

Understanding DoS protection services

Dissertation presented by
François DEDERICHs , Simon HARDY

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor
Ramin SADRE

Readers
Marc LOBELLE, Fabien DUCHÊNE

Academic year 2015-2016

First of all, we would like to thank our supervisor, Ramin Sadre, for his interesting master's thesis subject, his availability and his useful comments and remarks throughout the entire year.

We would like to thank Olivier Bonaventure for his excellent network courses which helped us a lot during this thesis, and for his availability during the year to answer our questions, together with Fabien Duchêne. Thanks to Alain Ninane and Quentin Hunin too for their explanations about the architecture of the UCL network.

Finally, we would like to thank Marc Lobelle and Fabien Duchêne for having accepted to be readers of our thesis.

Abstract

Among the various existing online attacks, Denial of Service (DoS) is the most problematic one. Thanks to amplification and reflection, coordinated attacks from large botnets can now generate several hundreds of Gbps of attack traffic, becoming a serious problem even for large server infrastructures. Given that traditional network defenses can't handle such attacks anymore, several companies, such as Cloudflare and Akamai, offer services to protect websites and other infrastructures from DoS attacks. But since these are commercial companies, the used technologies are not documented. This master's thesis studies available DoS protection services, analyses their strengths and weaknesses and tries to guess which mechanisms are used to filter legitimate traffic from large floods. Another subject discussed is the potential interest for malicious users to use such protection services in order to cover their illegal activities.

Contents

Introduction	6
1 DoS attacks in a nutshell	8
1.1 Definition of DoS attack	8
1.2 The impact of DoS attacks	8
1.3 Who launches DoS attacks ?	10
1.3.1 Botnets	10
1.3.2 Booter services	12
1.4 Resources affected by a DoS attack	12
1.5 Categorization of DoS attacks	13
1.6 Statistics about attacks	14
1.7 The open reflectors problem	17
1.7.1 DNS reflection	17
1.7.2 IoT-based reflection	19
1.8 Conclusion	19
2 Cloudflare	20
2.1 What is Cloudflare ?	20
2.1.1 Facts about Cloudflare	20
2.2 How does it work ?	21
2.3 DNS	24
2.4 Anycast	24
2.5 Architecture of a data center	25
2.6 Advanced DoS protection service	26
2.6.1 Automatic learning platform	26
2.6.2 Intelligent network protection	27
2.6.3 "I'm under attack" mode	27
2.7 Experiments on Cloudflare external-facing IPs	27
2.8 Automatic detection of data center hit	30
2.8.1 Based on latency	30

2.8.2	Based on <code>traceroute</code>	30
2.8.3	Based on the response to an anycast network	31
3	More about DoS protection services	32
3.1	Akamai	32
3.1.1	What is the Akamai Intelligent Platform ?	32
3.1.2	How does it work ?	33
3.1.3	Always-on security	34
3.1.4	Kona Site DDoS Defender [Akac] [Akaa]	35
3.2	Comparison	36
3.2.1	Scale	37
3.2.2	Setup	37
3.2.3	DNS servers	38
3.2.4	Load-balancing	38
4	General mitigation mechanisms	40
4.1	Network ACLs	40
4.2	Firewalls	40
4.3	IDS & IPS	41
4.4	Manual response	42
4.5	Overprovisioning	43
4.6	Black hole filtering	43
4.7	IP blacklisting	43
4.8	Upstream filtering	44
4.9	Working around the Linux kernel	44
4.10	Conclusion and dedicated DoS defense	45
5	State of the art DoS defenses	47
5.1	Source-based defenses for layer 3/4 attacks	47
5.1.1	Ingress/Egress filtering	47
5.1.2	D-WARD [MPR02][MPR03]	48
5.1.3	MULTOPS and TOPS [GP01] [AGCLS03]	49
5.1.4	MANAnet's Reverse Firewall [Cs316]	50
5.1.5	Conclusion	51
5.2	Destination-based defenses for layer 3/4 attacks	51
5.2.1	Without collaboration of other routers	51
5.2.2	With collaboration of other routers	54
5.2.3	Conclusion	55
5.3	Hybrid (distributed) defenses for layer 3/4 attacks	55
5.3.1	Hybrid packet marking and throttling/filtering mechanisms	55
5.3.2	DEFensive Cooperative Overlay Mesh (DEFCON) [MRR03]	57
5.3.3	Capability-based mechanisms	57
5.3.4	Active Internet Traffic Filtering (AITF) as a filter-based (datagram) mechanism [AC09]	58
5.3.5	Conclusion	59
5.4	Destination-based defenses for layer 7 attacks	59
5.4.1	DDoS-Shield [RSU ⁺ 09]	59
5.4.2	DAT [LC11]	59
5.4.3	Conclusion	60

5.5	Hybrid (distributed) defenses for layer 7 attacks	60
5.5.1	Speak-up [WVB ⁺ 06]	60
5.5.2	Differentiate DDoS flooding bots from humans (e.g. CAPTCHA)	61
5.5.3	Trust Management Helmet [YFLL09]	61
5.5.4	Conclusion	62
5.6	Defenses against specific attacks	62
5.6.1	Reflection-based attacks	62
5.6.2	DNS Amplification attacks	62
5.6.3	SYN flood	63
5.7	Our experiments	64
6	In the shoes of the bad guys	65
6.1	Experimental results	65
6.2	Interpretation	66
6.3	Lifetime of a phishing website under Cloudflare protection	66
6.3.1	Statistical interpretation	67
6.4	Tracking phishing websites under Cloudflare protection	68
7	Conclusion	71
	Bibliography	72
	Appendices	77
A	List of the most famous DDoS attacks	78
A.1	List of the network/transport-level DDoS flooding attacks	78
A.1.1	Flooding type of attacks	78
A.1.2	Protocol exploitation flooding type of attacks	79
A.1.3	Reflection-based flooding type of attacks (a.k.a DrDoS)	80
A.2	List of the Application-level DDoS HTTP flooding attacks	81
A.2.1	Session flooding attacks	81
A.2.2	Request flooding attacks	81
A.2.3	Asymmetric attacks	81
A.2.4	Slow request/response attacks	82
B	The IP spoofing problem	83
B.1	Introduction and motivations	83
B.2	Proposed solution	84
B.2.1	Egress filtering	84
B.2.2	Ingress filtering	85
B.3	Current and future deployment	86
B.4	Prevent source IP spoofing in an enterprise Wi-Fi network	87
B.5	Conclusion and alternatives	87
C	Code	89
C.1	dig_program	89
C.2	Source IP spoofing: Scapy commands	90

Introduction

Denial of Service (DoS) attacks are nothing new : they have been around for decades. However, over the years, with the development and generalization of the Internet, they became more powerful and easier to launch, to the extent that script kiddies (individuals with no computer knowledge) can now easily launch DoS attacks of several tens of Gbps for a few dollars. This is achieved by renting existing botnets or booter services, a practice known as "DDoS-as-a-service". On the other hand, more experienced individuals are able to launch much more powerful attacks, up to several hundreds Gbps of attack traffic. Such attacks can't be handled by traditional on-premise defenses anymore, as they would immediately fill all the bandwidth available and would overwhelm the different networking devices, filling their queues and forcing them to drop most of the received packets, resulting in a denial of service.

In this thesis, we highlight the need for coordinated defenses : mechanisms that allow the victim to communicate with the intermediate network in order to mitigate the attack as close to the sources as possible. Unfortunately, most of these mechanisms are still in the research field due to the difficulty to deploy them as they would require to patch a lot of Internet-connected devices such as routers.

Due to these problems, the solution that most of the companies chose to adopt today is to subscribe to a DoS protection service. At the beginning, those services were just Content Distribution Networks (CDNs) that allowed clients to lower the load on their servers by redirecting the requests to the different CDN data centers, allowing them to cache popular content. However, since all the requests are first redirected to the data centers of the CDN, they can also filter attack content themselves, thus protecting their customers against different types of attacks, and in particular DoS attacks. The problem is that most of the technologies used by those protection services are proprietary. The main goal of this thesis is to study the mechanisms used by those services, and try to guess how they really work, or at least how they should work according to what we learned about state of the art defenses.

We chose to analyse two DoS protection services in details, namely Cloudflare and Akamai. This choice was deliberate : on one hand, Cloudflare is a recently created but fastly growing company, thus communicating a lot to make themselves known, which was really useful to better understand the general functioning of a DoS protection service. On the other hand, Akamai is an older,

well-established company that demonstrated its effectiveness over the years, and that gathered famous clients such as eBay, PayPal, Apple or even Microsoft. We also decided to investigate a bit about Incapsula, another recent DoS protection service that had several interesting features. But before studying them, we ran a little script (`dig_program/dig_alexas.sh`, see C.1) on the well-known Alexa websites ranking in order to have a more concrete idea about the customers of those three services. The results are available in Figure 1 under the form of a cumulated graph. It shows the number of clients of each CDN versus the maximal Alexa rank considered, allowing to see that Cloudflare and Akamai have a similar number of customers among the 10 000 top websites, both way ahead Incapsula, but also that Akamai gathers more famous clients (since the blue curve is higher at the left side of the graph). We can also extrapolate these results and suppose that Cloudflare has a larger number of customers in total, which can among others be explained by the existence of their free plan.

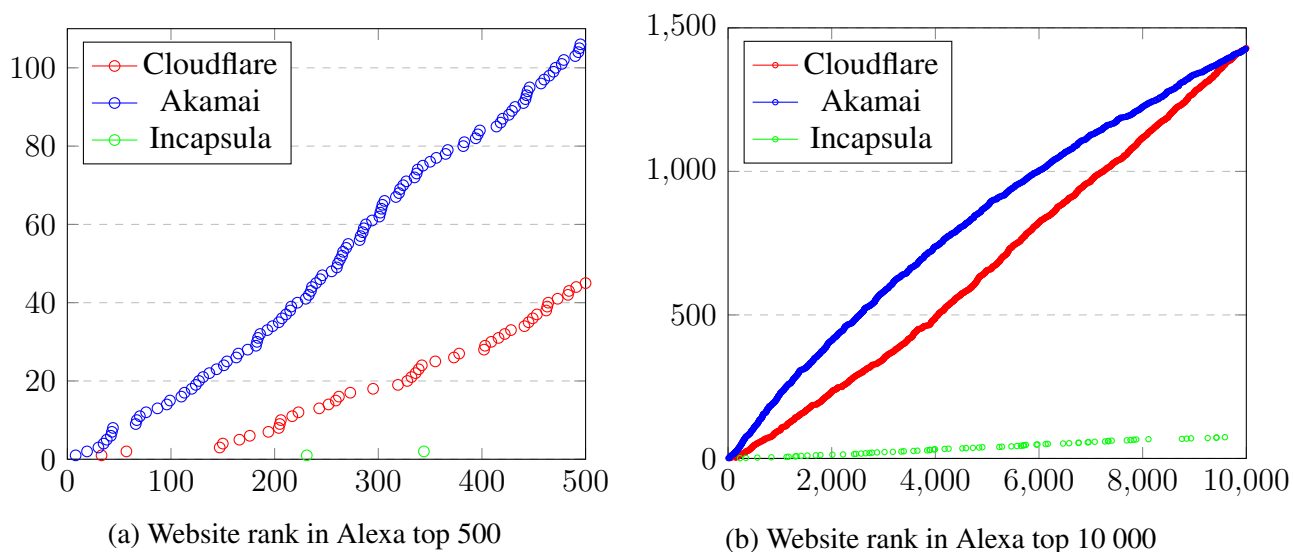


Figure 1: Cumulated number of protected websites from Alexa ranking (for each CDN)

In this report, we will first explain everything that has to be known about DoS attacks : how they work, the different types of attacks, their possible impact and so on (precise descriptions of the attacks are available in Annex A). Then, we will analyse in details the two CDNs mentioned previously, i.e. Cloudflare and Akamai, comparing their features with Incapsula, the third CDN. The next chapter will do a quick overview of general mitigation techniques that are not always adequate to fight against DoS attacks, but still interesting to know. Then, we will have a more technical chapter categorizing and explaining in details state of the art defenses existing in the literature nowadays. Finally, the last chapter will talk about potential misuses of DoS protection services, i.e. what potential interest would criminals have in using such services. The interested reader can also find more informations about the IP spoofing problem in Annex B, as this problem is at the core of the DoS problem.

Note that the different resources associated with this thesis, including the $\text{\LaTeX}$ files, the presentation slides and the different codes we wrote are available on BitBucket under the public repository <https://bitbucket.org/Sihardy/dos-master-thesis>. We will reference these different codes throughout this report: for example, the script `dig_program/dig_alexas.sh` mentioned above is available in the *Source* part of the repository, as the Bash script `dig_alexas.sh` in the folder `dig_program`. This script is also explained in Annex C.

DoS attacks in a nutshell

Before studying DoS defenses, we must understand what is a DoS attack, what are the different forms of attacks, which network resources can be affected by each of them, and what are the most dangerous and the most used attacks. The goal of this chapter is to shed light on these questions. Note that every DoS attack mentioned in this chapter (or anywhere else in this thesis) is categorized and further explained in Annex A.

1.1 Definition of DoS attack

A Denial of Service (DoS) attack is an attempt to make an internet service or resource unavailable to its users by overwhelming the capacity of response of the target. This attack can interrupt the service temporarily or even permanently.

A more precise term is the Distributed Denial of Service (DDoS) attack, where the attack is launched from multiple sources instead of one. This doesn't mean that all the attackers in a DDoS attack are automatically real people; we can have a DDoS attack launched by one single person from thousands or even millions of hosts, but this aspect will be detailed in Section 1.3. So strictly speaking, DoS is a general term and DDoS is a refinement, but in general the two terms will be interchangeable given that most of DoS attacks are distributed nowadays.

1.2 The impact of DoS attacks

At first glance, a DoS attack doesn't seem so harmful: by definition, its only purpose is to make the target service unavailable to its end-users. So why would people bother to launch such attacks, if the only effect on the victim will be to lose some uptime ? There are multiple various reasons, ranging from economical to personal revenge. The 2011 Worldwide Infrastructure Security

Report found that more than 35% of the DDoS attacks were performed for political or ideological motivation, and 31% for nihilism or vandalism motivation. And in the end, the consequences can sometimes be disastrous: frustrated customers, SLAs violated, company reputation tarnished, lost revenue (e.g. from e-commerce transactions or from advertisement), lost productivity, technical staff required to restore a site attacked, increased IT expenses to buy a better protection, litigation costs, legal liabilities (e.g. if your infrastructure was used as a reflector to attack someone else) and so on.

Now let's look at more practical examples. According to estimates from Forrester, IDC and the Yankee Group, a 24-hour outage for a large e-commerce company could cost them around 30 million dollars. And this is not only theoretical: in February 2000, some big websites (including Amazon, Yahoo and eBay) were hit by a series of DDoS attacks, causing huge loss to these companies (the Yankee Group estimated the cumulative loss to 1.2 billion dollars). Another example is Microsoft, that lost around 500 million dollars in January 2001 after having experienced a several days DDoS attack on their website [Sys16]. This shows that the stakes of DoS attacks are bigger than one might think.

Finally, let's look at Figure 1.1, showing the industries most affected by DDoS attacks on Akamai's network [Aka16a]. Not surprisingly, the gaming industry was recently the most hit, by more than 50% of the attack traffic. This can be explained by several factors:

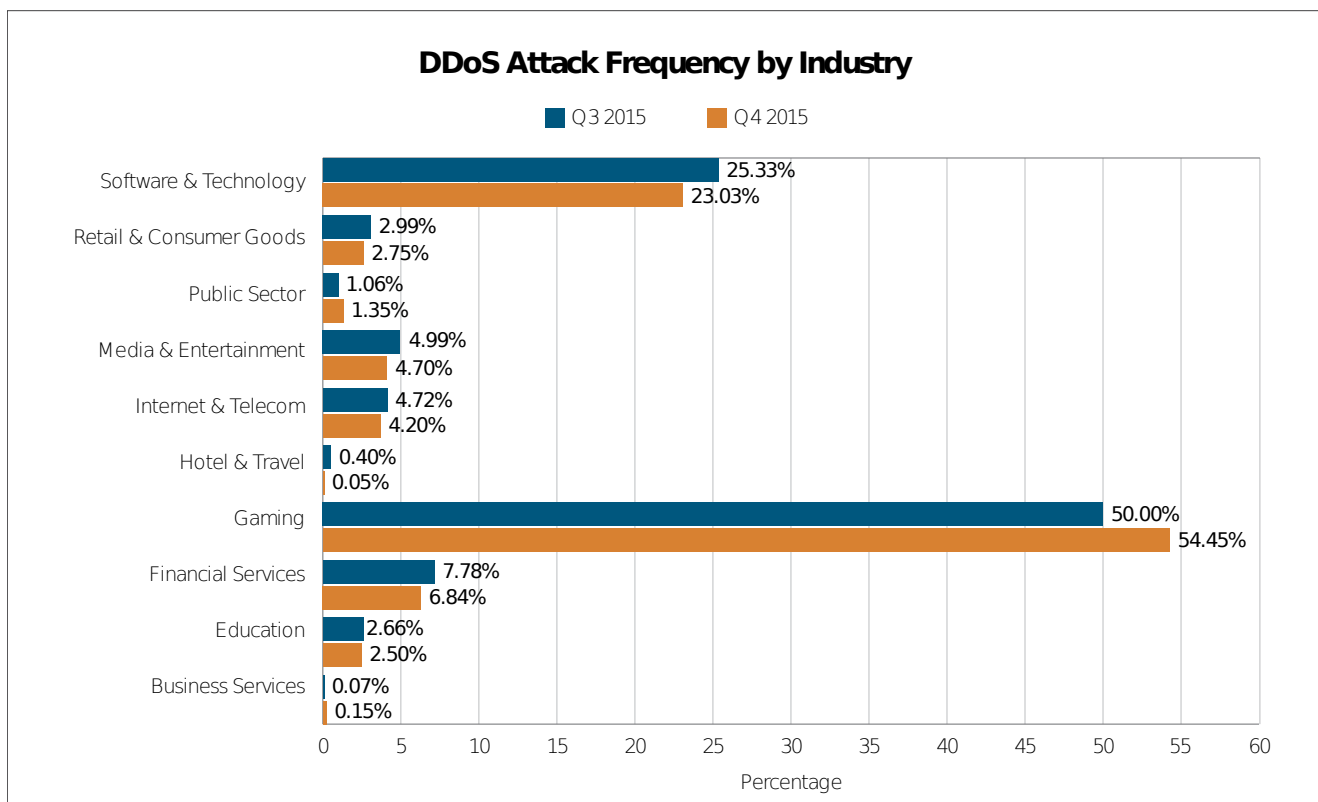


Figure 1.1: Graph from Akamai [Aka16a]

The rise of the video games industry In the recent years, the gaming industry has constantly evolved and expanded, especially the online gaming. Online games are played by millions of people every day. For example, Overwatch, a game developed by Blizzard out since May 24, 2016 has already over 7 million online users [Sep16], and League of Legends, a highly competitive game, has been reported to have more than 7.5 million of concurrent players at peak times [Wik16e]. Because of the online aspect and the need of constant connectivity to play those games, it has introduced a new single point of failure. DDoS attackers can target

the servers that host a game, for example to achieve instant Internet notoriety.

The rise of the betting websites for gaming Online competitive games like Counter strike: Global Offensive (CS:GO) have an entire economy that gravitates around them with what we call "skins". In short, skins are items that you can display in game and that cost real money. Betting websites quickly arose from these and it is now possible to bet your skins on pro matches that happen online or on LAN. When games happen online, a gambler can simply DDoS the winning team in order to make them lose, and in highly skills based games like CS:GO where the time reaction is key, even a slight "lag" can make the difference between a win and a loss.

The rise of the streaming industry (i.e. people streaming themselves while playing video games). To illustrate this, we can cite the famous streaming website `twitch.tv`, which was acquired by Amazon in 2014 for 970 million dollars. As streamers can make a lot of money, and often play competitive games, they are a target of choice for attackers: they can even watch the effect of the attack directly on the stream.

The use VoIP programs Skype is a VoIP program based on the peer-to-peer model. In consequence, the attacker can retrieve any Skype user's IP address given that he knows the Skype pseudonym of the victim and that the victim is connected on Skype. Since most gamers use Skype, this program is a common way to DDoS individuals such as streamers whose Skype pseudonyms would have leaked. This means that attackers can also target individual streamers or competitive players in addition to gaming servers.

1.3 Who launches DoS attacks ?

Given the resources (among others, the upstream bandwidth) necessary to perpetrate an average DoS attack, you can't really launch it from your home computer : you will need to use other machines to launch your attack. Currently, there are two techniques often used to launch big DDoS attacks : the botnet, and the booter/stresser tool. Note that in both cases, the current trend is "DDoS-as-a-service", meaning that you don't have to create your own botnet or booter tool; you can simply purchase the service, either for a one-time attack, or subscribe for repeated attacks during for example one month.

1.3.1 Botnets

A botnet (literally, combination of *robot* and *network*) is a set of hijacked Internet-connected devices (servers, computers, routers, mobile phones and so on) controlled remotely by the "bot master" without the knowledge of the devices' real owners. Historically, botnets have been used for different purposes such as sending spam emails or mining bitcoin, but more recently botnets are used a lot for performing DDoS attacks. There are two general architectures for a botnet: the client-server model, and the peer-to-peer model.

The client-server model is the architecture that appeared with the first types of botnets. Figure 1.2 is a Stacheldraht botnet diagram which perfectly represents a client-server model for a botnet. As you can see on this diagram, the attacker is in control of a few machines, the C&Cs (Command and Control, also called "handlers") and those machines are controlling infected computers, the zombies, that are on the same network. Communication between those entities can be done

through Internet relay Chat (IRC) or HTTP websites with domain generation algorithms creating new domains permanently. From there, a massive coordinated attack against a target victim can be performed. Nowadays, the client-server model isn't that popular anymore since it became too vulnerable and bot masters have moved towards the peer-to-peer architecture to lower the chances to be taken down.

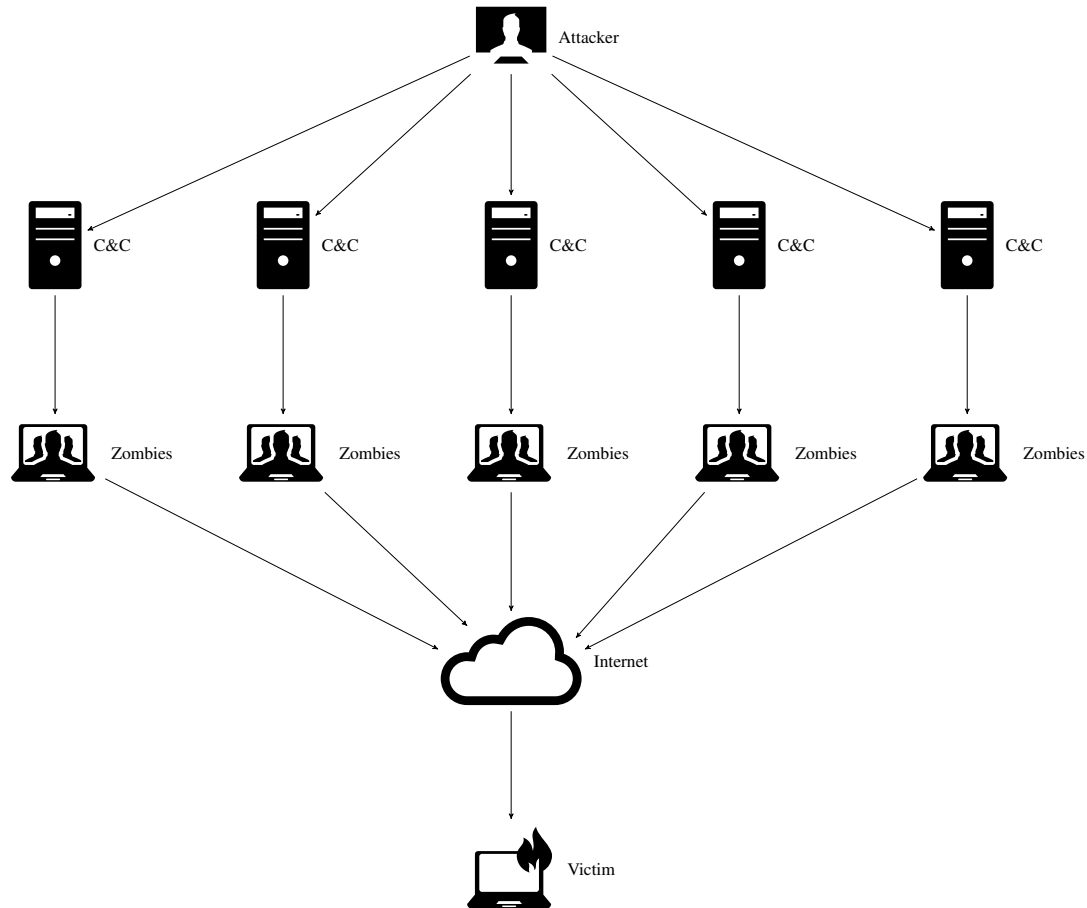


Figure 1.2: Stacheldraht botnet diagram

In the peer-to-peer model, the C&C is incorporated within the botnet itself, which means that attackers don't need to rely on external servers that can be shut down over time. Some of the P2P botnets use asymmetric cryptography to encrypt all the data. The Commanders can be identified using a cryptographic key, they will encrypt all the data with a public key and the bots will decrypt them with their own private keys. In this model, bots only know a list of peers where they can send commands, usually a 256 peers list. This allows to quickly communicate commands from one peer to another.

To end this subsection, we can cite some famous botnets in order to have a general idea of their scope. Botnets are usually named after the malware used to infect the "zombie" machines. For example, the XOR botnet is named after the Trojan malware *XOR DDoS* used to build it. It was recently taken down by the authorities in Asia, but it was made of more than one million Linux hosts, being able to launch more than 150 Gbps of attack traffic [Aka16a]. Another more famous botnet is *Zeus*, considered as the largest botnet that ever existed on the Internet, consisting of several million hosts, and being pretty sophisticated, having moved from the client-server model (where it was generating more than ten thousand domains per day for communication) to the peer-to-peer model.

1.3.2 Booter services

The alternative, easier way is to use booter/stresser tools. The advantage is that you don't have to build an entire botnet to perpetrate your attacks : you just need to rent some high-bandwidth servers. Those tools are theoretically used by administrators to load test their own servers (under the name of *IP stresser tools*), but in practice a lot of them are actually used to launch DoS attacks (hence the name *booters*). In that case, the attacks are often reflection-based so that it's harder for the victims to go back to the server. As noted earlier, booters usually use the "DDoS-as-a-service" model, and are often subscription-based. Figure 1.3 shows typical pricing plans for such services.

MONTHLY BRONZE	DOUBLE SILVER	V.I.P IRON BEST
19.99\$	49.99\$	599.99\$
⊗ XyZ Public Network	⊗ XyZ Public Network	⊗ XyZ VIP Network
⊗ 250Gbps Network Capacity	⊗ 250Gbps Network Capacity	⊗ 350-400Gbps Network Capacity
⊗ 27 Attack Methods	⊗ 27 Attack Methods	⊗ 27 Attack Methods
⊗ 1800(s) Stress Time Per Attack	⊗ 6000(s) Stress Time Per Attack	⊗ 12000(s) Stress Time Per Attack
⊗ 1 Months Membership	⊗ 2 Months Membership	⊗ 4 Months Membership
⊗ 1 Concurrent attacks	⊗ 1 Concurrent attacks	⊗ 2 Concurrent attacks
REGISTER	REGISTER	REGISTER

Figure 1.3: Examples of pricing plans for a booter service (booter.xyz)

1.4 Resources affected by a DoS attack

Different network resources can be affected by a DoS attack. Some attacks will target one specific resource, while other attacks will affect several ones at the same time. Here is a list of resources usually targeted by DoS attacks :

- Bandwidth of the link between the victim and its provider : requires a huge amount of large packets to fill the bandwidth more easily.
- Router or server processing capacity + firewall : here, the attacker will favour small packets, and the quantity of packets required will depend on the attack.
- Disk, database or more generally IO resources : a well-known example is sending a lot of heavy-processing SQL queries in order to make the database unavailable to other users (this attack can also affect the CPU of the server, or even the bandwidth, depending on the queries).
- Memory : some attacks will aim at filling the entire memory, but you can often make the server unresponsive to genuine requests by targeting specific portions of the memory. For example, you can try to open enough TCP connections (with a SYN flood attack) in order to fill the SYN queue with enough entries, preventing the server from opening new connections with other users. Some other attacks can also try to disrupt state informations already present on the server (for example with TCP RST packets).

