Automatically defend from DoS and DDoS traffic flood attacks

Yinon Cohen and Maor Shabtay

Advisor: Dr. Amit Dvir

Ariel University

September 2018

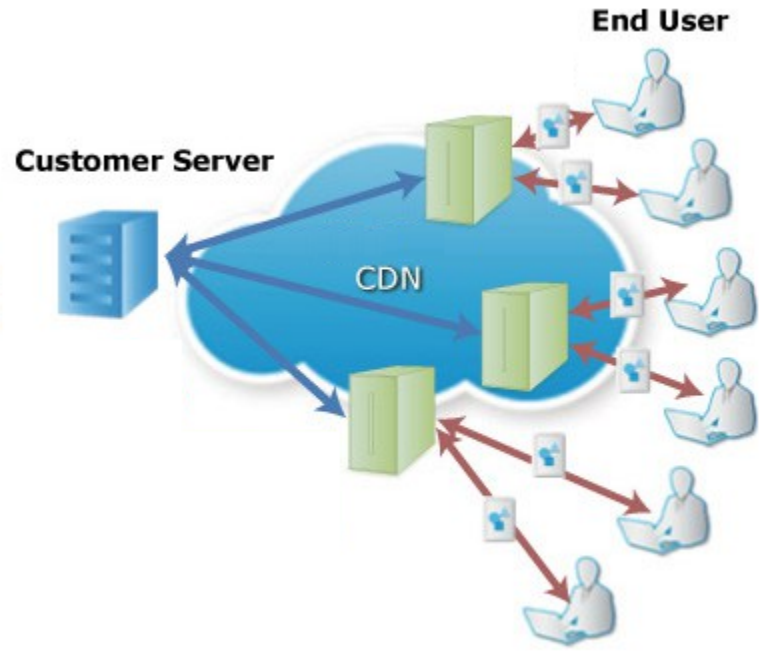
Contents

1. Abstract

Denial-of-Service attack (DoS) and Distributed Denial-of-Service attack (DDoS) attacks are attempts to exhaust server side assets, and designed to prevent client-to-server communication (denial of service). These attacks aim to both public and private sectors, and occur more and more frequently. In addition, lately the massive DDoS attacks are performing 100 Gigabits per second, and being more common than ever. These attacks are sowing fear among organizations and private server owners.

Our project deals with understanding and examining DoS and DDoS attacks, and what are the solutions for them. In particularly, we will discuss and handle with traffic flood attacks on web servers, and will try to develop our own software or algorithm to block or to give any immediately pragmatic solution.

Our main goal is to develop an automatic system that would identify and analyze a traffic flood attack, defend the server from it, and in need - will block any Internet Protocol (IP) or sub-net which the flood comes from. Our system should run on a Content Delivery Network (CDN) instead of on the server in order to save the server's performances providing service



2. Introduction

2.1. DoS

DoS attacks are attempts to exhaust server-side assets and designed to prevent client-to-server communication (denial of service). Simply, we can say that stealth server sabotage wires or even the server is denial of service, but in the context of data security we discuss about remote attacks and not physical sabotaging.

2.2. DDoS

DDoS attacks are very similar and sometimes even identical, and their intention is Distributed Denial of Service. In other words, the attack comes not from a single source, but from a large number of end stations – usually triggered by the attacker in the form of a king of virus located on these end stations. Most DDoS attacks are much more powerful and significant. It is important to understand that even an attack by two or three end stations is usually considered as a DoS attack, since there is really no significant flooding of the server.
Earlier this month Cisco released a white paper that [1] is part of the company's larger report, "Visual Networking Index Complete Forecast Update, 2015-2020." Here are some statistics from that white paper, relevant to distributed denial of service (DDoS) attacks:

- Frequency of distributed denial-of-service (DDoS) attacks has increased more than 2.5 times over the last 3 years.
- The average size of DDoS attacks is increasing steadily and approaching 1 Gbps, enough to take most organizations completely off line.
- Peak DDoS attack size (Gbps) is increasing in a linear trajectory, with peak attacks reaching 300, 400, and 500 Gbps respectively, in 2013, 2014, and 2015, at about 10 to 15 percent per year.
- In 2015 the top motivation behind DDoS attacks was criminals demonstrating attack capabilities, with gaming and criminal extortion attempts in second and third place, respectively.
- DDoS attacks account for more than 5 percent of all monthly gaming-related traffic and more than 30 percent of gaming traffic while they are occurring.
- Globally the number of DDoS attacks grew 25 percent in 2015 and will increase 2.6-fold to 17 million by 2020.

The DoS and DDoS attacks can be divided into two types:

- Attacks that flood and delay the service.
- Attacks that completely disrupt the service (and these we want to deal with in our project).

