DoS and DDoS Detection, Defense and Deterrence

Marinos Bernitsas
Department of Electrical & Computer Engineering
Carnegie Mellon University
marinos@cmu.edu
www.bernitsas.com

Abstract

DoS is the most debilitating attack in the Internet and yet it still remains an open research topic. In this paper, we storm through a sea of publications to examine the sophistication of DoS attacks, schemes to detect them, ways to defend against them and credible means to deter their occurrence. As the Internet provides no means of detection, defense, resilience and deterrence for DoS, we describe both commercial solutions and research proposals for addressing each of these security properties. We conclude that no incremental modification can efficiently address DoS, and emphasize the need for a clean-slate approach that integrates all of the necessary security properties to eliminate DoS.

1 Introduction

Above all else, the primary goal of every network is *availability*. It is the number one feature users demand from their providers. When the availability of the Internet is disrupted critical transactions fail and millions of dollars are lost in business. For instance, in 2008, going dark for 2 hours cost Amazon $31,000 per minute [36]. Today this cost is estimated to be over $100,000 per minute.

Denial of Service (DoS) and Distributed Denial of Service (DDoS) are the primary means of attacking availability on the Internet. DoS first became a major scare in February 2000, where a series of massive DoS attacks incapacitated several high-visibility sites, including Yahoo, Ebay and Etrade [15]. Figure 1 demonstrates the disruption these attacks caused in an Internet-wide basis. In January 2001, Microsoft faced a similar assault and was forced to pay Akamai to host part of its network for years.

Since the first public DoS scares, millions of websites have been attacked, ranging from commercial institutions to governmental organizations. In the past months, 10 years after the first publicized DDoS attacks, the international community has been a witness to large-scale DDoS attacks by Hacktivist group "Anonymous" in support of Wikileaks founder Julian Assange [5]. "Operation Payback", as it was dubbed, performed large-scale DDoS attacks on popular websites such as Mastercard, Visa, Paypal and PostFinance bank, rendering them unusable for hours. As more and more of our daily lives transition to the digital realm, the effects of such attacks become more and more devastating.

But what is it that makes this attack still possible? How have researchers not provided a solution for this problem, after all these years? To answer this question, we need to go back to when the Internet was invented, almost 3 decades ago. When the Internet was designed it was comprised by a number of hosts where each one knew and trusted each other. Sending packets that appeared to have come from another host (IP spoofing) was not an issue, because the hosts had no reason to do so. For this reason, the Internet never built in safeguards to prevent malicious hosts from affecting other hosts in the Internet. It also did not provide any accountability mechanism for reliably identifying where traffic came from.

In today's Internet we are still victims of the universal trust and lack of accountability design decisions. First, a host needs to trust that another host on the network will not attempt to launch an attack against it. Second, a host needs to trust that hosts have the IP addresses their packets claim they have. Today's Internet is a chaotic mass of devices across different geographical, legal, political and social institutions. As a result, neither of these levels of trust is attainable. In addition, packets flow so fast that it is extremely expensive and slow to verify or inspect each packet in a flow. For all the above reasons, attackers can inflict serious damage with impunity.

In the remainder of the paper, we first give the necessary background on DoS in section 2. We then examine different types of DoS attacks in section 3. We then proceed to perform a security analysis across three axes: Detection in section 4, Defense and Resilience in section 5 and Deterrence in section 6. We show that unless we move towards a clean-slate approach, by redesigning the Internet [7], DDoS attacks will always be a concern.

Date	Average (s)	Load Time (s)	Δ
7 Feb	5.66	5.98	-5.7%
8 Feb	5.53	5.96	-7.8%
9 Feb	5.26	6.67	-26.8%
10 Feb	4.97	4.86	+2.2%

Figure 1: Internet Performance for 40 important business sites during Feb 2000. Measurements are load times in seconds. Source: Keynote Systems

2 Background

Before proceeding to our analysis of DoS, it is important to define several key terms.

DoS DoS attacks occur when a large amount of traffic from one or more hosts is directed at some network resource. The artificially high load denies or severely degrades service to legitimate users of that resource [21]. DoS can either exploit a *design flaw* in a particular protocol or an *implementation flaw* (logical bug). The Internet, by default, has no way of distinguishing legitimate from illegitimate flows and thus provides no *detection* of the attack. Since the attackers flood the victim with traffic which the victim has no way of halting, the Internet provides no *defense* to DDoS. Because protocols behave non-linearly to congestion, the Internet does not provide *resilience* either. Last, since attackers can spoof their IP addresses and remain undetected, there exists no *deterrence*.

DDoS When DoS happens in a distributed fashion, i.e. multiple hosts sending attack packets at the same time, it is called a DDoS. DDoS can occur without the knowledge of the owner of the host, in which case the host is a zombie, part of a botnet. The DDoS becomes more effective as its hosts become more uniformly distributed across the Internet, as they become harder to identify. Since DDoS is a subset of DoS, the paper uses both terms interchangeably.

Flash Crowd When a major event attracts massive traffic to a particular server, that server experiences a flash crowd. While in a flash crowd the hosts sending the packets are well-intentioned, from the perspective of the server the view is similar to a DDoS, since it is receiving massive traffic from diversely distributed hosts across the Internet.

Zombie When a host has been infected by stealthy malware that opens remote root access, it is said to be a zombie, since the attacker can, at any time activate the host and perform instructions on it. The attacker needs to be careful not to create a noticeable load, since the user might have the computer checked and the malware removed.

Botnet A botnet is a collection of zombies. Modern botnets have hundreds of thousands of zombies at their disposal. The zombies are controlled through command and control servers that provide them with remote instructions to run. Botnets are crucial in performing DDoS attacks, since they allow the attacker to send packets from a huge number of uniformly distributed hosts simultaneously, while concealing his presence.

Security Analysis In this paper we perform a security analysis across four axes:

1. *Attack*: The most important DoS attacks as well as the resources the attacker needs to succeed
2. *Detection*: Commercial and research schemes the victim use to detect and trace the DoS attack
3. *Defense and Resilience*: Commercial and research schemes the victim can use to defend against DoS attacks and remain resilient after it has become compromised
4. *Deterrence*: Commercial and research schemes the servers can use to deter DoS attacks from happening in the first place

3 Attacks

Every attack on the Internet is based on a design flaw or on an implementation flaw (software bug). In the most fundamental sense, DoS leverages the Internet's fundamental design vulnerabilities to launch attacks with impunity. However, DoS is not just a matter of congesting a link. Attackers have become a lot more sophisticated and are able to perform attacks that are specifically tailored for breaking certain protocols. Below, we will see a wide breadth of attacks of varying sophistication, some very common and some that (currently) only exist in the research realm, but could have devastating consequences if ever attempted.

3.1 Bandwidth DoS

Bandwidth DoS is perhaps the simplest and most traditional DoS attack. The basic idea is pointing a large number of clients to a certain resource simultaneously. Since the requests arrive from different hosts, they are distributed uniformly across the Internet and the requests congest the ingress link to the victim's network. Because of the congestion, packets are dropped and connections are either very slow or impossible to maintain. In addition, for transactions where the response is larger than the request (e.g. HTTP), the victim server might congest its own egress link, and thus self-inflict Bandwidth DoS through its own responses.

This attack lacks sophistication and is simply based on the fact that networks are provisioned for a certain bandwidth based on their *average traffic* and not their peak traffic. Note that flash crowds essentially are also a

type of Bandwidth DoS since many users visit a certain website at the same time. The term "getting slashdotted" has been coined to denote the effect experienced when a certain website is linked to on `slashdot.com`, creating a flash crowd that cripples the linked server. Popular events (Michael Jackson's death) have also been known to cause such DoS attacks [11].

3.2 Resource Starvation DoS

In computational DoS the attacker depletes a resource that the victim critically relies on by flooding it with requests. This resource can be: computational power (Computational DoS) by forcing the CPU to perform complicated computations; memory, by forcing the victim to allocate all of its RAM, and therefore enter into a thrashing vicious circle; or storage, by making the victim deplete its allocated storage.

Computational DoS is particularly effective on servers that perform complex processing, e.g. a travel search engine, which has to mine and analyze large sets of data. Unless the victim has significantly over-provisioned, a few requests could bring down the entire system.

In memory DoS the victim's RAM is depleted, forcing it to store elements in secondary storage, which incurs a huge performance penalty because of thrashing. In addition, servers might automatically reboot after their memory is depleted, as they are unable to perform vital functions. If an attacker manages to reboot the victim, he can perform DoS by continuously forcing it to reboot.

In storage DoS, the victim is forced to deplete its storage on its own (i.e. without the attacker having to send gigabytes of storage, which is not realistic). This attack is particularly relevant to devices whose storage is limited, e.g. routers, firewalls, proxies and IPS. In order to force a victim to deplete its own storage the attacker can generate packets that will generate a large number of storage actions per packet. For example, the attacker can send packets that will create a large number of log entries in the victim's storage, or can force a firewall to create a larger number of flow filters than it can handle. The device will then either reboot or stop responding, creating a DoS.

A lot of DoS attacks that exploit application-layer vulnerabilities to perform DoS fall under Resource Starvation DoS. For example, the billion laughs DoS exploits weaknesses in XML parsers. ReDoS exploits regular expressions that take a long time to calculate. Buffer overflows can cause a victim to crash and experience DoS. Such vulnerabilities are typically fixed in a faster development cycle, as they are software fixes.

3.3 Algorithmic Complexity DoS

Algorithmic Complexity DoS is a sophisticated kind of Resource Starvation DoS attack which exploits specific vulnerabilities in the data structures used by network elements. Network devices need very efficient data structures to minimize the delay introduced by their component. While those data structures (usually Hash Tables and Binary Trees) have very desirable average runtime properties, in their worst case, they degenerate to linked lists, in which `add` becomes $O(n)$ and `lookup` becomes $O(n^2)$. In addition, the structures can be forced to constantly resize, incurring a huge performance penalty that during average runtime would be amortized. An attacker can specifically choose his packets to force these efficient data structures to reach their worst-case performance, thus bringing the victim down to its knees.

