Denial-of-Service Mitigation for Internet Services

Aapo Kalliola, Tuomas Aura, and Sanja Šćepanović

Aalto University, Espoo, Finland
{aapo.kalliola,tuomas.aura,sanja.scepanovic}@aalto.fi

Abstract. Denial-of-service attacks present a serious threat to the availability of online services. Distributed attackers, i.e. botnets, are capable of exhausting the server capacity with legitimate-looking requests. Such attacks are difficult to defend against using traditional filtering mechanisms. We propose a machine learning and filtering system that forms a profile of normal client behavior based on normal traffic features and, during an attack, optimizes capacity allocation for legitimate clients based on the profile. The proposed defense mechanism is evaluated using simulations based on real-life server usage patterns. The simulations indicate that the mechanism is capable of mitigating an overwhelming server capacity exhaustion DDoS attack. During attacks where a botnet floods a server with legitimate-looking requests, over 80 percent of the legitimate clients are still served, even on servers that are heavily loaded to begin with. An implementation of the mechanism is tested using synthetic HTTP attack traffic, also with encouraging results.

Keywords: denial of service, internet service, filtering, clustering.

1 Introduction

Denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks present an increasing threat to service availability on the Internet. Botnets can either exhaust the target server's processing capacity or its network bandwidth. Traditional methods for defending against a DDoS attack are based on manually configured filters. This filtering can either be done at the server itself or pushed to a firewall or a router further in the network.

As botnets have grown larger, it has become possible to overwhelm an online service by using perfectly legitimate-looking requests. These attack requests can be distributed among a large number of bots that make requests similar to those from normal users. It is difficult for a system administrator to define rules for differentiating between legitimate and attack traffic. For this reason, we look for automatic DoS filtering mechanisms which do not require manually configured rules, which work well against DDoS attackers, and which are benign to flash crowds.

The solution we propose builds on several ideas from the literature and uses a combination of machine learning and filtering. Our defense mechanism observes

K. Bernsmed and S. Fischer-Hübner (Eds.): NordSec 2014, LNCS 8788, pp. 213–228, 2014.
DOI: 10.1007/978-3-319-11599-3_13,

normal inbound traffic to the server and builds a profile of its distribution. When a DoS attack starts, i.e. when the server is overloaded, this profile is used for prioritizing requests. The assumption is that the attack traffic is not perfectly identical to the normal traffic. By giving priority to requests that most closely resemble the normal traffic, we are able to serve a large fraction of the legitimate clients and drop most of the DoS traffic.

In the learning phase, the system profiles the normal traffic as a set of clusters and their capacities. The clusters can be based on hierarchical traffic features such as IP addresses or request URLs. The most important feature is the client IP address. A filtering policy is created and updated continuously based on the observed normal traffic clusters. When the server comes under an attack, the learning component stops processing the incoming requests and filtering starts. The filtering policy proposed in this paper is based on comparing the amount of traffic in each cluster during normal operation and during the attack. Those clusters where the traffic volume does not increase significantly are the most likely to contain requests from legitimate clients.

DoS attacks vary widely in their level of distribution, bandwidth per attacker host, and attack traffic type and pattern. We therefore analyse the effectiveness of our DDoS mitigation mechanism using a range of different attack models. The simulation results indicate that once the server is attacked, our defense mechanism can retain good quality of service for over 80% of normal users during a request-flooding attack by a botnet, and for 40% to 70% of normal users during an overwhelming SYN flooding attack. Experiments with a prototype implementation also confirm the results.

2 Background and Related Work

Defending against denial-of-service attacks has been an area of active research. On a high level, the defense mechanisms can be divided into two main groups: host based and network based.

Network-based mechanisms often require significant changes to the communication mechanism between client and server. For instance in Phalanx by Dixon et al. [5] a massive packet forwarding swarm is set up to withstand botnet attacks. Others require the deployment of new mechanisms to core routers, as in pushback by Ioannnidis and Bellowing [8]. Peng et al. [13] present an overview of network-based defense mechanisms. Some of the host-based methods identify anomalous traffic at routers and communication endpoints with packet inspection [10].

One algorithm proposed for traffic clustering is *hierarchical heavy hitters* (HHH). The HHH algorithms were first presented in one-dimensional form by Cormode at al. [3]. HHH has also been used for DDoS detection by Sekaret al. [15].

Traffic filtering is commonly performed at a firewall near the server. Typically simpler filtering can also be done on routers and switches. The mechanism presented in this paper aims to be fully automatic and simple enough to deploy on

any of these network components. Our filtering mechanism can also be considered to bear similarities to priority-queue schemes such as that by Lin et al. [12]. IP-based filtering has also been discussed by Collins [2]. In contrast to work by Collins, our clustering mechanism produces dynamic and equal-bandwidth clusters with a fundamentally different goal for the filtering mechanism.