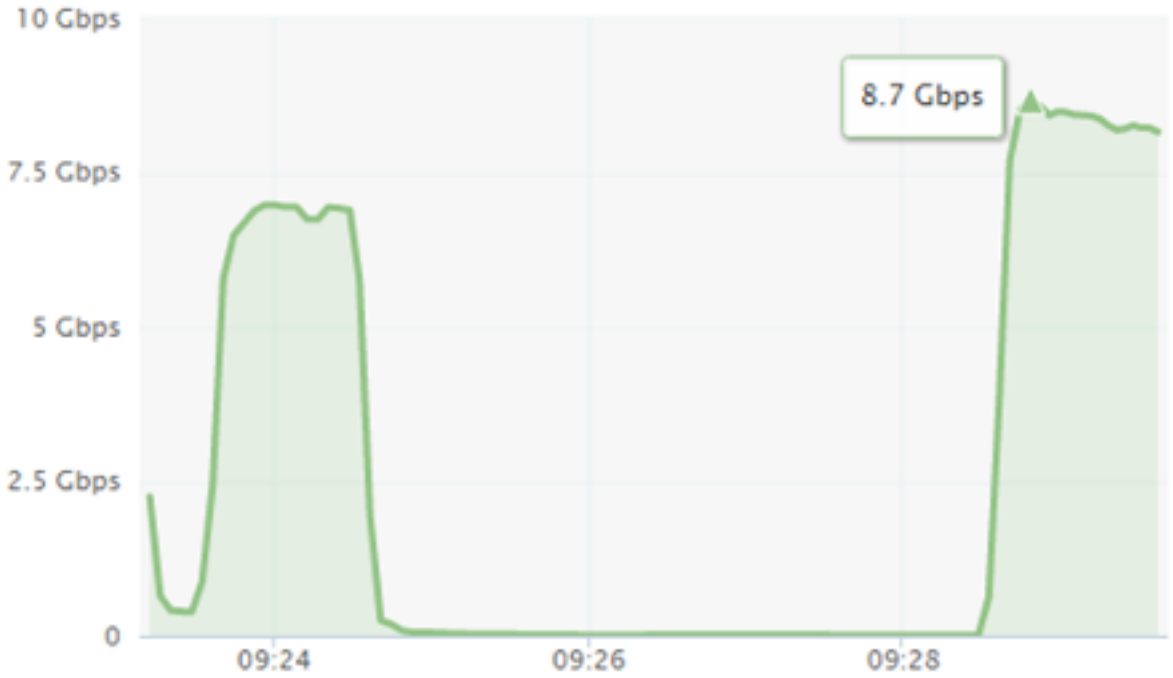


Figure 2.1: Sudden increase in server network traffic

2.3. DoS and DDoS attacks

Our research is including both types of attacks for DoS and DDoS attacks in general, and we will divide the types into different main types, based on the seven-layer-model (Open Systems Interconnection (OSI)) [2].

2.3.1. DoS and DDoS over Application layer

Attacks in the application layer are often generated by POST requests. They are also divided to sub protocols in the application layer – http / https.

- HTTP POST Flood - Creating and sending very large number of POST methods, to the extent that the server can't answer all requests, therefore service for real users of the server is compromised.
- HTTPS POST Flood - This is a flood of post methods that pass through Secure Sockets Layer (SSL) Session. The purpose of SSL is to take every message and decrypt it in order to inspect it. Flooding of these methods would harm the service.

- HTTP GET Flood - The attacker creates and sends to the server a huge amount of GET requests. The server needs to analyze all of them and return some data. Some people regard this attack as a Transport-layer attack, since sometimes the server would have to send a lot of data to the user attacker. Therefore, traffic and network bandwidth are flooded. Denial and service prevention depends on the server's capacity to getting and sending back packets. If it is able to handle a huge number of requests – the traffic will be damaged, and if it fails, the requests that it receives from real users will not be handled as the server falls.
- HTTPS GET Flood - Overflow of GET requests on HTTPS protocol requires a lot of work from SSL Session – decryption every message and hence load and sabotage the service.

2.3.2. DoS and DDoS over Transport layer

Flooding over Transport layer characterized mainly by packets that the server receives and is required to provide service – mostly by sending a requested data or any other response[2].

- Syn Flood - In this attack, the attacker takes advantage of the Transmission Control Protocol (TCP) principles that the server always wants to reach. When a server receives a Syn packet, it is a request from a client to open a connection, and it is obligated to respond to it and must return the client a Syn – Ack certificate. Each Syn message requires time from the server – analyzing the packet (understanding who created it, calculating 'Checksum' etc.), and then be able to reply. Therefore, flooding these messages is slowing down and compromising the server's serviceability.
- Rst Flood - Like Syn flood, the attacker takes advantage of TCP principles, including reliable communication. In case that a socket is closed or when one of the sides disconnected (and in few other situations), TCP has a solution. The connected side still wants to continue the communication (since there was no closing connection process), it sends a packet with a Rst flag and hence they have to re-open the connection. Like Syn packets overflowing, Rst packets overflowing also require a lot of work from the server and would sabotage the service.
- User Datagram Protocol (UDP) Flood - UDP floods are used frequently for larger bandwidth DDoS attacks because they are connectionless and it is easy to generate UDP messages from many different scripting and compiled languages. The attack can be initiated by sending a large number of UDP packets to random ports. As a result, the server would check for the application listening at that port, realize that no one is and reply with Internet Control Message Protocol (ICMP) packet saying 'Destination Unreachable'. Thus, for a large number of UDP packets, the server will be forced into sending many ICMP packets and much performance.

2.3.3. DoS and DDoS over Network layer

DoS and DDoS attacks over the Network layer are characterized with a large number of packets in order to overload the bandwidth and exhaust network resources. Network resources can be routers, firewalls and servers, and it is clear that their ability is final.

- ICMP Flood - ICMP protocol is typically used for error messages rather than data exchange between systems. Flooding messages with ICMP protocol – e.g. ping – is intended to overload the network.

2.3.4. DoS and DDoS over Link layer

DoS and DDoS attacks over the Network layer require access to the local network. Therefore, they are rare and more easy to detect.

- Media Access Control (MAC) Flood - A rare attack, in which the attacker has to be connected to the local switch. The attacker sends multiple dummy Ethernet frames, each with different invalid MAC address. Network switches maintaining their MAC table, and treating MAC addresses separately, and hence reserve some resources for each request. When all the memory in the table is used up, it either shuts down or becomes unresponsive.