- Server logical resources : alternatively, you can try to exhaust logical resources such as the thread pool of the server with slow request/response attacks (explained in A.2.4).
- Less common, attacks can target specific configurations, such as routing informations of a router.
- Vulnerability attacks can also directly target a program flaw to confuse a service or make it crash, or even target a physical component.

1.5 Categorization of DoS attacks

There are several ways to categorize DoS attacks. In this thesis, we chose the following categorization, which is a mix of several sources found on the web. But for example, [MR04] proposes a more technical comparison.

1. Vulnerability attacks
2. Flooding attacks
 - Network/transport-level attacks
 - Direct flooding attacks
 - Protocol exploitation attacks
 - Reflection-based attacks
 - Application-level attacks
 - HTTP attacks

Let's have a look at each category. *Vulnerability attacks* consist in general in sending a malformed packet to the victim in order to confuse some particular protocol or application. We won't develop them in this thesis as they are not very common and in general easily manageable with destination-based mechanisms only. We can still cite the most notable, such as Ping of Death (which consists in sending malformed ping packets) or the Teardrop attack (which uses fragmented IP packets that are overlapping and over-sized).

Flooding attacks are the main focus of this thesis, as it's almost impossible to defend against such attacks with only destination-based mechanisms : DoS protection services will be required to handle the toughest ones.

Among *flooding attacks*, we will distinguish two big subcategories : attacks on the network or transport layer of the OSI model (layer 3 or 4), and attacks on the application layer (layer 7). In general, *network/transport-level attacks* are characterized by a huge number of either large packets (in order to overwhelm the victim's bandwidth) or smaller packets (targeting for example the processing capacity of routers/servers).

Application-level attacks are in more stealthy (they look more like normal traffic), they consume less bandwidth and they will in general target one particular resource of the victim server (for example, the thread pool of a web server).

1.6 Statistics about attacks

Although Annex A describes a lot of DoS attacks, it's impossible to list all of them because they are too numerous, some of them are obsolete, and new ones appear all the time. In consequence, in order to build a good DoS protection service, we have to focus on the most frequent attacks, which represent the most easy to launch and/or the most damaging ones. Figure 1.4 illustrates the frequency of the different DoS attacks experienced by Akamai during the last quarter of 2015 [Aka16a]. It should be representative of the real trends in DoS attacks nowadays due to the large network managed by Akamai (they handle between 15 and 30% of the total network traffic).

The first observation that we can make is that network/transport-level attacks represent almost 97% of the attacks. The main cause is that application-level attacks require a successful TCP handshake, so source IPs can't be spoofed : this could reduce the life expectancy of the botnet/server hosting the booter/stresser tool, so they must have opted for network/transport-level attacks only, that are anyway easier to launch.

A second observation is that, among the 24 attack vectors displayed, four of them (UDP Fragment, NTP, SYN and DNS) represent almost 60% of all attacks. This illustrates the fact that, even if a lot of attacks are imaginable, a few of them only are really effective and easy to set up. In consequence, the different tools available on the Internet will in general only implement those attacks, which is the reason why we see such a big difference with other attacks. In consequence, a good DoS defense mechanism should always make sure to be optimized enough to be able to handle such attacks.

Another interesting observation is that a significant part of those attacks are reflection-based (among others, CHARGEN, DNS, NTP and SSDP, i.e. more than 30% even if we don't consider UDP fragments). This is due to the fact that reflection-based attacks naturally hide the source of the attack¹, and allow an amplification phenomenon. In consequence, most of the stresser/booter tools only allow reflection-based attacks, which explains their predominance in the graph. Note that UDP fragments are often the consequence of reflection attacks, due to sometimes huge amplification factors creating large packets that have to be fragmented to reach the destination.

In Figure 1.5, you can see a history of predominant attacks during the past five quarters. It's interesting to see that these attack vectors seem to have a cyclic nature : when defense mechanisms begin to get well used to one vector, attackers seem to switch to another, and maybe use the old one again later. Reflection-based attacks can also vary in function of the number of available public reflectors. We could have guessed that after years, those attacks would lower due to the diminishing number of reflectors available, but it seems that they (CHARGEN, DNS, NTP) have actually increased a lot. SSDP is an emerging reflection-based attack that was used a lot in the past quarters of 2015. It seems that it lowered a bit in the last quarter, but we can expect it to emerge again in 2016.

¹Compared to random source IP spoofing, reflection makes it even harder for authorities to traceback your identity because they have to go from the victim to the reflector, and then from the reflector to you. Moreover, some reflectors could not keep logs or be located in foreign countries, making the traceback even harder.

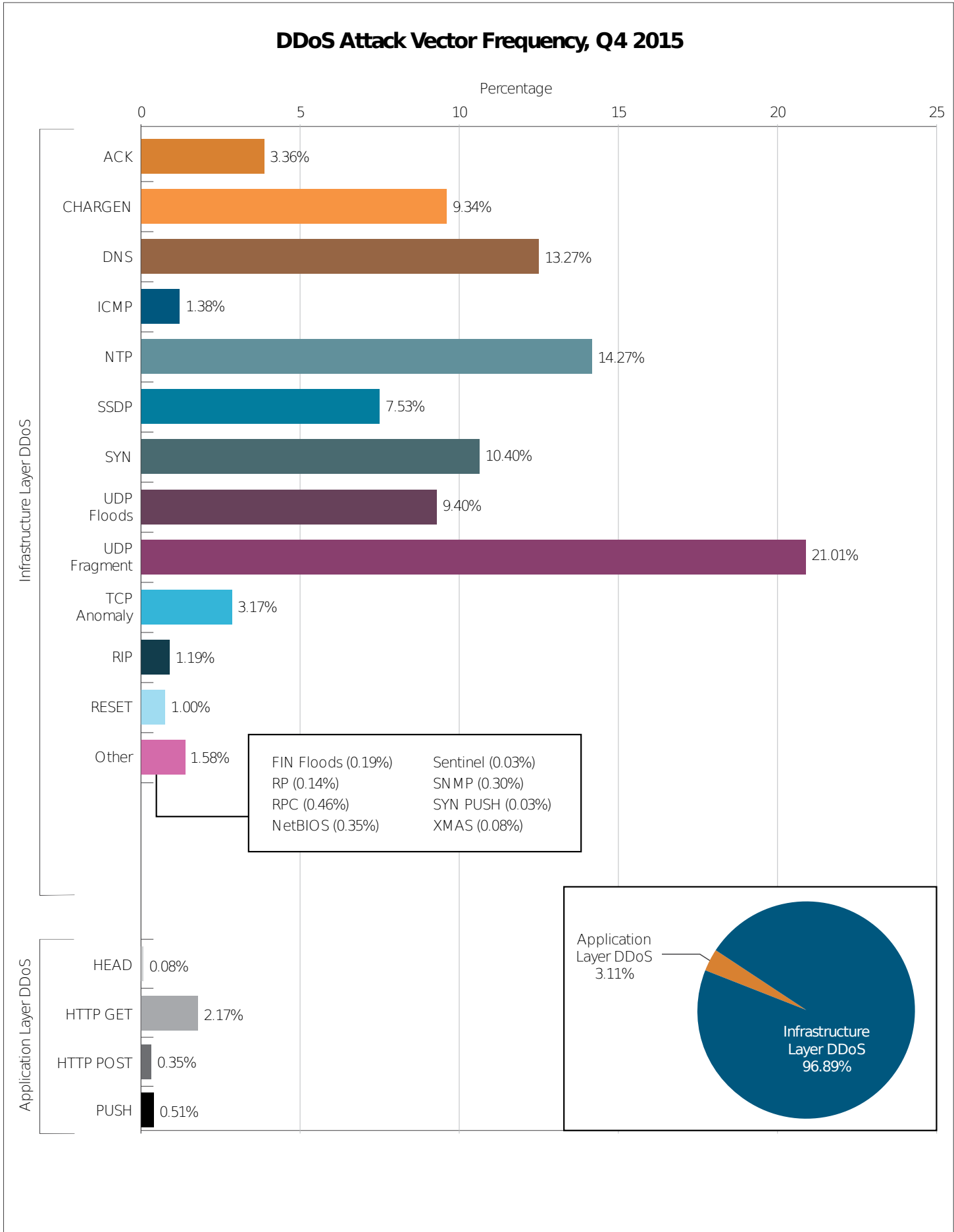


Figure 1.4: DDoS attack vector frequency (graph from [Aka16a])

10 Most Frequent Attack Vectors by Quarter

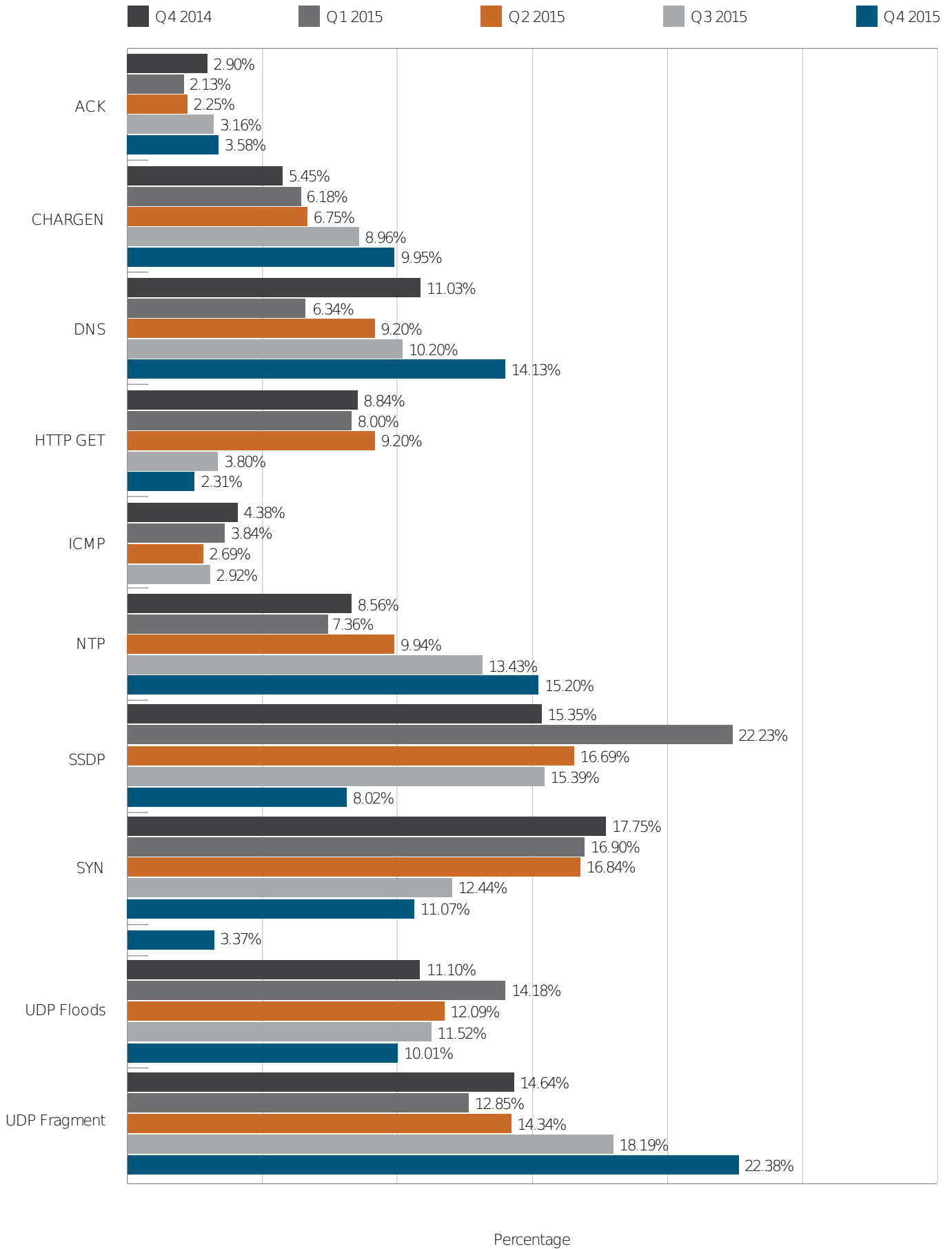


Figure 1.5: Ten most frequent attack vectors by quarter (graph from Akamai [Aka16a])

1.7 The open reflectors problem

As we previously saw, reflection is at the core of the denial of service problem. Reflection-based attacks represent the major part of DoS attacks, and for two main reasons. First, it allows to amplify the traffic, i.e. to create more traffic at the victim for a fixed amount of upstream bandwidth available at the source(s) of the attack. Secondly, it makes it more complicated for the victim or the authorities to identify the source(s) of the attack, as we explained in the previous section.

However, by definition, reflection requires available reflectors, so we could wonder how such huge reflection-based attacks are made possible ? Actually, open reflectors are everywhere, as we can see in Figure 1.6, showing the locations of vulnerable devices used in reflection-based attacks during Q4 2015 (based on logs of Akamai identifying more than 525,850 reflectors).

We can see that they are concentrated in the US, Asia and Europe, mirroring major population centers. Figure 1.7 shows the same information, but divided by ASN. In the remaining of this section, we will take two particular reflectors as examples : DNS reflectors, which have been used for a long time to launch DoS attacks, and IoT-based reflectors, which are emerging reflectors based on the Internet of Things. For more informations about reflection-based attacks, see A.1.3.

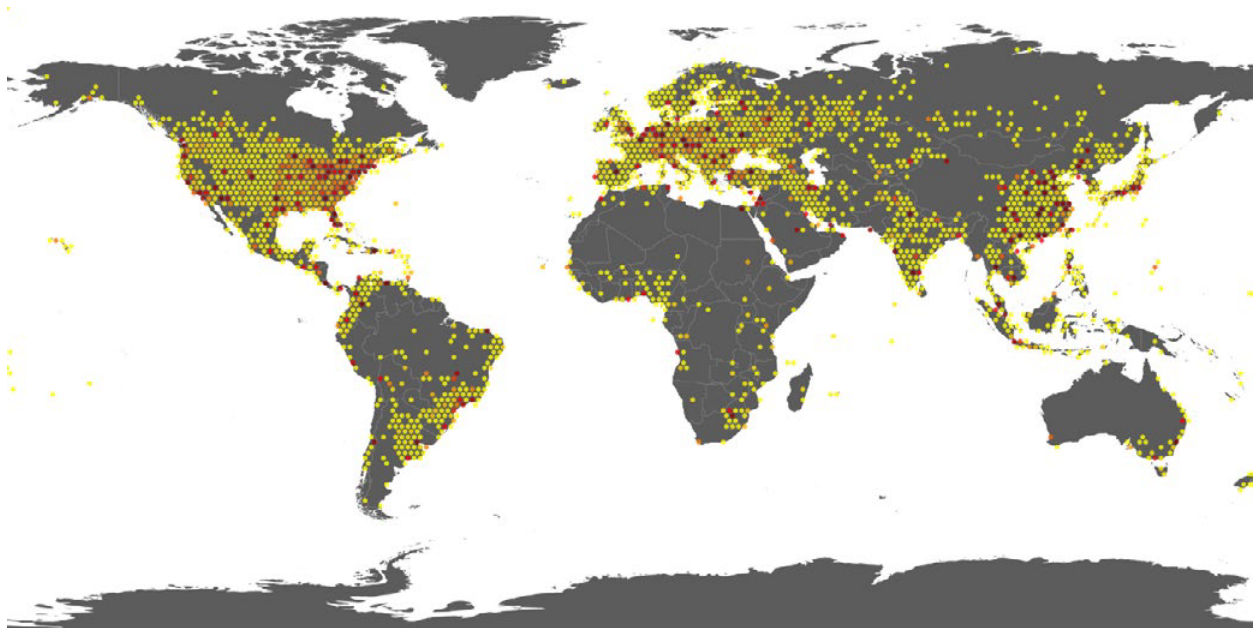


Figure 1.6: DDoS Reflector Heat Map (from Akamai [Aka16a]). Note that the reflectors considered here are ssdp, ntp, chargen, qotd, Sentinel and rpc protocols (no dns).

1.7.1 DNS reflection

While authoritative DNS servers can be used to launch DNS reflection attacks, the main problem comes from open resolvers (a.k.a. recursive DNS servers). As a reminder, resolvers are devices that allow you to fully translate a domain name into an IP address by recursively querying authoritative DNS servers. There are two types of resolvers : those who are operated by ISPs, and those who are operated by large companies such as OpenDNS or GoogleDNS. While the second category is by definition open to the world (anyone can do a `dig` request to one of them), in general they won't cause much problems because they have intelligent rate limiting setup in

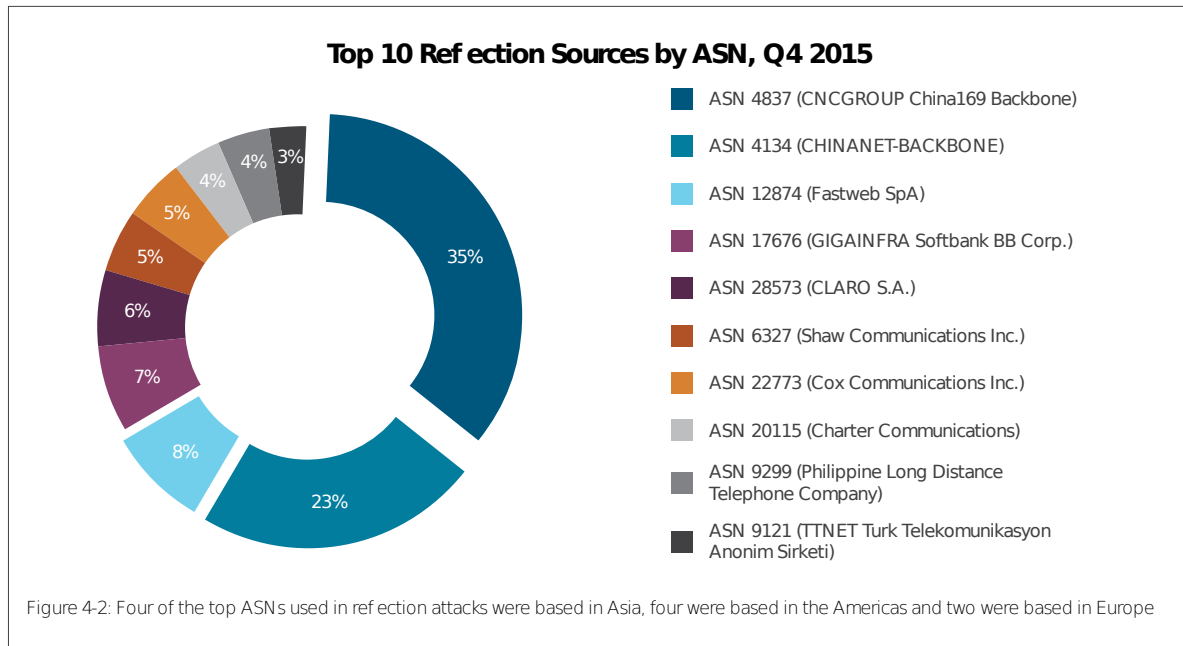


Figure 1.7: Graph from Akamai [Aka16a], and again for ssdp, ntp, chargen, qotd, Sentinel and rpc (no dns).

order to prevent abuse (for example, if you do several `dig any` requests in a row, they will stop answering you).

The major problems come with the first category : resolvers operated by ISPs. Theoretically, a resolver operated by an ISP should only answer requests coming from clients of the ISP running it (and ISPs have no interest of doing otherwise). But in practice, a lot of ISPs run misconfigured, "open" DNS resolvers that will answer requests from anywhere, without rate-limiting or other security mechanisms, then allowing DNS reflection attacks with large amplification factors.

According to the Measurement Factory and the Open Resolver Project, at the time of this writing, 17,624,678 open resolvers still exist on the Internet and answer queries. As you can see in Figure 1.8, this is better, but still unacceptably very high compared to the 27,904,057 that were available in the beginning of 2013.

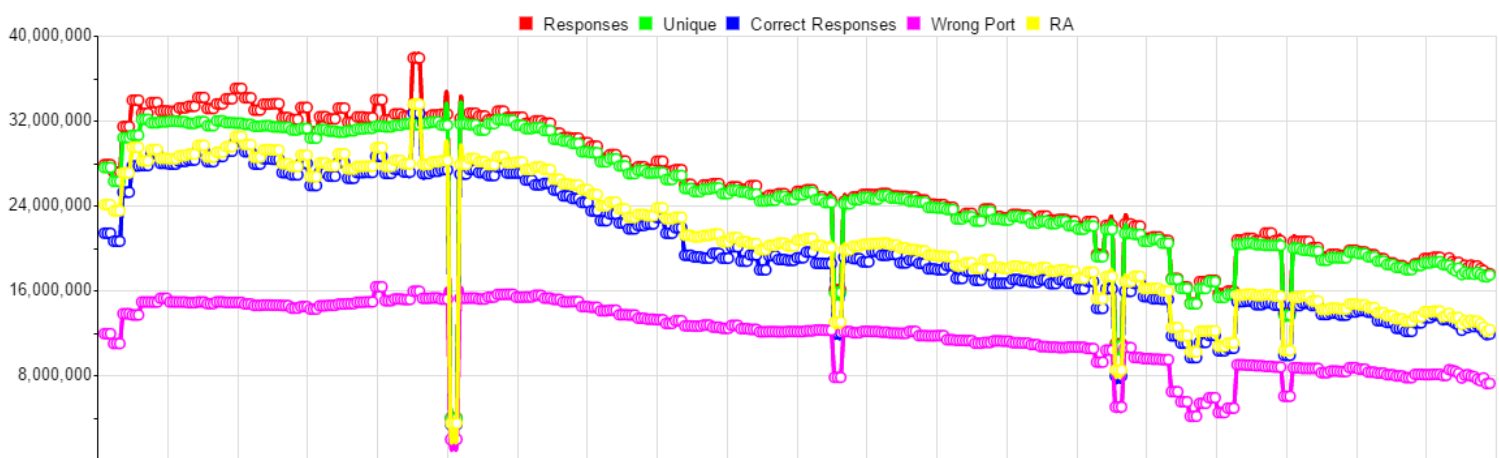


Figure 1.8: Number of open DNS resolvers from 2013-03-24 to 2016-05-22 (graph from the Open Resolver Project)

Moreover, these open resolvers typically run on big servers with fat pipes, allowing attackers to amplify way more traffic than they need to overwhelm most of their victims².

As a final note, we can mention the fact that Cloudflare recently submitted an Internet Draft to the IETF³ proposing to stop answering ANY queries in order to limit the potential amplification factor of such requests.

1.7.2 IoT-based reflection

With the rise of the Internet of Things, more and more devices are connecting to the Internet, often with new custom protocols that were not built with security in mind. In consequence, it's frequent to find small vulnerabilities that, once used on a lot of devices at the same time, can have disastrous effects.

A first example of IoT-based reflection attacks is the SSDP attack, described in A.1.3. It's an attack that emerged some years ago. Unlike most reflection attacks (dns, ntp, chargen, qotd) that are generally perpetrated via servers, the SSDP attack is based on home connections, and is made possible because of a vulnerability in some home routers. According to the Open SSDP Scanning Project from the Shadowserver Foundation, there are currently (2016-05-26) 9,572,379 distinct IPs reflecting SSDP probes. Among those, 3,525,954 come from China, that is to say more than one third of them. This is a good place to remind you that when you look at statistics about sources of DoS attacks, you must always take a step back and understand that an attack coming from China doesn't mean that the attacker is himself in China : maybe he is just using reflectors located in China, such as those exposed here.

Another example of IoT-based attacks (thought not reflection-based) was recently perpetrated by using CCTV (Closed-Circuit TeleVision) [Inc16b]. Attackers were able to create a botnet of 900 CCTV cameras spread around the world by simply using default factory credentials, and used them to generate an attack peaking at 20,000 requests per second. This again highlights the fact that security should be a priority when designing new IoT devices.

1.8 Conclusion

In this chapter, we highlighted the potential danger of DoS attacks, and we tried to raise awareness about the necessity for sensible companies - even small ones - to protect themselves against such attacks. We saw that DoS attacks are more and more easy to launch : with the recent trend of "DDoS-as-a-service", you can now launch DDoS attacks for a few dollars, even if you don't have any computer knowledge. This means that a few dollars spent in an attack could make some unprotected websites lose several hours of uptime. And we saw that for big websites, losing one hour of uptime can result in a loss of up to one million dollars. In conclusion, for those websites - and even for smaller ones -, designing or buying protection against DoS attacks is inevitable: the ROI (Return On Investment) is compelling, even if it means subscribing to a DoS protection service costing several hundreds of dollars every month.

²A table showing the number of known open resolvers for each AS can be seen at <http://dns.measurement-factory.com/surveys/openresolvers/ASN-reports/latest.html>

³Available at <https://tools.ietf.org/html/draft-jabley-dnsop-refuse-any-00>

Cloudflare

In this chapter, we will analyse in details a particular DoS protection service : Cloudflare. This will give us the intuition of how such a service should look like, and provide us with a comparison point for the next chapter, analysing other DoS protection services (especially Akamai).

2.1 What is Cloudflare ?

Cloudflare is an American company based in San Francisco, but it also has offices in London and Singapore. The company was founded in 2009 by Matthew Prince (CEO), Lee Holloway and Michelle Zatlyn [Wik16a]. The main function of CloudFlare is to provide a *content delivery network* (CDN) to its customers, i.e. a distributed system of proxy servers deployed in multiple data centers whose goal is to serve content from their customers to end-users with good performance.

The main goals of Cloudflare are to improve the performances and the availability of its customers' websites, while also protecting them from a large range of attacks. As we will see later, Cloudflare also provides distributed DNS services, which is a crucial point in the DoS protection. At the time of this writing, the company has 79 data centers (or "points of presence") worldwide, as you can see on their network map¹.

2.1.1 Facts about Cloudflare

Cloudflare is a young company, but they were able to gather a lot of big customers over the years. Among others, we can cite the news networking service Reddit, the multinational technology company Cisco, the international song competition Eurovision, the game League of legends or

¹You can consult it at <https://www.cloudflare.com/network-map/>

also the metal band Metallica².

These past years, Cloudflare faced some huge DDoS attacks. The most famous one is probably the attack against the spam-fighting group Spamhaus in 2013 [Clo16d] [Clo16e], that got attacked by the ISP CyberBunker because Spamhaus had decided to blacklist them. It was a DNS amplification attack (explained in A.1.3), that used reflection on over 30 000 open DNS resolvers to generate a final traffic of around 300 Gbps. That's huge, but unfortunately this kind of attacks happens a lot. However, it could be easily avoided by solving the source IP spoofing problem (see Annex B) and/or the open DNS resolvers problem already explained in Subsection 1.7.1.

Another example of huge attack happened in 2014 against a confidential customer of Cloudflare. This attack used NTP reflection (explained in A.1.3 too) and generated around 400 Gbps of attack traffic. This attack also hit OVH, who recorded 350 Gbps of attack traffic [Gig16].

2.2 How does it work ?

Before going any further, let's clarify some notions and illustrate them with an example : let's say that Bob has a website `example.com` currently hosted in his garage, together with his own DNS server. He uses the resolver of his ISP.

Hosting server It's the server hosting the website. It can belong to a web hosting provider, or to a private that bought the hardware himself. It's totally independent of Cloudflare. In our example, this server is located in Bob's garage.

DNS server (or name server) It's a server answering DNS queries for a given DNS zone for which it's authoritative. It can resolve domains names (i.e. translate them into IP addresses) for its zone, or give the IP address of the authoritative name server corresponding to the request. In our example, Bob bought his own DNS server, but in practice you can also use a DNS hosting service, sometimes provided by your web hosting provider.

DNS zone It's a portion of the domain name space for which a particular DNS server is responsible, "authoritative". In our example, the DNS zone for which Bob's DNS server is responsible is `example.com`, which contains for example `www.example.com`, `blog.example.com` and so on.

Root name server It's a DNS server responsible for the root zone.

Top-level name server It's a DNS server responsible for one of the top-level domains (TLD), for example `be`, `nl`, `com` and so on.

