# Flooding DDoS Mitigation and Traffic Management with Software Defined Networking

Aapo Kalliola*, Kiryong Lee†, Heejo Lee†, Tuomas Aura*
*Aalto University, Espoo, Finland
aapo.kalliola@aalto.fi, tuomas.aura@aalto.fi
†Korea University, Seoul, Republic of Korea
krlee@korea.ac.kr, heejo@korea.ac.kr

***Abstract*—Mitigating distributed denial-of-service attacks can be a complex task due to the wide range of attack types, attacker adaptation, and defender constraints. We propose a defense mechanism which is largely automated and can be implemented on current software defined networking (SDN) -enabled networks. Our mechanism combines normal traffic learning, external blacklist information, and elastic capacity invocation in order to provide effective load control, filtering and service elasticity during an attack. We implement the mechanism and analyze its performance on a physical SDN testbed using a comprehensive set of real-life normal traffic traces and synthetic attack traces. The results indicate that the mechanism is effective in maintaining roughly 50% to 80% service levels even when hit by an overwhelming attack.**

## I. Introduction

Distributed denial-of-service (DDoS) attacks are a difficult ongoing problem in the internet. Services ranging from banks to online entertainment to private blogs are frequently affected by attacks that degrade their availability to the point where they become inaccessible to normal users.

Various mechanisms for protecting against DDoS attacks have been proposed, and there are companies which provide network, hardware and software solutions for mitigating DDoS attacks. Unfortunately, the theoretical proposals often rely on network functionality which is not present in current network hardware, while the commercial solutions are expensive and may require the service to route traffic through the third party mitigation provider. If the targeted service operator is unable to use the commercial services due to financial or traffic sensitivity reasons, he may need to resort to manual attack mitigation, which is heavily dependent on the skills and persistence of the service administrators.

In the scope of this paper we propose a DDoS mitigation and traffic management system which is largely automated and can be deployed on current software defined networking (SDN) technology. Our mechanism combines automated hierarchical-clustering-based normal traffic learning, fixed or dynamic blacklist integration, and service distribution or additional server capacity invocation within the network. The mechanism is designed to be effective against packet and bandwidth flooding attacks, and can be used to defend both end hosts and network links.

In the following sections, we explain our proposed mechanism in detail and evaluate the mechanism on a physical SDN testbed using real web server usage history and synthetic attack traffic traces. We find the defense mechanism is effective against various flooding DDoS attacks, maintaining ≈80% service quality in a typical scenario and ≈50% in an extremely challenging scenario.

## II. Background and related work

The general topic of filtering distributed denial-of-service traffic has been examined by Collins et al. [5]. The concept of using software defined networking (SDN) capabilities for DDoS mitigation has been explored most recently by Sahay et al. [13]. A focused approach into clustering-based traffic prioritization on end hosts was published by Kalliola et al. [7].

The hierarchical heavy hitter algorithm, which is essential to the machine learning phase of our mechanism, has been presented in detail by Cormode et al. [6]. Blacklist aggregation algorithms for network devices have also been analyzed by Soldo et al. [14]. Invocation of additional server capacity in cloud environment based on server load has been proposed by Aliyev et al. [3] and extended by Mubarok et al. [9] with more details about the virtual server invocation and stopping process.

In contrast to previous work our main contribution is the implementation and evaluation of the automated learning and DDoS mitigation mechanism [7] in an SDN network. Additionally, in this work the mechanism is extended to accomodate optional input from external signature-based blacklist sources, and traffic management for elastic server capacity has been incorporated into the core mechanism.

## III. Traffic management

The core idea of our mechanism is to use automatic traffic learning and blacklists to assign probable quality values to packets arriving to a network. Based on the quality values and available server resources and traffic service goals, we are then able to smartly allocate traffic and server resources within the network when hit by a DDoS attack or a flash crowd. Our mechanism minimizes traffic hotspots inside a network, and it provides load control and distribution for the normal server and dynamically invocable replica server instances. In this section we describe the network environment in which the DDoS mitigation and traffic management mechanism can be deployed. We also explain the traffic handling and service distribution algorithms.

### A. Architecture

Our defense mechanism can be implemented on a network that supports software defined networking (SDN). A typical case is an autonomous system (AS) such as an internet service provider or a large company that completely controls its own network. Such a network commonly has at least two edge connection points to the wider internet and some degree of internal redundancy. In addition to the traditional network elements, an SDN-enabled network needs a centralized element with a view of the network traffic and control over the network switches. This centralized element is typically an SDN controller, or alternatively a sampling-based traffic viewer with separately implemented control of the switch forwarding tables.