2.3.5. Demonstrating APDoS Attack

The Advanced Persistent Denial-of-Service (APDoS) is an attack that combines many DoS and DDoS attacks, and is carried out by a lot of hostile elements over time. APDoS represents the worst Denial of Service attack that can occur. The idea behind it is a combination of many attacks from multiple endpoints, and over long period of time, hence its name Advances Persistent DoS. In this attack, the attackers usually attack several stations in order to create a distraction from the DoS defenses, but concentrate on one main victim in the organization.

2.4. DoS and DDoS defense solutions

2.4.1. DoS and DDoS solutions over Application layer

- HTTP POST Flood - There is a difficulty in distinguishing between legal traffic and attack. The most effective mechanism that exists today is by combining methods of characterizing the movement of requests and identifying the source user. When a random url is used, an exception check is required to understand that this was an attack and not an innocent use of the server [3]. Part of the exception check is to try to identify the source user that triggers the attack, and you may notice that sometimes a large part of the package signature and content is the same.

- HTTPS Request Flood - Using the BIG-IP system [4] and the F5 iRules scripting language. Now available via the F5 DevCentral online community, this iRule states that if a device tries to renegotiate more than five times in any 60-second period, the connection is silently dropped. The biggest benefit to this approach is that the attacker believes the attack is still working and in service, when in actuality, the server has ignored the request and moved on to processing valid user requests instead.

- HTTP GET Flood Today We know about two detection algorithms [3], one is focusing on a browsing order of pages and the other is focusing on a correlation with browsing time to page information size. that implement detection techniques and evaluate attack detection rates, i.e., false positive and false negative. The results show that our techniques can detect the HTTP-GET flood attack effectively.

2.4.2. DoS and DDoS solutions over Transport layer

- Syn Flood - We have a lot of solution [5] for this attack: Filtering ,Increasing Backlog,Reducing SYN-RECEIVED Timer,Recycling the Oldest Half-Open TCP,SYN Cache,SYN cookies,Hybrid Approaches,Firewalls and Proxies We will expand a bit on SYN cookie is a technique used to resist SYN flood attacks. The technique's primary inventor Daniel J. Bernstein defines SYN cookies as "particular choices of initial TCP sequence numbers by TCP servers." In particular, the use of SYN cookies allows a server to avoid dropping connections when the SYN queue fills up. Instead, the server behaves as if the SYN queue had been enlarged. The server sends back the appropriate SYN+ACK response to the client but discards the SYN queue entry. If the server then receives a subsequent ACK response from the client, the server is able to reconstruct the SYN queue entry using information encoded in the TCP sequence number.

- Rst Flood - A RST packet is accepted if the sequence number is in the receiver's window, or when a connection is closed (closed socket). This is slightly different from a FIN, which just says that the other endpoint will no longer be transmitting any new data but can still receive some.

  There are three types of event that cause a RST to be emitted. A) the connection is explicitly aborted by the endpoint, e.g. the process holding the socket being killed (just closing the socket normally is not grounds for RST, even if there is still unreceived data). B) the TCP stack receiving certain kinds of invalid packets, e.g. a non-RST packet for a connection that doesn't exist or has already been closed. C) An unexpected amount of RST packets gained and there is no stopping point when sending back even a few responses. Emmiting these senarios would mitigate the performance and engagement of the server with unwanted traffic. [6].

2.4.3. DoS and DDoS solutions over Network layer

- ICMP Flood - Reconfiguring your perimeter firewall to disallow pings will block attacks originating from outside your network, albeit not internal attacks [7]. Still, the blanket blocking of ping requests can have unintended consequences, including the inability to diagnose server issues. The Incapsula DDoS protection provide blanket protection against ICMP floods by limiting the size of ping requests as well as the rate at which they can be accepted.

2.4.4. APDoS solution

To combat APDoS, organizations require a single vendor, hybrid cyber security solution that protects networks and applications from a wide range of attacks. Ideally, such a solution [8] includes all the different technologies needed for effective detection and mitigation, including DoS/DDoS protection, behavioral analysis, Intrusion Prevention System (IPS), encrypted attack protection and web application firewall (WAF). Additionally, organizations also require new levels of partnership with their DDoS mitigation service provider and any Internet Service Providerl (ISP) that provides managed DDoS services to coordinate for the effective detection and mitigation of a multi-vector assault.

3. Related Work

3.1. Related Articles

3.1.1. Traffic flooding attack detection with SNMP MIB using SVM

DoS and DDoS attacks have become more and more destructive, and are threatening to various network services. Hence, the various methods of protection and monitoring and control of network traffic have also begun. However, most of the current modern detection systems are focusing on detail analysing for each packet's data, which causes late detection and can't handle high network traffic.
Simple Network Management Protocol (SNMP)[9] provides a universal method of exchanging data for purposes of monitoring systems that reside on a network. The use of SNMP is most dominant in the modern industry. But, to utilize SNMP for traffic flooding attack detection, we need to consider the following three points in the use of the SNMP MIB variables which affects the performance and accuracy of the detection system:

- Proper selection of SNMP MIB variables for attack detection
- Determination of the detection timing about when and how often
- Algorithm for attack detection using the selected MIB (Management Information Base) variables.