DNS resolver (or recursive DNS server) It's a server responsible for the full resolution of a domain name. In other words, you give it a URL, for example `blog.example.com`, and it will do the recursive resolution, i.e. query the root name server for the IP address of the top-level name server responsible for `com`, then it will query this IP address for the IP address of the name server responsible for `example.com`, and finally it will query this IP address for the IP address of `blog.example.com`. Note that resolvers can cache content, allowing to skip some of those steps. In general, by default you use a resolver of your ISP (it's the case of Bob), but you can also use another one, for example the well-known resolver of Google, available under the IPs 8.8.8.8 or 8.8.4.4. Some problems can arise when recursive resolvers are open to the Internet, as we saw in Subsection 1.7.1.

²A larger list, together with customer stories, can be found at <https://www.cloudflare.com/customers/>

To ensure that everything is clear, let's go through a small scenario (assuming there is no cached content). Let's say that an attacker enters `blog.example.com` in his browser. The flow is pretty straightforward : the attacker's DNS resolver will do the full resolution (using a root name server, a top-level name server and finally Bob's name server), and then give the IP address to the attacker. Then, a TCP session will be created with Bob's hosting server, that will finally send the needed HTTP packets and other optional stuff to the attacker. The flow can be seen in Figure 2.1

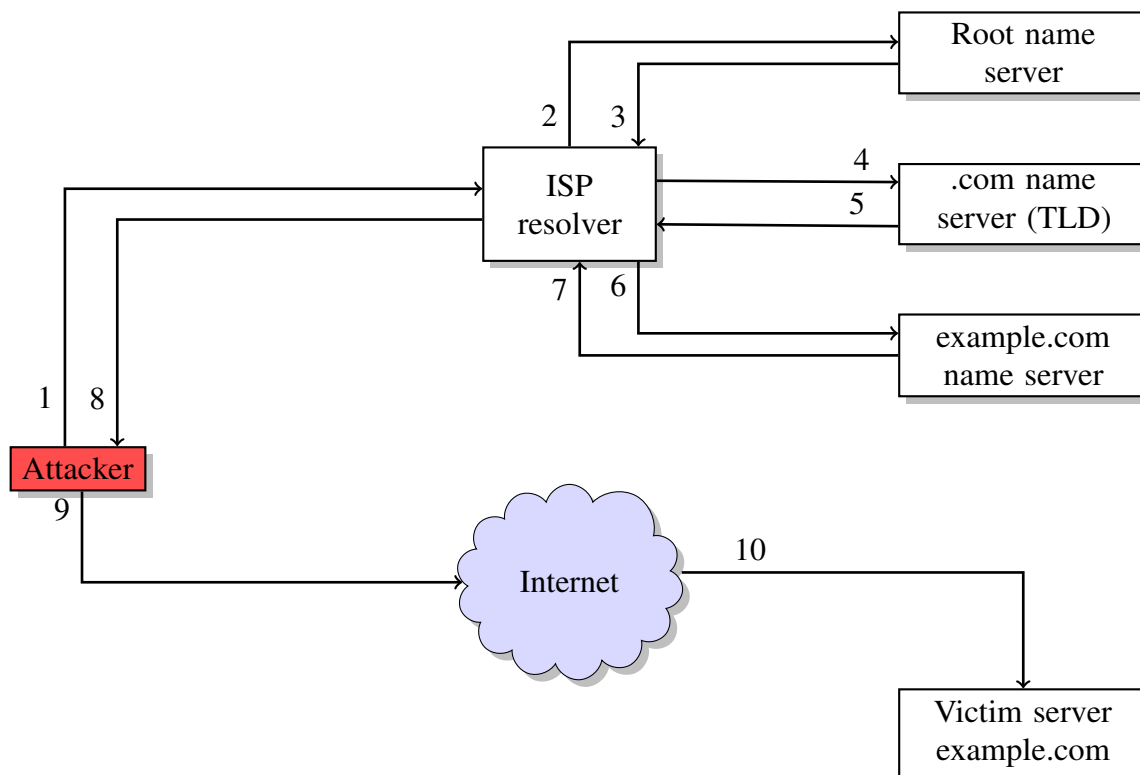


Figure 2.1: Flow without Cloudflare protection

Notice that now, the attacker has the IP address of Bob's hosting server : Bob is now vulnerable to direct-to-IP DoS attacks, since the attacker can reach Bob directly whenever he wants.

Later on, Bob decides to become a customer of Cloudflare. To subscribe, Bob has only one thing to do : give Cloudflare *authority* for the *DNS zone* corresponding to `example.com` (see next section for more details). Let's see what changed in the previous scenario (see Figure 2.2) : the attacker enters `blog.example.com` in his browser. His resolver will again do the full resolution, but this time, the top-level name server (corresponding to `com`) will tell him that `example.com` is managed by a Cloudflare name server (step 5). His resolver will then contact this name server, which will tell him that `blog.example.com` has a Cloudflare IP address (step 7). This IP address will redirect him to a Cloudflare data center (the closest one, see Section 2.4) for every subsequent request made to Bob's website (step 9). In other terms, the attacker will never get any information about Bob's hosting server (and especially, he won't be able to retrieve the IP address of Bob's server). The data center will do all the needed requests to Bob's hosting server itself (step 10). But the good point here is that the data center can also cache that content, so that each subsequent request for the same content (by the attacker or anyone else) to that data center will be served to the end-user directly, without doing a single request to Bob's hosting server, saving resources to Bob and making the response way faster for the end-user. Indeed, if

the attacker lives in Belgium and Bob in America, his requests will hit the closest Cloudflare data center, for example the one in Amsterdam : if the content is in cache, the resources saved are consequent.

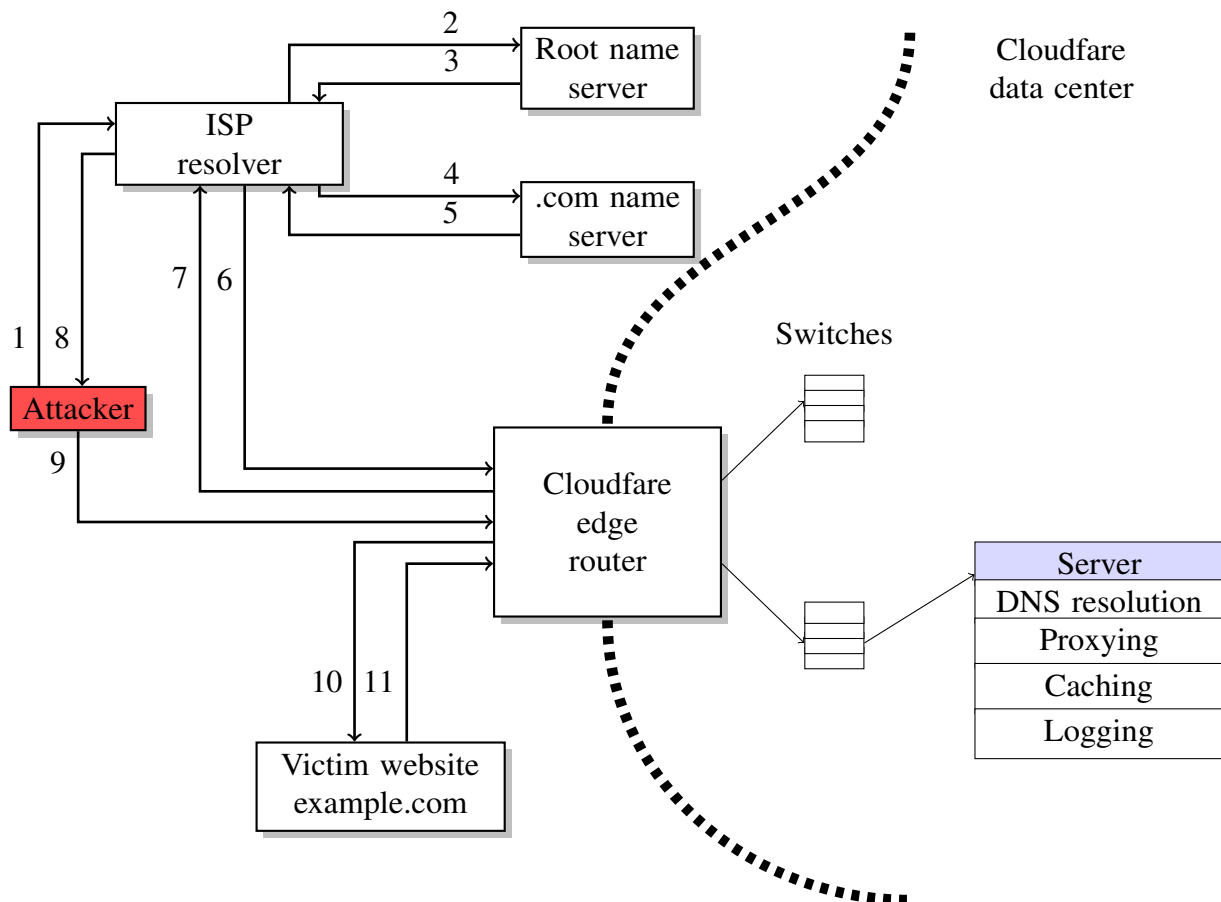


Figure 2.2: Flow with Cloudflare protection

So in conclusion, this "reverse proxy" architecture will allow Cloudflare to bring the following to subscribed websites:

Performance increase Thanks to caching, there will be less load on the original hosting server, which means better performances. Moreover, this architecture also allows Cloudflare to easily provide other services such as website optimization.

Protection Since all the traffic must go through Cloudflare before hitting the original hosting server, Cloudflare can filter most attacks before forwarding traffic. Moreover, the original hosting server is automatically protected against direct-to-IP DDoS attacks.

Availability The original hosting server will be less likely to be overwhelmed by genuine traffic (thanks to performance increase) and by attack traffic (see protection), and even if the original hosting server runs into problems, Cloudflare can sometimes keep the website alive by serving cached content to end-users.

Finally, one could say that in the absence of caching (for example if the website contains a lot of dynamic content), there will actually be a performance decrease due to traffic having to make a detour to a Cloudflare data center before hitting the original hosting server. This can be true, but we must also take into account that:

- Cloudflare has a lot of data centers, so the detour will in general be short;
- The data centers are often directly connected to an IX point, which can be reached by ISPs really fast;
- When the original hosting server is far away from the end-user, the detour can sometimes actually improve the latency due to better, direct routes to the destination provided by Cloudflare.

2.3 DNS

As we said previously, Cloudflare setup consists in giving Cloudflare *authority* for the *DNS zone* corresponding to our protected website. In practice, this is done by replacing our own DNS server by two name servers provided by Cloudflare. This means that Cloudflare becomes our new DNS provider (our old DNS server becomes unused), but we keep our resolver, hosting provider and registrar. As a final step, they need to have a copy of the DNS records of our old DNS server. In order to make it automatic³, they were pretty inventive and decided to implement a small utility that will scan our old DNS server for common subdomains (like `www`, `blog`, `forum` and so on) in a "brute force" way in order to get most informations very quickly (the scan is said to last 60 seconds). Then, later, we can still decide to remove or add some subdomains through the GUI⁴.

However, one year ago, Cloudflare decided to enable Virtual DNS [Clo16g]. The principle is similar to the approach explained in the previous paragraph, except that they will proxy our own DNS server instead of replacing it. This means that we don't need to bother with a scan or with manual updates on the GUI : everything will still be managed as before Cloudflare, and DNS queries will be redirected to our own DNS server after having been filtered / cached by them. Notice that this is also an option proposed by Akamai, and it will be described in more details in the next chapter, with some additional features proper to Akamai.

2.4 Anycast

We saw that requests to a website protected by Cloudflare are directed towards the closest data center. This can be done by several ways, often different from one CDN to the other, but the way Cloudflare chose (and which is probably the most efficient one) is the Anycast technology : most of Cloudflare external-facing IPs⁵ are anycasted, which means that any request to such addresses will hit the data center that is the closest to the source of the request according to BGP routing. In other words, it's the BGP decision process that will decide which data center will be hit, based on factors like `local-pref` parameters, shortest AS path and, inside a single AS, the shortest IGP path. In the end, the data center hit should always be the most appropriate one, unlike other techniques like geolocalisation that don't take into account the structure of the Internet. We will call this "WAN Anycast", because Cloudflare also uses Anycast locally in each data center in order to select the most appropriate server, a.k.a "LAN Anycast", as explained in Section 2.5.

³As a reminder, the strength of Cloudflare is that they can protect your website in 5 minutes in a completely automated way.

⁴The process is explained in <https://support.cloudflare.com/hc/en-us/sections/200820158-CloudFlare-101>

⁵They are listed at <https://www.cloudflare.com/ips>

Finally, Cloudflare also uses Anycast for their DNS servers : this will be classified in the "WAN Anycast" category, and has already been illustrated in the "flow with Cloudflare" example⁶.

This means that under heavy load, it's likely that the requests will be dispatched to a lot of different data centers, reducing the individual load of each data center, and thanks to caching, few or even no traffic will actually reach the hosting server. This is especially true for attacks launched from a botnet, as botnets are usually spread all around the world, so the attack is inevitably divided between a lot of different data centers. Even better : if the hosting server runs into troubles and crashes, Cloudflare can still be able to maintain it alive by serving cached content to the end-users.

2.5 Architecture of a data center

As said previously, Cloudflare uses (WAN) Anycast to load-balance the traffic across its 79 data centers worldwide. Now, let's see how those data centers look like [Clo16c].

Each data center contains one edge router to connect it to the Internet. And in each data center, this router announces all Cloudflare external-facing anycasted IP addresses. The connection is, whenever possible, done at Internet Exchange Points (IXs) so that, besides using ISPs, Cloudflare can also directly peer with other networks for better performances⁷. For example, their data center closest to the UCL is located in Amsterdam, and is connected to the AMS-IX, the well-known Amsterdam Exchange Point. Behind this edge router, we can find the servers, connected to the edge router with switches. Each server has 4 main processes : DNS resolution, proxying, caching and logging.

Now, the last question is how to load-balance the traffic between the different servers inside a single data center ? And the answer is : (LAN) Anycast. They use a software called Bird⁸ allowing each server to announce all the Cloudflare external-facing anycasted IP addresses (via BGP) to the router, with a custom weight for each route. When a packet arrives in the data center, the edge router will forward it to the server with the lowest weight. This allows a lot of flexibility : if a process or a server crashes, simple route withdrawals allow the edge router to fluidly forward the traffic to another server. If a server is under heavy load, the weights can be adjusted in order to deviate some traffic to other servers. If the edge router crashes, WAN anycast will automatically withdraw all the routes previously announced, and once again, the traffic will be fluidly redirected to another data center. Even if a data center is completely overwhelmed, they can send a BGP withdraw themselves in order to redirect traffic to other data centers. That's the strength of Anycast : it's harder to configure, but it's way more powerful and handles failures way better than other load-balancers.

However, BGP updates are not instantaneous : there is some propagation time that is in general counted in seconds, sometimes in tens of seconds before the information will be propagated in the FIB of every router on the Internet. This can be annoying, but it's not a major issue. The real problem arises when route flapping (i.e. a BGP withdraw followed by an update) occurs multiple times in a short period of time. Indeed, most routers nowadays implement the *BGP Route Flap Damping* [VCG98] algorithm that will penalize routes that flap often, not advertising them anymore as long as the *penalty* is larger than the *suppress threshold* (the penalty decreasing with a

⁶To end this classification, we can also mention that Anycast is widely used for DNS resolvers, allowing a company to have only 2 IP addresses (e.g. Google's 8.8.8.8 and 8.8.4.4) for a lot of different physical resolvers worldwide.

⁷An overview of all IXs worldwide can be seen at <http://www.datacentermap.com/ixps.html>, and the list of IXs to which Cloudflare is peered (more than 100!) is available at <https://www.peeringdb.com/asn/13335>

⁸<http://bird.network.cz/>

half-life time). For example, Cisco routers are configured with a *penalty* of 1000 on withdrawals, a *suppress threshold* of 2000 and a half-life time of 15 minutes. This means that after two route flaps, the route won't be advertised anymore during 15 minutes, which can be very annoying for popular websites. However, there is also a limit on the amount of time during which a route can be suppressed : for Cisco routers, it's one hour [PBP⁺14]. We will talk about potential vulnerabilities linked to this in Section 2.7.

2.6 Advanced DoS protection service

The *basic* DoS protection offered by Cloudflare roughly consists in the advantages offered by the CDN, in addition to some other reputation-based threat protection, and customisable protection (e.g. block visitors by IP or by country). To get an *advanced* DoS protection, you need to subscribe to the business plan⁹. In this section, we will detail this advanced DoS protection [Clo16a].

If we read their whitepaper [Clo16b], we can learn that Cloudflare has three important technologies to fight against DDoS attacks :

- Automatic learning platform (for layer 3 and 4 attacks);
- Intelligent network protection (for layer 7 attacks);
- "I'm under attack" mode.

The problem is that those technologies are proprietary, so we have very few informations about them. We will however explain what we know about them here. Then, in Chapter 4, we will analyse in details state of the art DoS defenses that could give us more concrete ideas about what is actually used by Cloudflare behind the scenes, or at least which defenses we would use if we were in their shoes.

2.6.1 Automatic learning platform

This platform was built to handle layer 3 and 4 attacks, such as DNS flood attacks (see A.1.1), TCP SYN flood attacks (see A.1.2) or amplification attacks (see A.1.3).

In their whitepaper [Clo16b], it's written that "this technology analyzes network traffic and metrics in real-time in order to identify anomalous or malicious requests. Once a new attack is identified, Cloudflare automatically starts to block that attack type for both the particular web property and the entire Cloudflare network community."

From this, we can suppose that they use both statistical anomaly-based defenses (to identify *anomalous* requests) and signature-based defenses (to detect *malicious* requests). Moreover, this would allow them to feed the database with new signatures (and new suspicion scores of the corresponding IP addresses) each time they detect a new attack thanks to the anomaly-based mechanism. Besides, we know they also apply rate limiting when facing a huge DDoS attack, and that they have alarms (e.g. triggered when traffic is larger than usual).

⁹An overview of Cloudflare plans can be seen at <https://www.cloudflare.com/plans>

2.6.2 Intelligent network protection

This technology aims at fighting against layer 7 attacks such as the slow read attack (see A.2.4).

In their whitepaper [Clo16b], we can read that this technology, thanks to the large amount of traffic analysed in their network, allows them to "quickly identify new types of malicious traffic and compromised botnets that are being used in both Layer 7 denial of service and Layer 7 application vulnerability attacks". It's also said in [Clo16f] that it's "constantly monitoring and reacting to changes in your site's traffic patterns".

Again, we can suppose that there is a statistical anomaly-based part in this technology (trained on the traffic passing through their network, and looking at the changes in traffic patterns to your website), and maybe also a signature-based part to identify famous attacks (such as those defined in Annex A.2).

2.6.3 "I'm under attack" mode

This mode is the last line of defense, acting against application layer attacks, that should be used only if the other layers could not stop an attack (even if it can also be used preventively). It takes effect in less than 30 seconds after activation, and aims at verifying that the requests made to the website under protection are genuine, i.e. from real users. One easy way to implement this could be to use a CAPTCHA, but it can prove annoying for the user, so Cloudflare opted for a series of tests lasting approximately 5 seconds, using Javascript and cookies. Note that this technique does not block search engine crawlers.

After some research, we found out that this mode actually consists in a challenge page containing a math problem that the client browser has to solve, which can take a bit of time to compute. If it passes the test, the browser will receive a cookie so that it won't be challenged again. It has several benefits :

- An attacker forging packets does not use his browser, so he needs to understand this mechanism in order to bypass it. Moreover, Cloudflare can try to obfuscate the equation and run through some other tricks that make it hard to find the answer if you're not actually rendering the Javascript. And since Cloudflare is a cloud service, they can patch it each time an attacker understands it so that attackers have to start from scratch each time;
- Cloudflare could make the equation very hard to compute but very easy to verify the result (much like the Bitcoin proof of work) such that attackers wishing to open a lot of connections would be limited by their CPU rather than their upload bandwidth. And moreover, this would slow down the visits to the website, which can be useful under attack.

NB : "I'm under attack" mode also contains mechanisms after that test for devices creating a lot of connections, such as rate limiting.

2.7 Experiments on Cloudflare external-facing IPs

As explained in Section 2.4, Cloudflare relies on Anycast to load-balance the traffic between their different data centers. According to Cloudflare, "At the WAN level, every router in all of CloudFlare's [79] data centers announces all of our external-facing IP addresses [Clo16c]. This

would be great, ensuring that it's impossible for an attacker to target one data center in particular: any attack would be spread out among their 79 data centers, avoiding than even one of them gets overloaded. And since the largest DDoS attacks (of several hundreds of Gbps) are always launched from botnets that are spread worldwide, it would be even better. So we decided to verify their claim, and [warning, spoiler] we were able to find several IP addresses unicasted to a single data center ! Explanations below.

Our protocol is based on the fact that packets can't travel faster than the speed of light, which is about 300 000 000 m/s in vacuum and about 200 000 000 m/s in optical fiber [Wik16b]. In practice, packets will flow slower than that due to a multitude of other factors, but it's not a problem as we only need an upper bound. Based on that, we can easily deduce a lower bound on the time needed for a packet to flow from Louvain-la-Neuve to, let's say, Los Angeles : 42 ms¹⁰, which makes a round-trip latency¹¹ of 84 ms. Again, in practice it will be much more (we measured a latency of 180 ms with a ping on ucla.edu - the website of the University of California, Los Angeles -, but it's a good lower bound).

The protocol is the following : open a terminal in Louvain-la-Neuve (e.g. from the INGI department) and open another one in the west coast of the USA (e.g. using an Amazon AWS EC2 instance). Take the list of all Cloudflare external-facing IP addresses¹² as input, and send a ping to all of them from both terminals. Then, for each IP address, compare the results of the two different pings :

1. If the result is < 42 ms in both, the IP is anycasted. Indeed, it's physically impossible to have that result with a unicasted IP.
2. Otherwise, the IP is certainly unicasted. Indeed, Cloudflare has a lot of data centers in Europe and on the west coast of the USA, so the latency is theoretically always way better than that. As an illustration, when we pinged from Louvain-la-Neuve the list of all Cloudflare websites extracted from the Alexa top 10 000, we never got a latency lower than 30 ms, and every request was redirected to the Amsterdam data center, directly linked to the AMS-IX.

So we made a small program called `ping_anycast.sh`, available under the `ping_program` folder. It pings all Cloudflare external-facing IPv4 addresses, and reports the ones that give a latency of more than 42 ms, together with the latency obtained. The results are available under `ping_program/final_outputs`, and actually, we didn't even need to run the program in a second location to show that a lot of IPs are not anycasted : while most of these IPs have a latency of less than 42 ms, several hundreds of them have a latency of more than 170 ms.

To better understand these results, we manually ran `traceroute` commands to some of them, and looked at the one before last hop of the result. Here are some of them :

1. 103.22.200.1 - 282.138 ms : if-ae-2-2.tcore2.TV2-Tokyo.as6453.net
2. 103.22.200.11 - 278.525 ms : if-ae-2-2.tcore2.TV2-Tokyo.as6453.net
3. 162.158.0.1 - 340.700 ms : ten-0-0-0.bdr02.mel07.vic.VOCUS.net.au
4. 173.245.48.1 - 171.969 ms : las-b21-link.telia.net

¹⁰This travel time and the following ones were measured by WolframAlpha, e.g. <https://www.wolframalpha.com/input/?i=distance+louvain-la-neuve+los+angeles>

¹¹In this section and the following one, "latency" will always refer to the round-trip latency instead of the one-way latency, so that we can measure it from a single point

¹²Publicly available at <https://www.cloudflare.com/ips/>

5. 188.114.96.1 - 239.509 ms : et7-1-0-0-grtsanem4.net.telefonicaglobalsolutions.com

6. 199.27.128.1 - 170.193 ms : sjo-b21-link.telia.net

The two first IPs stopped just after Tokyo, so they obviously hit a data center in Japan, probably the one in Tokyo or Osaka. We can guess that the third IP hit the Melbourne data center in Australia (confirmed by the fact that the previous hop seemed to be in Sydney). The 4th IP seems to have hit the Los Angeles data center or another one near it (under the hypothesis that "las" refers to Los Angeles, and confirmed by the fact that the previous hop was in "ash" - probably Ashburn - 62 ms sooner). The 5th one is trickier : it could refer to San Jose, San Francisco, San Diego or maybe another one. Anyway, we can guess that the last hop was on a city on the south of the west coast of the USA, because according to previous hops, it followed the path Paris -> London -> New York -> Miami. The last one is obviously San Jose. Note again that those results are not 100% precise, but they make sense due to the union of several factors : interpretation of the names, comparison with big cities and cities in which Cloudflare has a data center, and the latency of the different hops of one `traceroute` command.

In the future, we could consider making a program able to extract those informations automatically, based on the factors mentioned and on databases of major cities and regular expressions to understand the signification of these abbreviations, but this was outside the scope of our thesis.

Finally, we tried to figure out whether these unicasted IPs were associated to websites or if they were just pointing to something else in the considered data center. The problem is that reverse DNS lookups are not very convenient (see Chapter 6 for more informations on that), so we took another direction : we took the list of the 1387 Cloudflare websites figuring in the Alexa top 10 000, and we wrote a little program (`ping_program/dig_ping.sh`) extracting their IP addresses and testing all of them.

The result was good for Cloudflare : all of these IPs were anycasted. In order to confirm our guess, we played with the `nmap` utility which allows us to discover open ports on the target host. For example, IPs associated with websites under Cloudflare protection always have four open ports : 80 (http), 443 (https), 8080 (http-proxy) and 8443 (https-alt). But when we try `nmap` on previously found unicasted IPs, we don't find any open port (they are all filtered). This confirms the fact that those IPs are not used by websites, and we can suppose that they are used internally at Cloudflare, certainly for inter-data centers communication, or maybe also as "honeypots" in order to get informations about random scanning devices.

In conclusion, we were able to show that several Cloudflare IP addresses were unicasted, seemingly pointing towards one particular data center. We showed that those unicasted IPs are not associated to any website, but as they are pointing towards one data center, it could be used to target those data centers with DDoS attacks, in order to weaken their network and the connectivity of end-users near those data centers. For example, by targeting the Sydney and Melbourne data centers at the same time, an attacker could force Cloudflare to redirect all Australian end-users to Auckland or Singapore, adding sensible latency to their traffic. It could even create a domino effect if too much traffic was redirected to the same data center, and if it was not designed to handle it (but given the architecture of their data centers, we guess this should not happen).

Another possibility is to attack these data centers one at a time with huge traffic attacks. This could force Cloudflare to send BGP withdrawals from the data center attacked in order to redirect traffic towards other ones, in order to avoid high latency and packet losses to their customers. Theoretically, this would not be very harmful because they would send a BGP update to readvertise the routes as soon as the attack stops. But if we repeat it multiple times, BGP Route Damping

could prevent those data centers from advertising their routes again during up to one hour, which would lead to the same result as the previous attack, but without having to attack two data centers simultaneously.

2.8 Automatic detection of data center hit

In this section, we will analyse the following problem : given an anycasted network (in our case, Cloudflare network), and given the complete set of their data centers, can we find precisely which data center will be hit by our traffic from a given location in the world ? We will develop three possible ways to solve this problem:

- Based on latency (e.g. via the `ping` command);
- Based on informations given by the `traceroute` command;
- Based on the response to an anycast network.

2.8.1 Based on latency

The principle is pretty simple : use the `ping` command to get the latency between you and the data center hit, and then use this information compared to other latencies previously gathered to try to geographically localise the data center. Nevertheless, this strategy has several drawbacks:

- Latency is not always linked to geographic localisation, at least for small distances, for which other factors (such as the presence of optical fiber) will have a higher impact. Moreover, BGP can decide to route your request towards a further, more optimal data center. For example, from the UCL, the geographically closest data center is Düsseldorf, but the data center hit in practice is Amsterdam.
- Even if it was the case, the difference would be too small to be significant. For example, from the UCL, four data centers can theoretically be reached in less than 1 ms (Düsseldorf, Amsterdam, Frankfurt, Paris). Even London can be theoretically be reached in 2 ms (considering, as previously, the speed of the light in fiber, even if this speed is not reached in practice).