In [12] Crossby and Wallach demonstrate the effectiveness of such attacks on Perl, the Squid web proxy and the Bro intrusion detection system. Using bandwidth less than a dialup modem, after six minutes the Bro server was dropping as much as 71% of traffic and consuming all of its CPU. Figure 2 shows the cumulative dropped packets by the Bro intrusion detection system, with carefully selected traffic of only 16kb/s. At time A, only 6 minutes after the attack, Bro's processing latency starts increasing more than the packet interarrival time. At time B, Bro is catching up on its backlog and essentially dropping every packet. At time C the hash table is resized, and the attack commences again.

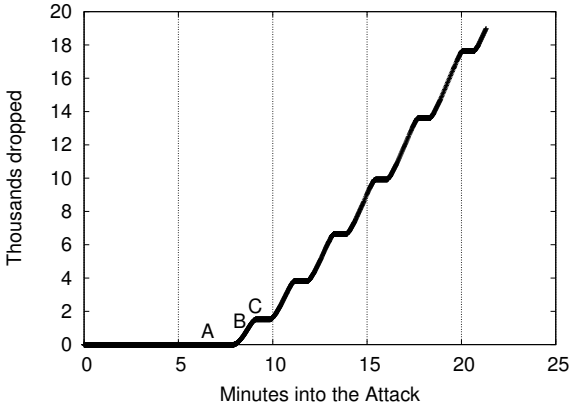


Figure 2: Cumulative dropped packets by the Bro IDS, 16kb/s [12]

When increasing the attacker's bandwidth to only that of an ISDN connection 64kb/s, point B starts after only 1 minute into the attack and more than 45,000 packets are dropped in only 6 minutes.

3.4 Reflector Attack

A Reflector Attack allows a single host with limited processing power and bandwidth to force a legitimate intermediary network to perform DoS on the victim, while simultaneously framing the intermediary. The reflector

attack is more of a method than an attack itself, as it requires a protocol that generates a reply when specific packets are sent to it. The attacker generates a packet that seems to have come from the victim and addresses it to a multicast address in a network. This packet is specifically crafted to induce a response from the intermediary to the victim. When each host in the reflecting network receives the packet on its multicast interface, it follows the protocol's specification and generates a reply directed to the source of the packet, which in this case is the victim.

With a single packet and a single machine the attacker has managed to perform an amplification of the size of the reflecting network. When the packet flood reaches the victim, either the victim's bandwidth or resources are depleted, and DoS is experienced. Note that this technique is attractive because the victim thinks that the attack is coming from the reflecting network, so the attacker can conceal his presence.

3.5 Smurf DDoS

A Smurf attack takes the Reflector attack even further. Because in a traditional reflector attack the responses seem to come from a single network, it is easy to perform filtering for one specific flow aggregate to block all requests from the attacker's network. In order to circumvent this problem, the attacker uses a botnet to direct reflector packets to a large variety of intermediaries. Each of the intermediaries perform the reflection, but since the attack packets are coming from so many different networks, it is impossible for the victim to filter them and effectively distinguish attack packets from legitimate packets. Therefore, the victim experiences DoS.

Because of its simplicity and ability to conceal the attacker, Smurf DDoS has been demonstrated to be extremely effective and hard to defend from. Even worse, because of its distributed nature, this attack may lead to Bandwidth DoS on links that are before the victim's network, forming a bottleneck before the traffic even reaches the victim. As a result, the victim can't even defend from the attack since he never sees it.

3.6 TCP DoS

Since TCP was created in order to provide reliable communications and was not designed with security in mind, there are fundamental weaknesses in the protocol that can be used as attack vectors for DDoS. Its most frequent exploitation is forcing a victim to respond with RST messages as part of a Smurf DDoS attack. All the RSTs congest the egress link of the router creating DoS.

But there are more sophisticated and effective attacks against TCP. It is important to note that the only security available to TCP is the 2^{32}-bit Sequence Number (SN), which the attacker has to guess in order to inject bogus packets into a connection.

SYN Flooding Upon a SYN request, a server needs to reserve state for establishing a connection. This can be the Initial Sequence Number (ISN) of the connections, connection options/preferences, the IP and port of the client etc. Typical TCP implementations allow 1024 or 2048 connections in their SYN buffer (i.e. incomplete connections for which only a SYN has been received). By flooding the victim with SYN requests, the attacker can exhaust the SYN queue and thus evict all legitimate connections. Assuming a uniform probability and a 1024 size queue, the attacker only needs to send

$$\sum_{n=0}^{1023} \frac{1024}{1024-n} = 7689.4 \tag{3.1}$$

packets. As a result, SYN Flooding performs DoS by never allowing new connections to be established.

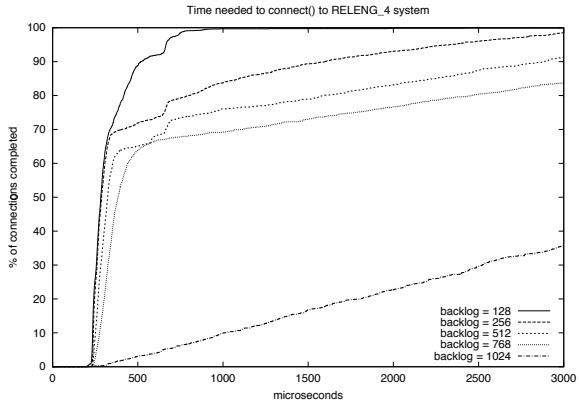


Figure 3: Time needed to `connect()` to a RELENG_4 system under SYN Flooding [19]

Figure 3 shows how after a SYN flood that causes a 1024-byte backlog the machine's TCP performance completely deteriorates. Because of TCP's congestion control, DoS happens very abruptly and non-linearly.

TCP Reset The attacker can send TCP resets to constantly reset existing connections. When the TCP protocol receives a RST packet from a client, it checks that the Sequence Number is within acceptable bounds (SN+TCP Window) and if it is, it terminates the connection given by the $< port, src >$ pair.

As pointed out in [6], if the victim's TCP implementation uses a predictable algorithm for determining SNs, the attacker can simply open a legitimate connection, look at the SN and then create RSTs for every SN that comes after it, denying access to every subsequent communication request.

But even if the SNs are not predictable, in [35] Watson shows that because the typical TCP window is 2^{16}, only $\frac{2^{32}}{2^{16}}$ different SNs need to be generated for a RST packet to succeed. As shown in Figure 4, it only takes half a second for a 45mbps line to reset a connection with a known port, and 25 seconds for 50 source ports.

This allows an attacker to very quickly evict established TCP connections and perform DoS.

Fragment Flooding Both TCP and IP allow single packets to be fragmented into smaller units (fragmentation and reassembly). This allows packets to pass through networks that may require smaller packet sizes than the sender estimated. In order to reassemble the stream, the attacker has to first allocate space for the original TCP packet and then place each fragment in the right memory slot. Only after all fragments have arrived can he read the packet received. But the attacker can simply break a TCP packet into n fragments each of size s_n bytes and only send the n-th packet. In that way, he can achieve the storage allocation of having sent an $n*s_n$ byte packet, with only sending s_n bytes. The server's resources are quickly exhausted, leading to DoS.

Attacking TCP Congestion Control One of TCP's fundamental operations is its ability to perform congestion control and dynamically adjust the throughput of a connection. During a DoS attack, the TCP stack can dynamically throttle an attacker who is sending a flood of data through TCP, therefore limiting the potential for exploitation. However, in [29], researchers present *TCP Daytona*, three ways by which an attacker can receive more than its fair share of bandwidth in TCP. These are based on logic and implementation errors in most TCP stacks.

With *ACK division*, an attacker can send n ACKs for every packet received. The server misinterprets the extra ACKs as a sign of lower congestion, thus allowing the attacker to send and receive more data. With *DupACK*, an attacker sends multiple ACKs for every packet, again tricking the server into accepting larger throughput from the server. Last, with *Optimistic ACKing* the attacker sends ACKs for packets it has not yet received, again increasing its available throughput.

As shown in Figure 5, TCP Daytona can hinder the sender's ability to determine the congestion of a line based on the ACKs it receives. This makes a very powerful DDoS, since the server under DDoS will be forced to increase its send rate instead of throttle it.

3.7 ICMP DoS

ICMP provides various control and maintenance commands for networks. Since the echo and ping commands induce responses by the hosts they are sent to, they provide ideal candidates for Smurf and Reflector attacks,

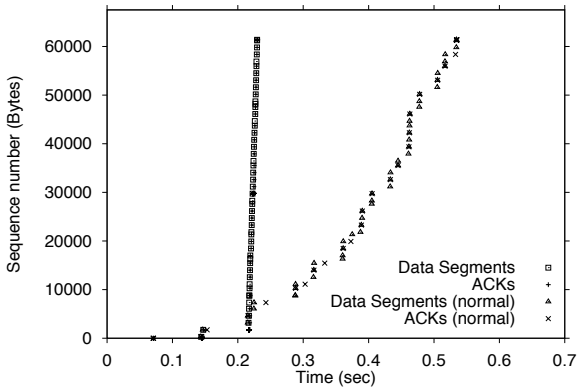


Figure 5: TCP Daytona using ACK division [29]

leading to the term "Ping Flood".

However, ICMP packets are vulnerable to more sophisticated DoS attacks. Since they are not authenticated, any attacker can spoof them. As a result, the attacker can send Destination Unreachable or Time to Live Exceeded commands to reset existing TCP connections, as long as it knows the source and destination port [6]. In addition, he could force routers to make other kinds of routing changes that would deny the victim a route to the Internet. Past implementation vulnerabilities also include the *ping of death*, where an attacker could send a malformed ping packet to crash a host, and the *teardrop attack*, where a host would crash when receiving overlapping TCP fragments.