3.1.2. Change trend of averaged Hurst parameter of traffic under DDoS flood attacks

DoS and DDoS flood attacks are great threats to the internet though various approaches and systems have been proposed. Hence, Intrusion Detecting System (IDS) and Intrusion Preventing System (IPS) are desired. The DDoS flood attack sends packets upon a server with a huge amount of traffic. It never tries to break into the server's system, which makes the servers's security defenses irrelevant.
The solutions given by misuse detection are primarily based on a library of known signatures to match against network traffic . Hence, unknown signatures from new variants of an attack mean are hard to be recognized. Therefore, anomaly detectors (exceeded routine detectors) play a role in detection of DDoS flood attacks.
It is important considering the Hurst parameter - H in characterizing exceptions of traffic series in packet size under DDOS flood attacks. This paper specifically [10] studies how H of traffic is changing under DDoS flood attacks. It it important understanding the following:

1. Whether H of traffic when a server is under DDoS flood attacks is much different from the regular one?
2. How is H changes when a server suffers from DDoS flood attacks?

Answering these questions might give us better understanding for detecting and protecting from DDoS flood attacks.

3.1.3. Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments

Cloud computing develops rapidly due to its essential characteristics. Cloud computing would not be possible without the underneath support of networking. Recently, Software-Defined Networking (SDN) [11] has attracted great interests as a new paradigm in networking. In SDN , network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications. Denial of Service (DoS) attacks and Distributed Denial of Service(DDoS) flooding attacks are the main methods to destroy availability of cloud computing. Although the capabilities of SDN make it easy to detect and to reactDDoS attacks in cloud environments, the separation of the control plane from the data plane in SDN introduces new attack planes. SDN itself may be a target of some attacks, and potential DDoS vulnerabilities exist across SDN platforms. For example, an attacker can take advantages of the characteristics of SDN to launch DDoS attacks against the control layer, infrastructure layer plane and application layer of SDN .

3.2. Related Productions

3.2.1. Cisco - 'DDoS Mitigation'

Cisco had developed a System which delivers a complete DDoS protection solution based on the principles of detection, diversion, verification, and forwarding to help ensure total protection[12]. The solution maintains the business continuity by:

- Detecting the DDoS attack.
- Diverting the data traffic destined for the target device to a Cisco appliance for treatment.
- Analyzing and filtering the bad traffic flows from the good traffic flows packets, preventing malicious traffic from impacting performance while allowing legitimate transactions to complete.
- Forwarding the good traffic to maintain business continuity.

3.2.2. Check Point - 'DDoS Protector'

Check Point had developed a System called 'DDoS Protector'[13], which keeps businesses running with multi-layered, customizable protections and up to 40Gbps performance that automatically defends against network flood and application layer attacks with fast response time against today's sophisticated denial of service attacks.
DDoS Protector Appliances offer flexible deployment options to easily protect any size business, and integrated security management for real-time traffic analysis and threat management intelligence for advanced protection againstDDoS attacks. The product provides multi-layer protections, handles network and traffic flood and has a management system.

3.2.3. Our work

In order to implement our system, we will must focus on preventing the DoS and DDoS attacks from our side.
Similar to 'doi' system (Traffic flooding attack detection with SNMP MIB), we have to analyze the traffic coming from the network,
and to determine which traffic and data are meant to harm our system. Just like they did, we must realize which second third and more identical packets are caused due to timeouts or part of some DoS or DoS attacks.
In contrast, determining which 'H' traffic is under DDoS flood attack, in our opinion won't help us developing a system which would prevent our server to handle requests.
In addition we note that no other related workers implemented their system upon a CDN server which is separated from the server.
This is a very important point because our whole goal is to assist our server in cases of DDoS attacks.
If the server would analyze the requests arriving and running our algorithm, he will not be able to handle the requests themselves.
Remember, the server might deal with its databases, algorithms and even complex calculations.
The server hypothesis should be that most of the requests are legal and not meant to harm it.
Thats where our system's importance. In order to manage building a proper solution for DoS and DDoS traffic flood attacks, the system must contain the following:

- A durable CDN server which could communicate both with the server and the clients.
- Proper algorithm which can identify traffic status and mark and block specific ips and subnets.
- Experiment DoS and DDoS attacks with and without our algorithm in order to determine that our algorithm is properly working and mitigation any attack.

4. Difficulties

When we developed and even planned our system, we had a few technical and theoretical issues.

4.1. Building our own system

In order to determine the specific behavior and requests handling of the server, we must develop our server:

- Coding a server which listens and handles every request properly.
- Developing a CDN server, which accepts requests from clients, following them to the server, and when answer from the server arriving, return it to the right client from many clients.
- Coding a client end-point which could send regular request or multiple flooding requests over the server.
- Developing an algorithm which can identify and analyze the requests and deciding either to handle the request, mark the specific client, or block it.

4.2. Creating a proper network environment

In addition, to determine that our system really providing a proper solution for DoS and DDoS attacks, we had to implement such attacks:

- Creating a network simulation of our network architecture.
- Use it in a third-party application which could demonstrate a real DoS and DDoS attacks.

4.3. Experimenting the system

After planning, developing and using our system, we should decide if it is good enough for a proper solution for web servers. Our goal is to provide continuous activity of the server even when it is under DoS or DDoS attack.

- Experimenting our system, and analyzing it against other existing solutions.
- Integrating our system with existing web servers.

4.4. Future difficulties

Beyond our project, there is a whole world of DoS and DDoS attacks. In our future progressive work, we need to give other solutions to a variety of DoS and DDoS attacks:

- Building a mitigation for other layers except Application-Layer (e.g. Transport-Layer).
- Making our system available even for apache servers and not only web or desktop servers.
- Determining which is the best platform for mitigating DoS and DDoS attacks - Asynchronous single-threaded or Synchronous multi-threaded. There are many difference opinions in this case.