So in conclusion, the latency trick can be useful in cases where the distance between the 2 locations is large (typically when you cross the Atlantic Ocean, or even when you go from one coast of the USA to the other), like we did in previous section, but inside Europe it's not a good idea.

2.8.2 Based on `traceroute`

We already talked about that in the previous section. As a reminder, there are several informations usable in a `traceroute` output :

- The names associated to the IPs of routers crossed;
- More precisely, the abbreviations of the cities in which these routers are located, especially for the one before last hop;

- The latency between each hop in order to break ties for previous point (for example, if the latency of one hop is close to the latency of the previous hop, we can suppose that the two cities are close to each other, or at least on the same continent).

These informations could be extracted with regular expressions, but this would require databases of big cities, rules for common abbreviations, and we would have to train our algorithm on a lot of traffic in order to get used to conventions of the different telecommunication companies. Moreover, `traceroute` patterns are not always the same (sometimes the one before last hop is not relevant and we have to check the one before), sometimes router IPs are not associated with any name, sometimes the names don't give us informations about the city in which it's located. In conclusion, it could be possible, but would require a lot of time, data, training and "trial and error", and even after that it would not work in all cases. But still, this could be an interesting way to do.

Note : it's also possible to run a `whois` command on the one before last hop IP address to try to get informations about it. For example, as Cloudflare likes connecting their data centers to big Internet Exchange Points (IXs), this IP will often belong to that IX and you can read this information in the `whois` result. The problem is that some companies run several IXs themselves (for example, Equinix). In that case, it's sometimes still possible to determine which IX was hit by looking on their website for the list of (IX name, IP range), if it's available.

So that's also an interesting way to do, but it has several drawbacks :

- Cloudflare data centers are not always connected to an IX;
- Difficulties when a large company runs several IXs;
- Not easy to do automatically;

2.8.3 Based on the response to an anycast network

This is by far the most interesting technique, but also the most difficult to set up in practice [Cox16]. Indeed, you will need your own anycast network with as many servers spread worldwide as possible. The principle is the following : send a packet from one of your anycasted servers to the Cloudflare anycasted IP address, and listen back for the response on all of your anycasted servers. The server that receives the response is the closest to the data center hit by your packet. Now, to be really accurate, you would need to set up one server close to each Cloudflare data center, and you could precisely determine the data center hit by a request from anywhere in the world (if you don't have a server at the location you want to test, you can still spoof the source IP address if your ISP allows it - see Annex B for more information about source IP spoofing).

Note that this technique also allows you to check if the destination IP is anycasted : in order to do that, all you have to do is to send the packet from several of your anycasted servers, and see if you receive all the responses on one server (which means it's unicasted), or on different servers (which means it's anycasted).

The problem with this technique is that it requires an anycast network. And this is neither easy nor cheap to set up : you need to register a /24 IP block and an Autonomous System (AS), then you need to set up all of your servers worldwide. Actually, the easiest way to do would be to rent the service of an already existing anycasted network (for example... Cloudflare !).

More about DoS protection services

In this chapter, we will first analyse another large DoS protection service named Akamai. Then, we will present a general comparison between Cloudflare, Akamai and another smaller DoS protection service named Incapsula.

3.1 Akamai

3.1.1 What is the Akamai Intelligent Platform ?

Akamai was founded in 1998 by Daniel M. Lewin, an MIT graduated student and Tom Leighton, an MIT applied mathematics professor. Their headquarters are located in Cambridge, Massachusetts in the United States. Akamai is one of the world's largest distributed computing platforms that can deliver a wide variety of web content and applications. The Akamai Intelligent Platform contains more than 150 000 servers located in more than 2000 locations over 92 countries. This platform handles more than 2 trillion requests per day, which represents around 15 to 30 percent of all the web traffic. They have a lot of big companies for customers, including Apple, Facebook, Bing, Twitter, eBay and `healthcare.gov`.

Even if Akamai was originally created to be a large CDN, it has evolved into a provider of both network and application-layer security. Its scale and overall connectivity provide some natural benefits when defending against most of the attacks present today on the Internet. Similarly to Cloudflare, the Akamai Intelligent Platform acts like a man-in-the-middle between the end-user and the web application, but this aspect will be more detailed in the next section.

3.1.2 How does it work ?

Similarly to Cloudflare, when a content is requested from Akamai, the user is redirected to the optimal server. In [Jan14], the author presents two ways to decide the optimal server. The first one, Anycast, is the one used by Cloudflare and it has already been explained in Section 2.4. The second way is to direct the user to the optimal server based on the resolver it receives the request from. In other words, when a user sends a request to a DNS-based CDN, a record will be returned depending on the resolver where the request comes from. This is called "mapping" and that's what Akamai is currently using. They also use other criteria in order to determine the optimal server such as:

- Latency;
- Throughput;
- Packet loss;
- CPU load on the server;
- HD space;
- Network utilization.

Now let's consider an example to better understand what happens when a user sends a request to a website, `www.example.com`, which is a client of Akamai. As you can see in Figure 3.1, the request is sent by the user to his ISP resolver (step 1). Then, resolver will do the recursive lookup until reaching the authoritative name server (step 2). Now the resolver will contact the concerned Akamai Name Server (step 3) which will be told the source of the request and will be able to decide which data center will be used to serve the content of the website. The IP address of the optimal data center will be communicated to the ISP (step 4) and down to the end-user (step 5) and from now, all the requests from this particular user will be forwarded directly to the local, optimal data center (step 6).

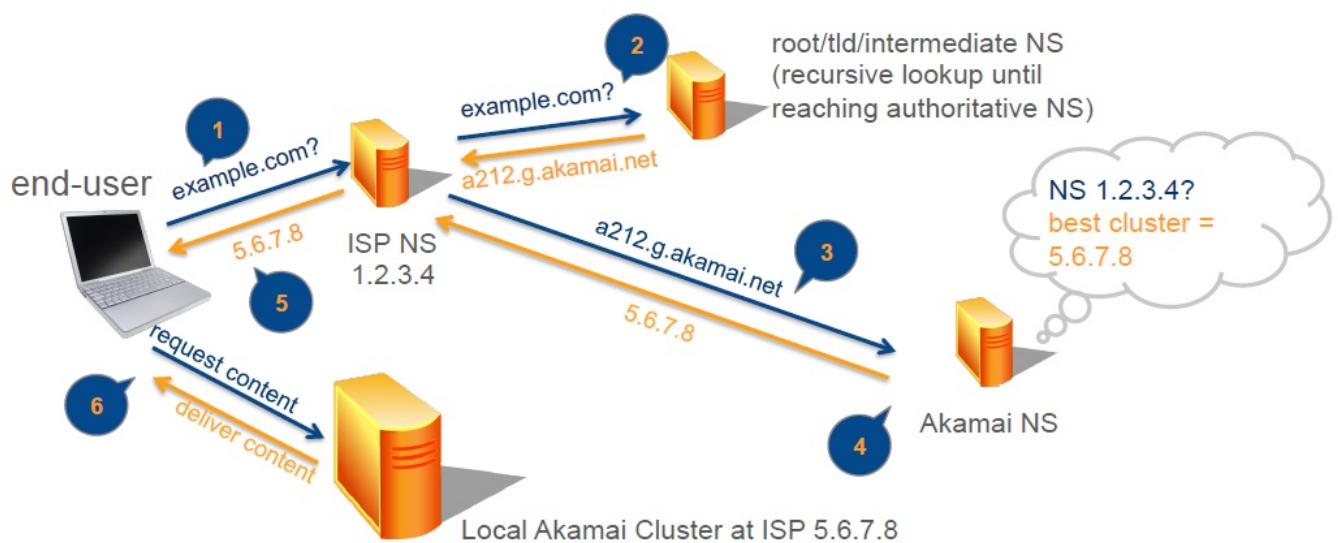


Figure 3.1: Basic CDN mapping [Jan14]

But what happens if the user decides to use a third party DNS resolver like OpenDNS or GoogleDNS ? This could result in a big imprecision on the geographic location, which could dam-

age the overall experience of the user. Indeed, while an ISP's resolver is generally representative of the real position of its clients, third party resolvers don't necessarily have a physical resolver close to each end-user. For example, suppose that the end-user lives in Southern Brazil and uses OpenDNS as resolver service. Since OpenDNS uses Anycast, the request of the end-user will be directed to the closest OpenDNS physical resolver, which is based in Miami. In consequence, when the OpenDNS resolver will forward the query to Akamai, the only informations available to Akamai will be the back-end address of the Miami resolver, and the Brazilian end-user will then be directed to the Akamai data center located in Miami, whereas Akamai had several data centers in Brazil.

To solve this problem, Akamai uses EDNS0 client-subnet. This will include the end-user's prefix in step 3 of Figure 3.1. By doing so, the CDN will now choose the optimal data center based on the prefix of the end-user instead of the resolver from which the request comes from. Currently, GoogleDNS and OpenDNS send *client-subnet* as /24 prefix for caching-efficiency and privacy reasons. An example of the use of EDNS0 with GoogleDNS as resolver can be seen in Figure 3.2

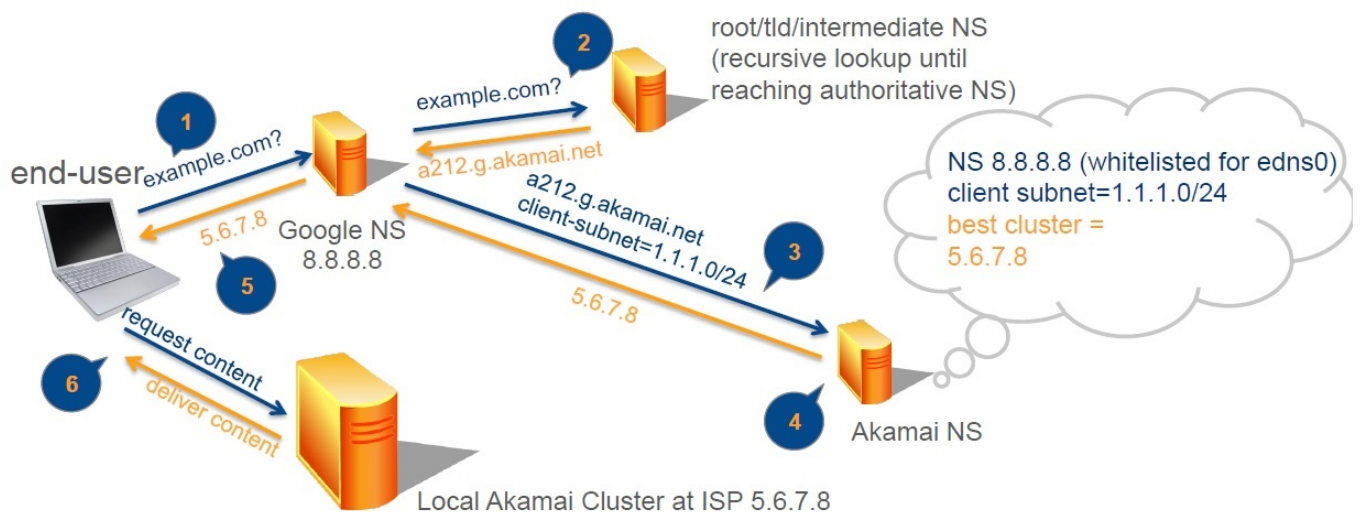


Figure 3.2: Akamai's EDNS0 example [Jan14]

This inline and distributed architecture provides two benefits when you want to counter network or application layer attacks. Firstly, all the traffic that goes through the Akamai Intelligent Platform in direction to the application can be identified and analysed by Akamai. This allows Akamai to react appropriately to mitigate the incoming malicious traffic. Secondly, because of its overall large scale, it provides Akamai with global reach to detect and stop DDoS attacks.

3.1.3 Always-on security

One of the main advantages of the Akamai Intelligent Platform is that it is always on, this provides a big advantage over other security solutions. This means that the clients of Akamai don't even need to know that they are under attack before they can defend against them. Thanks to the 24/7 Security Operation Center (SOC), all the attacks can be detected, analysed and then mitigated as soon as they occur. Akamai has reported that its 24/7 SOC team were mitigating 10 to 15 attacks every day.

3.1.4 Kona Site DDoS Defender [Akac] [Akaa]

Kona Site Defender is the security solution that Akamai has developed in order to guarantee the availability of all the websites that use the services of Akamai. Kona Site Defender is built on the same edge network as the one of the Akamai Intelligent Platform. Kona is able to detect, identify and mitigate traffic that corresponds to DDoS attacks or web application attacks. The DDoS defenses implemented by Kona Site Defender are designed to be always on. This greatly improves the flexibility and the response time during a DDoS attack [Akac]. This package can also be sold as a standalone for use with websites that don't have DDoS protection.

Defenses against network-layer DDoS attacks

All the network and transport layer attacks are automatically stopped by dropping the packets at the edge of the Internet. The Akamai Intelligent Platform is designed as a reverse proxy and only accepts packets on ports 80 and 443 for HTTP and HTTPS respectively. Dropped traffic includes UDP fragments, ICMP, SYN, ACK, RESET, UDP floods. Kona also implements a Web Application Firewall (WAF) which enables the ability to define custom IP white and blacklists. Those lists can be updated across the entire network of Akamai in within 45 minutes which means a rapid capacity of response when under attack.

Defenses against Application-layer DDoS attacks

The attacks that operate on the application layer can't be automatically dropped. Most of the attacks communicate with the website in a legitimate way, with legitimate requests but they will attempt to provoke a denial of service of the application by overwhelming the server with incomplete or malformed requests (see Annex A, Section A.2 for more informations about those attacks). Kona Site Defender will defend web applications against these kinds of attacks in two ways.

Firstly, the Akamai Intelligent Platform is able to absorb most of the largest attacks, which are, HTTP flooding (see Section A.2) and DNS flooding (see Section A.1.1), before they even reach any of the targeted websites.

Secondly, Kona Site Defender provides tools to automatically identify the attacks and block corresponding traffic to the application. Those tools are:

Static and advanced caching Akamai Intelligent Platform is distributing its web content across their 150 000 edge servers in order to make it harder to disrupt protected web applications.

Adaptive rate controls Akamai is heavily monitoring the behaviour of the clients of their protected application and they will automatically block the clients that have suspicious behaviour. For example, if a user sends a lot of requests in a short amount of time, or if they detect that the pattern of a request is identical to the one of other clients, then the user will be considered as an attacker and will be stopped. These automatic rate controls will act as deterrent on the traffic before it reaches the web application.

IP whitelists and blacklists Akamai will allow or deny specific IP addresses. Those lists are propagated across Akamai's whole network within 45 minutes, which enables a relatively rapid response to DDoS attacks.

Geo blocking This tool will allow Akamai to block traffic from specific regions to mitigate localized DDoS attacks.

WAF Akamai provides pre-defined and configurable application-layer firewall rules. On top of that, Akamai allows its customers to create some *Akamai metadata-based rules* and those will be enforced after the application layer rules.

HTTP Inspection Akamai is able to inspect HTTP request/response *headers* but also HTTP POST request/response *bodies* by passing them through a series of cascading REGEX rules.

eDNS (Enhanced DNS) [Akac] [Akab]

Not to be confused with EDNS0, which has already been explained in Section 3.1.2, Enhanced DNS is an optional feature of Kona Site Defender. The eDNS has been designed in order to not directly expose the primary DNS server of the customer to the end-user. The use of eDNS requires no change to the already existing DNS while providing the customer with an always available, scalable and high performance DNS resolution service thanks to the large size and high level of distribution of the Akamai Intelligent Platform.

Figure 3.3 represents the eDNS components. The Zone Transfer Agents (ZTAs) are processes that run on multiple machines and are physically separated from the Akamai name servers. Only the customer that has enabled the eDNS knows the IP addresses of these ZTAs. All the data that go from the primary DNS of the customer have to pass a validation/verification test, done by the ZTAs, before being transferred to the Akamai name servers. The zone transfer supports Secret Key Transaction Authentication for DNS (TSIG), see RFC 2845 for more information.

The customer can configure the interval time between zone transfers and they also have the choice to configure eDNS to accept NOTIFY requests in order to reduce the time gap between zone transfers. ZTAs are distributed in all the locations of the Akamai Platform in a redundant fashion. ZTAs balance their processes by dynamically electing a leader that will decide of the load balance over all the other ZTAs. With this technique, Akamai ensures constant availability of ZTAs for their customer who would like to propagate updates.

Akamai name servers are responsible for resolving the end-user DNS request. They are distributed in various locations from the United States to Europe and Asia. They will receive an update from the ZTAs each time that the customer has made some changes on his primary name server. One important aspect is that these name servers will never be able to serve as primary name servers because there are not allowed to initiate zone transfers. Another important aspect is the use of IP Anycast for these top-level name servers. This means that the end-user will always be routed towards the closest name server but different end-users with a different network location may get routed to different ones.

On top of all of that, the customer can activate the DNSSEC security extensions for the eDNS of Akamai. This will allow the customer to digitally sign his zone transfers using public key cryptography. The main goal of DNSSEC is to prevent DNS cache poisoning and DNS hijacking.

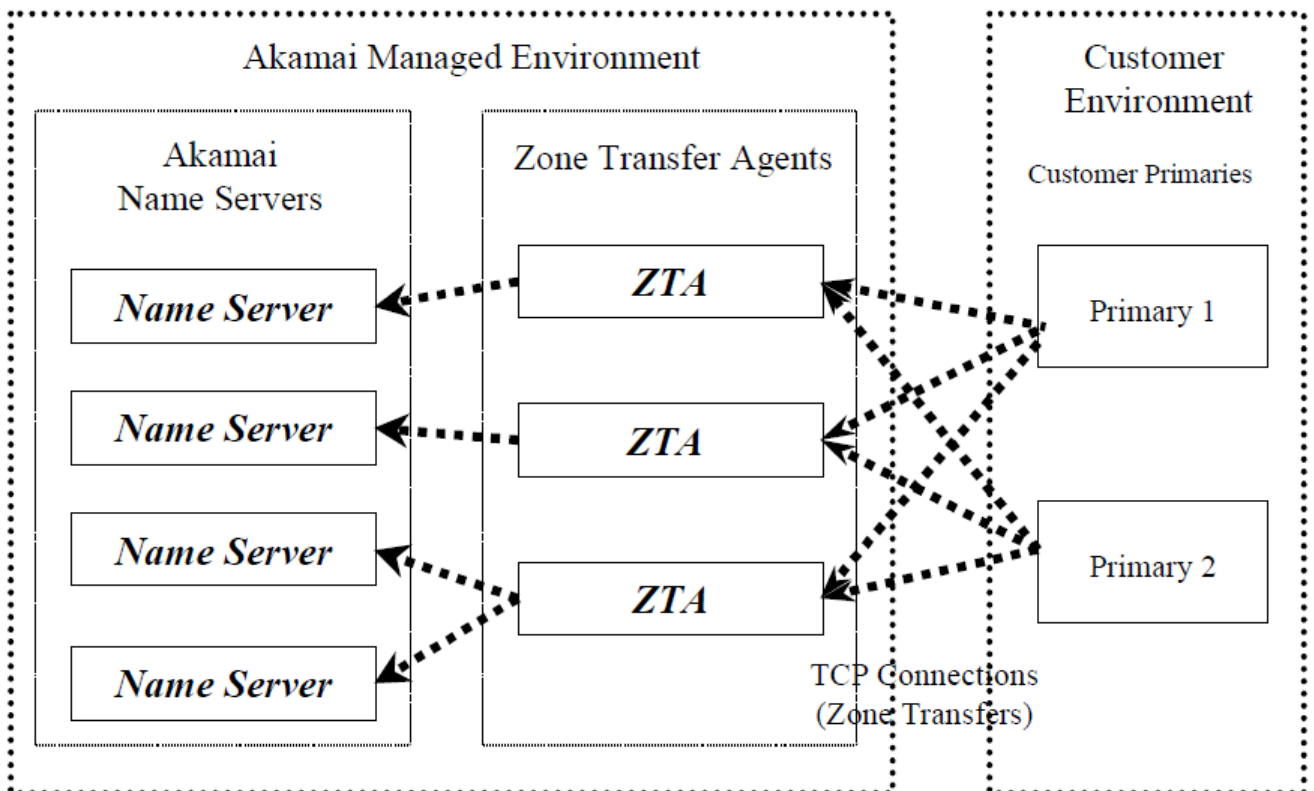


Figure 3.3: Enhance DNS Service Components

3.2 Comparison

Let's conclude this chapter by returning on important features of DoS protection services, listing the advantages and drawbacks of each possible solution, and analysing the choice of the different protection services mentioned so far (Cloudflare, Akamai and Incapsula).

3.2.1 Scale

At the time of this writing, Cloudflare has 79 data centers worldwide, each one in different cities¹. Incapsula has 30 data centers worldwide, again each one in different cities². Finally, Akamai claims that its CDN is composed of "more than 216,000 servers in over 120 countries and within more than 1,500 networks around the world"³. Given those informations, it's clear that Akamai has the largest CDN.

As a reminder, in the introduction, Figure 1 showed us that Akamai and Cloudflare have both much more customers than Incapsula, and while Cloudflare has probably more overall customers thanks to the better accessibility of their service, Akamai has more important clients (i.e. clients in the first places of the Alexa websites ranking).

¹For up to date informations, see <https://www.cloudflare.com/network-map/>

²You can consult them at <https://www.incapsula.com/incapsula-global-network-map.html>

³See <https://www.akamai.com/us/en/about/facts-figures.jsp>

3.2.2 Setup

There are basically two ways to set up a CDN :

1. Put a CNAME record from your website (www version of the domain, e.g. `www.example.com`, since CNAMEs are forbidden on the root domain) to the CDN.
2. Replace the authoritative NS of your website (root domain, e.g. `example.com`) by a NS of the CDN.

The first solution is convenient since you can configure it directly from your DNS server, it's just a record to add. Nevertheless, it has some security issues. Firstly, your DNS server is exposed to end-users since their first request (to your root domain) will inevitably reach your own DNS server. Moreover, since you can't use a CNAME on the root domain, your web server will also be exposed since you will have to give its IP address to the end-user. After that, you can use HTTP redirection to redirect the end-user to the www version, which will redirect him to the CDN thanks to the CNAME record, but the damage is already done : the end-user knows the IP addresses of both your DNS server and your web server, and direct-to-IP attacks are now possible.

The second solution is more annoying to use, since you have to pass by the registrar of your domain to do the modification. However, it's way more secure : even the first DNS request of the end-user (to your root domain) will directly reach the CDN, obviously it's also the case for all the subsequent requests. In consequence, with such a setup, it's theoretically impossible for an attacker to find the IPs of your own servers.

As explained above, Cloudflare privileges the second solution, even if they also allow you to use the first one if you want to (at your peril). Akamai uses both solutions, but it seems that most of their customers have currently the first one. For Incapsula, the default solution is the first one, but with a minor change : they also provide you one IP address to put as a "A" record for your root domain. In consequence, your web server is not vulnerable anymore since it will redirect the end-user to an Incapsula IP instead of the IP of your web server. Incapsula also proposes the second solution (NS protection) but it's an extra feature (you have to pay for it), unlike Cloudflare where it's even included in the free plan.

NB : the fact that your DNS server is exposed is not necessarily a problem, since most of the time you don't have your own DNS server, but you subscribe to a DNS service or use the DNS servers of your registrar. Nevertheless, they can still be the target of big DDoS attacks themselves.

3.2.3 DNS servers

If you choose the second option here above, there are two ways to implement it :

1. The CDN completely replaces your DNS server, which becomes unused. In consequence, the future modifications will have to be done directly via the CDN;
2. The CDN proxies your DNS server, which means that you can keep managing it like before.

As we saw in Section 2.3, the default solution of Cloudflare is the first one (remember their original scan utility), but they also propose the second solution, known as "Virtual DNS". Akamai opted for the second solution, using eDNS as seen in Subsection 3.1.4. Incapsula also uses the second solution.

3.2.4 Load-balancing

This part is very interesting, because the three CDNs use three different mechanisms :

- Cloudflare uses Anycast for everything, which means that they rely on BGP to find the closest data center;
- Akamai uses EDNS0, so they rely on the /24 prefix of the end-user to localise him, and then they use proprietary heuristics to find the closest data center to him;
- Incapsula uses MaxMind's geolocalisation tool to find the precise location of the source address of the DNS requests, which means that they locate the resolver used by the end-user to contact them.

Our opinion is that Anycast is by far the best mechanism since it relies on the architecture of the Internet to find the most optimal data center. Moreover, everything is done automatically, you have automatic fail-over and so on. The only drawback is that it's less customizable: BGP chooses for you, you can't change this decision except by using BGP withdraw or update announcements, but you must be careful at BGP Route Damping as explained in Section 2.7.

General mitigation mechanisms

This chapter covers simple defenses and mitigation techniques, which are in general easy to set up but either obsolete, inefficient or quite useful but just not sufficient [Wik16d]. In the end of the chapter, we will try to establish the specifications of an optimal dedicated DoS defense.

4.1 Network ACLs

One of the most basic (and most limited) defenses is the network Access Control List. Often implemented on routers or switches, it basically consists in a list of IP addresses and port numbers allowed or forbidden to pass through the machine. Some ACLs can also filter traffic based on other parameters such as the destination IP, the destination port or the protocol (e.g. filtering ICMP traffic in order to avoid ping flooding attacks), but the filtering will always be stateless, so in most cases a more advanced filtering technique will be required (for example, stateful firewalls introduced in next section).

However, an example of application successfully handled by ACLs is *bogon filtering* : it consists in dropping packets with a source IP address belonging to the *bogon space*, i.e. the set of IP addresses not yet allocated by the IANA (or a delegated Regional Internet Registry). Indeed, those addresses should never be seen on the public Internet, and come de facto from randomly spoofed IP packets. Note that smart attackers won't use those addresses during their spoofing attacks, but it's still good practice to set up this defense.

4.2 Firewalls

We can classify firewalls in three categories, from the least to the most sophisticated [Wik16g] :

- Packet filters, a.k.a. stateless firewalls. They will filter based on layer 3 and 4 headers, but

they have to process each packet independently.

- Stateful firewalls. The major difference with the previous ones is that they have a memory so they can use new criteria based on the connection state to filter packets.
- Application layer firewalls. They can inspect packets based on any OSI layer, payload included, and can then detect most attacks by "understanding" protocols such as FTP, DNS or HTTP. They are often implemented using application proxies, meaning that two TCP connections are established : one between the packet source and the firewall, and another between the firewall and the packet destination. In consequence, we will often have one by type of application. A good example is the Web Application Firewall (WAF), which will focus on the HTTP protocol. Note that CDNs often use a distributed WAF (i.e. cloud-based) : it's the case for Cloudflare, Akamai and Incapsula.

In practice, stateful firewalls are preferred to stateless ones because of their connection tracking capabilities. The problem is that stateful firewalls need to maintain a connection state memory, and this memory can be the target of DoS attacks aiming at filling it with thousands of fake connections. In consequence, the overwhelmed stateful firewall won't be able to accept new legitimate connections. You can try to bypass that by aging out inactive connections faster (by sending a RST from the firewall to the user), but it's not a perfect solution. Moreover, there are other disadvantages to firewalls : each rule will add some load on the machine, potentially leading to its CPU getting overwhelmed. In the end, against DoS attacks, stateful firewalls are often the bottleneck as a first line of defense, so more advanced mechanisms can be required to detect attacks more easily, or to filter them more efficiently.

Application layer firewalls primary goal is to block general attacks such as SQL injections, XSS and other application-specific attacks, and they are really good at doing that. They are also good at stopping application-layer DoS attacks. But they should obviously not be used as first line of defense.

Note that to defend oneself against DoS attacks, a default deny policy will always be preferred, i.e. close all ports except those in use in order to decrease the potential attack range. And that's indeed the strategy adopted by, for example, Cloudflare: if you run `nmap` on any website under their protection, you will see that there are always only four open ports : 80 (http), 443 (https), 8080 (http-proxy) and 8443 (https-alt).

4.3 IDS & IPS

An Intrusion Detection System is a sophisticated device that will monitor the network for malicious activities or policy violations, and provide feedback to a management station [Wik16h]. An IDS can be :

- Host-based or network-based. We will focus on the network-based, since we are interested in identifying DoS traffic.
- Hardware-based or software-based. An hardware-based IDS will generally be more efficient, but it's obviously more costly to deploy.
- Online or offline. The goal of an offline IDS is to feed a database with new attack signatures for latter use. But we will rather focus on online IDSes, as they allow to react more quickly to new types of attacks.