Optionally, our mechanism also supports a view into the load status of the servers. Analysing the actual load on the server may in some cases result in different load control cases than the network traffic volume view. However, in the scope of this paper we focus on the load information from the network perspective. A minimal example network of this type is shown in Fig. 1.

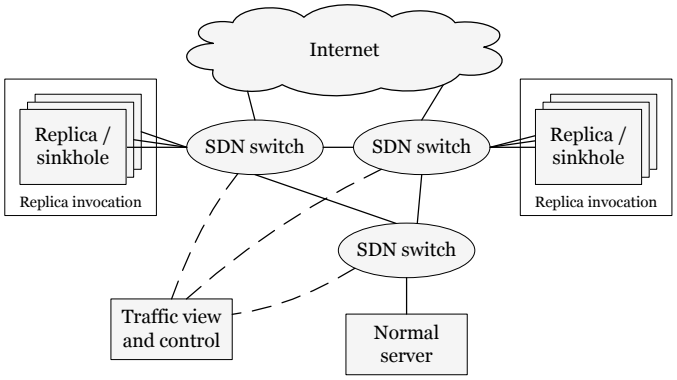



Fig. 1: Network architecture

The SDN switches provide a set of standard capabilities typically defined in the different OpenFlow protocol versions. We require the bare minimum of capabilities: source-IP-address based forwarding.

Replica servers are placed in cloud locations near the network edge. These servers model the capability to flexibly invoke additional server capacity up to some limit in order to serve more incoming traffic than would be possible with only the normal server while also avoiding internal network link congestion. In addition to providing similar service as the normal server, the traffic can be also directed to the replicas for other purposes such as attack-time traffic analysis.

### B. Implementation elements

In order for the DDoS mitigation and traffic management to work efficiently, we identify certain key implementation elements. These are detailed in this section.

*1) Traffic clustering:* At the heart of the traffic management mechanism lies the clustering system that produces network prefixes for traffic forwarding in the SDN switches. When the protected servers are receiving normal amounts of traffic, we build a learning set of the normal traffic features over a time window. Since we want to do prefix-based filtering and routing, it is natural to select the traffic source IP address as the base for building the normal traffic model. While the source IP address can be spoofed, it is the most difficult traffic feature for the attacker to convincingly modify, because the attacker does not have detailed information about the normal users of the service. This leads to there being only partial overlap between the normal user clusters and the attacker clusters.

The algorithm we use for clustering the normal traffic is the hierarchical heavy hitter (HHH) algorithm [6]. This algorithm operates on the prefix tree formed by a set of IP addresses. Using the HHH algorithm and a defined percentage threshold ($TH$) of the total traffic ($TOT$), we divide the normal traffic into non-overlapping clusters. A cluster contains a number of IP addresses, depending on the prefix length and possible excluded child clusters, and a hit count of $TOT * TH \leq HITS < 2 * TOT * TH$. In the case of leaf nodes, the hit count can be arbitrarily large. Thus, while the number of IP addresses within a cluster can vary significantly, the normal traffic volumes of most of the clusters are roughly similar.

Fig. 2 shows an example of the clustering results for a 3-bit prefix tree with a total of 100 units of normal traffic (sum of normal hits in leaf nodes). In real life, the algorithm operates on the 32-bit IPv4 address prefix tree, but the principle is identical. In the figure, traffic has been clustered with a 10% threshold, resulting in clusters that contain ≥10 units of traffic. These clusters are highlighted with thick solid or dashed lines. In real life, the threshold would be 1%...0.01%, resulting in ≤100...10000 clusters. The clusters are non-overlapping, i.e. the traffic values of a cluster are not counted in the traffic values of the parent node. As a modification to the HHH algorithm, we also include the root node so that all traffic sources in the address space match one cluster. For further details of the learning algorithm we refer to a previous publication [7].

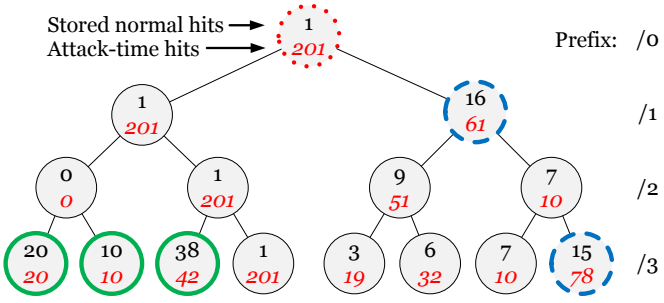



Fig. 2: Traffic clustering and allocation

We periodically run an accurate version of the algorithm on the learning data and store the resulting clusters. In effect the mechanism is incrementally learning about the normal traffic within the learning time window. When the service is attacked, i.e. the defined capacity limit is exceeded, we stop the clustering and thus also the learning. The final set of stored clusters is then used as a model of the normal traffic. The quality of our clustering result cannot be evaluated in a vacuum: in addition to the normal traffic it also depends on the attack traffic. Analysing the results of this interaction is a key goal in section IV-D.