5. Our algorithm

5.1. Calculating approved count of times for each file

Algorithm 1

function MAX-FILE-REQUESTS($file$)
    Let smallestFile be the smallest file in the server
    Let smallestCount be his number of approved requests for it.
    Let biggestFile be the smallest file in the serverr
    Let biggestCount be his number of approved requests for it.
    if $file == biggestFile$ **then**
        return return biggestFile
    else if $file == smallestFile$ **then**
        return return smallestFile
    else
        Let y be the ratio between the files: $y = \frac{file - small}{big - small}$
        Let expectedCount be number of ratio between them:
        $ansRequest = smallRequest - \frac{(smallRequest - bigRequest) * Y}{100}$
        $expectedCount = round(expectedCount)$
        return expectedCount
    end if
end function

5.2. Handle request for tracking table

Define tracking-table (ip, file, time, count) Define max-time – maximum time between several requests

Algorithm 2

```
1: function HANDLE-TRACK-TABLE(ip, file, time)
2:     Let ip be the requested ipr
3:     Let file be the requested file
4:     Let time be the current time
5:     if tracking − table[ip]! = null then
6:         count ← 1
7:         Insert to the table: ip, file, time, count
8:     else if tracking-table [ ip ] not contains file then
9:         count ← 1
10:        Insert to the table: file, time, count
11:    else
12:        Let firstTime be the first time the ip requested this file.
13:        if |firstTime–time| > max − time then
14:            tracking − table[ip][time] ← time
15:            tracking − table[ip][count] ← 1
16:        else
17:            count + +
18:            if count > max − file − requests(file) then
19:                return false
20:            end if
21:        end if
22:    end if
23:    return true
24: end function
```

5.3. Handle request

Define blocking-table: (ip, ExpIncreament, degree)

Algorithm 3

```
1: function BLOCKING-IP(ip, file, time)
2:     if blocking − table[ip]! = null then
3:         Close the socket
4:     else if Handle − track − table(ip, file, time) == false then
5:         blocking-table [ ip ] [ ExpIncreament ] ++
6:         if ExpIncreament >= 3 then
7:             Block user
8:         else
9:             blocking-table [ ip ] [ degree ] ← 2^ExpIncreament
10:        end if
11:    else
12:        Handle the request
13:    end if
14: end function
```

5.4. Intervals

Algorithm 4 Intervals

1: Every 15 seconds downgrade every degree.

2: Every 3 minutes reset all tables.

5.5. Advantages and disadvantages

5.5.1. Advantages

- Working on every server platform (web servers, desktop server or cloud servers).
- Does not require a specific operating system.
- Does not requires any external packages.
- Calculating blocking users together with requested file sizes.

5.5.2. Disadvantages

- Taking time to implement.
- Protective only for Application-layer (OSI).

6. Related mitigations

6.1. Mod evasive

6.1.1. Explanation

mod evasive is a module for Apache that provides evasive action in the event of an Hypertext Transfer Protocol (HTTP) DoS or DDoS attack or brute-force attack.
The module tracks HTTP connections and verifies how many requests for a page are done within a given time frame. If the number of concurrent requests exceeds a specified threshold, then the request is blocked. This blocking is done on an application level. The requester gets a forbidden answer to the request.
It allows you to add your management and proxy networks to the DOSWhitelist setting so that you do not block your own network.

6.1.2. How to use

On Ubuntu, run:

```
$ sudo apt-get install libapache2-mod-evasive
$ sudo mkdir /var/log/mod_evasive       // creating log folder
$ sudo a2enmod evasive                  //enabling mod_evasive
```

6.1.3. Advantages and disadvantages

Advantages

- Very easy to use.
- Very effective.

Disadvantages

- Working only on Apache servers.
- External package use.
- Requires Ubuntu operating system only.
- Protective only for Application-layer (OSI).

6.2. Information security / network-layer approach

6.2.1. Explanation

A far more practical way, considering we are deciding for the fate of the packet right before it reaches the other end or even if it had already completed the 3-way handshake. consider these examples:
a. restrict the number of concurrent connections per IP on your firewall. this will give you the ability to define how many connections can a client requests simultaneously. in iptables, you can do this by:

```
$  iptables -p tcp --syn --dport 80
   -m connlimit --connlimit-above 20
   --connlimit-mask 24 -j DROP
```

in openbsd, you can do this in pf.conf:

- Limit the absolute maximum number of states that this rule can create to some number (e.g. 200)
- Enable source tracking; limit state creation based on states created by this rule only
- Limit the maximum number of nodes that can simultaneously create state to some number (e.g. 100)
- Limit the maximum number of simultaneous states per source IP to 3

Example in bash (OpenBSD operating system):

```
pass in on $ext_if proto tcp to $web_server \
     port www keep state \
     (max 200, source-track rule, max-src-nodes 100, max-src-states 3
```

6.2.2. Advantages and disadvantages

Advantages

- You're preventing the attack even before it arrived to the 3-way-handshake.
- Quite easy to implement.

Disadvantages

- You need to know the average amount of users per hour.
- You might limit some users when not needed.
- Access to the ip-tables required.