- Signature-based or statistical anomaly-based. In the first case, it works like an anti-virus, using a library of known signatures in order to detect DoS attacks. Note that this library will have to constantly evolve, as the attacks and softwares used are constantly changing. In the second case, it will try to detect attacks by comparing current traffic patterns to previous parameters learned on "normal" traffic. The advantage is that it can detect new, never seen attacks (so we don't have this "lag" phenomenon from a signature-based IDS), but it will in general generate more false positives.
- A simple IDS that will just produce reports or signal alarms, or an IPS (Intrusion Prevention System) that will also react to the threat. For example, an IPS can decide to change the attack's content, or to reconfigure the firewall in order to block the attack. When you just have an IDS, a possibility is also to ask your ISP to filter traffic upstream based on the patterns found by the IDS.

Let's now list typical evasion techniques used by attackers in order to bypass an IDS :

- Encryption : if the content of a packet is encrypted, the IDS won't be able to detect threats in that part of the packet.
- Fragmentation : if a threat consists in a combination of multiple factors, the attacker can fragment his packets so that the IDS doesn't consider them as a whole.
- Avoiding defaults : sometimes, attacks are detected because the attacker always uses the same value in some particular fields of his packets. A good way to bypass that for an attacker is to randomize all the unnecessary fields of the packet.
- More precisely, an attacker can be identified with its source address : an obvious evasion technique is to spoof it (e.g. randomly), or to use different proxies.
- More generally, an attacker can try to use changing attack patterns in order to mislead a signature-based IDS. There are several ways to do this, but we won't detail them here as we encroach on the domain of anti-virus.

To end this part of the defenses, we will have a look at Figure 4.1, which is a graph from the 2013 security report of Radware¹. It shows that in 33% of the cases, the first device overwhelmed by a DoS attack is the firewall or the IDS. This confirms the fact that those devices should not be used as a first line of defense, and highlights the need for dedicated DoS defenses as we will see in the end of this chapter.

4.4 Manual response

Upon manual detection of an attack, try to mitigate it by yourself, either by creating new rules in your firewall, or by asking help to your closest upstream connectivity provider (e.g. your ISP, hosting provider...). For example, you can ask them to identify the true source of the attack, but it could require cooperation among many providers, which makes it complicated and time consuming. And in general, you don't want to wait so long, hence the need for automated mechanisms.

¹<https://blog.radware.com/security/2013/05/can-firewall-and-ips-block-ddos-attacks/>

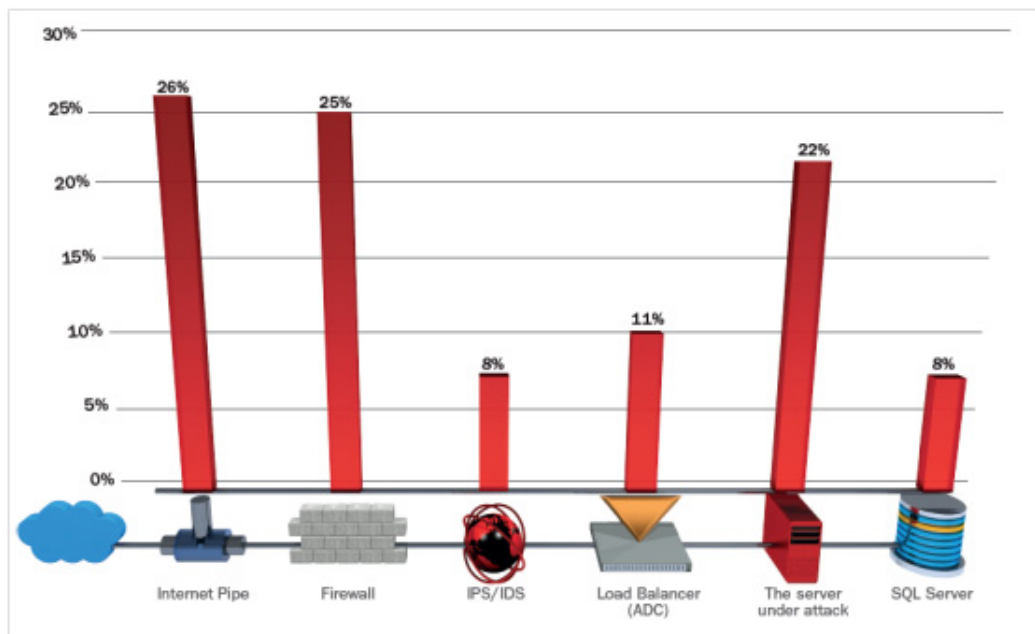


Figure 4.1: Main bottlenecks during DoS and DDoS attacks

4.5 Overprovisioning

Simple principle : add bandwidth, redundant network interfaces and devices in order to "absorb" the spikes of traffic generated by the attack. The problem is the cost : you will have to invest a lot of money for equipment that will only be used during attacks, so it's not really worth it.

4.6 Black hole filtering

It consists in sending every packet destined to a specific IP address to a "null route" : in other terms, all the packets will be dropped before they reach the bottleneck of the attack (e.g. the firewall, the server...). This technique is regularly used by ISPs in order to save their network (for their other customers), and is very efficient for this purpose since the filtering can be done more easily, and further upstream where the traffic is not too heavy yet. The problem is that the victim will lose all his traffic (bad traffic as well as legitimate one), resulting in a real denial of service for him, so it doesn't solve the original problem, it's just the last resort.

4.7 IP blacklisting

IP blacklisting can be automated (using a tool identifying itself the threatening IPs, e.g. based on pps per client) and/or manual (let the user choose himself what to blacklist). It can be based on single IPs (banish one attacker at a time), on IP ranges (e.g. if the attackers are all under the same ISP, or if ingress filtering prevents him from spoofing IPs from other networks) or even on countries (for example, the owner of a small Belgian website could decide to only allow Belgian IPs to access his website). The blacklist can also be divided into several parts: for example, one category for each identified botnet, one category for known disreputable proxies and one

for known open recursors. When you are under attack, you can then identify the threat (e.g. reflection-based or directly from a botnet) and act in consequence (only activate one category of the blacklist).

Note that an alternative, sometimes scaling better is the IP whitelisting : see Subsection 5.2.1 for an example of algorithm based on that principle (*History-based IP filtering*).

4.8 Upstream filtering

It's the use of a "cleaning center" to filter your traffic before it hits your real server. That's the technique used by Cloudflare, Akamai, Incapsula, and more generally every other CDN providing DoS protection². The advantage is that you don't have to invest a lot of money in hardware: you just have to pay a monthly subscription in order to benefit from already existing, state of the art equipment.

4.9 Working around the Linux kernel

This idea comes directly from Cloudflare [Clo16h]. It's based on the observation that the Linux kernel networking stack, due to constraints such as numerous unavoidable interruptions, is becoming the new bottleneck in current networks. Indeed, modern 10 Gbps NICs can usually process at least 10 Mpps, while Vanilla Linux can hardly manage 1 Mpps (and in average, it's rather 50 kpps per core). In consequence, the obvious way to handle larger packet floods is to work around the Linux kernel - let's call it "kernel bypass".

The kernel bypass proposed by Cloudflare is the following : during packet floods, offload selected flows (packets belonging to a flood) to a user-space application. This application will then be able to filter those packets really quickly, in general dropping most of them, but identifying and reinjecting valid packets in the kernel afterwards so that there is no loss. A general schema of the situation can be observed in Figure 4.2. Receive Side Scaling (RSS) is used by the NIC to distribute packets to the different RX queues so that packets belonging to different flows can be processed by different CPUs. But when a flood occurs, one or several queues will be isolated by modifying the *indirection table* and by using *flow steering* to specifically direct the flooding flows to those isolated queues. Then, we have to link those isolated queues to a user-space application, that will be able to filter them faster than the kernel network stack. The goal is to let the kernel process normal packets (since it can do it alone efficiently), but to use the user-space to filter the flooding packets (because most of them will be dropped, so few of them will have to be passed back to the kernel, and the filtering job can be done more quickly by a user-space application).

Now, how to put that into practice ? For Solarflare network cards, there is already an existing proprietary API (`ef_vi`) that works well. But for non-Solarflare network cards, there was no existing open-source solution that could do the kernel bypass without taking over the whole network card (and what we want is a single RX queue kernel bypass, i.e. let the kernel process "normal" flows even during a flood). In consequence, Cloudflare decided to patch Netmap with a single RX queue mode. The code is available under <https://github.com/luigirizzo/netmap/pull/87>. The only "obscure" part is the identification of the flooding flows, which is

²For information, here is a list of other famous ones : A10 Networks, Arbor Networks, AT&T, F5 Networks, Neustar Inc, Radware, Sprint, Staminus Communications, Tata Communications, Verisign, Verizon [Wik16d].

most probably a proprietary technology. Indeed, in their examples they execute the flow steering manually, but in practice they must have a dedicated program or piece of hardware that listens to the traffic and sends flow steering instructions to the NIC. This could either take place ahead the Linux machine (in order to react immediately to the flood), or after (which would produce a small "lag" phenomenon before they are able to react, but would probably be easier to deploy).

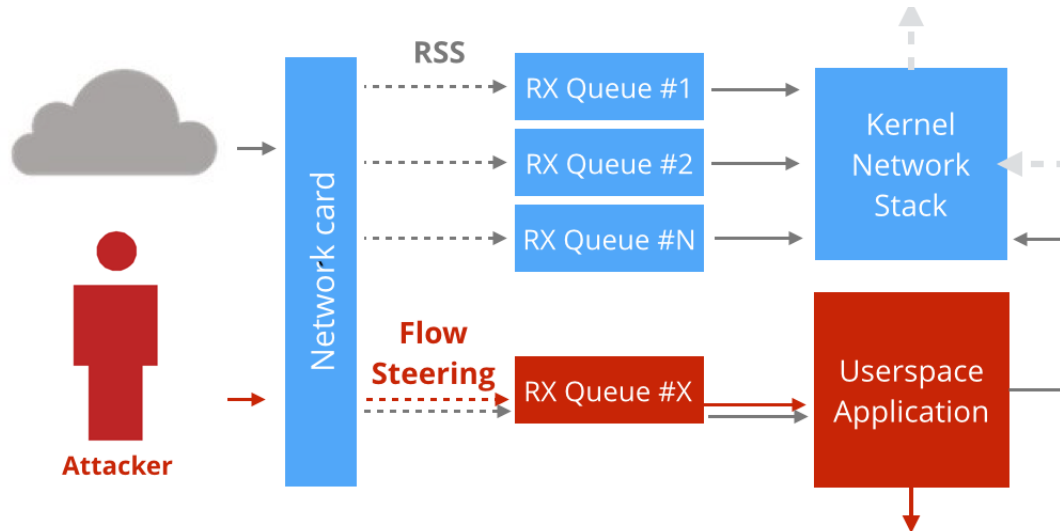


Figure 4.2: Single RX queue kernel bypass (figure from [Clo16h])

4.10 Conclusion and dedicated DoS defense

In conclusion, most of the techniques mentioned above are not sufficient to counter today's huge DDoS attacks. This highlights the need for more advanced defenses specifically designed to defend against such attacks. What we need is a protection mechanism that would (ideally) have the following features :

- This mechanism should be the first line of defense;
- It should be able to detect a flood of valid packets with the malicious intent of overwhelming a service, so it should not consider packets independently;
- It should be totally automated, i.e. behavior-based anomaly detection and immediate response without human intervention;
- The detection could occur at the same place as the filtering, or better, they could occur at two different locations;
- The device(s) could be software-based, or maybe better, hardware-based;
- It should inspect network and application headers, but not perform full DPI;
- It should be a stateless mechanism in order to be able to handle millions of connection attempts without requiring connection table entries or exhausting other system resources (i.e. not have the flaws of a stateful firewall);
- The policy should be to only blocking known bad traffic (in order to avoid false positives);
- The mitigation should work at line rate, i.e. for tens of Gbps.

The next chapter will expose state of the art DoS defenses that could be part of such a mechanism. But before analysing them, we can quickly go through a proprietary example designed by Cisco and meeting most of those requirements. Their solution set consists of two hardware devices, the Cisco Traffic Anomaly Detector (TAD) XT and the Cisco Guard XT [Sys16]. The first one is an early warning system; it passively monitors network traffic, looking for deviations compared to a "normal" behaviour previously learned. Upon detection of an attack, it will make a detailed report to the Cisco Guard XT. The Guard can for example be deployed next to the edge router of a data center or customer in order to protect it. Upon reception of an attack report (from the Cisco TAD XT or another interoperable device such as an IDS), traffic destined to the edge router will be directed to the Guard for a multiple-stage analysis and filtering process aiming at dropping malicious packets while forwarding good packets back to the router. This multiple-stage process includes static (user-configurable) filters and dynamic filters (based on observed behaviour). It contains a source-authentication module ensuring that incoming packets have not been spoofed. The central part is then an anomaly recognition module that compares traffic to a baseline and looks for deviations that would identify the source of the attack (which should be correct thanks to the source-authentication module). Then comes the protocol analysis module that looks for application-layer attacks. There is also a rate-limiting module penalizing sources consuming too many resources for a too long period.

State of the art DoS defenses

This chapter will talk about advanced, state of the art DoS defenses. These ones will be divided into different sections depending on the layer of application (network/transport layer, or application layer) and on the deployment location (at the source, at the destination or hybrid, i.e. distributed defenses) [ZJT13]. At the end of the chapter, we will also expose some defenses against specific DoS attacks, and present some experimental results about one of the destination-based defenses.

5.1 Source-based defenses for layer 3/4 attacks

Generally speaking, source-based defenses can use two different mechanisms to weaken a potential outgoing attack. First, they can block packets with a spoofed source IP. Secondly, they can limit or block traffic from a machine not respecting some flow model. In this section, we are going to present in details a few of the most famous source-based defenses against layer 3/4 attacks.

5.1.1 Ingress/Egress filtering

This defense is explored in details in Annex B. The following is just a summary.

In short, *egress filtering* consists in verifying that outgoing packets (i.e. packets going from a private network to the Internet) have, among others, a valid source IP address. By valid, we mean at least an IP address belonging to the IP block allocated to the private network, but we can do better, e.g. by verifying that this IP is currently allocated to a user of the private network.

Ingress filtering is more general as it can be implemented on any router in the Internet. It consists in dropping an incoming packet if it doesn't come from the network associated with its source IP address. The general technique used to verify that is unicast Reverse Path Forwarding (uRPF) that has a basic mode (Strict RPF) and relaxed modes (Feasible RPF, Loose RPF). In Strict RPF, the

interface from which the packet comes from must match the interface which would have been chosen by the FIB to send a packet to this IP address. But since routing is not always symmetrical, this can cause problems, so relaxed modes were designed, for example allowing interfaces from the N best paths based on the RIB (Feasible RPF), or even allowing any interface as long as the IP address is reachable (Loose RPF).

5.1.2 D-WARD [MPR02][MPR03]

D-WARD is deployed at the source-end of the networks and it's able to automatically detect and stop attacks. The detection is done by constantly monitoring the traffic flows between the network and the rest of the Internet plus a periodic comparison between these flows and normal flows models. Once a malicious flow is detected, the flow rate is dynamically limited. This rate limitation is adjusted over time with the changes in the behaviour of the flow. With this techniques, false positive flows will be able to recover quickly but if the flow corresponds to a real attack then it will be severely limited.

The D-WARD defense system must be deployed on routers that are network entry points. The more routers implement the D-WARD, the more effective the defense will be. One advantage to be deployed at the source is that the routers are less likely to relay big traffic flows which allows for more complex and more accurate detection mechanisms to be performed.

Figure 5.1 represents the overall architecture of D-WARD. D-WARD is a self-regulating reverse system. First, the router will monitor all the packets that go through, then it will gather statistics about the traffic and it will send them to the D-WARD module. From there, the observation component will have to compare these statistics with the hard-coded models to classify the traffic (see next subsection). Those results are transferred to the throttling component which will adjust and transmit the new rate-limit rules to the source router.

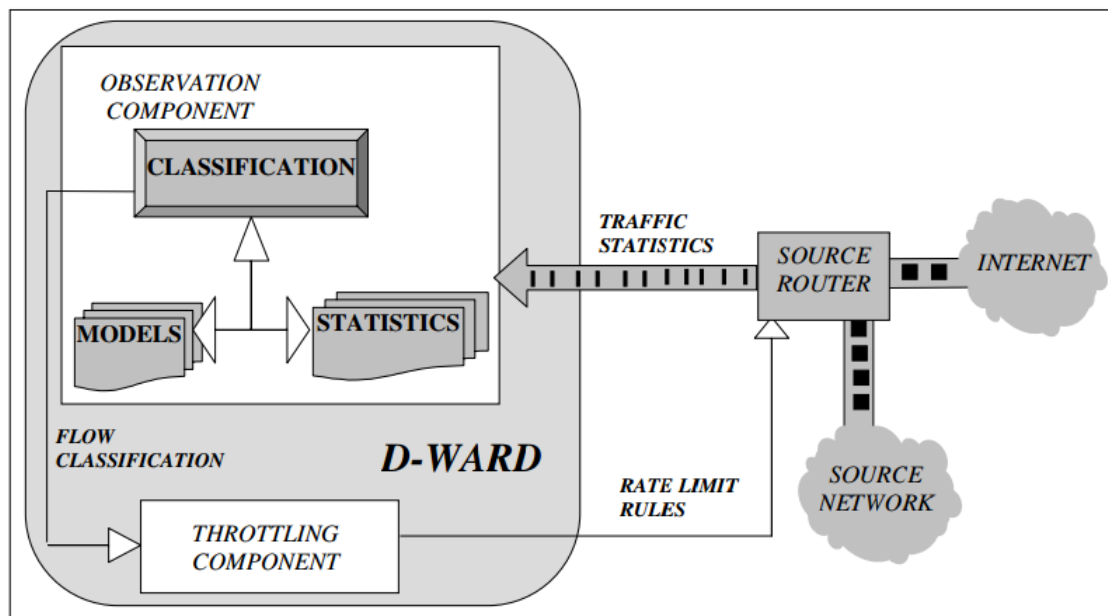


Figure 5.1: D-WARD architecture

Observation component

As we previously explained, in the observation component, the statistics will be compared with normal flow models and then classified as either normal, suspicious or attack flows. But we still didn't concretely define what's a normal flow model. To do so, we are going to explain the criteria that define a normal flow model for different types of traffic, namely, TCP, ICMP and UDP traffic. In each case, D-WARD defines a parameter that will serve as a threshold which will determine if the flow is a genuine one.

TCP normal traffic model. D-WARD's TCP flow model defines the following parameter: TCP_{rto} which corresponds to the maximum allowed ratio between the packets that are sent and received in the TCP flow by the source.

ICMP normal traffic model. D-WARD's TCP flow model defines the following parameter: $ICMP_{rto}$ which corresponds to the maximum allowed ratio between timestamp, information request and echo request and the corresponding reply.

UDP normal traffic model. D-WARD's TCP flow model defines the following parameters:

- n_{conn} an upper bound on the number of allowed connections.
- p_{conn} a lower bound on the number of allowed packets per connection.
- UDP_{rate} a maximum allowed sending packets rate per connection.

The flow will be considered as an attack flow if at least one of these three thresholds are exceeded. An advantage of such an architecture is that D-WARD doesn't need to be trained on normal flows, since the rules are already hard-coded and regulated via input parameters. On the other hand, the drawbacks of D-WARD are that it consumes a lot of memory space and CPU cycles than most of the other source-based defenses.

Throttling component

In order to properly regulate the rate limit of the flow, D-WARD applies TCP congestion ideas: fast exponential decrease when the flow is categorized as an attack, slow recovery of rate-limited flows for a certain time period and finally, fast recovery once flows proved that they are genuine.

5.1.3 MULTOPS and TOPS [GP01] [AGCLS03]

MULTOPS is a heuristic and tree data-structure that will monitor certain traffic characteristics to detect and filter DDoS attacks. With this method, each router will maintain a data-structure MULTOPS which stands for Multi-Level Tree for Online Packet Statistics. As you can see in Figure 5.2, there is a node for each subnet that contains packet rate statistics. The detection of an attack is done by using a simple heuristic: a significant difference between the packet rate that comes in and out of a host or a network. Of course, this is based on the assumption that normal operations on the Internet generate the same packet rate in both directions.

MULTOPS will fail if the attack consists of deploying a lot of proportional flows that all target the same victim. It won't detect any anomaly here because the number of packets that come in the network will be the same as those that leave it.

Moreover, the number of levels in the tree indicates the level of precision of the defense but the

more levels the tree has, the more memory is needed. So for MULTOPS to be really effective, it is capital to not restrict its memory which can sometimes be problematic. This is the major drawback of this method and the main reason of the introduction of the next defense, TOPS.

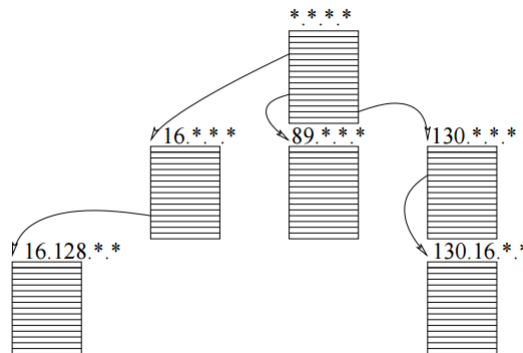


Figure 5.2: MULTOPS tree example

TOPS, standing for Tabulated Online Packet Statistics, uses a compact, fixed-size structure thanks to a few effective heuristics such as hashing. It can detect bandwidth attacks with a high rate of precision and low rate of false positives. The advantages of this method are that it uses relatively low computational resources and it doesn't require more resources when under attack which means that it won't slow down when an attack is performed.

Both methods are based on the hypothesis that the incoming and outgoing packet rates are proportional which can be a wrong assumption from the get-go. For example, packet rates for streaming are highly non-proportional, which would create a lot of false positives. Moreover, clever attackers can superficially increase their incoming or outgoing traffic in a genuine manner in order to remain undetected.

5.1.4 MANAnet's Reverse Firewall [Cs316]

A traditional firewall will protect a network from incoming packets. The reverse firewall, on the other hand, will do exactly the opposite, that is, protecting the outside from packet flooding attacks that would originate from this network. This can be extremely interesting for network owners such as ISPs, universities or even corporate networks.

The main principle is pretty simple, it will limit the rate at which the network sends packets which do not correspond to replies to previous recently received packets. Keep in mind that the reverse firewall still has to allow some packets to get out of the network which are not replies, e.g. to start new connections and so on but these packets don't need to be transmitted at high rate.

The main drawback of the reverse firewall is that it requires administrators to be involved to configure the firewall manually. On top of that, most of the ISPs seem to miss the benefit of implementing a reverse firewall simply because there is not a direct gain for the source network (see discussions in Annex B for more informations about that).

5.1.5 Conclusion

Source-based defenses are efficient against DDoS attacks since they can block them before they reach the Internet, while they are still split around different source networks. Moreover, it's the easiest place to detect and filter source address spoofing, which can weaken most attacks and facilitate traceback and investigations by the victim.

However, source-based defenses aren't the most effective for three reasons. One, the sources of a DDoS attack are, most of the time, highly distributed which makes the detection accuracy pretty low overall. Two, the rate of false positives is high because it's hard to make a clear differentiation between malicious and genuine traffic at the source since the closer we are to the source, the less traffic volume we will be able to analyse. Three, there is no real benefit for the source network to deploy such costly mechanisms as it will mainly benefit to other networks (see Annex B for a detailed discussion about that).

Source-based defenses should be implemented by ISPs in order to prevent their clients from attacking other networks. We think that every ISP should at least implement ingress filtering in order to prevent its clients from perpetrating spoofing attacks. Nevertheless, it's not necessarily sufficient since they can still launch attacks with their real IP. In order to avoid this scenario, we would advise them to implement at least one of the three remaining defenses presented in Section 5.1, or even a few of them deployed simultaneously. Nevertheless, we would avoid using MULTOPS or TOPS since they are pretty old and based on outdated models of the Internet.

5.2 Destination-based defenses for layer 3/4 attacks

In this section, we are going to detail some defense-based mechanisms against layer 3/4 attacks. We divided them into two categories : defenses that can be applied directly, without collaboration with other routers, and defenses that need to collaborate with them, which means patching them with new protocols, which can be difficult to deploy in practice.

5.2.1 Without collaboration of other routers

History-based IP filtering

With this mechanism, during an attack, the victim will filter traffic on an edge router based on the history maintained previously, during "normal" periods. In other words, the victim will create an *IP address database* made of IP addresses that appeared *frequently* during previous, non-attack periods. Upon detection of an attack, the filter will be activated, denying any IP not belonging to the database. Considering a "training", i.e. "normal" period of N days, an IP address will be considered as "frequent" and be added to the database based on [PLR03] :

- The number of different days d in which it appeared during the training period;
- Considering only occurrences with a number of packets (per flow) u greater than some threshold. For example, by setting the threshold to 3, you make sure the IP address was not spoofed, and by setting it to 5, you make sure it was a meaningful connection (3 packets for the connection establishment, at least 1 for the information transfer, and one for the connection release). Packets not meeting this rule are considered as anomalous and come in

general from a network scan, a spoofing attack or a network failure.

Note that the period of N days can be viewed as a sliding window. In their experiments, they chose $N = 14$ days in order to have a trade-off between completeness, memory requirements and rapidity of the look-ups.

The main advantages of this mechanism are:

- Blocks almost all the attack traffic during a spoofing attack with random source IP addresses;
- Doesn't require cooperation between several entities, as it's an entirely destination-based defense.

The main drawbacks are:

- It's only destination-based, so the edge router implementing this defense could be overwhelmed if it can't handle the traffic of a very large DDoS attack;
- There will always be some unavoidable false-positives;
- An attacker knowing that this defense is in place, and knowing the typical IP ranges accessing the network, could try to spoof his IP addresses according to that;
- An attacker knowing that this defense is in place could incorporate his own IP addresses in the database by sending normal traffic several days before the attack. A possible countermeasure is to increase the period over which IP addresses must appear in order to be considered as frequent.
- The effectiveness of this mechanism is lowered due to the fact that most ISPs give dynamic IP addresses to their customers. Nevertheless, customers often keep the same IP during several days. We can also take that into account and lower the period from 14 to 1 or 2 days. But this could make the filter too difficult to pass. An alternative would be to allow all the IP addresses in the subnet of a frequent IP address.

See last section of this chapter for our experiments and more discussions about this defense mechanism.

HCF [WJS07]

Hop-Count Filtering is a defense mechanism against spoofing attacks. The defense is based on the fact that even if an attacker can randomly spoof his source IP, it is impossible to do it while keeping consistent TTL fields simply because he can not forge the number of hops the packet takes to reach its destination. In order to infer the spoofed IP addresses from the genuine ones, a server has to use a mapping between the IP addresses and their hop-counts. This hop-count will be inferred from the TTL value of the packet at the destination. The only tricky part is that at the destination, you can only see the final value of the TTL field, and different OSes can use different initial TTL values.

However, under some empirical hypotheses such as that most people use only a small number of different OSes, or that most Internet hosts are apart by less than 30 hops, it's in general possible to guess the initial TTL value, and then to infer the real hop-count.

The default mode of the defense is the *learning* state, where HCF only monitors the IP2HC mapping behaviors but it won't discard any packet. If a spoofing attack is detected then HCF switches to its second mode, the *filtering* mode, and only then it will proceed to filter and stop

spoofed IP packets.

During the *learning* phase, HCF will build an accurate IP2HC (IP to hop-count) mapping table in order to detect and stop spoofed IP packets. The implementation of this method is done with the pseudo-code described in Figure 5.3. Note that the mapping table will be updated only with packets belonging to established TCP connections so that it is impossible for attackers to poison the table.

```
for each packet:
    extract the final TTL  $T_f$  and the source IP address  $S$ ;
    infer the initial TTL  $T_i$ ;
    compute the hop-count  $H_c = T_i - T_f$ ;
    index  $S$  to get the stored hop-count  $H_s$ ;
    if ( $H_c \neq H_s$ )
        the packet is spoofed;
    else
        the packet is legitimate;
```

Figure 5.3: Hop-count inspection algorithm

MIB [Cea01]

First of all, note that MIB (Management Information Base) is a database describing formally a set of network objects, used for example for SNMP (Simple Network Management Protocol). The mechanism proposed in [Cea01] is a kind of IDS; it tries to detect DDoS attacks as early as possible by using exclusively MIB traffic variables, such as the packet and routing statistics, which are recorded when a DDoS attack occurs. Examples of MIB variables typically used are `udpInDatagrams` which represents the total number of UDP datagrams delivered to UDP users, and which is the key precursor of UDP flood attacks, or `icmpInEchos` which represents the total number of ICMP Echo (request) messages received and which is the key precursor of Ping flood attacks.