The speed of the clustering algorithm is not performance-critical: the clustering is only ran during normal operation

and thus it is not necessary for reacting to the attack traffic. In our test cases running the clustering algorithm took some seconds with a full learning window, but existing research into HHH algorithms indicates that line-rate clustering would also be possible with limited loss in accuracy [8].

*2) Blacklist integration:* In addition to the automatic clustering, our mechanism supports the integration of fixed or dynamic blacklists consisting of individual IP addresses. This blacklist can, for instance, be provided by a third party or it can be produced inside the network by an intrusion detection system (IDS). The blacklist integration enables support for traditional signature-based traffic filtering, which complements the information provided by the machine learning algorithm. Signature-based detection allows us to filter traffic that is obviously malicious, such as ping flooding attacks, while the clustering system provides better results with attack traffic that mimics the normal traffic types.

Our approach to blacklist integration begins with the aggregation of blacklist entries in similar to what has been outlined by Soldo et al. [14]. This aggregate set of filter rules is prepared but not deployed to switches during normal operation. The reason for not deploying the blacklist filtering to switches in normal operation is to avoid the aggregation-inflicted collateral damage to normal traffic when there is no overload necessitating packet drops. Individual blacklist entry filtering is placed into the processing pipeline before the clustering mechanism, i.e. blacklisted sources are never included in the learning dataset. Since all possible source entries for the whole IPv4 address space can be mapped to a $2^{32}$bit data structure for extremely fast lookup, we do not need to use aggregate blacklist entries where not constrained by the limited flow entry budget of switch hardware.

Using this approach, the traffic that is known to be from malicious sources is removed from the learning dataset before the more complex learning mechanism deals with the traffic in which attack packets cannot be easily distinguished. On an overload event caused by an attack the aggregate blacklist entries are deployed to switches, and the corresponding aggregate discard rules are used in discarding features before they are processed by the traffic allocation and control phase.

*3) Traffic allocation and control:* When an attack starts, we have the clusters from the learning period, their corresponding average normal traffic volumes, and the current attack-time traffic volumes. While we cannot separate the normal traffic from attack traffic, given that they are potentially mixed inside the clusters, we can calculate the ratio between the stored normal traffic average and the attack-time traffic inside each cluster. In Fig. 2 the stored normal traffic averages are shown as the top number inside a node, while the attack-time traffic is the bottom number. We observe that out of the six clusters three have ratios at or close to 1:1 (solid thick line), two have ratios around 1:4-1:5 (dashed line) and one has a ratio of 1:201 (dotted line).

The clusters with the best normal-to-attack-time ratios are the ones most likely to contain normal traffic during an attack, so we allocate the available server capacity with a greedy algorithm starting from the cluster with the best ratio until all server capacity is used. While the greedy algorithm may not be exactly optimal, it is fast, and the high number of clusters in real life limits the impact of possible nonoptimality to well below the random effects caused by traffic volume fluctuations.

In the Fig. 2, assuming we have a normal server of capacity 100 and two replica servers with capacity 80 each, we would allocate the best three clusters to the normal server, the two medium-quality clusters to the replicas, and the worst cluster would be dropped outright. In real implementations, the *serve* or *do not serve* quality threshold related to additional capacity invocation is service-dependent. In some cases, providing the service may be critical even if it means serving increasingly poor-quality traffic, while in other cases, the service might not be valuable enough to warrant the costs of providing any additional capacity.

*4) Server consistency:* Since the traffic allocation mechanism is dependent on the ratio of normal traffic to attack-time traffic within a cluster, we must define how we want the algorithm to behave in the case when the service location of a cluster would change due to attack-time traffic rate fluctuation.

The simplest solution would be to redirect the traffic to original server or replica server according to the clustering result regardless of the previous allocation state. In many cases this would, however, have undesirable side effects. A large part of internet services maintain state between the client and server, at the very least in the form of TCP connections, and therefore changing the server may cause extra overhead in connection reestablishment and it may also cause problems on the application layer.