7. Technical work

7.1. Our systems

7.1.1. Building our own server and CDN server

In order to determine the specific behavior for the CDN server and original server when packets are arriving, we had to code our own servers. This was expressed in the following cases:

- Paying attention in every type of http request, (text/plain , text/html , application/-javascript etc.).
- The server had to calculate the file sizes and give priority for each one of them.
- The CDN server should gain the priority table, and to mark significant users from flooding the server. In addition it should close the socket for every flooding user, and to count it.
- Determining which end-point is a non-regular user, and be able to block it if needed.
- Creating registration for gained requests, and thus we were able to experiment our system.

For that, we used NodeJS platform for building the servers. Asynchronous single threaded servers are usually very fast than regular servers, and that helped us to handle huge amount of request per second.

Usage: node Client [options]

```
$ node server --port <port>

  Flags:
  --port              define argument of PORT listener.

  Args:
  <port>              Port value for server's listener.
```

Usage: node Server [options]

```
$ node request -t <ip> -p <port> -f <path> -ht <requested file>

  Flags:
  -t                   define argument of IP destination.
  -p                   define argument of Port destination.
  -f                   define argument of path folder.
  -ht                  define argument of requested file

  Args:
  <ip>                 IP address for attack.
  <port>               Port destination for attack.
  <path>               path of folder to download files
                       (default /dev/null/)
  <requested file>     path to requested file on the server.
```

Usage: node CDN [options]

```
$ node CDN --server <server_ip> --serverPort <server_port>
  --port <port> --no-protect
  Flags:
  M - Mandatory
  --port                define CDN server's port argument
                        (4400 by default)
  --server        M     define original server's IP argument
  --serverPort    M     define original server's Port argument
  --no-protect    M     define CDN mitigation

 Args:
  <server_ip>           IP address of Server.
  <server_port>         Port destination for server
  <port>                Port destination for CDN.
```

7.1.2. Create System

We are discussing and investigating all about DoS and DDoS attacks. In order to really simulate a DDoS attacks, we need several different end-points to flood our servers. Create system do just that.
With NS file system (Network simulator), we're creating virtual nodes representing our server, CDN sever, and clients. This language was built in order to simulate virtual nodes and be able to link between them – creating virtualized vetwork.
On each end-point we installed Ubuntu (Linux Distribution), and on each one of them installed NodeJS. The nodes on Create system are not connected to any external network, thus the have no internet connection. In order to install it we had to define Create's proxy connection and then we were able to get the external files.
Because these nodes are virtual, there is no real built-in gateway on the router, and for each end-point we need to simulate a router and a gateway in order to route packets to the right direction.

Example for node set up:

```
$ tb-set-node-os $server Ubuntu1404-32-STD
$ set routerServer [$ns node]
$ tb-set-node-os $routerServer Ubuntu1404-32-STD
$ set link55 [$ns duplex-link $routerServer $gatewayServer
  1000Mb 0ms DropTail]
$ set MainLink [$ns make-lan "$gatewayCDN $gateway1 $gateway3
  $gateway2 $gatewayServer " 100Mb 0ms ]
```

In addition, we simulated one link, which will represent a huge connected system between several end-points.

Figure 7.1: Gateways in general are being provided for each router, but when defining virtual routers, they are our way to route packets to the right end-point.

7.2. Experiments

After implementing our system, we've made many experiments in order to test and prove our system's success with mitigating DDoS attacks. For each experiment, we first turned on the servers, and then started to run the client sides.

1. Multiple end-points without any mitigation: The server gained in average 3600 requests in 3 seconds (1200 per second). After 12 seconds of flooding, the server was shut down.

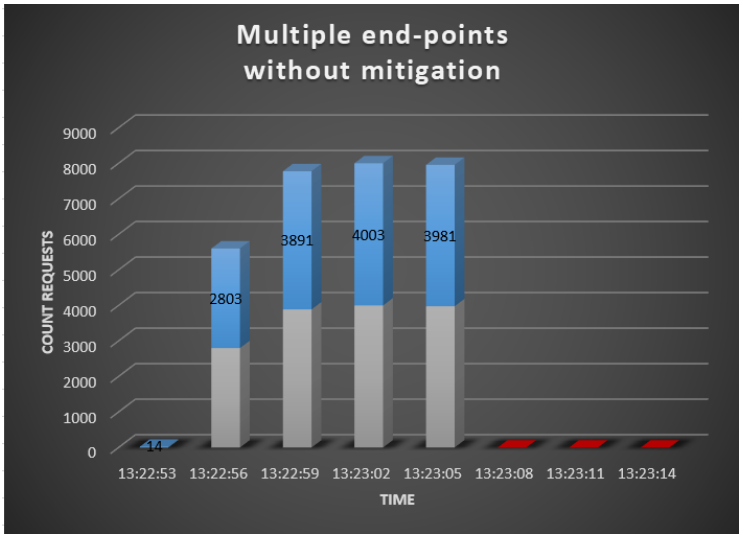


Figure 7.2: DDoS traffic flood attacks without our mitigation.

2. Single end-point without mitigation: The server gained in average 2000 requests in 3 seconds (670 per second). The server could handle all of the request although the huge amount.