The extraction of all the relevant MIB variables is done automatically by using Statistical Tests for Causality, at the attacker and victim locations. Once these variables are extracted, they can be analysed and compared to an Anomaly Detection scheme, based on a simple model of the normal rate of the key MIBs in order to detect DDoS attacks before it shuts down the victim. Concretely, the anomaly detector can be implemented by looking at "jumps" in the MIB variables, i.e. checking the absolute values of the difference between one variable at two different times, and comparing it to the largest jump encountered for the same variable between two different times during a training phase.

At the moment, this defense has only been tested on a local test bed with controlled traffic loads (only three different attack types). It would require to be further tested in more realistic conditions before its real deployment.

Packet dropping based on the level of congestion

Destination-based methods can drop malicious packets if the network throughput is congested until a certain threshold. One example of this kind of defense methods is Packetscore [KLCea06].

This method is based on the key idea of prioritizing packets that have a good score. This score estimates its legitimacy related to the attribute values it carries. The score of a packet is computed using the "Conditional Legitimate Probability" (CLP) which is based on the Bayesian theorem. CLP is produced by comparing the packet flows during an attack with previously monitored legitimate traffic. This approach is based on the fact that some traffic characteristics stay stable during normal traffic operation.

Once the score of a packet is computed, it will be dropped or not based on a dynamically adjustable threshold. This threshold will be adjusted based on two things :

- The current score distribution of recent incoming packets;
- The current level of congestion of the system.

The advantages of this method are that the general principle is very simple, and it adapts well to network conditions. Moreover, it's a completely independent mechanism (i.e. it doesn't require cooperation with other entities). One possible drawback is that it requires a "training" phase.

5.2.2 With collaboration of other routers

IP traceback mechanisms

The idea behind IP traceback methods is to trace back the spoofed IP addresses that were used in a DoS attack to their true sources. There are already a lot of existing methods that can be categorized in two main classes.

The first category is **packet marking** mechanisms. One example of defense in this category is the Router Interface Marking (RIM) [CPM06]. With RIM, routers will mark packets in a probabilistic manner (which means that only a small portion of the packets - representative of the attack - will be marked) with an interface's identifier. Once an attack is performed, a victim can gather the informations that RIM routers have marked inside some packets to trace back the source of the attack.

An important thing to see here is that RIM-enabled routers don't mark packets with their IP, but with the interface used by the packet. This approach has several advantages, including requiring relatively small number of marked packets to be able to trace back, fast traceback, last-hop traceback capability, low computation overhead and low false positive rate. Finally, it is possible to update RIM to counter marking field forgeries.

The second category is **link testing** mechanisms. Link testing mechanisms usually start from the edge router of the victim and iteratively contact upstream routers to determine which link was used to carry the attack traffic. Hence, they can do their way upstream until they reach the true source(s) of the attack. One example of link testing mechanism is presented in [BC00]. In this article, the authors present a technique to trace back spoofed packets to their real source host without needing a cooperation between different ISPs. The first step is to map all the possible paths from the victim to all possible networks. After that, they will test each link (each branch of the tree) and observe changes in the rate of incoming packets. This will allow them to locate

the sources of network load (for examples, real attackers or reflectors); usually they process by elimination and this allows them to isolate a small handful of potential attack sources, from which they will have to find the real source. This technique is based on the assumption that the attack comes from a single source network.

Path Identifier (Pi) [YPS03]

Path Identifier (Pi) is a new packet marking technique that embeds a path fingerprint in each packet. By doing this, it enables a victim to identify packets that go through the same paths on a per packet basis even if the source IP is spoofed. This marking technique is a per-packet deterministic mechanism, which means that each packet that travels on the same path of a network will have the same identifier. *Pi* marking and filtering are really light-weight operations which make it effective against large-scale DDoS attacks that have thousands of attackers under the condition that about half of the routers in the crossed networks actually mark the packets.

The main advantage of this technique is that it's able to identify the true sources of the attack without even looking at the IP address, so it's really convenient against spoofing attacks. The main drawback is that it requires at least half of the routers to be patched with the protocol, which is not easily doable in practice.

5.2.3 Conclusion

The main advantage of destination-based mechanisms is that they defend your own network against attacks from the Internet, so you have strong incentives to deploy them. Moreover, it's easier to detect an attack at the destination than at the source.

The main drawback is that it's sometimes too late, since huge attacks launched from large botnets are already merged together in a wave of traffic that most devices can't handle. However, for most attacks, smart defenses can still do the job, especially those which collaborate with upstream routers in order to mitigate the attack in the intermediate network instead of purely at the destination. The problem is that those mechanisms require upstream routers to be patched, which can be quite hard to do in practice.

5.3 Hybrid (distributed) defenses for layer 3/4 attacks

This section contains most of the hybrid defenses proposed in the literature so far. They are called hybrid because of their communication characteristics. As you can see in Figure 5.4, hybrid mechanisms that will be presented here are deployed at multiple locations among source, destination and intermediate network, so they require strong cooperation among those different deployment points in order to work properly.

5.3.1 Hybrid packet marking and throttling/filtering mechanisms

Here we present methods that usually deploy their detection mechanisms near the victim, but the filtering mechanisms are rather deployed near the source of the attack. These techniques only limit the traffic of malicious packets while leaving the genuine traffic untouched.

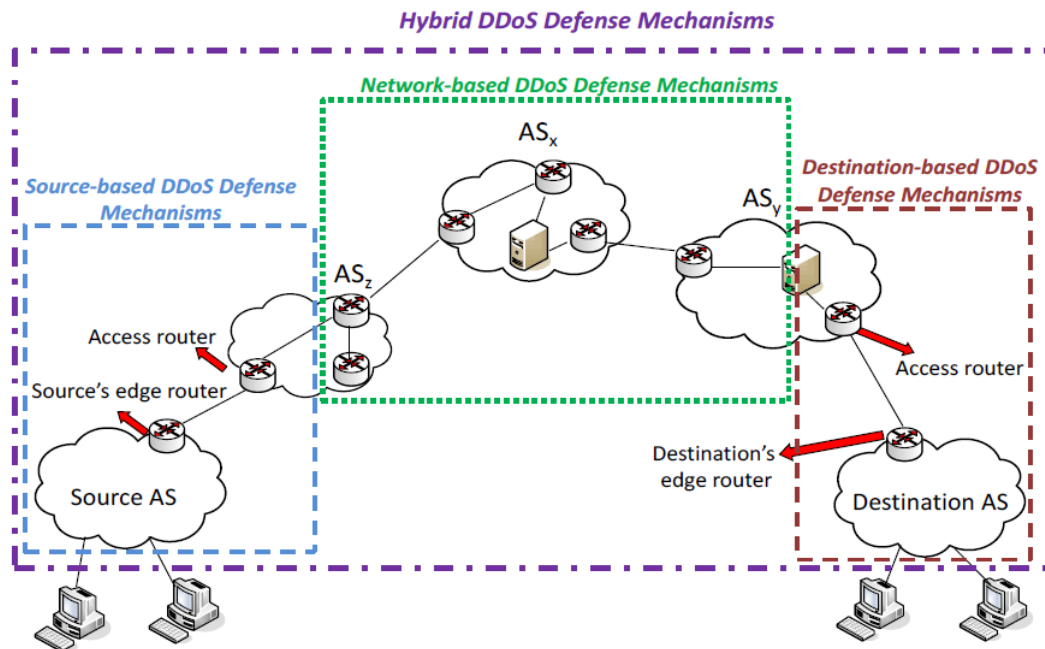


Figure 5.4: Defense mechanisms location [ZJT13]

Aggregate-based Congestion Control (ACC) and pushback [MBF⁺02]

Instead of rate-limiting IP addresses, which are most of the time spoofed, ACC rate limits *aggregates* which are portions of the traffic sharing some particular characteristics such as the source/destination IP/port or the protocol. Typical examples of aggregates are TCP SYN packets or ICMP ECHO packets.

Upon detection of an attack, this mechanism first tries to identify flooding aggregates. Then, the filtering will be done through a *pushback* mechanism. *Pushback* is a cooperative mechanism used between contributing routers to control the identified flooding aggregates. The victim will ask its closest upstream router to rate-limit the concerned aggregates to some value. From there, if needed, the request will be propagated further upstream, but only to interfaces from which a large part of this aggregate traffic is coming.

The problem with this defense is that it's not effective against numerous attack sources (such as in the case of a botnet) because traffic will be uniformly distributed between all the inbound interfaces of every router. Moreover, it requires that the routers to which the requests are sent understand the protocol, like other cooperative mechanisms.

Attack Diagnosis and parallel-AD [CP05]

The Attack Diagnosis (AD) defense mechanism is activated by the victim when an attack is detected. The activation is done by sending AD commands to the nearest router. Once the command is received, the router deterministically marks each packet intended to the victim with the input interface information. With these additional informations, the victim can trace back the origin of the attack easily. From that follows the filtering phase that will take place close to the source.

The problem with this defense is that it isn't effective against large-scale attacks, which include a

lot a different sources that launch the attack. To solve this issue, [CP05] proposed the extension Parallel Attack Diagnosis (PAD). PAD is able to trace back and filter traffic from multiple sources simultaneously.

5.3.2 DEFensive Cooperative Overlay Mesh (DEFCON) [MRR03]

As you can see on Figure 5.5, DEFCON is a distributed framework that consists of peer-to-peer defense nodes that can communicate easily between each other to dynamically implement a cooperative defense. There are four types of nodes, the core defense nodes (black nodes), the classifier nodes (dark grey nodes), the alert generator nodes (light grey nodes) and finally, the legacy router (white node).

The defense nodes are placed as close to the source network as possible. Once an attack is detected, the alert generator node propagates the alert message to all the nodes in the peer network. Only the nodes that propagate the traffic to the victim will receive the alert message. DEFCON will then determine the tree topology of the network and will make the concerned nodes cooperate to deploy rate-limiting of the attack traffic. The classifier nodes are there to differentiate genuine traffic from malicious traffic.

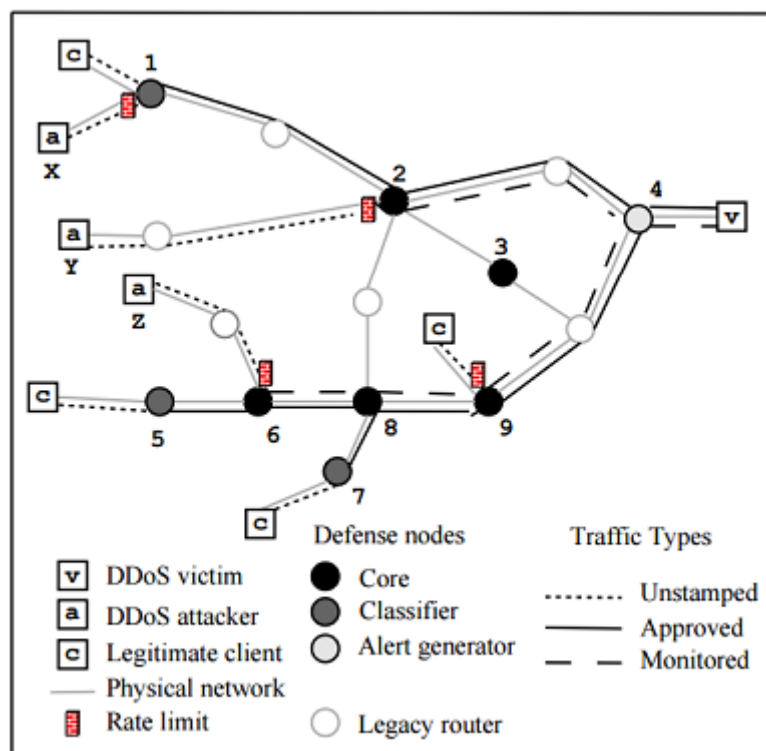


Figure 5.5: DEFCON overview [MRR03]

5.3.3 Capability-based mechanisms

Capability-based defenses regroup all the mechanisms that let the victim explicitly authorize the traffic it wants to receive. This means that the sender will receive an authorisation from the receiver and this authorisation will be stamped on each packet sent by the source.

Stateless Internet Flow Filter (SIFF) [YPS04]

This technique allows the destination to select which flow will reach its network. All the traffic is divided into two classes : privileged, where the traffic is ordered by levels of priority and unprivileged which is all the remaining traffic. Privileged traffic will be prioritized at routers and will be allocated more bandwidth such that they should never be dropped by unprivileged packet flooding.

A normal traffic can become a privileged one by completing a capability exchange handshake (in an unprivileged channel). This handshake is represented in Figure 5.6. First, the client sends an EXPlorer packet to the server that is marked with a tag α . The server will respond with its own EXPlorer packet with the α and β embedded in the capability reply field. The client is now able to send his first data packets, those will have both markers embedded in the capability reply field.

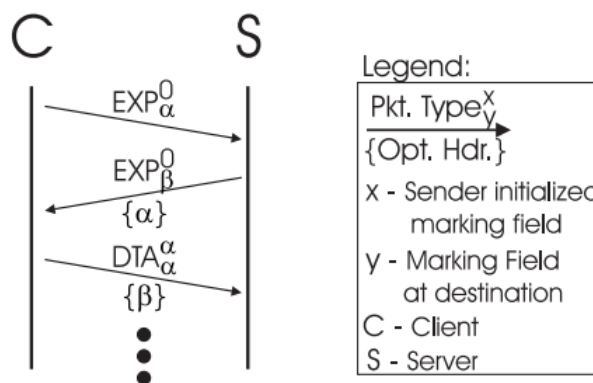


Figure 5.6: SIFF handshake

The capabilities are dynamically verified by the routers in the network. The problem of this method is that it is still vulnerable to flood attack if the attacker is using previously acquired capabilities. Another potential vulnerability is the flood of unprivileged channels, preventing new clients to complete their handshake and so to become privileged.

5.3.4 Active Internet Traffic Filtering (AITF) as a filter-based (datagram) mechanism [AC09]

AITF is a filtering-based mechanism, which means that the destination will accept all the traffic by default, but then apply *filters*, i.e. stop traffic specifically detected as malicious.

With AITF, the victim is able to directly contact the source of the attack and ask it (or rather its ISP) to stop sending traffic. Each network that has been identified as a source of malicious packets is policed by its own ISP. If the ISP doesn't police its clients, then it risks losing access to the complaining receiver. This is a great way to force all the ISPs to cooperate, even more if the receiver is a very popular service. In short, the original feature of this mechanism is that it creates a collaboration between the victim and the ISP of the source of the attack.

In [AC09], the authors have shown that even two networks that would deploy AITF could maintain their connectivity when they are under a DDoS flooding attack. AITF also uses a three-way handshake for verifying the validity of the filter requests. However, this method of validation can be incomplete if the flooding link is outside the AS of the victim because the three-way handshake

packets will travel through the same flooded link. This can really jam the whole defense system, because if the three-way handshake is not completed, then there won't be any filter deployed to stop the flooding attack.

5.3.5 Conclusion

Hybrid-based defenses are theoretically the best defenses against DoS attacks since they can detect the attack when it reaches the destination but the appropriate response takes place at the source or the intermediate network. Another benefit of the approach is that, since it is spread among multiple locations, it will have more resources available at each of them to counter the attack.

The drawbacks are the overall complexity and overhead of the communication schemes that the different parts have to implement in order to share all the informations. Moreover, this communication system must be secured and always available even under DDoS attack. Finally, there is sometimes a lack of incentives, similarly to source-based mechanisms.

5.4 Destination-based defenses for layer 7 attacks

Unlike the previous destination-based defenses presented (for layer 3/4 attacks), the two defenses of this section involve *only* the destination, so they don't require cooperation with other entities such as intermediate routers.

5.4.1 DDoS-Shield [RSU⁺09]

This application-layer defense aims at mitigating HTTP-based attacks, using rate-limiting as its primary defense mechanism. First, the suspicion mechanism assigns a continuous valued measure to each client session. This value will be based on the statistical deviations from a "normal behavior" trained offline on logs of legitimate client sessions, and more particular parameters such as *session inter-arrival time*, *request inter-arrival time* or *session workload-profile*.

Then, the scheduler uses this value to determine if and when to schedule a session's requests. While basic schedulers generally use mechanisms like FIFO or Round Robin, DDoS-Shield incorporates the suspicion value in its scheduler, resulting in two possibilities : the "Lowest Suspicion First" scheduler that could starve sessions with high suspicion probability, and the "Proportional to Suspicion Share" scheduler that uses a *max-min fair algorithm* to assign forwarding rates to sessions in proportion to their confidence probabilities. The scheduler is typically placed in a reverse-proxy such as a DoS protection service such that it can intercept requests belonging to malicious sessions before they overwhelm the real server.

5.4.2 DAT [LC11]

DAT stands for *Defense Against Tilt DDoS attacks*. Unlike DDoS-Shield that only defends against HTTP-based attacks, DAT holds for any layer 7 DDoS attack, but with a focus on attacks targeting the CPU of the end-server through high-complexity requests. Basically, DAT monitors users's features (such as instant traffic volume, session behavior) during a connection session to determine

whether users are malicious or not. This is decided by attributing a "degree of behavior deviation" to each user, comparable to the suspicion value of DDoS-Shield. Based on those results, it will provide differentiated services to different users : legitimate users will be guaranteed a certain level of service even during an attack, while malicious users could for example experience rate-limiting. This will be achieved by using a scheduler, and by giving it the users' behaviors as parameters.

5.4.3 Conclusion

We can see that there is a marked difference with previous destination-based defenses. While the main problem of layer 3/4 defenses was to fight against source IP spoofing, application-layer attacks require non-spoofed IPs, so the corresponding defenses can take a new form: they can try to analyse the behaviour of the different users throughout a session, since we know that one session corresponds to one user.

5.5 Hybrid (distributed) defenses for layer 7 attacks

Finally, we will end with hybrid defenses for application-layer attacks. All of the defenses presented in this section require new cooperation schemes between the destination and the source, except CAPTCHAs that use already existing communication protocols.

5.5.1 Speak-up [WVB⁺06]

Unlike other defense mechanisms that try to slow down the bad clients, Speak-up encourages good clients to send more traffic so that they will have more chances to be heard (under the hypothesis that bad clients are already using most of their upload bandwidth so they can't react to the encouragement). This is a strange idea, but it can be justified by the fact that if bad clients manage to look exactly like good ones but outnumber them, then traditional defenses can't work. The best alternative is then a fair allocation between good clients and bad clients.

Concretely, when the victim server detects an attack, it will encourage all its clients to use more bandwidth by sending large HTTP POST requests called "bandwidth payments". Then, the server will allocate its protected resources in proportion to the upload bandwidth spent by clients (among others, the bandwidth payments).

Despite the fact that the overall traffic to the server inflates, good clients will be much better represented in the traffic than before and will then capture a larger fraction of the server's important resources than before. For example, if 10% of the clients are bad, without this mechanism they could almost use all the server's protected resources because other clients would not use all their upload bandwidth. With Speak-up, if the remaining 90% good clients have the same total upstream bandwidth as the 10% bad ones, then bad clients will be limited to 10% of the server's protected resources, which is a really good improvement.

As a reminder, this defense works only against application-level attacks, and thanks to a collaboration with the clients (which need to understand the protocol, which is the same drawback as other collaborative defenses).

Another version of this defense mechanism, DOW (Defense and Offense Wall), was proposed by [YLCC07]. It's the combination of Speak-up with an anomaly detection method based on

K-means clustering. The goal is to get even better results by dropping suspicious sessions while encouraging good clients to send more traffic (under the hypothesis that there will still remain some bad clients). However, this mechanism seems to be too complex and resource-consuming compared to the possible performance gains [ZJT13].

5.5.2 Differentiate DDoS flooding bots from humans (e.g. CAPTCHA)

CAPTCHA stands for "Completely Automated Public Turing test to tell Computers and Humans Apart". In the first versions of CAPTCHAs, the user had to read distorted letters in order to prove he is actually a human. But with the rise of machine learning, they demonstrated to be insecure, so new versions of CAPTCHAs tend to use image recognition instead of text distortion (i.e. identification of simple objects in the images presented). Indeed, this task is typically more complex to perform than text recognition, and should in consequence be more secure to machine learning attacks.

The problem with CAPTCHAs is that they require users to solve them, which can result in a significant waste of time. Moreover, it can be really annoying for disabled people. In consequence, alternatives have been proposed in the literature. For example, [OM09] proposed a new mechanism based on three aspects of human behaviour :

- Request dynamics : they learned typical human interaction dynamics, detecting bots that show a more "aggressive" behaviour;
- Request semantics : they learned transitional probabilities of user requests, detecting bots that generate valid but low-probability sequences;
- They embed human-invisible content into server replies, detecting bots that visit them.

Another recent example is the new version of Google's *No CAPTCHA reCAPTCHA* that now consists only in a small box in which you need to click in order to prove you're a human. The detection is based on several parameters such as the time spent on the page, the exact movements of the user's mouse (especially when it approaches the box) and other confidential parameters. When the click doesn't produce a conclusive response, an old-school CAPTCHA will pop to clarify the situation.

However, all the methods of this section share the same inconvenient : they block web crawlers, preventing for example search engines to index the content of the website.

5.5.3 Trust Management Helmet [YFLL09]

Trust Management Helmet (TMH) is similar to *History-based IP filtering* in the sense that under attack, it will favor good, trusted clients. But since this one is a defense against application-layer attacks, it can be improved a bit by using the fact that clients need to establish sessions before being able to communicate with the server.

In consequence, instead of identifying trusted clients based on their IP address, it will use a more complex mechanism: their visiting history and several other informations will be stored in a *license*, stored at the client, and cryptographically secured against forgery and replay attacks. When a client revisits the server, he attaches his license to the session connection request, and based on the license and the current level of congestion, TMH will decide to accept or reject the request (and will also update the license for the next time).

5.5.4 Conclusion

As you can see, hybrid defenses against application layer attacks can be based on various different mechanisms, but like destination-based defenses, they still try to police an entire session instead of individual requests, which is efficient since it's necessarily the same user during the entire session.

Nevertheless, we are not convinced by all of these defenses: Speak-up is not a perfect solution, and could even worsen the situation if misconfigured or under special circumstances. TMH could work, but requires cooperation with the clients in order to store and send back the certificate, so it could be hard to deploy in practice. On the other hand, CAPTCHA-like mechanisms seem to be a good alternative. Of course they should not be used permanently because it slows down the user experience, but a good compromise would be to activate them only when the server is under attack, and to use lighter versions like Google's *No CAPTCHA reCAPTCHA* or mechanisms such as Cloudflare's "I'm under attack" mode (explained in Subsection 2.6.3).

5.6 Defenses against specific attacks

This section contains a list of defense mechanisms designed to handle one specific type of DoS attack. They can be used to create a defense in depth, i.e. add redundancy layers to the existing defenses, or to fill a gap in the current system. Note that the goal of this section is not to provide an exhaustive list (it would be too long and pointless), but rather to show an alternative way to the previous sections.

5.6.1 Reflection-based attacks

There are several ways to defend against such attacks, depending on where you stand :

- At the source of the attack : use *egress filtering* to prevent source IP spoofing, and therefore reflection.
- At the intermediate network (before the reflector) : use *ingress filtering* (same reason)¹.
- At the open reflector used by the attacker : patch your reflector such that it only responds to queries from authorized clients (e.g. only from the IP range of your company), and/or try to decrease the amplification factor attainable by your reflector (e.g. by disabling `dig ANY` requests for a DNS server or resolver).
- At the destination of the attack : you could try to establish lists of IPs of known open reflectors, and under attack, block packets from those IPs as far upstream as possible. Otherwise, use more advanced algorithms, depending on the type of reflection used (see for example next subsections).

5.6.2 DNS Amplification attacks

This is a defense specific to DNS reflection attacks proposed by [KMGG08]. The general idea is to detect "orphan pairs", i.e. DNS responses not corresponding to any of the victim's requests.

¹See Annex B for more informations about egress and ingress filtering.

Concretely, they employ the IPtraf tool, a monitor that will record both DNS requests and responses. Their algorithm (called DAAD) will process the captured network data on the fly and store them in a MySQL database. Then, upon reception of a DNS response, the DAAD module checks for the existence of a corresponding request by performing a SQL lookup, and if no match is found within a given timeframe, the response is marked as suspicious. When the number of suspicious messages is larger than a defined threshold, an alert is generated in order to block the suspicious IP addresses in the firewall of the edge router.

As this defense is based on the exceeding of a threshold, there will inevitably be some false negatives at the beginning of the attacks, but as soon as the threshold is exceeded, all the falsified DNS responses should be dropped on the edge router.

5.6.3 SYN flood

A first, easy way to defend against SYN flood attacks is called "SYN-proxy" [Hol16]. When the firewall detects a SYN flood attack, it switches into "SYN-Proxy" mode, which means that upon reception of a SYN, it will directly send a SYN+ACK to the client and wait for the final ACK. Since it doesn't forward anything to the server before the completion of the three-way handshake, the server won't be affected at all by the SYN flood. The advantage is that it can save substantial resources (CPU and memory) to the server, but the drawback is that the problem is just transferred to the firewall : firewalls can also have CPU issues, and worse, their connection table can also fill up very quickly.

Another, maybe better way to protect the SYN queue is to activate SYN cookies when a SYN flood attack is detected. It basically consists in a particular choice of initial TCP sequence numbers so that the server can discard the corresponding SYN queue entry, as all the information required to validate the handshake is in this sequence number. Concretely, the SYN cookie (i.e. the initial sequence number chosen by the server) will be based on :

- t , a slowly incrementing timestamp;
- m , the MSS (Maximum Segment Size);
- $s = hash(\text{server IP, server port, client IP, client port, } t)$ on 24 bits.

In the end, the sequence number will be $t \bmod 32$ followed by m (encoded on 3 bits) followed by s . The big advantage of this technique is that it's not an extension of TCP (just a particular choice of parameters), so it's easy to deploy.

The drawbacks are that there are only 8 possible values for the MSS, and the server must reject options (since there is no SYN queue entry to store them), but this is not a big problem since this mechanism will only be deployed under DoS attack, so the performance loss is negligible compared to the potential damages of the attack [Wik16f].

Note that another defense mechanism, called "TCP Cookie Transactions" (TCPCT) was discovered in 2011 and defined in RFC 6013 (Experimental). It was based on an extension of TCP, thus requiring support in the client as well as in the server TCP stack. Nevertheless, it didn't work as well as expected and was removed from the Linux kernel in 2013.

5.7 Our experiments

Finally, we decided to experiment on one of those defenses in order to evaluate its efficiency. We decided to skip defenses requiring cooperation between several entities since it was impossible for us to test it in real life conditions. In consequence, we decided to experiment on a destination-based defense. We first wanted to implement the hop-count filtering, but unfortunately we couldn't find training data suitable for this defense (the only data available were Netflow data from the UCL network, which don't contain the TTL field). We then decided to implement the history-based IP filtering.

The code is available in the folder `ip_filtering`. The data used (`ip_filtering/to_ucl_sorted.txt`) are Netflow data representing all the flows that reached `130.104.5.100`, i.e. the IP address corresponding to `uclouvain.be`, during a 4 hours interval (those data are not available on the public BitBucket for confidentiality reasons). We chose to divide those data in two sets: a training set of 3 hours, and a test set consisting in the last hour. Given the small training set, we chose to accept any IP appearing at least once in the training set, but only if it's a TCP flow consisting of at least u packets, with u varying.