All of the implementation vulnerabilities have been fixed, but ICMP still remains an attractive target for attacks due to its lack of authentication and its ability to provoke responses by victims.

3.8 Coremelt Attack

Coremelt [31] is an attack that attempts to perform DoS not just on a single target, but on the Internet Core, i.e. the core routers at the heart of the Internet that the majority of traffic passes through. Such an attack disrupts the operation of the entire Internet and the authors show it is possible to do so with a cotnet of average size. This is especially dangerous in the wake of the emergence of a form of Internet terrorism, which could use it to disrupt millions of communications, banking transactions and time-critical transmissions.

In Coremelt, an attacker generates traffic between the hosts on a botnet. Since the zombies are topologically uniformly distributed, the traffic flows pass through the core of the Internet, creating a large load across all the core routers. Among N routers there are $O(N^2)$ connections, which can cause significant congestion on the network core, creating an Internet-wide DoS.

In Coremelt, the attacker is limited by the size of the botnet, the uniformity of the distribution of bots and the

Operating System	Initial Window Size	Packets Required
Windows 2000 5.00.2195 SP4	64,512	66,576
Windows XP Home Edition SP1	64,240	66,858
HP-UX 11	32,768	131,071
Nokia IPSO 3.6-FCS6	16,384	262,143
Cisco 12.2(8)	16,384	262,143
Cisco 12.1(5)	16,384	262,143
Cisco 12.0(7)	16,384	262,143
Cisco 12.0(8)	16,384	262,143
Windows 2000 5.00.2195 SP1	16,384	262,143
Windows 2000 5.00.2195 SP3	16,384	262,143
Linux 2.4.18	5,840	735,439
Efficient Networks 5861 (DSL) v5.3.20	4,096	1,048,575

Figure 4: Initial Window Sizes for different OSs

amount of traffic each bot can generate. In the most realistic simulation performed (step network model), the attackers are able to create an Internet-wide DDoS using about 330,000 bots of the CodeRed botnet dataset where each bot has 128 kbps, as shown in Figure 6.

3.9 Control Plane Attacks

So far we have only seen DoS from the perspective of the Data Plane, in which we generate messages from the Data Plane to attack the Data Plane. However, we could also use the control plane to perform DoS attacks in a more sophisticated and hard to recover from manner. Since the control plane determines how traffic is routed, attacks on the control plane can affect the global reachability of a host instantly. Control Plane attacks are therefore the largest threat to the Internet.

3.9.1 Using the Control Plane to attack the Control Plane

The default control plane protocol for Inter-AS routing is BGP. Someone who manages to attack BGP can create DoS by: *Blackholing* the victim, i.e. forcing all of its traffic to be lost; *Redirection* of the victim's traffic to another AS; and inducing *Instability*, whereby the route constantly changes [27].

Attacks launched through BGP can be deadly, particularly because BGP updates are Internet-wide and take a long time to recover from. The attackers can perform *prefix hijacking*, where they claim to have a longer prefix to the victim, thus attracting all of the victim's traffic [10], preferably into a black hole, creating DoS. In addition, they can perform *link flapping*, where a route is withdrawn and re-announced a sufficient number of times to cause all routers to avoid that route (a procedure called route dampening), creating DoS through the supposed failure of critical links. Other instabilities as well as congestion-induced BGP session failures can also lead to DoS.

3.9.2 Using the Data Plane to attack the Control Plane

We have shown that we can use the Control Plane to attack the Data Plane and perform DoS. It is even more alarming, however, that we can also use the Data Plane to attack the Control Plane and achieve DoS. Since the data plane is where all of user traffic lies, this essentially means that simple users can perform control-plane disruptions that lead to devastating DoS attacks.

The Coordinated Cross Plane Session Termination attack (CXPST) [30] can melt the current Internet core by performing targeted attacks on BGP links in a strategic manner, through the data plane. Since BGP relies on the announcement of routing updates to all neighbors, updates propagate globally. By flapping certain carefully-chosen links with a large volume of data-plane traffic (through the use of a botnet), we can induce such a high rate of updates and queuing delays that BGP control-plane traffic is evaluated at a delay of 100 minutes, essentially breaking down the BGP protocol. CXPST relies on the fact that control and data planes are often sharing the same medium, and therefore it is possible to manipulate one plane through the other plane. This is because usually control and data traffic are separate flows sharing the same physical link.

The most important advantage of this attack is its strategic design: blindly flooding the Internet with packets will cause congestion at the border ASs and leave the core unaffected (in addition, the core will have the chance to route around such bottlenecks). But by strategically congesting certain links, the attack can very quickly bring the core routers to a standstill. The attack is immune to almost all countermeasures currently implemented (graceful restart, flap dampening and minimum advertisement intervals), as well as denial of ser-

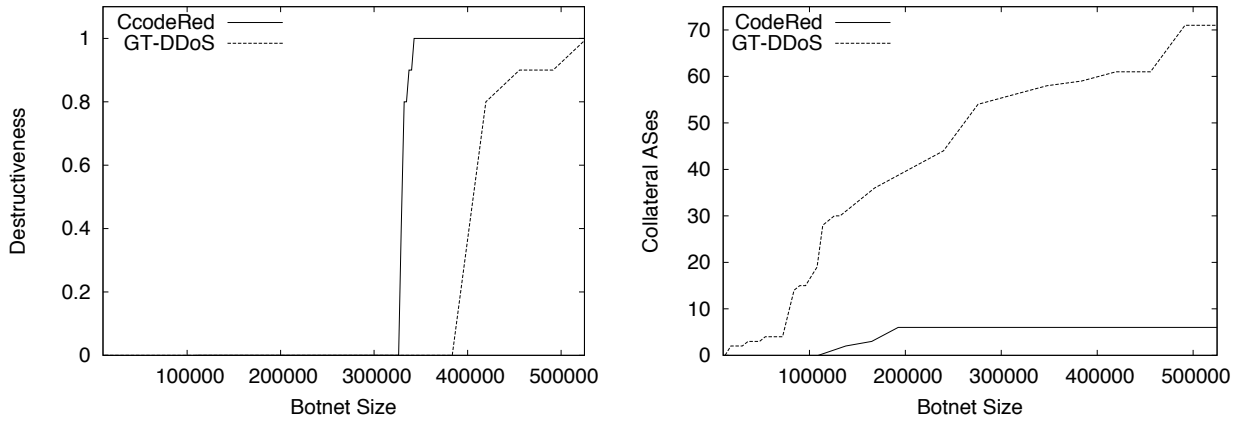


Figure 6: Step Network Simulation 128 kbps per bot [31]

vice defenses (because the botnet units that serve as the sinks actually requested the traffic).

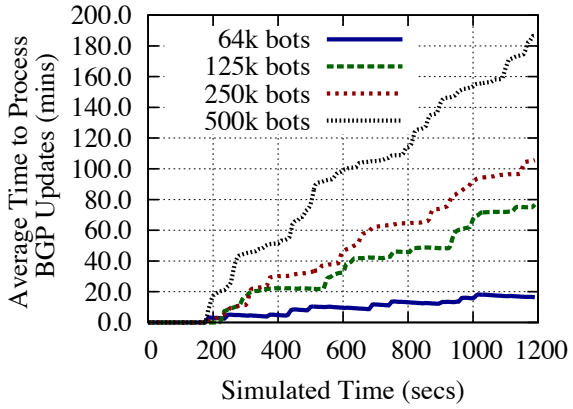


Figure 7: The average time to process a BGP update for core ASes under attack by botnets of various sizes [30]

As seen in Figure 7, the speed at which this attack can function is impressive: in 20 minutes, a 500k botnet creates a processing delay of 180 minutes and a 250k botnet creates a processing delay of 100 minutes. Since these effects will tend to multiply globally, recovery from such an attack would have to involve rebooting essential components of the Internet or shutting down links until they stabilize, and would create a global, sustained disruption of service.

Such targeted attacks on certain BGP links can be the next-generation way of performing DDoS. While this attack only exists in the research domain, its severity could be huge once it reaches the real-world.

4 Detection

The first step in defending against a DoS attack is detecting it. This is particularly hard, for a number of reasons. First, DoS may congest some of the links before the victim's routers, and therefore the victim will have no idea that its reachability by the world has been affected. Second, even when there is a perceived increase in traffic at the victim's network, it is hard to tell if it due to the result of a flash crowd or a DoS attack. Third, systems that are designed to detect such attacks may easily be tricked into not detecting the incident and administrators have been trained to ignore alarms due to their large number of false positives.

4.1 Intrusion Detection Systems

The easiest way for a host to detect if its network is a victim of a DoS or any other attack is through a Network Intrusion Detection System (NIDS). Such systems are usually installed in a non-inline manner (i.e. traffic does pass through them but beside them) so they sniff the network in a fail-open manner. Commercial NIDS are available by Cisco, Juniper, Barracuda and most network vendors. There are also open-source versions, the most popular being Bro, Honeypot and Snort.

The NIDS will look for specific attack vectors and compare them against a database. If the attack is in the database, it will raise an alarm and notify the network administrator of the intrusion. The administrator should then proceed to one of the defensive measures outlined in the next section.

NIDS effectiveness The most important question is to look at how effective NIDS systems are in terms of correctly predicting an attack. As shown in [4], NIDS are vulnerable to the base-rate fallacy experienced by Bayesian statistics, i.e. in order to increase their effectiveness ($P(I|A)$), we have to get the false positive rate to an unattainably low level.