In order to maintain a good level of server consistency we maintain the original server allocation of a cluster unless the cluster quality decreases to the point where it is dropped altogether. In a sense, this means that beyond the point of initial service allocation we treat the normal and replica servers as one mass of server capacity. While perfect server consistency cannot be guaranteed due to the fundamental nature of elastic server capacity, this appoach minimizes the server re-allocation overhead.

*5) Replica server invocation and stopping:* The essence of replica invocation is that we have the capability to increase the overall server resources in the network up to the point where we deem that serving the increasingly worse traffic clusters is no longer worth it. In practice, server instances are started as needed based on current incoming traffic volumes and stopped as the attack traffic diminishes or changes in such a way that less capacity is needed to serve the clusters meeting the value criteria.

The details of replica server invocation and stopping are not within the scope of this paper, and we refer to previous publications [3], [9] for more information.

## IV. Prototype evaluation

### A. Testbed

The networking core of our testbed consists of three Pica8 1Gbps/10Gbps switches in the Open vSwitch operation mode. Two of the switches are connected to the traffic generator and replica servers while one is connected to the normal server. All the switches are connected to each other and to the sampling traffic and test orchestration network. The traffic network and

the sampling and management network are kept completely separate.

The testbed setup is shown in Fig. 3. The traffic-path switches are interconnected as shown in the network architecture Fig. 1. The out-of-band traffic analysis and filter generation elements are shown separately in this figure for purposes of clarity; in the testbed they reside on the same physical device as the testbed command and control functions.

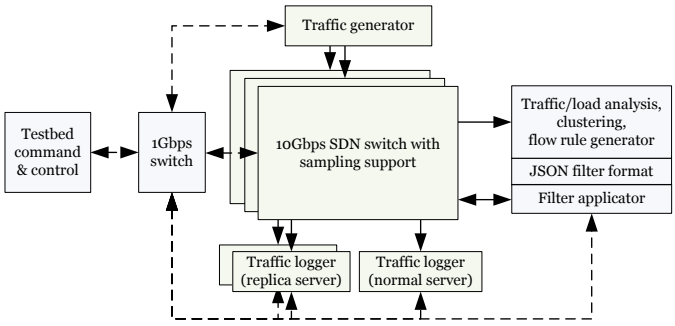



Fig. 3: Testbed setup

Traffic generation is done by using pre-recorded and pre-created network traffic traces and sending them to the switches using *tcpreplay* running on a modern quad-core Linux computer. The traffic generator can saturate the 10Gbps links when using large package sizes, though in practice the tested traffic rates remain below the maximum link capacity.

The server functionality is modeled by logging the incoming traffic on the target server and on the replica servers. Server capacity, and thus also the attack detection threshold, is defined as a packets per second (PPS) capacity limit.

### B. Software implementation

As indicated by the central algorithms, the novel software implementation components are located mainly in the traffic view and control network element. In the testbed this element contains the sampled traffic handling and clustering, blacklist integration, and flow rule generation for traffic control on the switches.

Traffic sample input into the system is handled by a minimal sFlow [11] collector implementation. Each traffic sample is timestamped on reception. These samples are accumulated into one-second time slice counters. These counters are managed in two sliding windows: the short-term window and the long-term learning window. The short-term window is used for the dynamic traffic control adjustments, and therefore its length should be in the range of a few seconds or tens of seconds. The learning window duration should be much longer, preferably at least as long as the expected attack duration. Typically this means that the learning window length should be at least 24 hours.

The clustering module implements the modified accurate HHH algorithm for the traffic features in the learning window and produces a set of address prefix clusters based on the threshold value. When under an attack, the traffic features in the short-term window are placed into the clusters and the normal-to-attack-time traffic ratios are calculated.

Based on the ratios and current traffic values, the clusters are allocated to the servers, and an intermediate JSON format filter file is created. This file is read by the filter deployment module, which pushes the filtering rules to the switches on the network. In the blacklist pipeline case, the blacklisted sources are not included in the traffic windows, and the blacklist entries are applied to the network traffic before the cluster allocation.

It is important to note that while the packet sampling ratio must be decreased during a heavy flooding attack, the resulting accuracy does not suffer as long as the number of samples per second remains the same [12]. Thus, if the attack traffic greatly exceeds the normal traffic we may see relatively few normal traffic samples, but the traffic limiting and priorization system is not adversely affected.