Here are the results we first got (where TP stands for True Positive rate, i.e. the percentage of valid IPs that would indeed be allowed, and FP is the False Positive rate, i.e. the percentage of spoofed IPs that would be allowed):

u	TP	FP
3	8.27%	1.94E-4%
5	7.41%	1.84E-4%
10	4.38%	1.63E-4%

We can see that the percentage of valid IPs allowed to pass is not very good (8.27% at best). This can be explained by the fact that we only got a training set of 3 hours, compared to 14 days in the article. We can suppose that with a better training set, we would have approached the results of the article (as their training set is 114 times larger than ours). But given that the false positive rate is very low, we also thought about an improvement to this algorithm : when the training set is small (as in our case), instead of storing single IP addresses in the database, store its subnet and during an attack, allow any IP belonging to a subnet contained in the database. This will significantly increase both the TP and FP rates : indeed, with $u = 3$, we got a TP rate of 15.02% and a FP rate of 0.031%. In the end, we got a TP rate two times better, and a FP rate 160 times worse. But as the FP rate is still very low, this result can be considered as better than the previous one, as we will still block most of the attack traffic, but we will allow two times more legitimate clients to access our service under attack.

As a further work, we would like to test this algorithm on bigger and more various data in order to verify our extrapolations. Moreover, we would like to test it on data containing the TTL field of the received packets in order to be able to compare the results with the Hop-Count Filtering algorithm explained previously in this chapter.

In the shoes of the bad guys

Now that we know more about DoS protection services, we can study the following question : could "bad guys" have an interest in using such services ? And if yes, what would be the gain ?

6.1 Experimental results

To answer this question, we ran the previous `dig_program/dig_alexas.sh` (that we used on the Alexa websites ranking in the introduction), but this time with a big list of phishing websites as input. Why phishing websites ? Because phishing is a very frequent attack vector on the Internet, and it requires the attacker to host some content on his own website, making it possible for him to use a DoS protection service.

The list of phishing websites was retrieved from <http://data.phishtank.com/data/online-valid.csv>, which is a list of websites that were manually confirmed as phishing websites, and that was filtered out to keep only websites still online (even if this was not completely accurate, as only 90% of them were still online when we ran our program, see later). This list is available in `dig_program/input/verified_online_stats`. We used our program to check how many of those 28980 websites were protected by each studied DoS protection service. Here are the results:

- Websites under Cloudflare protection : 590 (189 different domains)
- Websites under Akamai protection : 23 (4 different domains)
- Websites under Incapsula protection : 2 (2 different domains)

We can already observe that a lot of phishing websites subscribed to Cloudflare ($\sim 2\%$ of them), while almost none of them did it with other services. And actually, after manually reviewing the websites under Akamai and Incapsula protection, we noticed that they were "corner cases", i.e. false positives. For example, in Akamai, we find `onedrive.live.com` and `sharepoint.com`,

that are websites that allow the user to host some content on Microsoft servers (that are themselves under Akamai protection). We also have `terbish.com`, that seems to be a legitimate website that got hacked, allowing a hacker to put some phishing content on it, which resulted in this website being listed on phishtank even if it's not a real phishing website. In the end, we came to the conclusion that the only DoS protection service to which they really subscribed is Cloudflare.

6.2 Interpretation

These results can be explained by two main reasons. First, Cloudflare is the only service (among the 3 mentioned) providing a *free* and *automated* service: without paying anything, your website is protected within 5 minutes following the subscription. This is very convenient for "bad guys": indeed, with Incapsula the cheapest plan costs 59\$ per month, and Akamai is even more elitist: you have to contact them to have informations about their pricing plans. Secondly, Cloudflare completely hides your IP by default (as they become your free DNS provider), unlike the two other services where this feature is rather an extra.

So in conclusion, using Cloudflare allows them to freely hide their true IP address behind an IP belonging to Cloudflare. This has two advantages: first, it can allow them to bypass IP blacklists for a while, since they receive a fresh IP from Cloudflare at subscription. Moreover, this IP will be categorized as American, which can be more subtle than their true IP. Secondly, as their true IP is hidden, it becomes more difficult for their enemies (authorities, people who got caught or others) to locate and identify them: they would have to collaborate with Cloudflare (to get the true IP address) and then with the corresponding ISP (to get their real identity). It adds a "layer of anonymity".

Note that those results don't only hold for phishing websites. As an example, we can take the booter/DDoS website `booter.xyz` that is actually a client of Cloudflare !

In the following sections, we will try to answer these two questions :

- Does the lifetime of a phishing website increase thanks to Cloudflare protection ?
- Can we track phishing websites under Cloudflare protection ?

To answer these questions, we ran a little script called `phishing_program.sh` every day during 30 days (and then once again after 90 days for the first question). The input was, on one side, the list of the 189 phishing websites protected by Cloudflare, and on the other side, a list of 984 randomly chosen phishing websites that were not protected by Cloudflare. Every day, we checked if those websites were still online, what were their IP addresses, and so on.

6.3 Lifetime of a phishing website under Cloudflare protection

Among the 189 retrieved phishing websites, only 170 were still online on day 1. This can be explained by two reasons : first, the list from PhishTank was not necessarily totally up to date. Secondly, our definition of "online" is not necessarily the same as them : we define a website as "offline" if it doesn't respond to a curl query within 15 seconds, after 2 tries. After 30 days, 168 of them were still online. And after 90 days, 164.

Among the 984 selected phishing websites not using Cloudflare, 893 were still online on day 1 (for the same reasons as in the previous paragraph). After 30 days, 863 were still online. And after 90 days, 803.

So, what can we conclude from these results ? Let's first use percentages (compared to day 1) in order to have a general idea:

After	30 days	90 days
C	98.8%	96.5%
NC	96.6%	89.9%

At a first glance, we can conclude that there is a difference between those two proportions. But is this difference really significant ? In order to answer this question, we need to use a statistical test.

6.3.1 Statistical interpretation

We have two categorical variables X and Y, each of which has two subcategories : X (under Cloudflare protection [CF] or not [NCF]) and Y (still online or not after three months). The results we obtained can be presented in a two-way contingency table:

X	Y		
	online	offline	
CF	164	6	170
NCF	803	90	893
	967	96	1063

In order to know if the two variables X and Y are independent, or more precisely to see if the proportion p_1 of phishing websites under Cloudflare protection still online after three months is equal to the proportion p_2 of other phishing websites still online after three months, we can use a chi-square test of association.

The null hypothesis is $H_0: p_1 = p_2$.

In a chi-square test, the alternative hypothesis is $H_1 : p_1 \neq p_2$.

The expected table (obtained under the null hypothesis) is given by

X	Y		
	online	offline	
CF	154.65	15.35	170
NCF	812.35	80.65	893
	967	96	1063

The value of the chi-square statistics (i.e. the chi-square distance between the observed table and the expected table) is 7.4556.

Since the p-value obtained by performing the chi-square test (0.006324) is less than the significance level (α , chosen as 0.05), we reject H_0 and we conclude that the proportions p_1 and p_2 are significant. (We can even say that they are *very significant* because H_0 is still rejected with

$\alpha = 0.01$.)

The conditions to perform a chi-square test are fulfilled because the size of our sample is 1063 (larger than 30) and all the counts are at least 5 in each cell of the expected contingency table.

An alternative way to perform the same test is to use the Fisher exact probability test. That test is not based on a statistics whose law is known when n is sufficiently large, but it computes directly an exact p-value.

If we use the same hypothesis (bilateral H_1), the p-value is quite the same (0.00505). So the conclusion is the same. But in our case, it is more relevant to use a unilateral test because we want to know if $p_1 > p_2$. So we perform the Fisher test with the alternative $H_1 : p_1 > p_2$. The p-value is 0.0025, which is even better than the previous result. So we again reject H_0 and conclude that $p_1 > p_2$, the result being very significant again. In the end, we conclude that the proportion of phishing websites under Cloudflare protection still online after three months is very significantly larger than the proportion of other phishing websites still online after three months. [Fis70]

However, it should be noted that other factors could have influenced those results. For example, quality phishing websites will maybe be more likely to subscribe to Cloudflare, so they could have lived longer even without Cloudflare. Secondly, there are inevitably some false positives in the Cloudflare list (websites like `onedrive.live.com` with Akamai) that have in consequence no reason to be shutdown, disturbing a little bit the results.

6.4 Tracking phishing websites under Cloudflare protection

As explained earlier, when you subscribe to Cloudflare, the DNS server of your website is replaced by the DNS servers of Cloudflare, and the visitors will never directly reach your website: all the communications are done through a Cloudflare data center. In consequence, the visitor will *never* have access to the real IP address of your server (and should, theoretically, never have access to any other information about the server hosting your website). Nevertheless, there are some tricks to bypass this :

- Some websites want to keep their DNS server. The only alternative they have is to use a CNAME to Cloudflare. The problem is that, theoretically, they can't put a CNAME on your root domain : the best they can do is to put it on the `www` version of their website (i.e. if your domain is *domain.com*, the CNAME covers *www.domain.com* but not *domain.com*). In consequence, a DNS request to the root domain would reach the original server, and give its real IP address. Nevertheless, Cloudflare recently implemented an algorithm called *CNAME flattening* allowing to "virtually" put a CNAME on the root domain while staying RFC-compliant [Clo16i], but it's not used on all Cloudflare's customers yet.
- If you request some dynamic content (or static content that was not in cache in the Cloudflare data center hit, or if you use a `no-cache` HTTP header), then you can get an idea of the geolocalisation of the server by measuring the latency of your request (it will be routed to the closest Cloudflare data center, then to the original website, and then it will take the reverse path up to you).
- If the website unsubscribes from Cloudflare (in general unwillingly, because the authorities noticed that the website was illegal and asked Cloudflare to stop protecting it), then we can get access to the real IP address of the website.

This third point can actually be pretty interesting under the following hypothesis : "if a server is hosting a phishing website, then it's likely that it's hosting other malicious websites as well". We were able to verify this hypothesis experimentally by taking our list of 984 non-Cloudflare phishing websites (together with their IP addresses on day 30 - `nocloudflare-30.txt`), and creating a Map of those websites indexed by IP address. We noticed that a little bit less than 50 IP addresses were shared by at least two different phishing websites, and six of them were shared by more than five different phishing websites (see `phishing-mapping.txt`).

But for example, among those six IP addresses (98.139.135.129, 74.125.136.132, 212.83.173.208, 141.8.224.221, 212.83.128.22, 98.139.135.198), only half of them (the first, third and fourth ones) were blacklisted (and on only one of the many available lists on the web : either `sbl.spamhaus.com` or `bl.spamcannibal.org`). We can already see that there is a problem here : we were able to identify several malicious IP addresses not listed anywhere on the Internet. Now, it will be way more difficult when the websites are hidden behind Cloudflare... But before going further, let's explain two new notions :

- We say that an IP address is blacklisted if it appears on some well-known IP address blacklists on the web. In our case, we use `http://whatismyipaddress.com/blacklist-check` (one of the several available blacklist checkers on the web), and we will pay more attention at `zen.spamhaus.org`, which regroups all the different blacklists of Spamhaus (IP addresses of spam emails, of hijacked machines, and more importantly, of domains found in spam emails: these are in general phishing domains).
- We call *domain neighbor* of some domain X, another domain Y that shares the same IP address as X. There is no general protocol allowing to find the domain neighbors of a particular domain (reverse DNS does not do that), but some websites can detect some of them by caching forward DNS requests and storing them in some kind of hash table indexed by IP addresses. In our case, we use `http://www.tcpiputils.com/browse/ip-address` to find the domain neighbors of a particular domain.

We could verify our hypothesis pretty quickly with non-Cloudflare phishing websites : in general, they are listed on most IP blacklists, and when you look at their domain neighbors, you can often discover several other phishing websites (that are also listed, as they share the same IP).

Now for phishing websites protected by Cloudflare, it's harder : in general, they are *not* blacklisted, as they are associated to a Cloudflare IP, that is in general clean as explained previously. Nevertheless, if the website unsubscribes from Cloudflare, the real IP address of the attacker's server is exposed. Nevertheless, this IP address is in general not blacklisted neither : this could either be due to the fact that this website was the only malicious one hosted on that server, or to the fact that the attacker was careful to hide all of his malicious websites behind structures like Cloudflare.

In both cases, we now have the IP address, that was our primary goal. But we can do more : we can check the domain neighbors of the unsubscribed website, i.e. the domains hosted on the exposed IP address. This could allow us to (1) discover new phishing websites not yet known by traditional filters, and (2) discover more informations about this server : for example, the attacker could also be hosting legitimate websites on the same server, and this could help us to gather informations about the true location of this server.

Now, let's apply these ideas to our data. As said earlier, we studied 189 phishing websites under Cloudflare protection. Every day during 30 days, we logged the IP address(es) of each of those

websites (the root and the www version). This allowed us to check if they were still behind Cloudflare, and if not, to get their real IP addresses.

Among the 189 websites, during these 30 days, 14 of them lost the subscription at Cloudflare at least one day, and 9 of them never recovered it. We then had to make the difference between real phishing websites, and websites that just got hacked / that were used to host some phishing content during a short time. In the end, we only got two real phishing websites having completely lost their Cloudflare subscription:

- helpcentre.us

http://www.phishtank.com/phish_detail.php?phish_id=3537889

This website is clearly a phishing website, trying to steal your Netflix account. It was first totally hidden behind Cloudflare, but then, on day 5, it lost its Cloudflare subscription and its real IP address got exposed : 173.201.1.16. After 30 days, the website is still online and still under this same IP address, but this IP is not blacklisted on the web ! It has two domain neighbors: <https://manshaa.com/> and <https://zeecomp.com/>, but it's difficult to say if they are legitimate or not.

- www.maugoyang.ga

http://www.phishtank.com/phish_detail.php?phish_id=3795309

This website is a fake Facebook in a foreign language, but was shutdown pretty fast. What is interesting here is that from day 7 on, the IP address changed from 68.65.120.217 to 141.8.224.221. While both IPs are not blacklisted, we noticed that the second one was used by a lot of other Swiss phishing websites not under Cloudflare protection (for example : dcccfile.ga), but it seems like they are all shutdown today.

In conclusion, the data collected allowed us to validate our hypotheses, but unfortunately we couldn't really go further due to the fact that most phishing websites never lost their subscription to Cloudflare.

Conclusion

At the beginning of this thesis, we tried to raise awareness about the potential consequences of DoS attacks. Nowadays, Denial of Service is part of the Internet, and you have to act accordingly if you want to stay competitive. Indeed, every minute of uptime lost can cause huge revenue losses, but also alter the public image of your company. In consequence, we highlighted the necessity for sensible companies - even small ones - to protect themselves against such attacks.

There are basically two ways to protect yourself against DoS attacks. Firstly, you can try to mitigate the attacks yourself, for example by using general mechanisms such as those described in Chapter 4, or even some of the more advanced mechanisms of Chapter 5. The problem is that on-premise solutions can quickly become ineffective when the traffic grows, as even your first line of defense could end up overwhelmed by this huge amount of traffic. Hybrid defenses could overcome it, but they are most of the time pretty challenging to deploy due to the collaboration required with other devices of the Internet.

The alternative is to subscribe to a DoS protection service that will handle everything for you. It's a kind of "cleaning center" that will route all your traffic to their own data centers first, filtering malicious requests before forwarding genuine traffic to your server. Even if you have to pay a monthly subscription to benefit from this service, the ROI will generally be high due to the potential losses of being shutdown by an attack.

In this thesis, we studied two DoS protection services in particular : Cloudflare and Akamai. While both of them are based on the general architecture of a CDN, we noticed several interesting design differences between them. For example, Cloudflare uses Anycast to route requests to the closest data-center, while Akamai chooses it based on the source IP of the end-user. Another important point is that Cloudflare always protects its customers at the DNS level, which means that visitors are unable to discover the real IP of websites under Cloudflare protection. Combined to the fact that Cloudflare offers a free plan to any client without manual verification, it explains why so many malicious websites subscribed to Cloudflare. In Chapter 6, we performed some experiments on phishing websites, showing among others that phishing websites under Cloudflare

protection have a longer life-time, probably due to this additional layer of anonymity.

With the continuous development of the Internet and the generalisation of DoS tools, we can expect DoS attacks to keep growing in number and in traffic in the coming years. If we want to stop them without having to use a protection service, several steps will have to be followed. Firstly, every ISP should implement source-based defenses such as ingress filtering. For example, if we could prevent source IP spoofing on the Internet, we would remove at least one third of the attacks from the equation (i.e. reflection-based attacks), and it would also weaken other kinds of DoS attacks. Secondly, we need to support hybrid defenses, because the best way to defend against such attacks is to collaborate with the multiple devices forming the Internet. Detecting the attack at the destination and propagating appropriate responses upstream to the sources of the attack is currently the most promising solution to mitigate large attacks, but it requires strong cooperation between the destination, the intermediate routers and the different sources of the attack, which is pretty challenging to deploy in practice.

Bibliography

- [AC09] K. Argyraki and D. R. Cheriton. Scalable network-layer defense against internet bandwidth-flooding attacks. *in IEEE/ACM Trans. Netw.*, pages 1284–1297, 2009.
- [AGCLS03] S. Abdelsayed, D. Glimsholt, S. Ryan C. Leckie, and S. Shami. An efficient filter for denial-of-service bandwidth attacks. *in Proc. of the 46th IEEE Global Telecommunications Conference (GLOBECOM03)*, 3:1353–1357, 2003.
- [Akaa] Akamai. Akamai cloud security solution: Comparing approaches for web, dns and infrastructure security.
- [Akab] Akamai. Akamai enhanced dns: Service description.
- [Akac] Akamai. Akamai kona site defender service description.
- [Aka16a] Akamai. State of the internet report - q4 2015 security report. <https://www.akamai.com/us/en/multimedia/documents/report/q4-2015-state-of-the-internet-security-report.pdf>, accessed April 24, 2016.
- [Aka16b] Akamai. Ssdp reflection ddos attacks. <https://www.akamai.com/fr/fr/multimedia/documents/state-of-the-internet/ssdp-reflection-ddos-attacks-threat-advisory.pdf>, accessed May 26, 2016.
- [BC00] H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. *in Proc. of the USENIX Large Installation Systems Administration Conference*, pages 319–327, 2000.
- [BS16] F. Baker and P. Savola. Ingress filtering for multihomed networks. <https://tools.ietf.org/html/bcp84>, accessed April 13, 2016.
- [Cea01] J.B.D. Cabrera and L. Lewis et al. Proactive detection of distributed denial of service attacks using mib traffic variables - a feasibility study. *Integrated Network Management Proceedings*, pages 609–622, 2001.

- [Clo16a] Cloudflare. Advanced ddos protection service. <https://www.cloudflare.com/ddos>, accessed April 11, 2016.
- [Clo16b] Cloudflare. Advanced ddos protection whitepaper. <https://www.cloudflare.com/media/pdf/cloudflare-whitepaper-ddos.pdf>, accessed April 11, 2016.
- [Clo16c] Cloudflare. Architecture of a cloudflare data center. <https://blog.cloudflare.com/cloudflares-architecture-eliminating-single-p/>, accessed April 11, 2016.
- [Clo16d] Cloudflare. Ddos attack against spamhaus (1). <https://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet/>, accessed April 11, 2016.
- [Clo16e] Cloudflare. Ddos attack against spamhaus (2). <https://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho/>, accessed April 11, 2016.
- [Clo16f] Cloudflare. Features security. <https://www.cloudflare.com/features-security/>, accessed April 11, 2016.
- [Clo16g] Cloudflare. Cloudflare virtual dns. <https://blog.cloudflare.com/announcing-virtual-dns-ddos-mitigation-and-global-distribution-for-dns-traffic/>, accessed May 18, 2016.
- [Clo16h] Cloudflare. Single rx queue kernel bypass in netmap for high packet rate networking. <https://blog.cloudflare.com/single-rx-queue-kernel-bypass-with-netmap/>, accessed May 19, 2016.
- [Clo16i] Cloudflare. Introducing cname flattening: Rfc-compliant cnames at a domain's root. <https://blog.cloudflare.com/introducing-cname-flattening-rfc-compliant-cnames-at-a-domains-root/>, accessed May 27, 2016.
- [Cox16] Ben Cox. Detecting anycast addresses. <https://github.com/benjojo/AnyCatch>, accessed April 12, 2016.
- [CP05] R. Chen and J. M. Park. Attack diagnosis: Throttling distributed denial-of-service attacks close to the attack sources. *IEEE International Conference on Computer Communications and Networks (ICCCN'05)*, pages 275–280, 2005.
- [CPM06] R. Chen, J. M. Park, and R. Marchany. Rim: Router interface marking for ip traceback. *IEEE Global Telecommunications Conference*, pages 1–5, 2006.
- [Cs316] Inc Cs3. The reverse firewall: Defeating ddos attacks emanating from a local area network. http://www.cs3-inc.com/pubs/Reverse_FireWall.pdf, accessed May 26, 2016.
- [Dis16] Dennis Distler. Performing egress filtering. <http://www.sans.org/reading-room/whitepapers/firewalls/performing-egress-filtering-32878>, accessed April 13, 2016.
- [Fis70] R. A. Fisher. *Statistical Methods for Research Workers*. Oliver and Boyd, 1970.
- [FS16] P. Ferguson and D. Senie. Network ingress filtering: defeating denial of service

- attacks which employ ip source address spoofing. <https://tools.ietf.org/html/bcp38>, accessed April 13, 2016.
- [Gig16] Gigaom. Record-breaking ddos attack. <https://gigaom.com/2014/02/11/record-breaking-ddos-attack-struck-on-monday-according-to-reports/>, accessed April 11, 2016.
- [GP01] T. M. Gil and M. Poletto. Multops: a data-structure for bandwidth attack detection. *in Proc. of 10th Usenix Security Symposium, Washington*, pages 23–38, August 2001.
- [Hol16] David Holmes. Syn-floods and countermeasures-part 2. <https://devcentral.f5.com/articles/syn-floods-and-countermeasures-part-2>, accessed May 20, 2016.
- [IAN16] IANA. Ingress filtering for multihomed networks. <https://tools.ietf.org/html/rfc3330>, accessed April 13, 2016.
- [Inc16a] Incapsula. Incapsula ddos protection - attack glossary. <https://www.incapsula.com/ddos>, accessed April 23, 2016.
- [Inc16b] Incapsula. Cctv botnet in our own back yard. <https://www.incapsula.com/blog/cctv-ddos-botnet-back-yard.html>, accessed May 26, 2016.
- [Jan14] Matt Jansen. Edns0 client-subnet for dns based cdns, 2014.
- [KLCea06] Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao et al. Packetscore: A statistics-based packet filtering scheme against distributed denial-of-service attacks. *IEEE Trans. On Dependable and Secure Computing*, 3(2):141–155, 2006.
- [KMGG08] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis. Detecting dns amplification attacks. *Critical Information Infrastructures Security Lecture Notes in Computer Science*, 5141:185–196, 2008.
- [LC11] H. I. Liu and K. C. Chang. Defending systems against tilt ddos attacks. *6th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, pages 22–27, 2011.
- [MBF⁺02] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review*, 32(3):62–73, 2002.
- [MPR02] J. Mirkovic, G. Prier, and P. Reiher. Attacking ddos at the source. *Proc. of the 10th IEEE International Conference on Network Protocols*, pages 312–321, 2002.
- [MPR03] J. Mirkovic, G. Prier, and P. Reiher. Source-end ddos defense. *Proc. of the 2nd IEEE International Symposium on Network Computing and Applications*, pages 171–178, April 2003.
- [MR04] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.
- [MRR03] J. Mirkovic, P. Reiher, and M. Robinson. Forming alliance for ddos defense. *in Proc. of New Security Paradigms Workshop*, pages 11–18, 2003.
- [OM09] G. Oikonomou and J. Mirkovic. Modeling human behavior for defense against flash-crowd attacks. *Proc. of the 2009 IEEE international conference on Communications*

(*ICC'09*), pages 625–630, 2009.

- [PBP⁺14] C. Pelsser, R. Bush, K. Patel, P. Mohapatra, and O. Maennel. Making route flap damping usable. IETF RFC 7196, May 2014.
- [PLR03] T. Peng, C. Leckie, and K. Ramamohanarao. Protection from distributed denial of service attacks using history-based ip filtering. *ICC '03*, 1:482–486, 2003.
- [Pro16a] MIT ANA Spoofer Project. Everyone should be deploying bcp 38! wait, they are... <http://www.senki.org/everyone-should-be-deploying-bcp-38-wait-they-are/>, accessed April 13, 2016.
- [Pro16b] MIT ANA Spoofer Project. Spoofer project. <https://spoofer.caida.org/summary.php>, accessed May 27, 2016.
- [Rio16] RioRey. Dtaxonomy of ddos attacks. http://static1.squarespace.com/static/5548bab5e4b08ecb6652391c/t/554923d6e4b03d6d83ac7c5e/1430856662717/RioRey_Taxonomy_DDoS_Attacks_2.6_2015-1.pdf, accessed April 23, 2016.
- [RSU⁺09] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, and E. Knightly. Ddos-shield: Ddos-resilient scheduling to counter application layer attacks. *IEEE/ACM Trans. Netw.*, 17(1):26–39, 2009.
- [Sep16] Timothy J. Seppala. Overwatch' already has 7 million people aiming at your face. <http://www.engadget.com/2016/06/03/overwatch-seven-million-players/>, accessed june 3, 2016.
- [Sys16] Cisco Systems. White paper - defeating ddos attacks. http://www.cisco.com/c/en/us/products/collateral/security/traffic-anomaly-detector-xt-5600a/prod_white_paper0900aecd8011e927.html, accessed April 26, 2016.
- [VCG98] C. Villamizar, R. Chandra, and R. Govindan. Bgp route flap damping. IETF RFC 2439, November 1998.
- [Wik16a] Wikipedia. Cloudflare. <https://en.wikipedia.org/wiki/CloudFlare>, accessed April 10, 2016.
- [Wik16b] Wikipedia. Optical fiber. https://en.wikipedia.org/wiki/Optical_fiber, accessed April 12, 2016.
- [Wik16c] Wikipedia. Egress filtering. https://en.wikipedia.org/wiki/Egress_filtering, accessed April 13, 2016.
- [Wik16d] Wikipedia. Denial of service attack. https://en.wikipedia.org/wiki/Denial-of-service_attack, accessed April 22, 2016.
- [Wik16e] Wikipedia. League of legends. https://en.wikipedia.org/wiki/League_of_Legends, accessed June 3, 2016.
- [Wik16f] Wikipedia. Syn cookies. https://en.wikipedia.org/wiki/SYN_cookies, accessed May 27, 2016.
- [Wik16g] Wikipedia. Firewall (computing). [https://en.wikipedia.org/wiki/Firewall_\(computing\)](https://en.wikipedia.org/wiki/Firewall_(computing)), accessed May 6, 2016.

- [Wik16h] Wikipedia. Intrusion detection system. https://en.wikipedia.org/wiki/Intrusion_detection_system, accessed May 6, 2016.
- [WJS07] H. Wang, C. Jin, and K. G. Shin. Defense against spoofed ip traffic using hop-count filtering. *IEEE/ACM Trans. On Networking*, 15(1):40–53, 2007.
- [WVB⁺06] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. Ddos defense by offense. *SIGCOMM Computer Communications Review*, 36(4):303–314, 2006.
- [YFLL09] J. Yu, C. Fang, L. Lu, and Z. Li. A lightweight mechanism to mitigate application layer ddos attacks. *Proc. of the 4th International ICST Conference*, pages 175–191, 2009.
- [YLCC07] J. Yu, Z. Li, H. Chen, and X. Chen. A detection and offense mechanism to defend against application layer ddos attacks. *The third International Conference on Networking and Services (ICNS’07)*, page 54, 2007.
- [YPS03] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against ddos attacks. in *IEEE Symposium on Security and Privacy*, pages 93–107, 2003.
- [YPS04] A. Yaar, A. Perrig, and D. Song. Siff: a stateless internet flow filter to mitigate ddos flooding attacks. in *Proc. of the 2004 IEEE Symposium on Security and Privacy*, pages 130–143, 2004.
- [ZJT13] S.T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE Communications Surveys and Tutorials*, 15(4):2046–2069, 2013.

List of the most famous DDoS attacks

Note that all the attacks listed below are *DDoS flooding attacks*. The other category of DDoS attacks is called *vulnerability attacks* : in general, it consists in sending a malformed packet to the victim in order to confuse some particular protocol or application. Vulnerability attacks are developed here as they are not very common and in general easily manageable. We can still cite the most notable, such as Ping of Death (which consists in sending malformed ping packets) or the Teardrop attack (which uses fragmented IP packets that are overlapping and over-sized).