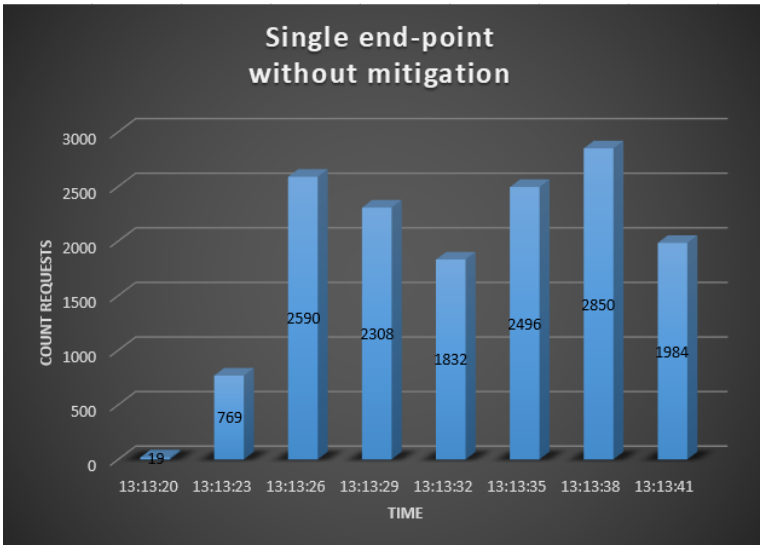


Figure 7.3: DoS traffic flood attacks without our mitigation

3. Multiple end-points with our mitigation: he server gained in average 55 requests in 3 seconds (18 per second). No significant flood on the server.

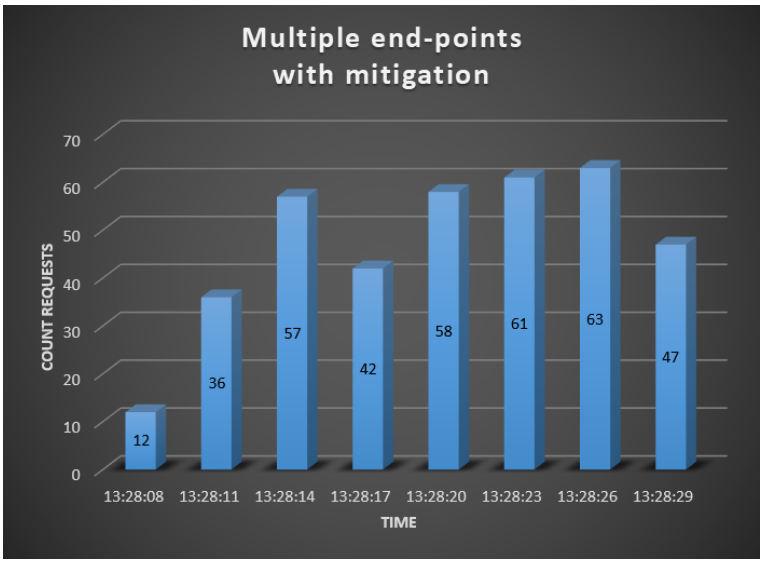


Figure 7.4: DDoS traffic flood attacks with our mitigationn

7.3. Our Innovation - Conclusion

DoS and DDoS protections have existed for many years, and many companies know how to respond to a variety of attacks. Also for Application-Layer (OSI) flood attacks, many mechanisms of companies that specialize in protections for web servers, contains also a protection for these attacks.
However, there is no protection intended only for Application-Layer (OSI) flood attacks, and an attack that can be so simply done (a few lines of code), requires a specific protection without any enveloping system.
Our only preoccupation with Application-Layer (OSI) flood attacks is unparalleled on the net, and no company is dealing only with Application-Layer (OSI) flood problem. In this situation, many web servers may lose the ability to serve customers easily, and only answer thousands of dollars of a large system that will protect the site from attacks that would probably never arrive.

In order to build our system, we successfully meet the following requirements:

- Investigating what is DoS and DDoS attacks, and focusing on Application-Layer - http flood attacks.
- Researching about other existing solutions, learning from other companies which already invented such defenses / algorithms.
- Developing our own servers (regular and CDN), and end-points representing clients or attackers.
- Testing and experimenting our system with several tools - Integration tools and multi - end points servers to demonstrate a real DoS or DDoS attack.
- 'Post Mortem' - Analyzing the results - does our system give a proper solution for web servers?

We note that our system has a few innovations that no other company implemented. It starts with the CDN implementation. Each one of them used their system within the original server. It means that their server should focus both on mitigating any DoS or DDoS attack, and also continue serving its clients properly and all that implies (interpret each request, deal with the databases and to perform complex calculations).
In addition, each one of our related workers considered the amount requests coming from an end-point. Our mitigating algorithm considers both the amount of requests and the specific request. This means that when the request has a minor significance in meaning of server calculations and network traffic, it will get the same treatment in our mitigating algorithm.
(see our calculation file algorithm)

7.4. Developing environment

7.4.1. Git

In order to work in parallel and to maintain all the work we've done, we used git as our version control manager for the project, and GitHub as our git server containing our repository. It helped us to keep well organized work, reverting changes when mistaken and track each other work. (link)

7.4.2. Tests

In order to know that the project is working as expected, we wrote tests for each module, and each functionality of the project.
We used Jasmine as our test runner, a very conventional test runner.
Our project contains 57 tests – 52 unit tests and 5 integration tests.

- Unit tests are covering our utility functions – validators, filesystem requests and programming stuff.
- Integration tests are covering our end-to-end tests: The whole steps since the server is on, CDN server connects to the server, and the client sending requests for them.