Overall, our approach to mitigating the impact of capacity-exhausting DDoS attacks combines aspects of host and network based countermeasures. We analyse the traffic reaching the server and use this information to activate a filtering mechanisms on the server itself or further upstream.

3 Filtering DoS Attacks

3.1 DoS Attacks

In this paper, we aim to filter DoS attacks in which the attacker overwhelms an online server, such as an HTTP server, with an excessive amount of requests. The attack may be executed from a small number of hosts, either inside or outside the target organization, or from a botnet with millions of computers distributed randomly around the Internet. The attacker may mount a brute-force attack to exhaust the CPU power or network bandwidth of the server, or it may intelligently request web pages that require expensive processing from the front-end web server or the back-end database. Consequently, the server must drop most of the incoming requests. Our goal is to be smart about which requests to drop at the server.

An attacker that is unable to exhaust the server capacity with legitimate-looking requests may turn to more primitive attack methods such as packet-flooding and connection-flooding. Most packet-flooding attacks can be filtered by a stateful firewall, which only allows through established connections and the initial packets, such as SYN packets for new TCP connections. This leaves the attacker with the possibility to open millions of connections to the server from botnet hosts or to send SYN packets from spoofed IP addresses. Our secondary aim is to protect against these types of attack by selecting intelligently which SYN packets are dropped at a firewall or upstream routers.

3.2 Filtering Defense

The architecture of our defensive system is shown in Figure 1. Inbound requests and connections are tapped into at the web server or in a separate device such as a firewall, router or switch. Depending on the bandwidth of the traffic flow, we can use either random samples or full traces of web requests and network connections. An out-of-band host analyzes the tap data and profiles the normal traffic. The system also monitors the server load and deploys the filter when a DoS attack is detected, i.e. when the server load exceeds 100%. The server may simply drop the filtered request or it may respond with a "Service Temporarily Unavailable" message.

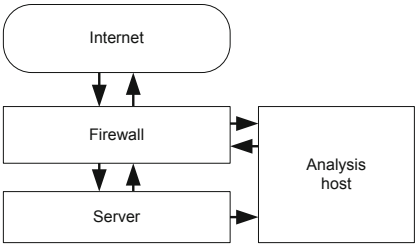


Fig. 1. Example of physical architecture elements

For high-bandwidth attacks that involve connection or SYN flooding, the filter needs to be deployed also at the firewall. In the experiments reported in this paper, though, the traffic tap and filter were implemented only on the server itself. We believe this to be sufficient to demonstrate the principle since the filter can be expressed as simple firewall rules. Nevertheless, there may be further implementation and performance details to consider in a high-bandwidth firewall or router, which require further investigation.

The analysis host continuously updates a profile of the normal traffic and prepares to deploy traffic filters based on this profile. The traffic profile and the filter have two components. Firstly, both define traffic classes called *clusters*. These are defined by sets of traffic features such as IP header fields and HTTP request parameters. Secondly, the traffic profile includes the measured *normal request rate* in each cluster, and the filter defines a *capacity allocation* for each cluster. When the filter is active, each cluster is limited to the given capacity and excess requests are dropped randomly.

As mentioned, *attack detection* in this paper simply means detecting when the server load exceeds 100%. No difference is made between DoS attacks and legitimate server overload such as flash crowds. Since some requests must be dropped during server overload, the filtering of legitimate requests causes no further damage. The effect of our filtering policy on legitimate overload is that the regular clients of the service will be favored over new or infrequent ones.

3.3 Feature Selection

It is desirable to find traffic features that differentiate legitimate requests from attack ones. The attacker, on the other hand, may try to *mimic* legitimate requests. The mimicking is made easier if the attacker gets feedback of it success, i.e. whether each request was served or dropped, as this enables an *adaptive* attack strategy.

Much of the past research on DoS attack detection uses timing features, such as inter-arrival times [14] and ramp-up behavior [7], to differentiate between an attack and legitimate server overload. These techniques do not, however, help to classify individual requests into legitimate and attack classes. Moreover, it can

be assumed that attackers are capable of mimicking such static features of the traffic if defenses based on them are widely deployed.

We considered various features in the request packets, such as IP header fields and HTTP request parameters, and initially expected to select a broad combination of these. Almost all such features can be mimicked by an attacker, especially since the server gives feedback by serving some requests and rejecting others. Finally, we settled for using the client IP address as the only feature. This is because the attacker cannot easily adapt its IP addresses to match the distribution of the IP addresses of the server's legitimate clients. Even a large botnet will lose most of its attack capacity if it uses only the bots that fall into the same subnets as the legitimate clients of the target service. Another advantage of IP addresses is that they are relatively stable. The distribution of application-level features such as the request URL may change much faster than the distribution of client IP addresses.