### C. Traffic traces

We evaluate our mechanism using three different normal traffic sources: traces from medium-traffic university and small business web servers, and a non-governmental organization (NGO) campaign web server. These datasets are selected based on their different normal traffic clustering and traffic level stability features:

| Source | Clustering level | Normal traffic profile |
|---|---|---|
| University | Good | Stable |
| Business | Poor | Stable |
| NGO | Variable | Flash crowd |

The clustering level indicates how well the normal traffic clusters using the HHH algorithm. In a well-clustering set of data, a large portion of the clusters are at the /25 prefix level, which has been observed to be a boundary of consistent address block usage in the internet [4]. In a poorly clustering dataset, the clusters are predominantly either single IP addresses or relatively large clusters (/16.../6). The traffic source stability is also an interesting dataset feature: while the mechanism is expected to work well with a relatively stable user base, it should ideally also work when a suddenly popular website is also subjected to a DDoS attack.

Our attack traces are generated synthetically based on real-life attack cases. In order to thoroughly evaluate the performance of the system we created a comprehensive set of attack packet traces:

| Attack type | Unique attack source IPs |
|---|---|
| Naive flood | 1 |
| DNS reflection | 1000 |
| Botnet, valid requests | ≈114,000 |
| Randomly spoofed | >4,000,000,000 |

In the DNS reflection dataset, the attack source IP addresses are randomly selected from a public open resolver address database [1], and the botnet addresses are attack source addresses collected by a honeypot. Naive flood and random spoofed source addresses are completely random. With the possible exception of the naive attacker, all of these cases are

relevant to real-life attack strategies. As detailed in Akamai's State of the Internet report [2], the consistently popular DDoS attacks include potentially spoofed address flooding by UDP and TCP as well as DNS reflection attacks and HTTP GET floods from real source addresses.

From the point of view of our mechanism it is essential to evaluate the performance with a highly varying number of attack source addresses. This must be done in order to establish the service quality provided by the defense when the number of normal traffic clusters tainted by attack traffic ranges from very few to almost all. While evaluating all possible real-life attacks is impossible within the scope of any paper, these attack cases cover the performance implications of a very broad range of possible flooding attacks.

### D. Results

Our implementation of the mitigation mechanism uses traffic sampling, which introduces a random element in the clustering and traffic-ratio calculations. In order to ensure result reliability, we have done multiple iterations of each presented test case, and selected median cases for presentation. The maximum variation in results between test runs is in the order of 5%. Blacklist integration is used only in the test shown in section IV-D3 and replica servers only in section IV-D5. Unless otherwise mentioned, the results are shown for the following scenario: university normal traffic dataset, one day learning period, one day attack period, $\leq$100 clusters, normal server load 50%, attack volume $\approx$700% server capacity, no replica servers. The number of clusters is lower than what would be supported by our test hardware. With the low cluster count we demonstrate the applicability of the mechanism also for environments where the forwarding rule budget on network devices is limited.

Normal and attack traffic traces are played into the test network at a stable rate in order to maintain an unambiguous concept of normal server load. This does not affect the results, since the important factor for traffic variability is the change of traffic volume inside individual clusters, and this varies heavily even when the overall traffic rate is constant. The NGO dataset evaluation is an exception: the normal traffic during the flash crowd event was played at twice the usual rate. The university normal traffic dataset is selected as the baseline for graphical presentation since its good degree of traffic clustering results in graphs that are visually simple to analyze, while the other datasets had more service quality variance during a test run.

*1) Attack scenarios:* Figures 4(a-d) present the service quality measurements for the different attacks. The traffic measurements are shown as points in the scatter plot. The calculation for random packet drops, which would be the case without our defense mechanism, is shown as a graph line. It is important to note that our defense mechanism allocates the available capacity to certain clusters of clients. The benign clients in these clusters get perfectly normal service while others are not serviced. The average normal traffic served values during the attack are as follows:

| Attack type: | **Naive** | **DNS** | **Botnet** | **Random** |
|---|---|---|---|---|
| Service quality: | 98.7% | 74.1% | 80.5% | 81.4% |

The naive flood case is very well defended against, which is to be expected as all the attack traffic is coming from one source address and thus can only affect a single cluster. The results for botnet and random spoof cases are quite similar because, in both cases, the attack traffic is very widely and relatively thinly spread in the address space, and the traffic volume overlapping with normal user clusters is quite small. DNS reflection attack is the most effective of the attack cases, since the traffic volume per attacker address is heavier but the addresses are still spread wide enough that the attack impact cannot be limited by dropping a very small number of clusters like in the naive flood case. Overall, in all of the realistic attack cases the percentage of normal traffic served remains in the range of 74%...81%.