To illustrate this, consider an example IDS with 0.01% false positive rate and 0.01% false negative rate. The box logs 1.1 million total records and each actual

attack is reflected in 10 records in the NIDS.

$$P(A|\bar{I}) = 10^{-4} \tag{4.1}$$
$$P(\bar{A}|I) = 10^{-4} \tag{4.2}$$
$$P(A|I) = .9999 \tag{4.3}$$
$$P(I) = \frac{10 * 3}{1.1 * 10^6} = 2.7 * 10^{-5} \tag{4.4}$$
$$P(\bar{I}) = 1 - P(I) = .99997 \tag{4.5}$$
$$\Rightarrow P(I|A) = \frac{P(A|I)P(I)}{P(A|I)P(I) + P(A|\bar{I})P(\bar{I})} = .21259 \tag{4.6}$$

As shown by (4.6), only one out of 5 alarms denote an actual intrusion. Bringing the false positives to the unattainably low 0.0001 rate gives a large improvement.

$$P(A|\bar{I}) = 10^{-5} \tag{4.7}$$
$$\Rightarrow P(I|A) = .72972 \tag{4.8}$$

As a result, NIDS fail to provide a trusted means of reliably detecting a legitimate attack, and may be ignored by network admins.

Insertion/Evasion Attacks NIDS can be circumvented. Insertion/Evasion attacks [28] take advantage of the fact that the NIDS is unaware of the state of the end-host and thus may be manipulated into interpreting the packets differently from the end-host. The NIDS would either generate a false alarm or generate no alarm for an attack packet. In an insertion attack, the attacker manages to insert data that the NIDS sees but the victim drops, thus obfuscating the attack signature. In an evasion attack, the NIDS drops data that the victim keeps.

There are many ways to create an insertion/evasion attack. The attacker can use a TTL that will reach the NIDS but not the victim, thereby creating a packet that will contain more segments when seen by the IDS than by the victim. This allows the attacker to insert garbage that will not match an attack signature, but when the packets reach the host, the garbage is removed through the low TTL and the host experiences an attack. Another popular evasion technique is setting the ECN bit (Explicit Congestion Notification). Some hosts will drop packets that have this bit set, while others will keep them. If the IDS and the victim host differ in the behavior, the attacker can inject garbage with the ECN set, which will evade the IDS but attack the host.

IP fragmentation and TCP stream reassembly introduce serious storage requirements for IDSs, since a packet can only be examined after all fragments from both the TCP and IP layer have been received. If the IDS does not allocate storage efficiently, an attacker could craft the packets to exhaust the IDS's storage resources as shown in section 3.2.

In addition, both types of fragmentation have varying behavior across operating systems, especially when it comes to overlapping transmissions: for IP fragmentation, Windows NT and Solaris always favor old data for overlapping fragments, while other systems favor new data for forward overlap; for TCP fragmentation, Windows NT always favors old data, while other systems favor new data for forward overlap. These differences in behavior leave the IDS vulnerable to insertion and evasion attacks, in which packets reassemble differently on the IDS than the host, and thus the intrusion is undetected.

Practically every element in the IP and TCP header can lead to an insertion/evasion attack if the host and the NIDS interpret it differently. The most recent of them is the TCP split handshake attack [32]. In the context of DoS and DDos, an attacker can leverage insertion/evasion attacks to successfully avoid the IDS from reporting the attack. This will make administrators believe that this is a simple flash crowd.

Attacking the NIDS itself The first thing an attacker will do is try to take down the NIDS itself. Because NIDS are mostly created in order to detect specific exploits instead of DoS attacks, they will crash under the heavy load of a DoS, and therefore go offline. Algorithmic Complexity attacks such as the ones shown in 3.3 are particularly effective in terms of knocking out an IDS.

4.2 Traffic Measurement and Accounting

While a host can detect DoS attacks through a NIDS and bandwidth monitoring, ASs also want to be able to detect and perhaps trace DoS attacks. This is essential in order for them to be able to provide any kind of throttling and protect their own routers and customers. Detecting DoS from an AS standpoint is particularly hard, because each backbone router incurs an average of 10 million concurrent flows per hour. Simply counting the bandwidth each flow consumes is impossible. As Internet routers become capable of attaining higher and higher throughputs, the issue of actually measuring this traffic and classifying it becomes harder and harder to solve, especially when taking into account the memory requirements necessary.

As a result, ASs rely on probabilistic counting tools to identify heavy-hitters (flows that consume a lot of bandwidth and are thus possible DoS targets). The simplest approach is Cisco's Netflow, which samples every s packets and collects the data into a table. However, it is incurs sampling bias and memory overhead, and ironically upon a DoS network administrators have to switch it off because of the resources it depletes.

In [13] Estan and Varghese propose two new methods for measuring Internet traffic generated by large flows or flow classes:

Sample and Hold in which traffic is randomly sampled, but after an entity has been sampled its count is updated for every subsequent hit of the packet. Sample and Hold is able to achieve a low probability of false negatives (with 99% probability we measure a 5% variation in the flow's traffic). Sample and Hold dramatically improves the Sampled NetFlow approach, which is widely used in industry, as it "holds" on to the flow after it has been sampled, instead of just randomly sampling

Multistage Filters in which each flow is hashed using various hash functions (stages), and if the count corresponding to a flow ID in every stage is above a threshold, the flow is added into flow memory for further analysis. Multistage filters have no false negatives, while the multiple stages exponentially attenuate the false positives. To limit the false positives even further, the authors provide additional optimizations that can be used: "conservative update of counters", where counters are not updated if the flows are added to memory, and "serial multistage filters", where the stages occur in series instead of parallel

The two algorithms can work with memory in SRAM, compared to DRAM which is currently used. The relative errors of both algorithms scale with $\frac{1}{M}$ (M being the amount of SRAM), which is a big improvement compared to $\frac{1}{\sqrt{M}}$ offered by previous algorithms. In addition, both algorithms can operate in real-time.

Both algorithms provide cutting-edge methods for ASs to identify DoS traffic reliably and efficiently.

4.3 Inferring Denial of Service Activity

Another detection goal is for there to be some dedicated "DoS Observatories" that detect and report on DoS activity across the Internet.

In [25] researchers offer the first ever technique of measuring, extrapolating and analyzing DoS attacks on the Internet. They does so by monitoring a /8 address range for *backscatter*, i.e. responses from victims that received illegitimate packets from spoofed addresses. Since the /8 is $\frac{1}{256}$ of the Internet, samples can be easily extrapolated to obtain representative figures for the entire Internet. The researchers used a series of filters to classify their attacks and obtained statistically significant results for variation of DoS attacks by time, protocol, traffic rate, duration, victim name, TLD and AS, providing useful insights into DoS attacks.

Their analysis reveals very interesting observations on these attacks. For example, as figure 8 shows, the PDF

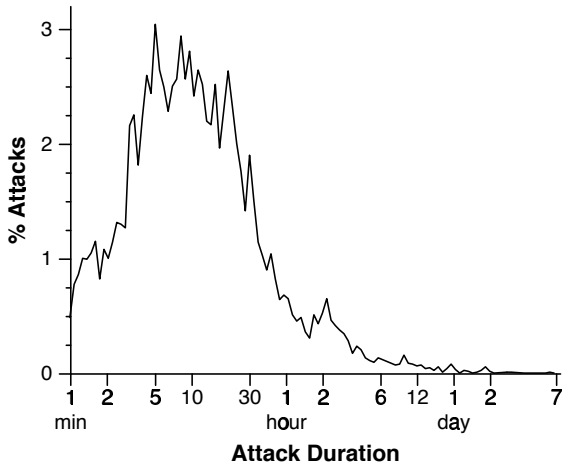


Figure 8: Probability Density Function of attack durations [25]

of attack durations presents spikes at rounded time durations for the attack for 5, 10 and 20 minutes. This is very strong empirical evidence of the measurements being reliable, as humans are likely to choose rounded time durations.

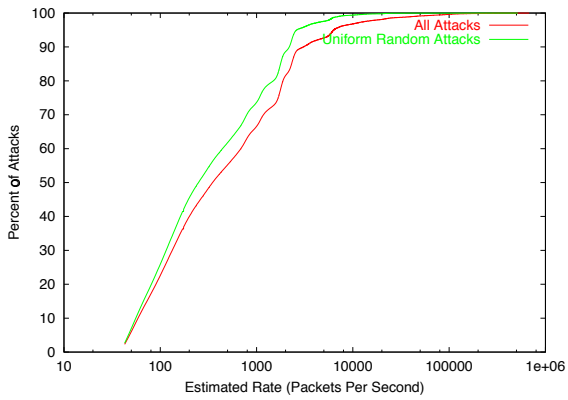


Figure 9: Cumulative distribution of estimated attack rates

Figure 9 provides alarming results: 50% of the attacks have a rate greater than 350 packets per second, with 500 SYN packets per second being enough to overwhelm a server. A small percentage of the attacks are even capable of breaking specialized firewalls designed to resist SYN floods.

5 Defense and Resilience

Defending against DoS and DDoS is hard, because without any mechanism predeployed, the victim cannot differentiate between the legitimate flows and the malicious flows. As a result, evicting a flow or throttling band-

width may even exacerbate the DoS because it will constrain the resources available to legitimate clients even further.

Resilience is also an important property, so that even if a host fails to completely defend against the DoS attack, its customers can still be given some level of service.

In this section we will examine a number of defenses for DoS, some of which are currently implemented, and some which exist only on the theoretical realm. As we will see, most real-world deployed defenses are incapable of successfully coping with DoS attacks.

5.1 Ingress Filtering

In ingress filtering each network drops all packets for which the source IP address does not belong to the network. This guarantees that a host in the network cannot spoof an IP address outside the network (note that the ingress filtering is only performed to the network-level granularity). If every network in the Internet performed ingress filtering, then a DoS victim would simply detect the network from which the packets are coming and filter it. This would allow the victim to maintain service for all non-attack packets and effectively defend against DoS.