A SYN-flooding attacker that uses spoofed IP addresses can, in principle, choose any source IP addresses for the SYN packets. Its total capacity will, however, be somewhat limited by ingress filtering [6]. Furthermore, the attacker will not get feedback on the success of each spoofed packet, which will make it difficult to adapt the attack to the filter. It also helps the filtering that, in reality, certain significant classes of attacks, e.g. DNS reflection attacks, send the attack traffic from a small number of addresses.

3.4 Clustering

Given one or more traffic features, such as the client IP address, it is not at all obvious how the feature space should be divided into traffic classes. Our approach is to use a data clustering algorithm that automatically extracts some structure from the feature space. This was initially inspired by Liao et al. [11] who mapped packet features into real numbers. However, continuous feature spaces are a poor match for discrete or hierarchical features. In particular, IP addresses are by nature hierarchically allocated and also used in that manner [1]. It is therefore logical to perform the clustering with an algorithm that takes advantage of the hierarchical structure.

From the various ways of clustering the IP address space, we use the *hierarchical heavy hitter* (HHH) algorithm, which produces clusters with roughly equal rates of legitimate traffic. The resulting clusters are in practise IP subnets defined by an address prefix. One subnet may be inside another, in which case the traffic counts of the smaller subnet are not included in the traffic counts of the larger subnet. The filtering operations for these clusters are done by longest-prefix matching and thus can be performed very efficiently with firewall or router hardware. This efficiency is critical because the traffic volumes for the filtering can be very high. An example of the resulting cluster structure can be seen in Figure 2. The bracketed values after the subnet identifier show the legitimate and attacker requests counts in each cluster during a simulation run. As an additional advantage of HHH, the learning itself can be executed at line speed without buffering the training data [4].

The HHH algorithm takes as input a threshold value which determines the minimum size of a cluster as a percentage of the training traffic. For example, if the threshold value is 1%, the output will be at most 100 clusters. As a slight modification to the normal HHH, we always include a top-level cluster (zero-length prefix) for catching all remaining traffic even if its volume does not meet the threshold.

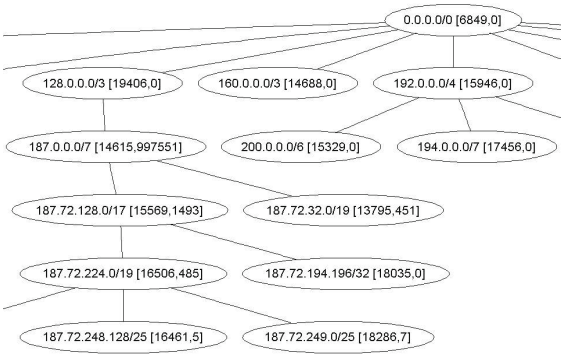


Fig. 2. HHH clustering example

The choice of an algorithm that produces clusters with roughly equal normal traffic rates can be argued as follows. IP addresses are un-uniformly distributed over the whole IP address space. This is especially true for IPv6. In order to minimize the number of configurable parameters, we do not want to give any a-priory knowledge about the IP address allocation as an input to the algorithm. Therefore, the only available information about the structure of the IP address space is the distribution of the client addresses during the learning period. By selecting clusters with equal normal traffic volumes, we use the given number of clusters relatively effectively to model the parts of the IP address space relevant to the particular server. It would, of course, be desirable to select the clusters in such a way that they maximally differentiate between the legitimate and attack traffic. Unfortunately, we cannot know in advance how the attacker IP addresses are going to be distributed. The only assumption we are prepared to make is that the distribution of source IP addresses in the attack traffic somehow differ from the legitimate traffic.

4 Simulation

4.1 System

We simulated a DDoS attack and the filtering defense on a web server that has a certain capacity for serving HTTP requests. The client IP address is used as the only traffic feature. Both the traffic monitoring and filtering implementations take place on the server itself. The main interest in the simulation is to

measure the effect of our filtering defense on the amount of legitimate traffic and clients that are served during an overwhelming DoS attack. The simulation is implemented in Python.

Real-life request traces from different web servers are used to generate the legitimate traffic. The graphs shown later are representative results that have been validated using the other data traces as well. The normal traffic data sets are roughly laid out in the following list.