*2) Traffic management on switch:* Fig. 5 shows the aggregate traffic management on the edge switches in the case of a random spoof attack. Ideally the clustering adjustment should keep the normal plus attack traffic forwarding level as close to 100% of the server capacity as possible: any less and some normal traffic mixed with attack traffic is being unnecessarily dropped, any more and excessive traffic is being allowed to the server resulting in random packet dropping.

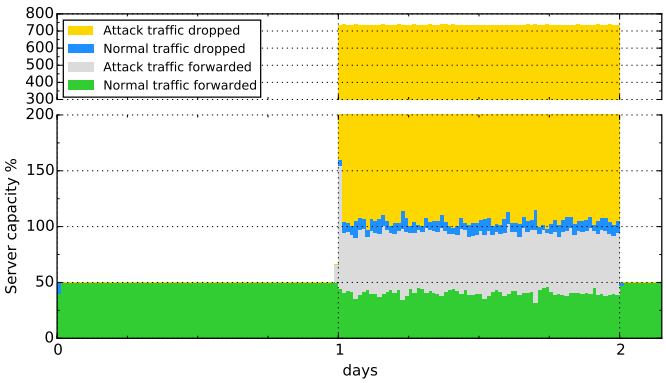
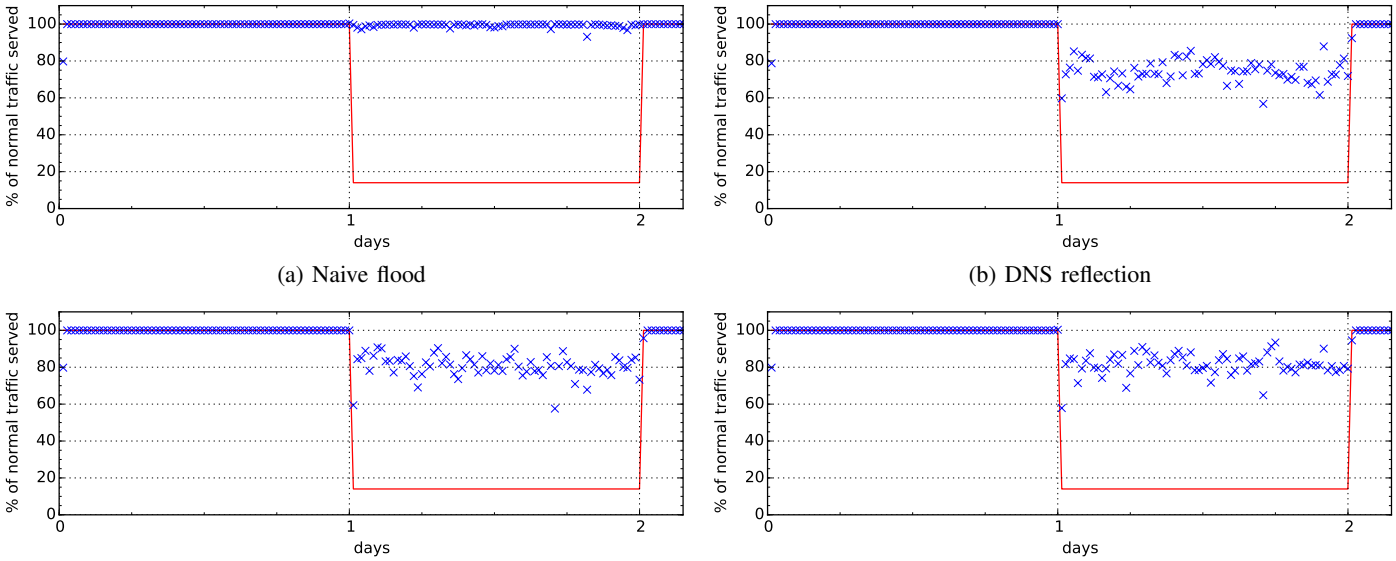



Fig. 5: Traffic management on edge switches

We can see that the mechanism effectively limits the traffic to roughly 100%. At the beginning of the attack all of the attack traffic is forwarded for a period of some seconds before the defense reacts and updates the flow entries on the switches.

*3) Blacklist integration:* Results derived from blacklist and attack scenarios are heavily dependent on the specified scenario. Thus, an exhaustive analysis of this topic is not within the scope of this paper. However, we do perform a test comparing the performance of blacklist plus clustering mechanism to using only the clustering mechanism.

The attack is a simultaneous DNS plus botnet attack, where the DNS source addresses are blacklisted but botnet source addresses are not. This reflects the incompleteness of real-life blacklists. The service quality with only the clustering defense is 74.8%, which, as expected, falls between the DNS and botnet attack results of section IV-D1. With both blacklist integration and the clustering defense the service quality improves to 82.6%. This result indicates that blacklist information, when available and accurate in relation to the attack traffic, improves the overall performance of the defense mechanism.