The problem is that unless every network performs it, ingress filtering is of no value. This is because it is a restriction imposed on a network that benefits networks other than the network it is deployed on. Since networks exist in different legal frameworks and countries, it is impossible to convince every network or ISP to perform ingress filtering, so the measure has limited success. However, it can be leveraged as part of peering relationships and contractual agreements, or legally mandated. In addition, ingress filtering is useless in DDoS attacks since the zombies do not spoof their addresses.

5.2 Pushback

Pushback [21, 18] works by proactively rate-limiting selected links with packets that would be dropped by the victim anyway.

Each router monitors incoming flows and tries to cluster traffic together to establish *aggregates*. The paper assumes that DoS packets can be clustered into aggregates by some feature of the packet (destination IP, protocol, network prefix). Upon a DoS attack or a flash crowd, the router identifies the aggregates that are responsible for the excess bandwidth and rate-limits them using Random Early Drop. The rate limit L is computed by formula (5.1) where $Aggregate[k].arrival$ is the arrival rate of the k-th aggregate:

$$\sum_{K=1}^{i}(Aggregate[k].arrival - L) = R_{excess} \quad (5.1)$$

Next, the router classifies each of its links as either contributing or non-contributing, depending on whether they consume a large or small fraction of the aggregate traffic. For all contributing links, the router sends a pushback message upstream to the neighboring router in which it instructs it to adopt a specific rate limit for all packets matching that aggregate description. The neighboring router will perform the rate-limiting and if it determines that there are aggregates of its own that are congesting the links, it will further propagate a pushback message upstream to its own neighbor. This creates a propagation of rate-limits upstream, greatly mitigating DoS from the predetermined aggregates.

Each of the routers that have received the pushback will periodically send their status to the victim. This includes the traffic that the victim would have received if the rate-limit was not in place. The victim will then periodically re-examine the status of each link and update or expire the rate-limit.

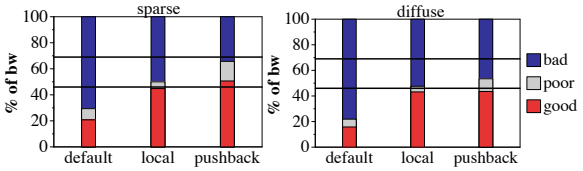


Figure 10: Bandwidth allocation for a link under DDoS with local rate-limiting and pushback [21]

Figure 10 shows how Pushback achieves a better allocation of bandwidth between good and bad flows. Note that poor flows, i.e. flows that are mistakenly attributed as attack traffic and rate-limited, also increase. The effects are stronger in sparse attacks than attacks where the attackers are more diffused across the network. This is expected, because in a diffused DDoS attack it would be hard to correctly classify the aggregates.

If DDoS attacks could be easily grouped into aggregates by some defining characteristic they share, pushback would be a great DDoS defense, because it could prohibit attack traffic from even reaching the victim. By essentially installing a distributed system for filtering, pushback can effectively prevent against aggregates. The problem, however, is that, as we saw in section 4, detecting and isolating aggregates is extremely hard, because DDoS traffic is so uniformly distributed. As a result, Pushback's effectiveness is significantly hampered.

5.3 Over Provisioning

Companies such as Akamai, Limelight and Amazon Web Services offer their customers services for which bandwidth is elastic, i.e. the service will automatically provide more bandwidth if a customer is experiencing high loads (and potentially DoS). These Content Delivery Networks (CDNs) have such huge resources available that it is impossible for them to be victims of DDoS. For example in recent DDoS attacks, Amazon's servers were not affected by a DDoS launched against it [26]. In addition, since CDNs offer points of presence close on the ISPs, attacks that may bring down some points of presence only impact the reachability from that ISP or ISP cluster.

Typically, the customer will pay for 95th percentile pricing, meaning he only pays for the bandwidth used by 95% of traffic during a billing cycle. He may also request an over provisioning margin above the average bandwidth in case of attacks or flash crowds. However, after prolonged DDoS attacks, the DDoS traffic quickly bleeds into the 95th percentile, and the cost becomes prohibitive. For this reason, a company has to decide on how much it is willing to pay to keep its website live during a DDoS.

In [9, 22] it is shown how organized crime performed racketeering by launching DDoS attacks on online betting websites right before the results were due, requesting a fee for the attack to stop. The extortion fee was carefully calculated to be less than what would be required for hiring a CDN to over provision (about $100,000 in [22]). The fact that most of these websites chose to pay off the attackers denotes how expensive a over provisioning can be.

5.4 IPS

Intrusion Prevention Systems (IPS) are IDS (section 4.1) that are placed inline and have the ability to thwart attacks in real-time, either on their own or by installing filters at the firewall. Since IPS use the IDS functionality to detect the attacks before they act upon them, they are limited by all the problems mentioned in section 4.1.

While an IDS crashing will simply not be able to detect attacks, an IPS failing will create downtime for the entire network, because IDSs are fail-closed. For this reason, IPS make attractive targets for DoS attacks. The attacker can simply find a way to exploit a bug or perform DoS on the IDS itself, thereby causing it to crash or reboot. While it the IPS is down, the clients of the victim network experience DoS.

In addition, since DDoS packets may not be classified as attack traffic (in fact, it can be totally legitimate traffic that does not map to an attack signature) the IPS will not detect the attack and is therefore useless in preventing it. IPSs are not only limited in their effectiveness to defend against DDoS attacks, but may be attack vectors in themselves. Algorithmic Complexity Attacks (section 3.3) may be particularly effective on bringing down an IDS.

5.5 CAPTCHAs

A CAPTCHA [33] is a Completely Automated Public Turing test to tell Computers and Humans Apart. The user is presented with a challenge that only a human can interpret and is asked to enter a response. If the response validates, the website has verified that he is human. CAPTCHAs prevent DoS on the basis that a computer program cannot itself solve the CAPTCHA and thus the attack cannot be scripted, but has to rely on a human. Since each human takes an average of 10 seconds to solve a CAPTCHA, requests cannot be sent at a fast enough rate to cause DoS. CAPTCHAs are particularly effective in websites where a few requests provide large computational load on the server (section 3.2), since they require each client to perform a small computation in exchange for the server investing computation time.

CAPTCHAs are relatively effective against DDoS because they cannot be pre-computed or solved in real-time in large numbers. However, attackers that have access to webservers with high throughput are able to perform a relay attack, whereby they relay the captcha to their visitor and play back the CAPTCHA to the victim. Typically attackers have access to high-demand porn sites where they can request each visitor to solve a CAPTCHA before playing a video or image. If they have sufficiently high number of customers per second, they may be able to solve CAPTCHAs in real-time, fast enough to cause a DDoS.

5.6 Capabilities

Capabilities are a way to provide resilience during attacks through flow differentiation, so that when DDoS takes place, the victim can police each flow differently. In the simplest case, there exist two flows, privileged traffic and unprivileged traffic. Privileged traffic is guaranteed bandwidth through QoS reservations, while unprivileged traffic remains best effort. Upon a DDoS, privileged traffic is unaffected and experiences no degradation, while unprivileged traffic experiences DoS as it competes with each other.

The victim periodically makes sure that each of the privileged flows is behaving properly. If not, it does not get its capability renewed and its flow is downgraded to unprivileged. Since the victim can identify all the hosts that have received a capability, it can easily block those hosts from accessing its network altogether, thus also protecting the unprivileged users from performance degradation.

Client Authentication Username-password authentication is the simplest way to establish a privileged flow through the application layer. Upon a DDoS, websites like Amazon may choose to give their authenticated users privileged flows, so that the DDoS is only experienced by unauthenticated users. In order for the attacker to perform DDoS at a high enough rate, he would need access to hundreds of thousands of accounts, which we may safely assume he does not have. As a result, the attacker is unable to affect the privileged flow. The site's customers are therefore not affected and the DDoS is less expensive for the victim.

A victim can also provide its customers with privileged flows through the use of client certificates, a method very popular in SSH. Upon connection request, the client sends its certificate to the server, who then promotes the client to a privileged flow. If the client misbehaves he is blacklisted for a short time and cannot perform DoS through that account.

SIFF SIFF [38] is a backwards-compatible incremental protocol that can be deployed over the current Internet to mitigate DoS attacks, by introducing a new class of privileged packets (vs unprivileged packets). Its success is phenomenal, managing to filter 97% of attack traffic on real data. It does so by adding to each privileged packet a capability field to the TCP header in which each hop adds its own (time-sensitive) marking for the upward and downward path. The protocol requires a two-way handshake to bootstrap, through the use of EXPLORER packets, while key rollovers can be performed at no additional overhead, by having each hop maintain a window of markings.

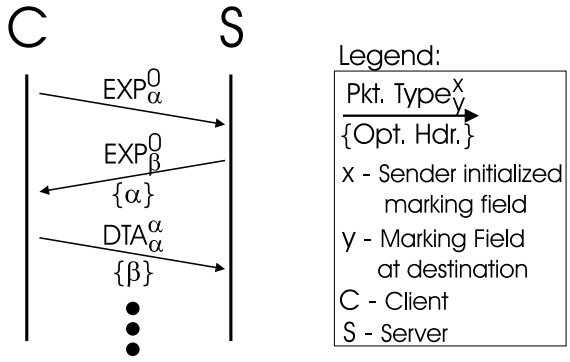


Figure 11: The SIFF handshake [38]

Figure 11 demonstrates the three-way handshake. The EXPLORER packet obtains marking α as it traverses each router. The sender then returns the marking though the Capability Update field if he wishes to establish a privileged connection with the client. The client then appends his returned capability to the data packet, along with the server's capability β for the downstream direction. As a result, a bidirectional privileged flow has been established.

What makes SIFF particularly attractive in the real world is its ability to be offered "as a service" running over the legacy Internet, by differentiating traffic into legacy/unprivileged and privileged. Even when a single routing path contains SIFF compatible and non-SIFF compatible routers, we can still achieve good attack filtering, provided the legacy routers do not reset any of SIFF's headers.