1. *University web site 1*: medium traffic, 3 month period
2. *University web site 2*: low traffic, 3 month period
3. *Small business web site*: medium traffic, 6 month period

The attacker requests are created synthetically with the help of real-life IP source data from honeypot servers. The real-life IP data set contains over 100000 unique IP addresses that are actively attempting to spread malware. While they are almost certainly not part of the same botnet we can surmise that their IP address distribution is reasonable for approximating a large botnet. The relative activity rates of these compromised hosts are also used in modeling the non-homogenous capabilities of bots. The reason for using synthetic attack data is that real-life DDoS attacks are rarely recorded for academic use.

The simulation starts with a *learning period* during which only legitimate traffic is sent to the server. The hierarchical clustering algorithm is applied to the data from the learning period to create the cluster-based profile of legitimate traffic. Synthetic attack traffic is then generated and merged with the legitimate traffic from the next time period. As this data is fed to the simulation, the defense system moves from the learning phase to the filtering phase. It uses the previously constructed profile of legitimate traffic in combination with current measurements of the attack-time traffic to generate a filter for the incoming traffic. The filters are computed following a policy in which *the clusters with the highest ratio of legitimate traffic to attack-time traffic are allocated capacity and the rest of the traffic is dropped*. That is, some clusters are fully served while others are not served at all.

The only configurable parameter in the defense mechanism is the number of clusters k. It represents the complexity of the profile which the defender builds of the legitimate traffic during the learning period. It is not always the case that a larger k is better. First, there is the danger of overfitting the training data. That is, a too accurate profile of the legitimate traffic during the learning period will not match well the legitimate traffic during the attack. One goal of the simulations is to determine the best range of k in this respect. Second, the parameter also determines the number of traffic classes in the filter that is deployed during the attack. The filtering platform, such as a firewall implementation, usually can handle only a certain number of traffic classes effectively.

4.2 Attack Scenarios and Predicted Outcome

Since we use the IP address as the traffic feature for clustering and filtering, the attack model also focuses on the attacker IP addresses. We consider several different attack-traffic models. The reasoning is that if the filtering works against a comprehensive set of different attack-traffic patterns, it will also be effective against a range of real-life attacks.

In the simulations, we considered the following request-flooding attack types:

1. *concentrated*: high-bandwidth attack from a small number of IP addresses
2. *random*: random source IP addresses
3. *botnet*: simulated botnet based on real-life malicious host data

In the concentrated attack model, a heavy attack originates from only a few IP addresses or subnets, which are not among the regular clients of the server and thus are not present in the training data. A significant portion of real DDoS attacks fall into this category. In this case, the attack traffic falls into relatively few clusters. Most of the legitimate clients are in the other clusters and thus are expected to retain close to 100% service levels.

The random attack model represents a SYN flooding attack with spoofed source addresses. It can also be seen as a rough estimate of the pessimistic scenario where the attack is so massively distributed that each attacker source IP address is used only once or a few times. Many of the spoofed IP addresses will overlap with normal traffic clusters leading to reduced filtering effectiveness.

The third attack model is the most realistic one. We simulate the IP address distribution of a large botnet by using real-life malicious IP source addresses and adjust the flooding-traffic bandwidth from these bots according to observed real-life behaviour. The expected simulation result for the realistic botnet attacker is something between the concentrated and random scenarios.

4.3 Simulation Results

In this section, we show the simulation results of the attack models explained above. In addition, we discuss the effect of the learning period on the results.

In Figures 3 and 4, we use one-month datasets from a university web server as the normal traffic. Each dataset contains approximately one million requests with monthly unique IP counts in the order of tens of thousands. The datasets for the learning period and the attack period are from consecutive calendar months. The server load was varied by compressing a longer dataset into a shorter time period. The server load in normal operations prior to attack is indicated with separate lines in the figures. During the attack period, synthetic attack traffic exceeding the normal traffic by a factor of 10^6 is merged with the legitimate traffic. We measure how well the defense mechanism is able to filter the attack. The metric used for this is the percentage of the normal traffic served during an attack. It should be noted that the unrealistic factor 10^6 was chosen for demonstration purposes to show that the filtering *algorithm* scales well to any volume of attack traffic.

First, we conducted the simulation with only 10 high-bandwidth attacker IP addresses sending the attack traffic. In this simulation, we served 87-100% of the traffic regardless of server load in normal situation. The cluster count for these percentage values was between 100-10000. These good results are explained by the fact that, with such a low number of attackers, the attack traffic is placed into very few clusters.