(a) Naive flood

(b) DNS reflection

(c) Botnet

(d) Random spoof

Fig. 4: Measurements of normal traffic served with different attacks. One-day learning period and one-day attack.

*4) Replica invocation:* The effect of replica invocation on service quality can be seen by comparing the cases in Figures 4(a-d) to the university dataset results of Table I. In the graphed cases there is no replica server involved, whereas the general evaluation case contains one replica server. The replica server effectively doubles the server capacity in the latter test scenario.

As an example, in the initial university normal traffic case the average service quality under botnet attack is 80.5%, which increases to 93.4% with the replica server. This indicates that the mechanism is able to benefit significantly from elastic server capacity, although increasing the capacity yields diminishing returns as the clusters allocated to additional capacity contain an increasing portion of attack traffic.

*5) General evaluation:* In addition to the previous examples that illustrate the behavior of the mechanism in different circumstances, we have performed average service quality calculations using all the normal-traffic to attack-traffic combinations. As previously stated, one replica server of 100% normal server capacity is included in these tests. The results are presented in Table I.

The learning period durations are different between the normal traffic datasets: the university dataset uses a one-day learning period, the business dataset two days and the NGO dataset five days. These learning periods have been selected based on the normal traffic characteristics: the worse the normal traffic clustering level and the larger the expected attack-time normal traffic in relation to the learning-period normal traffic, the longer the learning period needs to be.

In the NGO scenario, we define the normal plus replica servers to be at 100% capacity at the peak of the flash crowd event with the attack starting at the same moment of peak load. The NGO scenario is challenging for the defense mechanism because the flash crowd includes a very large number of users who are not present during the learning period.

| | | Attack traffic | | | |
|---|---|---|---|---|---|
| | | Naive | DNS | Botnet | Random |
| Normal | University | 98.9% | 85.0% | 93.4% | 92.4% |
| | Business | 98.2% | 94.4% | 88.4% | 84.6% |
| | NGO | 98.6% | 61.7% | 61.2% | 53.7% |

TABLE I: Normal traffic served averages with one replica server

It is apparent that the impact of a given attack type differs depending on the service. It is also not possible to easily identify the most effective attack. The fundamental reason for these obervations is that the effectiveness of the mechanism depends on the amount of address space overlap between the normal user and attacker addresses, and on the per-address traffic volumes. Nonetheless, our defense mechanism mitigates the attacks effectively even in the difficult NGO flash crowd with DDoS attack scenario.

*6) Effects of attack volume:* The attack volume in relation to server capacity has an effect on the resulting service quality. In the following we explore the effect of different attack traffic server overload factors on service quality with the university dataset normal traffic and botnet attack traffic:

| Attack factor: | **x1** | **x5** | **x10** | **x20** |
|---|---|---|---|---|
| Service quality: | 95.8% | 83.9% | 76.1% | 69.5% |

While the attack volume does affect the performance of our defense mechanism, the effect is relatively small compared to the increase in attack volume. Thus, we can state that the performance of our mechanism is relatively insensitive to the volume of the attack traffic.

## V. Discussion

Two existing techniques for traffic filtering in network devices are source blacklisting and large-flow detection. Solely using blacklists only works against attacks that are coming from the blacklisted addresses: spoofed attacks, previously undetected attack sources and flash crowds are not prevented from damaging the service availability. In large-flow detection, the network traffic is actively monitored and abnormally large traffic flows are blocked. This works well against attacks coming from relatively few sources, e.g. a DNS reflection attack using only tens or hundreds of DNS servers. Unfortunately large-flow detection does not offer protection against attacks coming from a large group of attack sources, i.e. a botnet, since the attacker is able to maintain reasonably normal data rates per attack source. Random source address spoofing also avoids this defense.

Like any learning-based defense mechanism, our mechanism is potentially susceptible to poisoning of the learning data by the attacker prior to the main attack. As separately discussed [7] this is not a critical problem in our mechanism, since it is extremely difficult for the attacker to gain a domineering effect on the learning data clusters without revealing the attack prematurely. Extremely volume- and source-variable attacks could potentially cause problems to the mechanism by overwhelming a cluster and roving to another cluster before the defense can react by blocking the overwhelmed cluster. This threat can be effectively controlled by limiting the maximum traffic rate of a cluster on the switch by using the meter functionality defined in the OpenFlow specification [10].