The tests are helping us to determine that every time we're pushing new code, the base code is still working as expected. (link)

7.4.3. Lint

Lint is a tool which helps us to manage our code validity, style and efficiency. As most of our project was built by NodeJS, we used ESLint as our Lint rule. Around 30 rules are on for errors or warnings. (link)

7.4.4. CI

Lint and tests are very good, but useless when a developer doesn't run them every time he is writing new code. Continues Integration is obliging us to run our tests and lint rules before any merging our code into master branch. We used CircleCI tool in order to connect our GitHub repository and to run our Jasmine tests and ESLint rules for any new code. (link)

7.4.5. CREATE

CREATE is a state-of-the-art scientific computing facility for cyber-security researchers engaged in research, development, and testing of innovative cyber-security technology. It helped us to simulate the DDoS attacks over multiple end-points. (link)

7.4.6. NodeJS

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code for servers and operation systems. With Node.js we've built our own server and CDN server, and thus managed to determine our desirable behavior for handling packets. (link)

7.4.7. NS

A network simulator is software that predicts the behavior of a computer network. With it we managed to simulate our large network and to make a real DDoS attacks, from several end-points. (link)

7.4.8. Tkinter

Tkinter is a Python open-source library for Graphical User Interface (GUI) software. With it we've built a simple use for our attacks. (link)

8. Future work

We've done a huge research work, and built multiple modules that could handle DoS and DDoS attacks when needed. However, we would like to suggest a few improvements for our system:

1. We would like to handle not only Application-layer DoS and DDoS attacks, but also attacks in transport-layer (Syn / Rst floods), and even on link-layer, dealing with routing tables, as some of our related workers did.

2. Within our algorithm, we would like to add "White List" – which would gain several ips or a whole subnet to ignore from mitigation. Might be effective for companies which would like to give access for internal employees. In addition, "Black List" could be effective too, or to prepare such mechanism if some client would request it.

3. To divide some implementations for web server, desktop servers and cloud servers. It might be effective when some special requests needed in each one of them.

4. To test implementations between node servers (Asynchronous Single Threaded), and other servers (Synchronous Multi-Threaded). What would give the best solution?

5. Future DoS and DDoS attacks – hackers are developing every day, and we need to be ready (and even predict) for future attacks.

Acronyms

APDoS The Advanced Persistent Denial-of-Service. 7, 9

CDN Content Delivery Network. 3, 12, 13, 19, 20, 23–25

DDoS Distributed Denial-of-Service attack. 3–7, 9–13, 17, 20–23, 25, 26, 28

DoS Denial-of-Service attack. 3–5, 7, 9–13, 17, 20, 22, 23, 26, 28

GUI Graphical User Interface. 25

HTTP Hypertext Transfer Protocol. 17

ICMP Internet Control Message Protocol. 6, 7, 9

IP Internet Protocol. 3

IPS Intrusion Prevention System. 9, 10

ISP Internet Service Providerl. 9

MAC Media Access Control. 7

OSI Open Systems Interconnection. 5, 16, 23

SDN Software-Defined Networking. 11

SNMP Simple Network Management Protocol. 10, 12

SSL Secure Sockets Layer. 5, 6

TCP Transmission Control Protocol. 6, 8

UDP User Datagram Protocol. 6

List of Figures

References

[1] Stephanie Weagle. *New Report Points to Alarming DDoS Attack Statistics and Projections*. URL: https://www.corero.com/blog/736-new-report-points-to-alarming-ddos-attack-statistics-and-projections.html.

[2] US-CERT. *DDoS Quick Guide*. URL: https://www.us-cert.gov/sites/default/files/publications/DDoS%20Quick%20Guide.pdf.

[3] GENERAL SECURITY. *Layer 7 DDoS attacks: detection and mitigation*. URL: http://resources.infosecinstitute.com/layer-7-ddos-attacks-detection-mitigation/#gref.

[4] D Holmes. *Mitigating ddos attacks with f5 technology*. URL: http://nl.security.westcon.com/documents/47064/mitigating-ddos-attacks-tech-brief.pdf.

[5] Livio Ricciulli, Patrick Lincoln, and Panka j Kakkar. *TCP SYN Flooding Defense*. URL: http://ai2-s2-pdfs.s3.amazonaws.com/51f2/6c450b44ee8c5ec15945b855e34b863328f4.pdf.

[6] Juho Snellman. *The many ways of handling TCP RST packets*. URL: https://www.snellman.net/blog/archive/2016-02-01-tcp-rst.

[7] CLOUDFLARE. *Ping (ICMP) Flood DDoS Attack*. URL: https://www.cloudflare.com/learning/ddos/ping-icmp-flood-ddos-attack.

[8] radware. *A Clear and Emerging Cyber-Security Threat: APDoS*. URL: https://security.radware.com/ddos-knowledge-center/ddos-attack-types/apdos-emerging-cyber-threat.

[9] Jaehak Yu, Hansung Lee, and and Daihee Park Myung-Sup Kim. "Traffic flooding attack detection with SNMP MIB using SVM". In: *Computer Communications* 31.17 (2008), pp. 4212–4219. DOI: https://doi.org/10.1016/j.comcom.2008.09.018.

[10] Ming Li. "Change trend of averaged Hurst parameter of traffic under DDOS flood attacks". In: *Computers and Security* 25.3 (2006), pp. 213–220. DOI: https://doi.org/10.1016/j.cose.2005.11.007.

[11] Qian Yan et al. "Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges". In: *Communications Surveys and Tutorials* 18.1 (2016), pp. 602–622. DOI: https://doi.org/10.1109/COMST.2015.2487361.

[12] Cisco System. *Defeating DDos attacks*. URL: https://www.cisco.com/c/en/us/products/collateral/security/traffic-anomaly-detector-xt-5600a/prod_white_paper0900aecd8011e927.pdf.

[13] Cheack Point. *DDoS protector appliance*. URL: https://www.checkpoint.com/downloads/product-related/datasheets/ds-ddos-protector-appliances.pdf.