Another great feature is SIFF's ability to select and filter individual traffic flows without requiring per-flow network state, thereby providing higher reliability to legitimate connections than other solutions like probabilistic filtering.

Another huge DoS issue is that incoming traffic is often blocked even before it reaches the victim's network, thus leaving the victim helpless to combat the DoS. Because each hop on the network decreases the probability that a legitimate packet with forged markings reaches the server, by the time the packets reach the victim, they have already been significantly filtered, therefore alleviating a single router from having to bear the burden of the increased packet volume.

During a DoS, privileged SIFF flows are protected, since their bandwidth is reserved. The victim can choose whether to renew or not an existing privileged flow, by choosing whether it sends back the updated marking. For flows that attempt to be established after the DoS has started, the client may need to resend its EXPLORER packet until it hears back from the server. For example, even with a packet loss rate of 90%, the legitimate client only needs to send 10 packets to get one through. After the privileged flow is established, he is free from any congestion.

For a router with x markings in its window, each of x bits, the probability that an attacker correctly generates a random marking is:

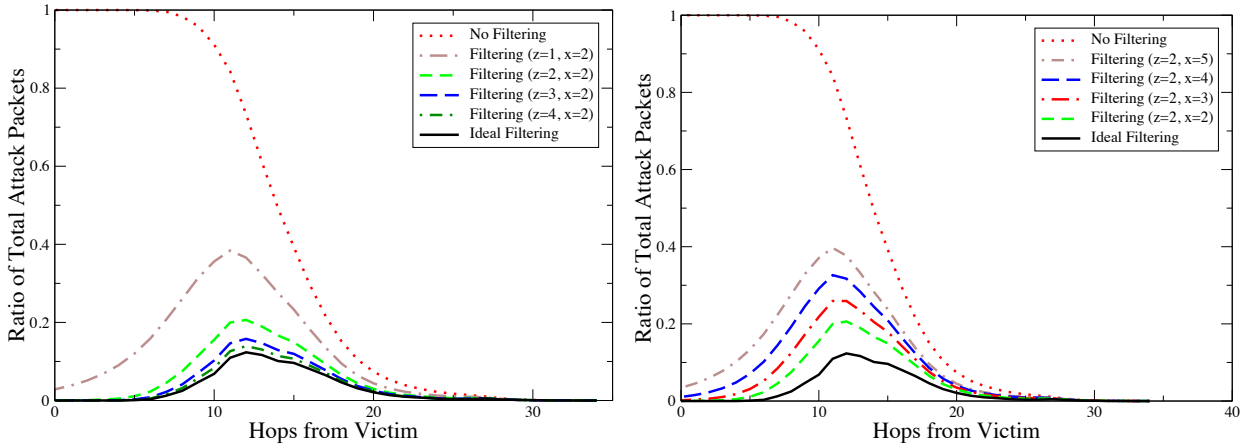
$$P(x,z) = 1 - \left(1 - \frac{1}{2^z}\right)^x \tag{5.2}$$

For a path of d routers, the probability of an attacker correctly generating a random marking is

$$P(x,z,d) = \left[1 - \left(1 - \frac{1}{2^z}\right)^x\right]^d \tag{5.3}$$

For a typical path length of 15 routers, the probability that an attacker correctly guesses a marking becomes $P(x=2,z=4,d=15) = 10^{-13}$ which is negligible.

Figure 12 shows how effective SIFF is in defending against DoS for different combinations of z and x. SIFF achieves this without significant per-flow state, and is thus a great way of providing capabilities to defend from DDoS.



(a) Performance for various values of z, ($x = 2$).

(b) Performance for various values of x, ($z = 3$).

Figure 12: Packet filtering performance for varying bits per router (z) and window sizes (x) [38]

5.7 Puzzles

Puzzles [34] are an excellent way for protecting against resource starvation attacks (section 3.2) on servers for which a single request induces significant load. The server can require the client to perform a small computation (solve a puzzle) before the server performs the expensive operation. While for the average client this is a minor unnoticeable cost, for an attacker performing DoS, sending packets of adequate throughput is impossible unless the client has access to atypically huge resources, such as a mainframe.

The typical puzzle works as follows: the server calculates the value T which is based on a MAC whose key is only known to the victim.

$$T = MAC_K(ClientIP, ClientPort, Timestamp, ISN) \quad (5.4)$$

The server conceals the first r bits of the value T and transmits it to the server along with the hash of T. The server then computes on average 2^{r-1} hash operations to find the r-bit value x needed to complete T.

$$S \rightarrow C : [T]_{m-r}, r, H(T) \quad (5.5)$$

$$C : H(x||[T]_{m-r}) = H(T) \quad (5.6)$$

$$C \rightarrow S : T, r, x \quad (5.7)$$

Puzzles allow the server to almost statelessly separate legitimate flows from attack flows. Some care needs to be taken so puzzles are not vulnerable to replay attacks and that the puzzle generation/verification does not turn into an attack vector in itself.

Without knowing the amount of processing power available to the attacker, it is hard to set up the puzzle. Puzzle Auctions are a way to fix this problem, by having each host auction off the amount of computation they are willing to make. This can be done by having each host auction the number of bits r it is willing to search, with an average of $2^{r-1} * 1\mu s$ computation required. Upon a DoS, the server allocates connections or bandwidth to the highest bidders, thereby maintaining fairness. This is done by mapping the each bid to a priority in a priority queue or a weight in Weighted Fair Queuing.

Because in most botnets the resources of the zombies are not significant, so as not to alert the user, zombies will be unable to commit to high values of r and the legitimate hosts will win. If legitimate hosts see that there is a DoS and they can't reach the victim, they will simply raise their bid until they outweigh the attack packets. Even if the attack packets attempt to compete with the user, DoS will not be possible because the high computational load will render the attacker unable to provide the high throughput necessary.

Puzzle Auctions have been shown to be fully compatible with TCP clients and even interoperable with unmodified TCP stacks. They can therefore be presently deployed without breaking network functionality.

5.8 SYN Cache and Cookies

When machines receive a SYN packet, they need to store some data in order to identify the connection and its options (window size, ISN, TCP header options). Even if they ignore those details, they still have to be able to retransmit their SYN-ACK if an ACK is not received within a timeout period specified by the TCP standard. Unfortunately, this gives rise to SYN flooding attacks, explained in section 3.6.

SYN Cache and Cookies [19] attempts to prevent DDoS during a SYN flood. During average load times, the server maintains a SYN cache, where each SYN packet is stored in a global hash table with fast lookup times $O(1)$. The hash value used is of the form

$$H = MAC_K(SrcIP, DstIP, SrcPrt, DstPrt) \quad (5.8)$$

where the value K is know only to the server. This is done to prevent Algorithmic Complexity Attacks (section 3.3). The hash table has an upper bound on the number of entries in the table and hence amount of memory used. If new entries overflow the per-bucket limit, the oldest packets are dropped first.

When SYN flooding is experienced, the server switches from using a SYN cache to SYN cookies. In SYN cookies the server *statelessly* encodes necessary information in the ISN of its SYN-ACK, so that when it receives it back with an ACK it can pick up the connection from where it was left. This requires no storage on the server during the connection establishment phase.

In [19], the client's MSS size is downsampled to 4 predefined values, denoted by M'. K_t is a time-dependent secret know only to the server. The 32-bit cookie C is then generated as follows:

$$H = H(SrcIP, DstIP, SrcPrt, DstPrt, K_t) \quad (5.9)$$
$$c_1 = ([H]_{24..0} \,||\, [\text{WindowIndex}]_{7..0}) \quad (5.10)$$
$$c_2 = SrcISN \quad (5.11)$$
$$c_3 = M' << 5 \quad (5.12)$$
$$C = c_1 \oplus c_2 \oplus c_3 \quad (5.13)$$

When the server receives back the ACK from the server, he first subtracts 1 from the ISN and XORs it with the cookie. He then XORs the TCP options to obtain c_1. He verifies the hash and that the key K_t has not expired and grants the connection to the client.

Because the DDoS attacker cannot hear the replies from the server directed to the spoofed address he sent the SYN from, the attacker will not be able to initiate a connection using an ACK. Since SYN cookies are stateless and the server never receives an ACK back, the SYN flooding attack is completely averted. In order to guess the correct cookie and cause the server to reserve a connection, the attacker needs to perform an average of 2^{23} hashes, each of $1\mu s$, which will take about 8 seconds to do. Since 8 seconds is longer than the timeout period for K_t, the attacker simply cannot perform a DDoS using a SYN flood.

Figure 13 shows the performance of SYN cache and SYN cookies during an attack. Most connections are completed within $300\mu s$. A simple comparison with

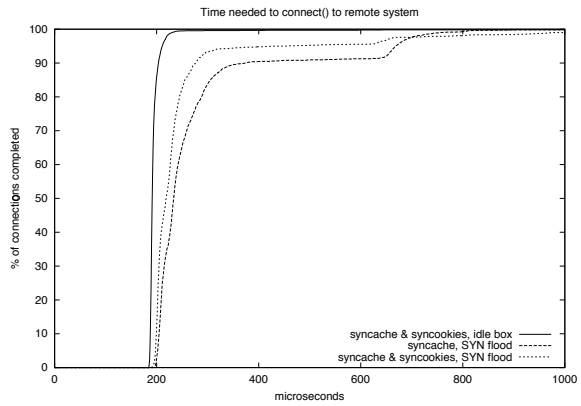


Figure 13: SYNCache and SYNCookies performance during SYN Flood. Compare with figure 3. [19]

Figure 3 in section 3.6 (which shows us the vast deterioration experienced by a system under SYN flood) proves the effectiveness of this measure against SYN flooding.

