gl/q2HHks

[9] How uk isps are charged for broadband - the cost of IPStream. [Online]. Available: https://goo.gl/ofEh1X

[10] O. Kharif. YouTube, Netflix Videos Found to Be Slowed by Wireless Carriers. [Online]. Available: https://goo.gl/QTGA9G

[11] A. Sardana, K. Kumar, and R. C. Joshi, "Detection and honeypot based redirection to counter ddos attacks in isp domain," in *Proc. of IAS*, 2007, pp. 191–196.

[12] J. Li, S. Berg, M. Zhang, P. Reiher, and T. Wei, "Drawbridge: Software-defined ddos-resistant traffic engineering," in *Proc. of SIGCOMM*, 2014, pp. 591–592.

[13] R. Sahay, G. Blanc, Z. Zhang, and H. Debar, "Towards autonomic ddos mitigation using software defined networking," in *Proc. of Security of Emerging Networking Technologies Workshop*, 2015.

[14] L. Krämer, J. Krupp, D. Makita, T. Nishizoe, T. Koide, k. Yoshioka, and C. Rossow, "AmpPot: Monitoring and Defending Against Amplification DDoS Attacks," in *Proc. of RAID*, 2015, pp. 615–636.

[15] S. Hares, D. Lopez, M. Zarny, C. Jacquenet, R. Kumar, and J. Jeong, "Interface to network security functions (I2NSF): Problem statement and use cases," RFC, Tech. Rep. 8192, July 2017. [Online]. Available: http://www.rfc-editor.org/rfc/rfc8192.txt

[16] P. Marques, N. Sheth, R. Raszuk, B. Greene, J. Mauch, and D. McPherson, "Dissemination of Flow Specification Rules," RFC, Tech. Rep. 5575, August 2009. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5575.txt

[17] L. Gao and F. Wang, "The extent of AS path inflation by routing policies," in *Proc. of GLOBECOM*, Nov 2002, pp. 2180–2184.

[18] A. Mortensen, F. Andreasen, T. Reddy, N. Teague, R. Compton, and C. Gray, "Distributed-denial-of-service open threat signaling (DOTS) architecture," WG Draft, Tech. Rep., 2019. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-dots-architecture/

[19] J. Krupp, M. Backes, and C. Rossow, "Identifying the Scan and Attack Infrastructures Behind Amplification DDoS Attacks," in *Proc. of CCS*, 2016, pp. 1426–1437.

[20] M. Aupetit, Y. Zhauniarovich, G. Vasiliadis, M. Dacier, and Y. Boshmaf, "Visualization of Actionable Knowledge to Mitigate DRDoS Attacks," in *Proc. of VizSec*, 2016, pp. 1–8.

[21] J. Krupp, M. Karami, C. Rossow, D. McCoy, and M. Backes, "Linking Amplification DDoS Attacks to Booter Services," in *Proc. of RAID*, 2017, pp. 427–449.

[22] L. Berti-Equille and Y. Zhauniarovich, "Profiling DRDoS Attacks with Data Analytics Pipeline," in *Proc. of CIKM*, 2017, pp. 1983–1986.

[23] The POX Network Software Platform. [Online]. Available: https://github.com/noxrepo/pox

[24] M. Gupta, J. Sommers, and P. Barford, "Fast, Accurate Simulation for SDN Prototyping," in *Proc. of HotSDN*, 2013, pp. 31–36.

[25] S. M. Mousavi and M. St-Hilaire, "Early Eetection of DDoS Attacks against SDN Controllers," in *Proc. of ICNC*, 2015, pp. 77–81.

[26] Q. Niyaz, W. Sun, and A. Y. Javaid, "A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN)," *EAI Endorsed Transactions on Security and Safety*, vol. 17, no. 12, 12 2017.

[27] N. McKeown, H. B. T. Anderson, L. P. G. Parulkar, S. S. J. Rexford, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[28] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A Survey of Security in Software Defined Networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 623–654, Firstquarter 2016.

[29] GNS3 — The Software that Empowers Network Professionals. [Online]. Available: https://www.gns3.com

[30] Open vSwitch. [Online]. Available: https://www.openvswitch.org

[31] Apache Parquet. [Online]. Available: https://parquet.apache.org/

[32] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proc. of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[33] K. He, J. Khalid, S. Das, A. Gember-Jacobson, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Latency in Software Defined Networks: Measurements and Mitigation Techniques," in *Proc. of SIGMETRICS*, 2015, pp. 435–436.

[34] M. Backes, T. Holz, C. Rossow, T. Rytilahti, M. Simeonovski, and B. Stock, "On the Feasibility of TTL-based Filtering for DRDoS Mitigation," in *Proc. of RAID*, September 2016.

[35] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we Ready for SDN? Implementation Challenges for Software-Defined Networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, July 2013.

[36] A. A. Aizuddin, M. Atan, M. Norulazmi, M. M. Noor, S. Akimi, and Z. Abidin, "Dns amplification attack detection and mitigation via sflow with security-centric sdn," in *Proc. of ICUIMC*, 2017, pp. 3:1–3:7.

[37] X. Xing, T. Luo, J. Li, and Y. Hu, "A defense mechanism against the dns amplification attack in sdn," in *Proc. of IC-NIDC*, 2016, pp. 28–33.

[38] C. C. Chen, Y. R. Chen, W. C. Lu, S. C. Tsai, and M. C. Yang, "Detecting amplification attacks with software defined networking," in *Proc. of DSC*, Aug 2017, pp. 195–201.