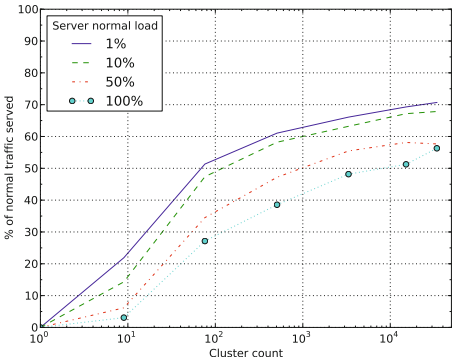


Fig. 3. Attack from random source addresses

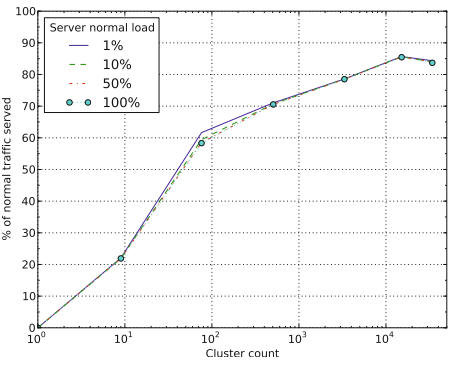


Fig. 4. Attack by simulated botnet

Next, we simulated the attack with random IP addresses. Figure 3 shows the percentage of legitimate traffic served during the attack. The x-axis indicates the number of clusters k, i.e. the granularity of the model which the defender builds of the normal traffic. The simulation results for the random (address-spoofing) attacker show that, even under the extreme DoS attack, the server is able to serve between 40% and 70% of the legitimate requests depending on the normal load. It is important to note that the service quality does not degrade equally for all clients. Those clients in the high-priority clusters are served all or most of the time and those in the low-priority clusters (i.e. ones where most attack traffic falls) are not served at all. We also observe that the normal load of the targeted server has a fairly small effect on the effectiveness of the defense.

In Figure 4, the attack traffic originates from a virtual botnet that contains about 110000 bots. The bots are geographically distributed based on real-life malicious hosts, as described earlier. In comparison to figure 3, we observe that the percentage of legitimate traffic served is higher, approximately 80% with manageable numbers of clusters.

We also analyzed the effects of overlearning, learning period duration and model aging to the filtering efficiency. We observed that overlearning can have a negative impact with very high cluster counts, as can be seen in the last data point of Figure 4. The learning period duration had relatively little effect after a number of requests in the order of tens of thousands was received. Model aging in relation to learning period had a significant effect on the filtering efficiency

with served traffic percentage dropping from an immediate value of 80% to 40% after the attack duration exceeded the learning period duration by a factor of 2. As the learning periods in real filter deployments are likely to be days or weeks, this would imply that an attack lasting several weeks is needed to degrade the service quality below 50%.

Overall, the simulation results indicate that the proposed clustering and filtering mechanism is effective in mitigating the effects of DoS and DDoS attacks. The worst case attack is one where the attacker sends requests from random IP addresses (i.e. spoofed SYN packets). Even then, the server is able to serve between 40% and 70% of the legitimate clients. Attacks by the simulated botnet and the concentrated attacker with only a few IP addresses were filtered even more effectively, with 80% to 100% of the legitimate clients served.

5 Implementation

5.1 Architecture and Implementation

Figure 5 shows the filtering-system architecture. The system is separated into several blocks, which can be divided into the performance-critical and non-performance-critical categories. The performance-critical components need to process in real time the requests arriving to the server. In order to not adversely effect the server throughput, these components on the path of the normal traffic to and from the server must be as simple as possible. Therefore, all the CPU and memory-intensive operations are performed on a separate analysis host connected to the traffic handling equipment over socket communications.

Traffic destined for the server is copied to the traffic tap and sent for feature extraction. The data from the tap is used both to create the normal traffic profile and to measure the server load levels. The feature extraction takes the IP addresses from the packets and sends them to feature-set bookkeeping. Both the traffic tap and the feature extraction may drop some data when the server comes under heavy load. This has little effect on the overall traffic profiling and filter generation process as long as the IP packets or HTTP requests are dropped randomly, so that the dropping does not bias the data.

Feature-set bookkeeping maintains a sliding window of data on the traffic features. The data in this window is fed to the HHH clustering algorithm. Our implementation of the HHH algorithm does not take advantage of the various approximation and memory-saving techniques presented in the literature. This is because we wanted to experiment with different variations of the algorithm. However, the slower algorithm is not detrimental in our implementation as the clustering algorithm is not placed in the performance-critical part of the system.