6 Deterrence

The reason DoS is still a debilitating attack after so many years is that the current Internet has failed to provide *Deterrents* to attackers. This is not because of unwillingness or lack of incentives, but because of its design, which permits anyone on the Internet to send packets to anyone, with any source IP and without being detected.

One way countries can provide deterrence to DoS is through legal deterrents, i.e. imposing large fines on attackers that perform DoS. In the US for instance, the Computer Fraud and Abuse Act includes penalties that include years of imprisonment. In the UK the Police and Justice Act of 2006 specifically outlaws DoS attacks and offers 10-year maximum imprisonment. Other countries have similar legislation.

There are three primary problems with legal deterrents. First, because of the global trust-all structure of the Internet, attacks can always be launched by countries that do not provide a legal framework against DoS attackers. Second, because attackers typically use zombies that are spread across the world and are infected without the knowledge of the owners, the host performing the DoS is usually not the attacker himself. Third and most importantly, the Internet does not provide *Accountability*; since any host can spoof its IP address and send packets, there is no way of holding someone accountable for a packet or tracing an attack back to its origin. Even when DoS attackers are caught, they are never caught by tracing the packets back to the attacker, but through other forensic residue that the attacker left behind.

In order to provide credible deterrents to DoS attacks, we have to provide accountability, specifically *Host Ac-*

countability. This allows us to hold a host accountable for its actions and reliably trace back a packet to its original sender. By providing accountability, we allow victims of DoS attacks to legally prosecute their attackers and demand compensation for the attack.

In the following sections we outline accountability protocols for both the current Internet and a clean-slate Internet.

6.1 Accountability for the Current Internet

Because the current Internet does not offer accountability by design, adding accountability incrementally is a dirty procedure that introduces significant overhead. Below, we present two protocols that attempt to do so.

Accountability as a Service Accountability as a Service (AAaS) [8] offers accountability as a first-class network service, separate from routing and addressing. This creates a new economic and legal model, where the client subscribes to an accountability service that requires an escrowed deposit.

The accountability service provides authenticated clients with identifiers that can be used to mark packets as accountable. A victim that is experiencing DoS can report that abuse is taking place from one of the accountability provider's clients by sending it a message with the traffic identifier of the unwanted data. The accountability provider can then choose to filter the portion of its client's traffic that matches that specific identifier. If the abuse is serious enough that it requires legal action, the accountability service may release the client's identity and use some his escrowed deposit to pay back the victim.

The authors propose two simple approaches for providing AAaS in the current Internet.

In the strawman protocol, the accountability provider gives its customers signed client certificates. Each sender then needs to sign each packet with its private key. The receiver retrieves the accountability provider's certificate to verify that the client's certificate is signed by a provider it trusts, and then verifies that the signature of the packet is correct. The strawman protocol provides non-repudiation, because only the host that signed the packet has the private key that corresponds to it. However, the use of public key cryptography for every packet is a significant performance impediment.

For this reason, the authors propose a more efficient protocol, in which each ISP and client has a certificate with a Diffie-Hellman public key. This allows the sender to generate pairwise keys between him and every router along the path, and only use symmetric cryptography for authenticating subsequent transmissions along the same path. To do this, the sender S first determines the ISPs along the path. It then uses the Diffie-Hellman calculation to derive pairwise keys. It uses those keys for computing MACs of the packet for each of the ISPs, and then appends its certificate as well as each MAC to the packet.

$$S : \{S, g^s\}_{K_{CA}^{-1}} \tag{6.1}$$

$$ISP_i : \{i, g^i\}_{K_{CA}^{-1}} \ \forall i \tag{6.2}$$

$$S : K_i = g^{si} \ \forall i \tag{6.3}$$

$$S \rightarrow ISP_1 : pkt, \{S, g^s\}_{K_{CA}^{-1}}, MAC_{K_1}(pkt), MAC_{K_i}(pkt) \ \forall i \tag{6.4}$$

$$ISP_i : \text{check cert, } MAC_{K_i} \tag{6.5}$$

Each router along the way verifies the certificate and MAC and forwards the packet. We have therefore established an accountability chain in a hop-by-hop fashion. In case of a DoS, the receiver can know exactly who sent the packet and request filtering or even legal action.

In terms of deployment, it is important to mention that AAaS does not require adoption by a significant portion of clients to provide a benefit to the users. Instead, it provides instant gratification. During a DoS, the victim can remain resilient by guaranteeing bandwidth for clients using AAaS and providing best-effort service for other flows.

Bootstrapping Accountability In [20] the authors propose IP made Accountable (IPA). The design uses DNSSEC to securely bind an IP Prefix to an ASs public key. At the AS level, certificates are distributed in-band, through BGP. In order to decrease the overhead from sending the public key to each of the neighbors every time a BGP update is made, IPA uses three caches. The cached certificates are stored in a tree, where the root is ICANN. Every new certificate or revocation has to validate in that tree, or it is discarded. With these secure prefix-to-key bindings, we can now have sender accountability for whoever is sending the packets.

IPA uses reverse DNSSEC queries to obtain public keys for each source network address range on the Internet. The hierarchical structure of DNS works really well in this case, as the parent network provides the child with sub-delegation certificates. In case network prefixes do not fall under CIDR bounds, a small modification to DNS queries is proposed that includes the subnet mask for the network, e.g. `2/24.10.10.in-addr.arpa`.

When a router in a network sends a packet, he signs it with its private key. The receiving router performs a reverse DNSSEC query and obtains the public key, signed by the sender's AS. The receiver retrieves the AS's key and verifies the signature. It then verifies the signature on the data. If the signatures verify, the packet is valid.

During a DoS, the victim can ask all senders to sign their packets if they want to connect. In order for the attacker to connect, he would have to reveal his identity and therefore face legal consequences.

In terms of deployment, IPA does not replace current functionality and is affordable to deploy, since a big percentage of the infrastructure is already in place. In addition, because IPA can be incrementally deployed, early adopters get a lot more value for adopting. However, as with any approach to add accountability to the current Internet, it increases overhead. There are also some race condition timing issues that can lead to a deadlock during revocation/key rollover.

6.2 Deterrence in the Next Generation Internet

We have seen how the structure of the current Internet does not permit accountability to be established in an attractive manner. For this reason, researchers have designed protocols to offer accountability using a clean-slate approach. Accountability has to be an integral part of the Next Generation Internet. Below, we present two protocols that offer accountability and deterrence in a clean-slate approach.

AIP The Accountable Internet Protocol (AIP) [3] provides a means of replacing the IP protocol to ensure accountability in a network, enabling us to detect and trace the source of spoofing and DoS, bootstrap identity into new protocols and provide additional security to existing protocols. AIP introduces two main security fixes for attacks that the current structure of the IP protocol is unable to not only prevent, but even detect: first, a procedure for verifying identities of packets, and second, a shut-off protocol that relies on trusted network cards' ability to halt packet transmissions and establish filters at the source of DoS attacks.

In AIP, each host and AS has a public key certificate. The address of each end host is the hash of its public key, or the Endpoint Identifier (EID). Likewise, the address of each AS is the hash of its public key, or the Autonomous Domain Identifier (AID) (note that AID is used here to avoid confusion, the actual paper calls this the AD: Autonomous Domain). By using the public keys themselves as the network addresses, AIP avoids the use of a PKI to bind an address to a public key and tremendously simplifies the task of providing accountability.

Verification of a packet's source address is done on a per-hop basis by using unicast reverse path forwarding. Each router along a path sends a verification packet to the source. The source is required to respond by providing a signature of the packet:

$$R_i \rightarrow S : V = \text{HMAC}_{K_t}(S_{AID} : S_{EID}, D_{AID} : D_{EID}, \text{interface}) \quad (6.6)$$

$$S \rightarrow R_i : \{K_{EID}, V\}_{K_{EID}^{-1}} \quad (6.7)$$

Because only the legitimate source has the public key, the identity of the source is established and cached by the router. Spoofing is not possible.

AIP also provides a shutoff protocol, through which a victim V of a DDoS attack can request the zombie Z's NIC to filter the packets directed to it. The shutoff protocol is of the form:

$$Z \rightarrow V : \text{Packet } P \quad (6.8)$$

$$V \rightarrow Z : \{K_V, TTL, Hash(P)\}_{K_V^{-1}} \quad (6.9)$$

Since the victim signs it with its private key, it cannot be spoofed. Since the victim has to show proof that it received a packet from that host (through $H(P)$), it prevents abuse by attackers that would use the shutoff itself to launch a DoS. In order for this to work, the zombie's NIC has to be untamperable, so that the zombie cannot ignore the shutoff packets without facing legal consequences.

AIP is a great way to provide accountability in a clean-slate approach without the use of a complicated PKI. A victim can reliably detect an attacker. In addition, through the shutoff protocol he can choose instruct the attacker to stop sending attack packets during a DDoS. As a result, AIP provides a credible deterrent for DDoS attacks.

SCION SCION [39] is a proposal for a new Internet architecture designed to provide, among others, explicit trust and accountability. SCION separates the Internet into a set of independent trust domains (TDs). A TD will most probably be a country or an alliance of countries. Each TD is completely independent from the other TDs, and actions in one TD cannot affect in another TD. Each TD has its own legal framework, which it strictly enforces. Therefore, SCION reduces the number of trust checks that a network needs to perform to a small number of trusted domains, instead of each node having to pairwise evaluate the other.