In terms of applicability we have evaluated the system against packet flooding attacks with a packets per second capacity limit. The mechanism is equally capable of bandwidth-based attack mitigation: the exact same principles and algorithms apply with the modification that in addition to packet count we also factor in the length of the packet. The defense mechanism operates on the IP layer, so attacks targeting higher layers are not within the scope of this defense. However, as observed in section IV-C, the majority of real-life DDoS attacks are based on flooding and are thus likely to be effectively mitigated by our defense.

The scalability of the mechanism is defined mainly by two factors: the feature processing capacity of the centralized traffic view element and the flow entry budget of the network switches. In feature processing the clustering, which is only done during normal operation, is not performance-critical. During a DDoS attack the full impact on feature processing can be effectively avoided by the fact that we can sample only a small part of the attack-time traffic without a significant loss in accuracy, as previously noted in section IV-B. The flow entry budget on the switches can also be effectively optimized by inserting only the pertinent flow entries, i.e. only inserting entries for traffic that actually flows through a given switch. As we have already demonstrated the effectiveness of our defense mechanism using a relatively low maximum number of flow rules it is unlikely that the flow entry budget would be a problem.

There are multiple options for gaining a sufficiently accurate view of the traffic flowing inside the network and controlling the switches accordingly. Our current implementation uses sFlow sampling for the view and direct switch manipulation for control. This could alternatively be done by using pure OpenFlow for both view and control, or hybridized by using sampling for network view, in order to avoid controller overload during an attack, and OpenFlow for switch control. Exploring the effectiveness of different network view and control schemes during a DDoS attack is a topic for future research.

## VI. Conclusion

We have presented and comprehensively evaluated a machine-learning-based DDoS defense mechanism which is designed for deployment on SDN-enabled networks. Our mechanism is capable of maintaining service quality for approximately 75% to 80% of normal clients when a service with a stable userbase is subjected to various flooding attacks. For a service subjected to simultaneous flash crowd and DDoS attack the mechanism maintains service for roughly 50% to 60% of benign users.

In addition the mechanism supports external blacklist integration and on-demand replica server invocation in a cloud environment. These features further improve the scenario-dependent service level for normal clients when the attack comes at least partially from blacklisted addresses or the network provides capabilities for server capacity elasticity.

## References


[1] Public DNS Server List, *http://public-dns.tk*, Feb. 2015.

[2] Akamai. Q4 2014 State of the Internet - Security, 2014.

[3] R. Aliyev, D. Seo, and H. Lee. DROP-FAST: Defending against DDoS Attacks using Cloud Technology. In *Int. Conf. on Security and Management*, 2013.

[4] X. Cai and J. Heidemann. Understanding block-level address usage in the visible internet. *ACM SIGCOMM Computer Communication Review*, 40(4):99–110, 2010.

[5] M. Collins and M. Reiter. An empirical analysis of target-resident DoS filters. In *Security and Privacy, Proceedings. IEEE Symp. on*, pages 103 – 114, 2004.

[6] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in streaming data. *ACM Trans. Knowl. Discov. Data*, 1(4):2:1–2:48, Feb. 2008.

[7] A. Kalliola, T. Aura, and S. Šćepanović. Denial-of-Service Mitigation for Internet Services. In *Secure IT Systems*, Lecture Notes in Computer Science, pages 213–228. Springer International Publishing, 2014.

[8] M. Mitzenmacher, T. Steinke, and J. Thaler. Hierarchical heavy hitters with the space saving algorithm. In *Proceedings of the Meeting on Algorithm Engineering and Experiments*, ALENEX '12, 2012.

[9] I. Mubarok, K. Lee, S. Lee, and H. Lee. Lightweight Resource Management for DDoS Traffic Isolation in a Cloud Environment. In *IFIP SEC*, 2014.

[10] Open Networking Foundation. OpenFlow Switch Specification 1.3.0, June 2012.

[11] P. Phaal and M. Lavine. sFlow Version 5, *http://www.sflow.org/sflow_version_5.txt*, 2004.

[12] P. Phaal and S. Panchen. Packet Sampling Basics, *http://www.sflow.org/packetSamplingBasics/index.htm*, 2006.

[13] R. Sahay, G. Blanc, Z. Zhang, and H. Debar. Towards Autonomic DDoS Mitigation using Software Defined Networking. In *NDSS Workshop on Security of Emerging Network Technologies*, 2015.

[14] F. Soldo, K. Argyraki, and A. Markopoulou. Optimal source-based filtering of malicious traffic. *Networking, IEEE/ACM Trans. on*, 20(2):381–395, April 2012.