In normal traffic situations, the traffic filter receives empty filter sets from the filter updater and thus permits all traffic. When the server load exceeds a defined threshold, the traffic filter generator rolls back to the previous clean cluster set, gives the clusters a priority order based on which are believed to contain proportionally most legitimate traffic, and blacklists those clusters that exceed the server capacity. The filter is updated periodically to match changes in

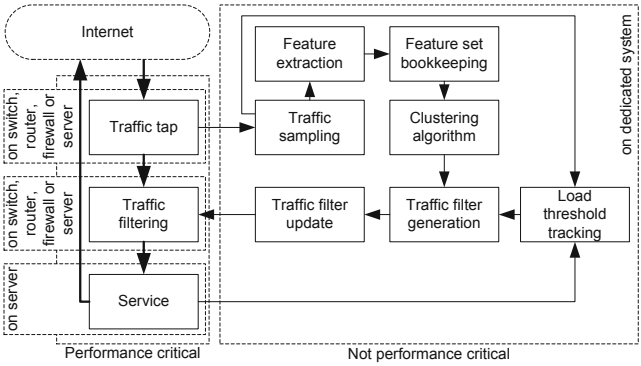


Fig. 5. Implementation architecture

the attack traffic. Once the attack subsides, the system observes the return of the incoming traffic to non-overload levels, removes blacklists from traffic filtering, and resumes normal learning operation.

We have implemented the tap and filter as user-space extensions to the iptables firewall. The traffic filters are black and whitelists of address prefixes. The clustering and traffic-filter-generation mechanism is an independent program running on a separate analysis host. The non-performance-critical clustering code is written in Python. The traffic tap could also be implemented as a separate network device, such as a switch, that mirrors the incoming traffic to the feature-set extractor though a monitoring port. Similarly, the filter could be deployed on a separate device. On the other hand, if we decided to use application-level features such as HTTP request parameters for the filters, it would be necessary to tap into that information on the web-server level. Further optimizing the implementation of the tap and filter components using for instance hardware support is left for future research.

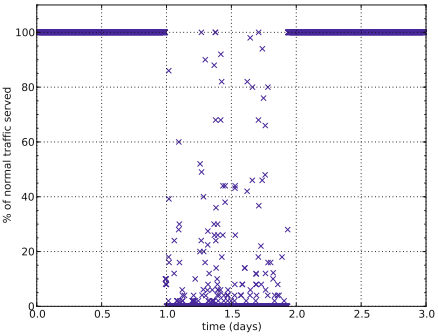
5.2 Test Setup

In order to test the functionality and performance of the implementation, traffic was fed to the server from two systems running traffic-generation software. One of the client systems generates traffic belonging to the legitimate clients based on data from recorded traffic logs while the other acts as the botnet. The network link between the client systems and the server is uncongested because our experiments are primarily concerned with server capacity exhaustion and not network congestion.

The performance of the clustering and filtering mechanism is evaluated on the client side. Both the legitimate client system and the attacker system analyze responses to their requests and generate reports on served and dropped HTTP requests. These reports are then analyzed and compared with the findings of the simulations.

5.3 Results

Figure 6 shows the implementation measurement results for the case of unprotected server coming under a one-day attack that exceeds the normal traffic by the factor 10. The normal server load before the attack is 50%. The unprotected server is unreachable almost the whole time from the perspective of the legitimate users.



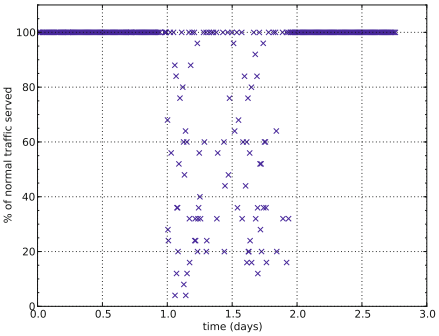


Fig. 6. Unprotected server, one day attack **Fig. 7.** Protected server, one day attack

Figure 7 shows the implementation measurement results for the case of a protected server subjected to the same attack as in Figure 6. The protected server remains mostly responsive to the legitimate users, serving an attack-time average of 69.8% of the legitimate requests. As in the simulations, this attacker with random source addresses is the worst case and attacks by a concentrated attacker or by a simulated botnet are filtered better.

6 Discussion

The simulation and implementation results show that the DoS defense mechanism proposed in this paper is effective in principle. The simple structure of the longest-prefix-matching filters is also an indication that it can be deployed in real systems.

One fundamental question is whether the local filtering of DoS attacks makes sense compared to distributed approaches. There are two kinds of distributed DoS solutions: network based and server based. The network-based solutions require broad changes to the Internet architecture or protocols and routers, which means that they cannot be deployed today to protect a specific service. The server-based solutions distribute the DoS protection or the online service itself globally, e.g. to a scalable server farm or cloud infrastructure. This kind of an approach is reasonable for large businesses that are able to pay for the higher availability. Small service provides and non-profit services with limited budgets,

on the other hand, struggle to meet the cost of such protection. Services that are frequent targets to DoS attacks may even be rejected by ISPs that consider them troublesome customers. Specialized DoS filtering services exist but at a premium price. For these reasons, we believe that there is a need for inexpensive locally-deployed DoS defenses that can mitigate the attacks at least to some extent.