As shown in Figure 14, each TD is comprised of multiple Autonomous Domains (ADs) which are the equivalent of today's ASs. ADs that have peerings with ADs in other TDs are called Core ADs. The set of all Core ADs for a TD makes up its TD Core. *Inter-TD* routing happens using established manually-configured paths between the TD Cores, since there is only a handful of TDs. *Intra-TD* routing happens using construction beacons, where the TD Core regularly polls each AD for a

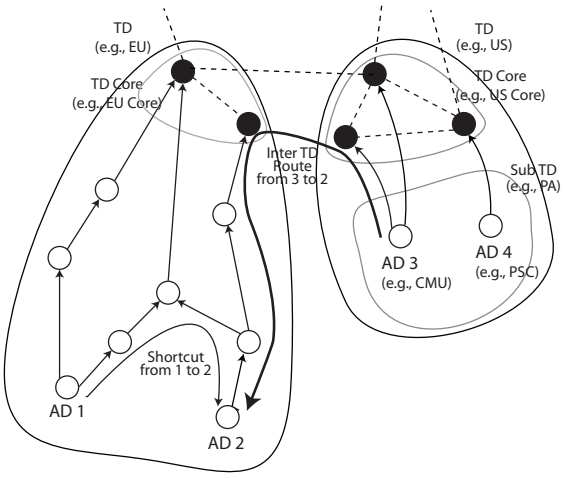


Figure 14: SCION trust domain architecture. Dashed lines indicate peering relationships while arrows indicate customer-provider relationships [39]

set of k up-paths it would like to use to connect to the TD Core. Since these construction beacons are authenticated, SCION prevents the control plane DoS attacks that were demonstrated in section 3.9, such as Prefix Hijacking, Routing Path Falsification and Blackholing. In addition, since each destination announces its up-paths, it gets to choose what paths incoming traffic will take, and thus route around failures and congestion, preventing DoS. As opposed to the current Internet where the sender has all the control, in SCION the control is on the receiver, as it ought to be.

Isolation between TDs allows enforcement of filters and restrictions on inter-TD traffic that prevents DDoS. An attacker can only attack nodes in his own TD and only at the data plane. But because each TD is a legally coherent domain, it can credibly enforce legal deterrents and pursue attackers. In addition, SCION uses AIP, so every attack can be traced back to its sender.

SCION provides not only an accountability system through its use of AIP, but also transforms the Internet architecture in such a way that a host is not vulnerable to attack by a host in a legal system it cannot control. For these reasons, attackers will be reluctant to perform DDoS, solving the problem once and for all.

7 Conclusion

We have started this discussion by looking at the types of DoS attacks currently available. DoS attacks are of varying sophistication and may have devastating consequences on the victims, such as monetary loss. For Data Plane DoS, we looked at Bandwidth, Resource Starvation, Algorithmic Complexity, TCP and ICMP as vectors for this attack. We also provided Reflector and Smurf as methods for performing the DoS. We also looked at Control Plane attacks as ways to target the Control Plane and the Data Plane for DoS.

As a result of poor design, the current Internet does not offer DoS victims Detection, Defense, Resilience or Deterrence when attacks take place. For this reason, we looked at a number of commercial and research approaches for each. In Detection, we looked at IDS, Traffic Measurement and Accounting, and Inference as methods of detecting DoS. In Defense and Resilience, we looked at Ingress Filtering, Pushback, Over Provisioning, IPS, CAPTCHAs, Capabilities, Puzzles, and SYN Cache and Cookies. In Deterrence, we looked at two incremental mechanisms for accountability in the current Internet, Accountability As a Service and Bootstrapping Accountability in the Internet We Have. We also looked at two clean-slate approaches for accountability, AIP and SCION.

We conclude by emphasizing that the only way to completely eliminate DoS in an efficient and effective manner is by adopting a clean-slate approach for the future Internet.

References

[1] Z. Ahmad, J.L. Ab Manan, and S. Sulaiman, *Trusted Computing based open environment user authentication model*, Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on, vol. 6, IEEE, pp. V6–487.

[2] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, *Holding the Internet accountable*, ACM HotNets-VI (2007).

[3] David G. Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker, *Accountable Internet Protocol (AIP)*, Proc. ACM SIGCOMM (Seattle, WA), August 2008.

[4] S. Axelsson, *The base-rate fallacy and the difficulty of intrusion detection*, ACM Transactions on Information and System Security (TISSEC) **3** (2000), no. 3, 186–205.

[5] BBC, *Anonymous wikileaks supporters explain web attacks*, `http://www.bbc.co.uk/news/technology-11971259`.

[6] S.M. Bellovin, *Security problems in the TCP/IP protocol suite*, ACM SIGCOMM Computer Communication Review **19** (1989), no. 2, 32–48.

[7] S.M. Bellovin, D.D. Clark, A. Perrig, and D. Song, *A clean-slate design for the next-generation secure internet*, Relatório técnico, Pittsburgh, PA: Report for NSF Global Environment for Network Innovations (GENI) Workshop, 2005.

[8] A. Bender, N. Spring, D. Levin, and B. Bhattacharjee, *Accountability as a service*, Proceedings of the 3rd USENIX workshop on Steps to reducing unwanted traffic on the internet, USENIX Association, 2007, p. 5.

[9] R. Chen, T. Giedgowd, and L. Greaves, *Understanding the Threat of Denial-of-Service Attacks in Society Today*.

[10] CNET, *How pakistan knocked youtube offline*, http://news.cnet.com/8301-10784_3-9878655-7.html.

[11] _____, *News sites swamped following michael jackson's death*, http://news.cnet.com/8301-1023_3-10273325-93.html.

[12] S.A. Crosby and D.S. Wallach, *Denial of service via algorithmic complexity attacks*, Proceedings of the 12th conference on USENIX Security Symposium-Volume 12, USENIX Association, 2003, pp. 3–3.

[13] C. Estan and G. Varghese, *New directions in traffic measurement and accounting*, ACM SIGCOMM Computer Communication Review, vol. 32, ACM, 2002, pp. 323–336.

[14] C. Farkas, G. Ziegler, A. Meretei, and A. L ”orincz, *Anonymity and accountability in self-organizing electronic communities*, Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society, ACM, 2002, pp. 81–90.

[15] L. Garber, *Denial-of-service attacks rip the Internet*, IEEE Computer **33** (2000), no. 4, 12–17.

[16] M. Handley, V. Paxson, and C. Kreibich, *Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics*, Proceedings of the 10th conference on USENIX Security Symposium-Volume 10, USENIX Association, 2001, pp. 9–9.

[17] The Tech Herald, *Themis: Looking at the aftermath of the hbgary federal scandal*, http://www.thetechherald.com/article.php/201112/6951/Themis-Looking-at-the-aftermath-of-the-HBGary-Federal-scandal.

[18] J. Ioannidis and S.M. Bellovin, *Implementing pushback: Router-based defense against DDoS attacks*, Proceedings of Network and Distributed System Security Symposium, vol. 2, Citeseer, 2002.

[19] J. Lemon et al., *Resisting SYN flood DoS attacks with a SYN cache*, Proceedings of the BSDCon, 2002, pp. 89–97.

[20] A. Li, X. Liu, and X. Yang, *Bootstrapping Accountability in the Internet We Have*.

[21] 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** (2002), no. 3, 62–73.

[22] A. McCue, *Bookie reveals $100,000 cost of denial-of-service extortion attacks*, http://www.silicon.com/technology/security/2004/06/11/bookie-reveals-100000-cost-of-denial-of-service-extortion-attacks-39121278/.

[23] J. McHugh, *Intrusion and intrusion detection*, International Journal of Information Security **1** (2001), no. 1, 14–35.

[24] J. Mirkovic and P. Reiher, *Building accountability into the future Internet*, Secure Network Protocols, 2008. NPSec 2008. 4th Workshop on, IEEE, 2008, pp. 45–51.

[25] D. Moore, G.M. Voelker, and S. Savage, *Inferring internet denial-of-service activity*, Proceedings of the 10th conference on USENIX Security Symposium-Volume 10, USENIX Association, 2001, pp. 2–2.

[26] Netcraft, *Operation payback aborts attack against amazon.com*, http://news.netcraft.com/archives/2010/12/09/operation-payback-aborts-attack-against-amazon-com.html.

[27] O. Nordstrom and C. Dovrolis, *Beware of BGP attacks*, ACM SIGCOMM Computer Communication Review **34** (2004), no. 2, 1–8.

[28] T.H. Ptacek, *Insertion, evasion, and denial of service: Eluding network intrusion detection*, Tech. report, SECURE NETWORKS INC CALGARY ALBERTA, 1998.

[29] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, *TCP congestion control with a misbehaving receiver*, ACM SIGCOMM Computer Communication Review **29** (1999), no. 5, 71–78.

[30] M. Schuchard, A. Mohaisen, D. Foo Kune, N. Hopper, Y. Kim, and E.Y. Vasserman, *Losing control of the internet: using the data plane to attack the control plane*, Proceedings of the 17th ACM conference on Computer and communications security, ACM, 2010, pp. 726–728.

[31] A. Studer and A. Perrig, *The coremelt attack*, Computer Security–ESORICS 2009 (2010), 37–52.

[32] Beardsley T. and J. Qian, *The tcp split handshake: Practical effects on modern network equipment*, `http://nmap.org/misc/split-handshake.pdf`.

[33] L. Von Ahn, M. Blum, N. Hopper, and J. Langford, *CAPTCHA: Using hard AI problems for security*, Advances in CryptologyEUROCRYPT 2003 (2003), 646–646.

[34] X.F. Wang and M.K. Reiter, *Defending against denial-of-service attacks with puzzle auctions*, (2003).

[35] P. Watson, *Slipping in the Window: TCP Reset attacks*, 2004.

[36] WSJ, *What does it cost amazon.com to go dark?*, `http://blogs.wsj.com/numbersguy/what-does-it-cost-amazoncom-to-go-dark-374/`.

[37] Y. Xiao, *Accountability for wireless LANs, ad hoc networks, and wireless mesh networks*, Communications Magazine, IEEE **46** (2008), no. 4, 116–126.

[38] A. Yaar, A. Perrig, and D. Song, *SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks*, (2004).

[39] X. Zhang, H.C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D.G. Andersen, *SCION: Scalability, Control, and Isolation On Next-Generation Networks*.