Among DDoS flooding attacks, we can distinguish two big subcategories : attacks on the network or transport layer of the OSI model, and attacks on the application layer [ZJT13]. The following attack descriptions were inspired from multiple sources [MR04] [ZJT13] [Rio16] [Inc16a].

A.1 List of the network/transport-level DDoS flooding attacks

A.1.1 Flooding type of attacks

Here, the victim is a particular server, and the attacks will in general target the bandwidth of the victim, but other network resources can also be affected, such as router or server processing capacity. Source IPs can be spoofed or not.

UDP floods:

The attacker sends a lot of UDP packets to random ports of the victim. For each received packet, the victim will then spend some time to understand that (in most cases) no application is running on that port, and will then have to send an ICMP "destination unreachable" packet back to the source.

ICMP (ping) floods:

The attacker floods the victim with ICMP Echo Request packets, for example using the

ping utility. If the victim responds with ICMP Echo Reply packets, this attack will consume both incoming and outgoing bandwidth from the victim.

DNS floods:

Specific case of the UDP flood. Here the attacker will send a lot of legitimate DNS requests to a specific DNS server in order to overwhelm its resources and make it unable to process other requests. Can target the DNS server's processing capacity, or the network link to the DNS server (with larger packets).

DNS NXDOMAIN:

It's a particular case of the DNS flood where the domains queried are non-existent. This one targets CPU in particular (it takes some time to understand that the domain doesn't exist), and it can also target the cache (the cache could fill with those useless requests).

VoIP floods:

Again, it's a specific case of the UDP flood where the attacker sends a lot of forged VoIP packets to a specific VoIP server in order to overwhelm either its bandwidth, or rather its CPU (since the server has to spend some time to understand if the connection is forged or not).

A.1.2 Protocol exploitation flooding type of attacks

Here, the victim is again the server, and attacks can again target different network resources, but bandwidth will be less affected since packets are in general smaller. Source IPs can be spoofed or not.

TCP SYN flood:

The attacker sends a lot of SYN requests to a victim server, without completing the three-way handshake. This will let a lot of connections half-opened on the server, and could in the end exhaust its resources (memory and/or CPU), making it unresponsive to legitimate traffic.

TCP SYN-ACK flood:

The attacker sends spoofed SYN+ACK packets at very high rate to a victim server. The server is going to dedicate a lot of processing power to understand that it's not valid (not corresponding to a previous SYN packet sent by itself). Moreover, the server has to send a RST packet back to the attacker.

ACK & PUSH ACK flood:

The attacker sends ACK packets to the victim server, which has to spend some CPU to try to match it. He can also set the PUSH flag in order to force the server to process the information inside the packet.

RST/FIN flood:

Attackers send a lot of RST (reset) or FIN (to end a TCP connection) packets from some spoofed IP addresses. It results in all the victim's resources being used to look for a match between those incoming packets and an existing TCP connection, which actually doesn't exist.

A.1.3 Reflection-based flooding type of attacks (a.k.a DrDoS)

Reflection-based attacks are usually accompanied by amplification. For this type of attack, source IPs have to be spoofed (using the IP of the victim). Note that some of these attacks could also fit in the Application-level category, but they share more features with the network/transport-level attacks. Due to possibly high amplification factors, these attacks can consume huge amounts of bandwidth.

Smurf attack:

The attacker sends a large amount of ICMP packets with the intended victim's spoofed source IP to a router's broadcast address within a computer network. Most devices on that network will, by default, respond to this IP address which will result by overwhelming the victim. Note that the broadcast feature allows the attacker to create an amplification phenomenon. Note that nowadays, most networks are immune to this attack (routers won't respond to this kind of traffic anymore).

Fraggle attack:

Same principle as the Smurf attacks but this time the attacker uses UDP traffic on port 7 (Echo protocol) or port 19 (CHARGEN protocol). In the case of port 19, the Character Generator Protocol will generate a heavy response containing a lot of characters, creating even more amplification.

DNS amplification attack:

An attacker sends a DNS query with a spoofed source address to a DNS server. This server will respond to this spoofed IP, overwhelming it with unwanted DNS informations. In general, the DNS responses are bigger than the requests, thus creating amplification. The attacker creates even more amplification by requesting big responses (using for example the ANY flag). This attack requires to have a lot of open recursive resolvers available : see Subsection 1.7.1 for more informations about the problem.

Network Time Protocol (NTP) attack:

NTP servers are used to synchronise multiple network client clocks. It also uses UDP packets and contains a monitoring command called `moonlist`. This command returns the addresses of the last machines that the NTP server has interacted with (up to 600 machines). Which means that by using reflection, an attacker can generate traffic with a huge amplification factor with a simple command. Similarly to the open DNS resolvers problem, there is also a urgent call to update publicly available NTP servers. According to the Open NTP Project, at the time of this writing, still more than 4 million unique IPs are still vulnerable to the `monlist` vulnerability.

SSDP (or UPnP) attack

SSDP stands for Simple Service Discovery Protocol. It's the basis of the discovery protocol of Universal Plug and Play (UPnP) whose goal is to allow devices in residential or small office environments to discover themselves without the need of server-based configuration mechanisms. It was described in an IETF Internet Draft in 1999, but although this draft has since expired, this protocol is still used in a lot of devices.

The SSDP protocol relies on multicast on IP address 239.255.255.250 at UDP port 1900. It allows devices to send a NOTIFY HTTP method to the multicast group in order to announce or withdraw themselves. Then, a client can use the M-SEARCH method to the

multicast group to discover available services on the network. The consequence is that all available services (that listen to the multicast address) will answer that request via unicast to warn the source of the M-SEARCH of their presence. The problem is that there is currently a vulnerability in a lot of home routers that allows an attacker to query home networks via those home routers, allowing them to receive all the responses on the source address of the query - that can then be spoofed with the IP address of the victim, creating a reflection attack with huge amplification [Aka16b].

SNMP attack:

Another reflection attack using the Simple Network Management Protocol, a protocol used for collecting and organizing information from network devices (e.g. servers, routers, printers).

A.2 List of the Application-level DDoS HTTP flooding attacks

In general, application-level DDoS flooding attacks are more stealthy and consume less bandwidth than network/transport-level attacks, and they will often target a particular resource of the victim server (for example, its thread pool).

In this category, attacks are based on the HTTP protocol which requires a successful handshake. In consequence, source IPs can't be spoofed. These attacks will target web server resources.

A.2.1 Session flooding attacks

Excessive VERB (a.k.a HTTP GET/POST flood):

The attacker uses bots to generate a large amount (usually at a higher rate than 10 requests per second) of non-spoofed valid GET request to the victim web server. The attacker usually uses GET requests but he can also use HEAD, POST, PUT, OPTION or any other valid HTTP request to perform the attack.

A.2.2 Request flooding attacks

Excessive VERB single-session (a.k.a single-session HTTP GET/POST flood):

Variant of the excessive VERB attack which exploits a feature of HTTP 1.1 to allow multiple requests within a single HTTP session.

A.2.3 Asymmetric attacks

Multiple VERB single request (a.k.a multiple HTTP GET/POST flood):

Variant of the excessive VERB attack, each packet will contain multiple requests. All those requests are then sent simultaneously which means that the attacker will be able to maintain a high load on the victim while keeping a low attack packet rate, bypassing most anomaly detection tools.

Faulty application:

The attacker takes advantage of faults in the website design by, for instance, employing

SQL-like injections to generate useless requests to lock up database queries. This kind of attack can quickly use all the server resources (CPU, memory and so on).

A.2.4 Slow request/response attacks

These attacks are generally based on the fact that Apache opens a new thread for each connection, and that connections are maintained as long as there is some traffic being sent. This can be used to exhaust, for example, the thread pool of the server.

Slow read attack (a.k.a slow response attack):

The attacker will simply open a lot of different connections, and prevent the server from closing them by sending packets very slowly across each of them. It's called "slow read" because the attacker fakes to slowly reading the responses of the server, e.g. by advertising a very small TCP receive window size, and by emptying the TCP receive buffer of the client slowly.

Slowloris attack:

The attacker sends HTTP requests with a partial set of headers that slowly updates but the request never completes. The victim server will keep many connections open until it reaches its maximum threshold, resulting in denying new and legit connections.

HTTP fragmentation attack:

The goal is the same as for Slowloris : attackers establish valid HTTP connections with the victim server, then they fragment legitimate HTTP packets into little packets which will be sent as slow as the server connection time-out allows.

Slowpost attack (a.k.a slow request bodies or R-U-Dead-Yet (RUDY) attack):

Named after an album of the melodic death metal band *Children of Bodom*, this attack is similar to the Slowloris attack. First, the attacker needs to detect embedded web forms in the victim's website. Once a form has been found, the attacker will send a valid HTTP POST request with a long content-length header field in order to prevent the server from closing the connection prematurely. But then, the payload is injected very slowly (for example, one byte every 10 seconds).

The IP spoofing problem

B.1 Introduction and motivations

The purpose of this annex is to study more deeply the problem of source IP spoofing. As a reminder, it consists in modifying the source IP field in the header of a packet, in order to replace the real source of the packet by either a random IP address (so that the receiver can't identify the real source), or by the IP address of a specific victim (in the case of reflection attacks). This problem is very important because, if we were able to completely solve it, we would eradicate a large range of DoS attacks :

- Reflection-based attacks would completely disappear, as they rely on the fact that the source IP address has been replaced by the victim IP address. And as seen in Chapter 1, these ones represent more than 30% of all DoS attacks.
- Other types of network/transport-level DoS attacks would become way less powerful as every packet coming from an attacker would contain the same source IP : to succeed, the attacker would need to have a very large botnet in order to bypass trivial defenses (such as dropping all packets from an IP that sends them at a suspect, too high rate)¹
- Nevertheless, it would not have much impact on application-level DoS attacks, as most of them require to successfully complete a three-way handshake, which is practically impossible if the source IP is spoofed. Nevertheless, those only represent a small portion of usual DoS attacks (approximately 3% as seen in Chapter 1).

Moreover, as a general matter of fact, DoS attacks would become weaker due to the fact that the sources of the attack could now be identified more easily :

- Before the attack by cross-checking incoming IPs with well-known blacklists on the web;

¹However, even without source IP spoofing, if the traffic is heavy enough, such attacks could still work against destination-based only defenses by overwhelming the first line of defense such as a firewall containing the filtering rules.

- During the attack by dynamically creating filters for the addresses involved in the attack;
- After the attack by using attack logs to improve the blacklists for later use.

In conclusion, solving this problem would not solve the general DoS problem, but it would certainly weaken a large part of them, and even eradicate the most threatening ones (i.e. reflection-based attacks, such as the well-known DNS amplification attack). So there is a real interest in studying this problem, and in deploying appropriate solutions.

B.2 Proposed solution

One general solution was proposed to solve this problem : *ingress filtering*. Basically, it allows routers implementing it to drop incoming packets if they don't come from the network associated with their source IP address. It's a general solution that can be applied on any router in the Internet and it will be more detailed in Section B.2.2. In the particular case of a router connecting a private network with its ISP, we call it *egress filtering* : it will ensure that outgoing packets (i.e. packets going from a private network to the Internet) have, among others, a valid source IP address. But in the case of *egress filtering*, we can do other verifications too, as we will see in Section B.2.1. An alternative way to perform *egress filtering* in a small network is to use the well-known Network Address Translation (NAT).

B.2.1 Egress filtering

This technique can be implemented in routers, firewalls or more generally any edge device between a private TCP/IP network and the Internet [Wik16c] [Dis16]. In general, it allows to deny packets leaving the private network if they don't meet some security policies that have been defined by the network administrator. These security policies can be based on the source IP, but also on the protocol (authorize only HTTP, email and DNS traffic for example) and a lot of other informations. In our case, it will help preventing packets with a spoofed source IP address from leaving the private network and accessing the Internet. The first and most important rule is the following :

Rule 1. Deny any packet whose source IP address is not part of the IP address block allocated to the private network.

This rule is easy to deploy and allows to make sure that, if an attack is launched from the private network, the victim will directly be able to identify that this network is the source of the attack, which will greatly simplify its mitigation (via adequate filters, and also by contacting the administrator of the network to explain the situation). Moreover, it prevents a reflection attack to be launched from that private network (or the attack would be redirected to its source, which would be pointless in most cases). This is theoretically enough to stop the most annoying attacks, but we thought about other rules that could add more security to the first one :

Rule 2. Also deny any packet whose source IP address is not currently allocated to a device by the private network.

This rule can be easily implemented with the cooperation of the DHCP server. It prevents the attacker from randomly spoofing the source IP with addresses inside the block allocated to the private server, since in general most of them won't be in use at that moment. In the end, it restricts the range of IPs that an attacker inside the private network can actually spoof. But we can do even

better :

Rule 3. Also check that the source IP address was indeed allocated to the source MAC address figuring on the packet header.

Using this third rule, an attacker that would stand in the private network and wanting to spoof his source IP address would need to know an IP address currently allocated, and the MAC address of the machine associated with this IP. It becomes way more complicated for the attacker to spoof an IP address from this private network. Nevertheless, some problems still remain :

1. The attacker could eavesdrop the network in order to get pairs of (MAC, IP allocated), which is the only informations he needs to impersonate another connected device;
2. The attacker could spoof his MAC address before asking a new IP address to the DHCP server, in order to create forged (MAC, IP) pairs.

But we can try to find counter-measures, such as ensuring that eavesdropping is impossible on the network, and registering the MAC addresses in advance in order to be able to get an IP address on the network.

B.2.2 Ingress filtering

As explained in the previous section, *egress filtering* allows to prevent outgoing packets from using a spoofed source IP address. In other words, it allows to avoid spoofing attacks *from* your network, but it doesn't protect it from incoming spoofing attacks. In order to protect your network against incoming spoofing attacks, or more generally to prevent source IP spoofing on the Internet, a more general mechanism is required. In this section, we present the most famous : *ingress filtering*, as defined in BCP 38 [FS16] and BCP 84 [BS16] (corresponding respectively to RFC 2827 and RFC 3704).

Unlike egress filtering that is implemented by private networks on their edge devices, ingress filtering is implemented by ISPs at most of their edge routers. The general methodology is the following : when a router receives a packet on some interface, it should check if the source IP address of that packet corresponds to the block of IP addresses associated with that interface. If it's not the case, the source IP address is certainly spoofed, and the packet should be dropped (possibly with a packet back to the sender warning him about that drop).

Now the remaining question is : how to determine the block of IP addresses corresponding to some particular interface ? Several methods exist to answer this question, and the most famous are listed below :

Ingress Access List It's a list of acceptable prefixes that has to be manually configured. The advantage is that it's deterministic, but it does not scale with large ISPs and numerous possible updates, so it's not advised (even if it was the one proposed in BCP 38).

Strict Reverse Path Forwarding The list corresponds to all IP addresses that are present in the Forwarding Information Base (FIB) and whose interface in the FIB (the interface which would have been used to forward traffic to the source of the packet) corresponds with the interface on which the router received the packet. The advantage is that it's dynamic (no manual configuration needed), but problems can arise when routing is not symmetrical : it will in general not be used between ISPs or with multihomed stub networks.

Feasible Path Reverse Path Forwarding Same as Strict RPF, but this time the router will use

the N best routes (where N is a parameter) instead of only the best one, for example by using the Routing Information Base (RIB). This method allows to handle pretty well asymmetric routing and multihomed stub networks.

Loose Reverse Path Forwarding Same as Strict RPF, but even more relaxed than Feasible RPF in the sense that it only checks for the existence of a route, without verifying if the interface matches for one of those routes. In the end, this method will only filter non-routed and special-use IP addresses (as defined in RFC 3330 [IAN16]).

B.3 Current and future deployment

Now comes the question of the deployment : why isn't it already implemented by everyone ? The problem is that it's not mandatory : ISPs can decide themselves if they want to deploy it or not. So let's try to compare the advantages and the drawbacks of deploying it for ISPs :

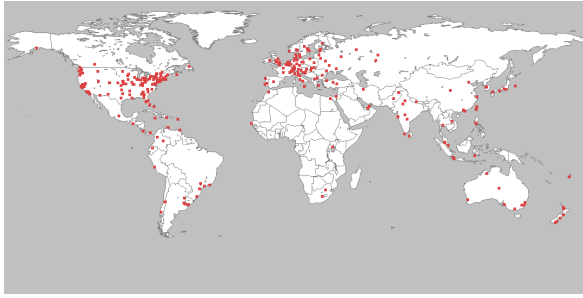
Pros

- It's a good neighbor policy, which relies on cooperation between ISPs for their mutual benefit.
- It's good for your reputation : it avoids other ISPs or enterprises to undergo an attack because of you.
- It should save you some money in the long term (better corporate image).
- It could save you some bandwidth by dropping packets as close as possible from the source.

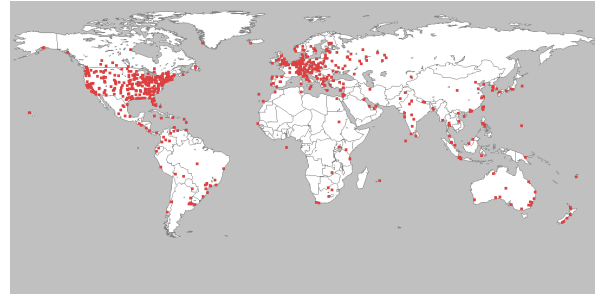
Cons

- It does not really benefit you directly, but it rather benefits your competitors.
- It has deployment and maintenance costs (at least in the short term).
- It should ideally be implemented for both IPv4 and IPv6.
- It could generate a small latency increase, and is not usable in MPLS networks.
- There are legitimate cases where IP spoofing is necessary, for example to stress test a website as if it was live.

The general opinion is that only a small part of the Internet actually deployed BCP38 (and, by extension, BCP 84), but surprisingly, according to a report from the MIT ANA Spoofer Project published in 2012 [Pro16a], "over 80% of the Internet cannot be used to send spoofed IP packets". This is good news, but not enough to erase spoofing attacks : the remaining 20% of the spoofable Internet is used daily by attackers, and it is the reason why we still see for example huge reflection attacks of several hundreds of Gbps nowadays, such as the attack against Spamhaus in 2013 [Clo16d] [Clo16e]. For information, Figure B.1 gives us an insight about the locations of networks not implementing ingress filtering.



(a) Locations of spoofable networks



(b) Locations of tested networks

Figure B.1: Those figures come from the MIT ANA Spoofer Project [Pro16b]

B.4 Prevent source IP spoofing in an enterprise Wi-Fi network

In private Wi-Fi networks, the protection mechanism commonly used is WPA-Personal (a.k.a. WPA-PSK that stands for pre-shared key). It doesn't prevent source IP spoofing, but in general it's used together with NAT, which solves the problem.

For enterprise Wi-Fi networks, you don't want to use NAT, but the problem will this time be solved via the protection mechanism used : WPA-Enterprise (a.k.a. WPA-802.1X). In this mode, there are basically three important parts :

- A RADIUS server that will authenticate the client (based on his own username and password) and generate a key distinct for each user and refreshed every hour;
- A DHCP server that will allocate the client his own IP address;
- A general Controller that contains a triplet of informations for each connected client : (IP address allocated, MAC address, key).

This means that if you try to spoof your IP address only, the Controller will notice that it doesn't correspond to your MAC address and will drop the packet. If you try to spoof both your IP and MAC address, the Controller will notice that this pair doesn't correspond to any key and will also drop the packet. Even if you know the username and password of someone else behind this network, you can't spoof his IP address because you won't receive the same key as him. In order to spoof his IP address, you would need to know his secret key, that is changed every hour. In the end, it's practically impossible to launch a spoofing attack from that kind of infrastructure.

In order to verify that, we did some experiments in the UCL network by using the tool *Scapy* to forge packets and an Amazon EC2 instance to check if the spoofed packets were actually leaving the UCL network. The detailed commands figure in the last annex, Section C.2. In the end, we never received any spoofed packet on the Amazon EC2 instance, proving that this protection mechanism works well.

B.5 Conclusion and alternatives

In conclusion, we saw that *egress filtering* allows stub networks to avoid being the source of spoofing attacks, and *ingress filtering* allows ISPs to avoid forwarding spoofed packets (either directly originating from a stub network not using egress filtering, or from another ISP not using ingress filtering itself). But we also saw that those two entities have few interest in doing such filterings. There is are options for networks wanting to defend themselves against spoofing attacks targeting them. For example, filtering IPs at the destination (current well-known techniques include *History-Based IP Filtering* and *Hop-Count Filtering*), or traceback mechanisms (that unfortunately require cooperation with upstream routers), but we won't go deeper here as it's the subject of Chapter 5.

Code

As explained in the introduction, all the codes written for this thesis are available on BitBucket under the public repository <https://bitbucket.org/Sihardy/dos-master-thesis>. This annex aims at explaining non-trivial scripts that were not already discussed in the thesis itself.

C.1 dig_program

This section explains in details the functioning of the Bash script `dig_alexas.sh` in the `dig_program` folder. This program was first used to plot the graphs of the introduction, but was also used afterwards for different purposes such as the experiments on Cloudflare data centers and the analysis of phishing websites (i.e. to check if they subscribed to Cloudflare).

In short, this script will take a file containing a list of domain names as argument, and will output one file by CDN, containing all the domains of the input that are associated with this CDN. How does it work ? It will make several `dig` requests to each of those domains, and for different versions of it : one on the naked domain name, one on the domain name preceded by `www` and one on the final URL that we get after following the possible HTTP redirections of the domain. It will then call a small Java program (called `Dig.java`) that will, for every domain, read the different outputs of the `dig` requests and look for informations related to a CDN. These informations are :

- For Cloudflare, as we had the list of all their external-facing IP addresses, we tried to match the A and AAAA records with those IP addresses;
- We tried to match NS records with the different domain names and name servers of the CDNs.
- We did the same with the CNAME records.

We think this method is exhaustive and, despite some possible special cases such as a DNS server

not responding in time, the results should be pretty accurate. The reason why we chose to use Java in addition of Bash is that we were more familiar with Java and `String` operations in general, and we thought Java code would be more readable than Bash code for this purpose.

C.2 Source IP spoofing: Scapy commands

First, connect to the EC2 instance and run `tcpdump` with the following command: `sudo tcpdump -i eth0 -v icmp`. Then, use *Scapy* to send ICMP packets to that instance.

Connect to Scapy `sudo scapy;`

Send an ICMP packet with your own IP `send(IP(dst="52.24.204.77")/ICMP());`

Send an ICMP packet with a spoofed source IP `send(IP(src="spoofed IP",dst="instance IP")/ICMP());`

Send an ICMP packet with a spoofed source IP and MAC `sendp(Ether(src="spoofed MAC")/
IP(src="spoofed IP",dst="instance IP")/ICMP());`

Abbreviations

ACC Aggregate-based Congestion Control

ACK Acknowledgment

ACL Access Control List

AD Attack Diagnosis

AITF Active Internet Traffic Filtering

AMS-IX Amsterdam Internet Exchange Point

ANA Advanced Network Architecture group

API Application Programming Interface

ASN Autonomous System Number

AS Autonomous System

AWS Amazon Web Services

bps bit per second

BCP Best Current Practice

BGP Border Gateway Protocol

CnC Command and Control

CAPTCHA Completely Automated Public Turing test to tell Computer and Humans Apart

CCTV Closed-Circuit TeleVision

CDN Content Distribution Network

CEO Chief Executive Officer

CF Cloudflare

CHARGEN Character Generator protocol
CLP Conditional Legitimate Probability
CNAME Canonical Name
CPU Central Processing Unit
CS:GO Counter Strike: Global Offensive
DAAD DNS Amplification Attacks Detector
DAT Defense Against Tilt DDoS attacks
DDoS Distributed Denial of Service
DEFCON DEFensive Cooperation Overlay Mesh
DHCP Dynamic Host Configuration Protocol
DNSSEC Domain Name System Security Extensions
DNS Domain Name System
DoS Denial of Service
DOW Defense and Offense Wall
DPI Deep Packet Inspection
DrDoS Distributed reflection-based Denial of Service
EC2 Elastic Compute Cloud
EDNS0 Extensions mechanisms for DNS
eDNS Enhanced DNS
FIB Forward Information Base
FIFO First In First Out
FP False Positive
Gbps Gigabit per second
GUI Graphical User Interface
HCF Hop-Count Filtering
HD Hard Drive
HTTP HyperText Transfer Protocol
IANA Internet Assigned Numbers Authority
ICMP Internet Control Message Protocol
IDS Intrusion Detection System
IEEE Institute of Electrical and Electronics Engineers
IETF Internet Engineering Task Force
IGP Interior Gateway Protocol

INGI Pôle d'ingénierie informatique

IO In/Out

IoT Internet of Things

IP2HC IP to Hop-Count

IPS Intrusion Prevention System

IP Internet Protocol

IRC Internet Relay Chat

ISP Internet Service Provider

IXP Internet Exchange Point

IX Internet Exchange Point

LAN Local Area Network

MAC Media Access Control

MIB Management Information Base

MIT Massachusetts Institute of Technology

MPLS MultiProtocol Label Switching

MSS Maximum Segment Size

MULTOPS Multi-Level Tree for Online Packet Statistics

NAT Network Address Translation

NCF Not Cloudflare

NIC Network Interface Controller

NS Name Server

NTP Network Time Protocol

OSI Open Systems Interconnection

OS Operating System

P2P Peer to Peer

PAD Parallel Attack Diagnosis

Pi Path identifier

pps packet per second

PSK Pre-Shared Key

RADIUS Remote Authentication Dial-In User Service

REGEX Regular Expression

RFC Request For Comments

RIB Routing Information Base

RIM Router Interface Marking
RIP Routing Information Protocol
RIR Regional Internet Registry
ROI Return On Investment
RPC Remote Procedure Call
RPF Reverse Path Forwarding
RSS Receive Side Scaling
RST ReSeT
RUDY R-U-Dead-Yet
RX queue Receive queue
SIFF Stateless Internet Flow Filter
SLA Service Level Agreement
SNMP Simple Network Management Protocol
SOC Security Operation Center
SQL Structured Query Language
SSDP Simple Service Discovery Protocol
TAD Traffic Anomaly Detector
TCPCT TCP Cookie Transactions
TCP Transmission Control Protocol
TLD Top-Level Domain
TMH Trust Management Helmet
TOPS Tabulated Online Packet Statistics
TP True Positive
TSIG Transaction SIGNature
TTL Time To Live
UCL Université Catholique de Louvain
UDP User Datagram Protocol
UPnP Universal Plug and Play
URL Uniform Resource Locator
uRPF unicast Reverse Path Forwarding
US United States
VoIP Voice over IP
WAF Web Application Firewall

WAN Wide Area Network

WPA Wi-Fi Protected Access

XSS Cross-Site Scripting

ZTA Zone Transfer Agent

List of Figures

1	Cumulated number of protected websites from Alexa ranking (for each CDN) . . .	7
1.1	Graph from Akamai [Aka16a]	9
1.2	Stacheldraht botnet diagram	11
1.3	Examples of pricing plans for a booter service (booter.xyz)	12
1.4	DDoS attack vector frequency (graph from [Aka16a])	15
1.5	Ten most frequent attack vectors by quarter (graph from Akamai [Aka16a]) . . .	16
1.6	DDoS Reflector Heat Map (from Akamai [Aka16a]). Note that the reflectors considered here are ssdp, ntp, chargen, qotd, Sentinel and rpc protocols (no dns).	17
1.7	Graph from Akamai [Aka16a], and again for ssdp, ntp, chargen, qotd, Sentinel and rpc (no dns).	18
1.8	Number of open DNS resolvers from 2013-03-24 to 2016-05-22 (graph from the Open Resolver Project)	18
2.1	Flow without Cloudflare protection	22
2.2	Flow with Cloudflare protection	23
3.1	Basic CDN mapping [Jan14]	33
3.2	Akamai's EDNS0 example [Jan14]	34
3.3	Enhance DNS Service Components	37
4.1	Main bottlenecks during DoS and DDoS attacks	43
4.2	Single RX queue kernel bypass (figure from [Clo16h])	45
5.1	D-WARD architecture	48
5.2	MULTOPS tree example	50
5.3	Hop-count inspection algorithm	53
5.4	Defense mechanisms location [ZJT13]	56
5.5	DEFCON overview [MRR03]	57
5.6	SIFF handshake	58
B.1	Those figures come from the MIT ANA Spoofer Project [Pro16b]	87

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve www.uclouvain.be/epl