In the sections below, we discuss some of the implementation and security issues that have not yet been answered in the paper.

6.1 Implementation Issues

For the reasons explained in section 3, we have used the client IP address as the only feature in the clustering and filtering implementations. Another feature that could be useful in addition to the IP addresses is the TTL field in the IP header, which has been been successfully used for filtering spoofed traffic [9]. If used in addition to the client IP address, it could improve the filtering accuracy against spoofed SYN packets in our system. In a network-based filtering system that protects multiple targets, such as a firewall in front of a data center, the target host may become another dimension for clustering. Application-level features such as the request URL could also be considered but these have the previously mentioned problem of being vulnerable to adaptive attacks.

The hierarchical-heavy-hitter algorithm works very well for one-dimensional hierarchical data. Multi-dimensional versions of the algorithm exist but they produce overlapping clusters, which is not acceptable in our application. HHH also does not work well with non-hierarchical data types such as port numbers. We initially designed algorithms for multi-dimensional clustering but they have not been thoroughly tested and the details are beyond the scope of the current paper.

It is arguable that our filtering strategy is unfair since it completely blocks access to some traffic classes that fall into the same clusters with heavy attack traffic. We believe this to be acceptable because TCP connections and most other network protocol fail anyway when most of their packets are dropped. However, it is trivial to change the filtering policy so that some small capacity is given to every cluster.

In the simulations, we have not given the lengths of the learning period and attack in normal time units such as hours or days. This is because we make no use of the timing of the recorded legitimate requests. Instead, the timing has been adjusted to get the desirable server load. This allowed us to experiment with a wide range of relative time periods and load levels. It would obviously be interesting to test the algorithms with naturally timed legitimate traffic and to observe the effects of natural time periods such as days and weeks on the learning algorithm.

We have also been somewhat vague about how the server overload is detected. In fact, in the simulations, we do not detect the attack but explicitly inform the algorithms to switch to attack mitigation mode. In the implementation, a threshold bandwidth for the traffic tap output was used to trigger the attack

response. The traffic threshold, of course, does not accurately tell when the server is overloaded, and a real deployment of the defense system should use some additional metrics. It is not trivial to accurately detect overload of a server because there are many resources, such as CPU, memory, hard disk, back-end database server, local network etc. that could become performance bottlenecks. This is a broader problem for web server management, though, and many ready solutions from that world exist. It remains to evaluate them and choose the right one for our purpose.

The filtering algorithm is based on longest prefix matching, which is also a key part of IP routing. Thus, much work has been done on its efficient implementation in both hardware and software. In the simulations, we have considered cluster numbers of up to 10^5 because the Internet routing tables are in that order of magnitude. In practice, we would not expect a local firewall or router to be able to handle filter tables of that magnitude with high-bandwidth traffic. The realistic cluster counts are one or two orders of magnitude smaller. As we have seen, cluster counts (and thus prefix counts) in the order of $10^2 \ldots 10^4$ already produce good filtering results.

In fact, we also experimented with a filtering implementation based on a DiffServ traffic classifier and scheduler with the hierarchical-token-bucket (HTB) queuing discipline. DiffServ supports at most 16 traffic classes. Nevertheless, it was able to successfully filter attacks from a limited number of IP addresses, with over 70% of the legitimate packets surviving.

Even then, there is the issue that real packet flooding (e.g. SYN flooding) DDoS attacks can reach bandwidths of tens of gigabits per second. While we have shown that the filtering algorithm scales well to almost any attack bandwidth, network hardware that can handle such amounts of traffic and execute prefix matching on every packet is going to be costly. One possibility for future consideration is to use our filtering policy for push-back, i.e. to deploy the filters in upstream routers at the ISP network.

6.2 Security Issues

Like in any security system, we need to also consider ways in which the attacker can circumvent the defense. One obvious way is to mount the SYN flooding attack from spoofed source addresses and try to *mimic* the address distribution of the legitimate clients. This requires the attacker to know how the legitimate clients are located in the Internet address space. The major disadvantage of the spoofing attacker is that the attack cannot be adaptive. That is, since the SYN packets are sent from spoofed IP addresses, the attacker gets no feedback on whether a particular address is served or not. Hence, the attacker must resort to manual research on the client-address distribution of the target server. That is possible but not at all as attractive as a fully automated attack, and one might expect many attackers to pass on the tedious work.

In addition to mimicking, another general concern in security systems that learn normal traffic profiles is the *poisoning* of the data. The danger is that the attacker slowly poisons the normal traffic set with malicious requests that remain

under the detection threshold. When the actual DoS attack then starts, attack traffic originating from the same IP addresses would be considered legitimate. With our mechanism, this attack is not extremely dangerous: even if some of the clusters contain attacker addresses, those clusters will experience the largest traffic increase during the DoS attack and, therefore, they will be served with the lowest priority. In addition, poisoning the normal traffic forces the attacker to begin his activity well in advance and risks alerting the defender about the attack beforehand. To achieve significant results, the attacker would need to do the poisoning for the majority of the learning period duration and to add a large amount of traffic to the normal traffic. Most attackers however prefer surprise to the risk of being detected early because the detection could alert the defender to take precautionary measures. Forcing the attacker to take such a risk is in itself an achievement.

7 Conclusion

While previous DDoS research has developed various solutions to different kinds of DDoS attacks, the problem of mitigating attacks consisting of perfectly valid request traffic has not been exhaustively researched. We have utilized machine-learning algorithms commonly used in data and traffic analysis to create cluster-based traffic profiles and used these profiles to optimize our allocation of server resources to different traffic clusters during the attack.

Our simulation results show that this filtering mechanism is capable of mitigating overwhelming attacks by botnets to such a degree that over 80% of the legitimate client requests are served. Under a SYN flooding attack, the same filtering algorithms enable us to serve 40% to 70% of the normal users. In practice, this would enable many of the regular clients of the service to continue to access it relatively undisturbed even during the attack. As a desirable side effect, the mechanism can also act as protection against flash crowds by favoring the regular users of the service.

Overall, we have presented and analyzed in detail a DDoS mitigation mechanism based on normal traffic profiling and attack-time filtering of traffic. By progressing through simulations on several different attack models to a proof-of-concept implementation, we have demonstrated the effectiveness of our mechanism in mitigating DDoS attacks that are based on flooding the server with valid requests or excessive numbers of connections or SYN packets.

References

1. Cai, X., Heidemann, J.: Understanding block-level address usage in the visible internet. ACM SIGCOMM Computer Communication Review 40(4), 99–110 (2010)
2. Collins, M., Reiter, M.: An empirical analysis of target-resident DoS filters. In: Proceedings of the 2004 IEEE Symposium on Security and Privacy (2004)

3. Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D.: Finding hierarchical heavy hitters in data streams. In: Proceedings of the 29th International Conference on Very large Data Bases, VLDB 2003, vol. 29 (2003)
4. Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D.: Finding hierarchical heavy hitters in streaming data. ACM Trans. Knowl. Discov. Data 1(4) (February 2008)
5. Dixon, C., Anderson, T., Krishnamurthy, A.: Phalanx: Withstanding multimillion-node botnets. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2008 (2008)
6. Ferguson, P., Senie, D.: Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2827 (Best Current Practice) (May 2000)
7. Hussain, A., Heidemann, J., Papadopoulos, C.: A framework for classifying denial of service attacks. In: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM 2003 (2003)
8. Ioannidis, J., Bellovin, S.: Implementing pushback: Router-based defense against DDoS attacks. In: Proceedings of Network and Distributed System Security Symposium, vol. 2 (2002)
9. Jin, C., Wang, H., Shin, K.G.: Hop-count filtering: an effective defense against spoofed DDoS traffic. In: Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003 (2003)
10. Lakhina, A., Crovella, M., Diot, C.: Mining anomalies using traffic feature distributions. In: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 217–228. ACM (2005)
11. Liao, Q., Cieslak, D.A., Striegel, A.D., Chawla, N.V.: Using selective, short-term memory to improve resilience against DDoS exhaustion attacks. Security and Communication Networks 1(4) (2008)
12. Lin, C.-H., Liu, J.-C., Jiang, F.-C., Kuo, C.-T.: An effective priority queue-based scheme to alleviate malicious packet flows from distributed DoS attacks. In: International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIHMSP 2008 (August 2008)
13. Peng, T., Leckie, C., Ramamohanarao, K.: Survey of network-based defense mechanisms countering the DoS and DDoS problems. ACM Comput. Surv. 39 (April 2007)
14. Ranjan, S., Swaminathan, R., Uysal, M., Knightly, E.: DDoS-resilient scheduling to counter application layer attacks under imperfect detection. In: Proceedings of the 25th IEEE International Conference on Computer Communications, INFOCOM 2006 (April 2006)
15. Sekar, V., Duffield, N., Spatscheck, O., van der Merwe, J., Zhang, H.: LADS: large-scale automated DDoS detection system. In: Proceedings of the Annual Conference on USENIX 2006 Annual Technical Conference, ATEC 2006 (2006)