Terminus: the end of the line for DDoS

Felipe Huici, Mark Handley
University College London
[f.huici|m.handley]@cs.ucl.ac.uk

Abstract

Denial-of-Service attacks continue to grow despite the fact that a large number of solutions have been proposed in the literature. The problem is that few are actually practical for real-world deployment and have incentives for early adopters. We present *Terminus*, a simple, effective and deployable network-layer architecture against DoS attacks that allows receivers to request that undesired traffic be filtered close to its source. In addition, we describe our implementation of each of the architecture's elements using inexpensive off-the-shelf-hardware, and show that we can filter very large attacks in a matter of seconds while still sustaining a high forwarding rate even for minimum-sized packets. We conclude by discussing initial deployment incentives.

1 Introduction

For the last few years, distributed denial of service (DDoS) attacks have been a significant problem for the operators of high-profile network servers and for Internet providers providing connectivity for them. There have been a number of well-publicized attacks, but also a huge number of less known attacks, such as those on many organizations dedicated to fighting spam and phishing.

Currently there is no automatic way to shut down traffic from the zombie hosts involved in such attacks, even though the actual IP addresses of these compromised machines are usually known. What is needed is a simple, practical and robust way to request from the ISP where attack traffic originates that this traffic is blocked.

Many solutions have been proposed to DDoS attacks, but few are actually practical for real-world deployment with incentives aligned appropriately so that deployment is likely. Often proposed solutions have fixated on solving all the corner cases of the problem, but the resulting solutions end up requiring too much change to the Internet for deployment to be likely. For example, Internet transit providers do not usually suffer during DDoS attacks - they have sufficient capacity, so they have no incentive to address the issue. Any solution therefore that requires changing routers in the network core would impose costs on players that do not receive any significant benefit. Thus, short of preventing hosts being compromised in the first place, viable solutions will primarily involve significant changes only at customer-facing ISPs or on the end-systems themselves.

There is a fundamental debate between anti-DDoS researchers about whether it is possible to distinguish good traffic from bad traffic. The people actually defending against such attacks find this debate futile - current attacks are obvious, and there is frustration at the lack of mechanisms to deal with them. For these people, the perfect is the enemy of the good [**?**], and has resulted in no progress in deploying mechanisms to tackle *any* of this problem space.

In this paper we describe simple and deployable mechanisms that would significantly *raise the bar* for DDoS attackers. Our belief is that there is great benefit in having a mechanism that can filter traffic from an attacking host close to its source. This would allow today's brute force attacks to be shut down with little effort. Likely this is not a complete solution; in response attackers will need to mimic flash-crowds, but this requires much larger botnets to inflict damage and moves the problem into the space where higher level solutions such as captchas[**?**] or similar mechanisms become viable.

2 Terminus Architecture

Possible DDoS solutions divide into proactive[**?**, **?**] and reactive[**?**, **?**] classes. While proactive solutions might in principle provide better defenses, our conclusion is that to be incrementally deployable without significant core-network changes, reactive mechanisms are most viable as general purpose solutions. Our architecture, Terminus, is just such a reactive system, in that it enables a victim to request that specified traffic be stopped close to its sources, before it has had a chance to aggregate.

We start from the assumption that the victim of an attack can tell with reasonable accuracy which traffic is bad. For the purpose of this paper, we refer to the detector as an intrusion detection system (IDS), but in many cases it may be the server itself. We then deploy filtering boxes near sources of traffic, since the size of botnets being reported [**?**] means it is not possible to defend against large flooding attacks near to the destination, even if the victim is connected to well-provisioned links. For the system as described so far to be viable, the following issues must be addressed:

- How to find the right filtering box to request a filter?
- How to validate filtering requests to ensure spoofed requests cannot become a channel for attack?
- How to avoid a spoofed traffic flood triggering a filter request that blocks legitimate traffic?
- How to provide incentives for early adopters, and especially how to provide incentives to deploy filtering boxes?

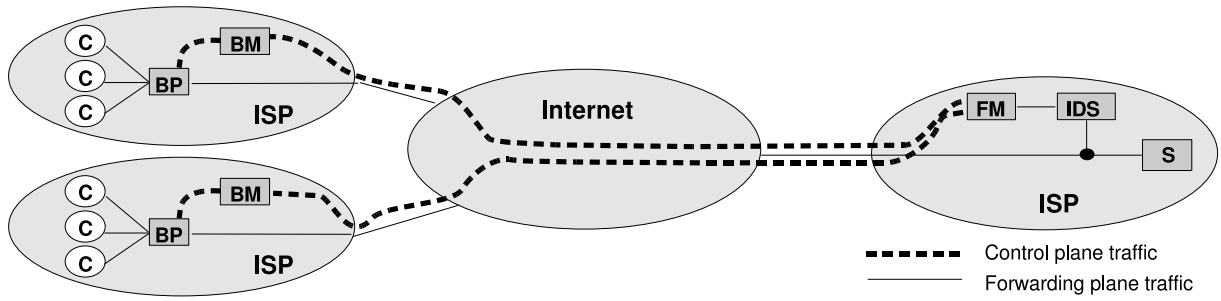


Figure 1: Terminus architecture showing the location of its elements. C stands for client, R for router and S for server.

From a deployment point of view, it is critical that the mechanisms will work even though the ISP of the bot and the ISP of the victim may be remote from each other, and have no prior business relationship. The only form of contractual arrangement that seems viable is that of pairwise service level agreements (SLAs) between neighbouring ISPs. Thus any architecture must assume that this is the contractual mechanism from which end-to-end filtering services are built. We primarily use such SLAs to distinguish spoofed traffic to avoid the second issue listed above. The rest of this section explains the architecture in greater detail, including solutions to all of these issues.

2.1 Edge Filtering

Terminus places special control points called *border patrols* (BPs) in ISPs, as close to the sources of traffic as possible (see Figure 1). An ISP deploying Terminus (a *"Terminus ISP"* in this paper) configures its network so that traffic from these sources is forced through border patrols. In this way, each BP can later be asked to install filters for traffic going through it, and, since it is close to the sources, the total aggregate throughput it forwards should be manageable.

With this mechanism in place, the victim of an attack would have to know which BP the traffic came through. In a perfect world, this would be as simple as looking at the source IP address of the malicious traffic and deriving a mapping between this and the correct BP. Unfortunately, because of spoofing, this information cannot be trusted.

The problem is not that most of the sources on the Internet spoof, but that a receiver cannot tell the difference between a spoofed packet and a non-spoofed one. All is not lost however, and we can use the border patrols mentioned above along with some added mechanism to combat this problem.

The idea is to use a "true source" bit in the IP header to mark whether the source IP address field in a packet is in fact the address of the host that originated the packet. As the packet travels from source to destination, Terminus ISPs have their ingress edge routers set or unset this bit depending on whether the packet came from a peering link to a Terminus or legacy ISP; the routers would know this through pairwise contractual agreements. In this way, if a packet traverses only Terminus ISPs on its way from source to destination, it will arrive with its true source bit set, and its source IP address can be trusted[1]. Of course, Terminus ISPs are assumed to perform ingress filtering, either at their routers or their border patrols.

Figure 2 gives a couple of different scenarios illustrating this mechanism. Packets originating at ISP A and going to the server hosted by ISP G will arrive with the true source bit set. Packets from ISP B, on the other hand, will have this bit unset by router E2, since it knows that its link connects to a legacy ISP. Finally, any packet from ISPs C and D will arrive with the bit unset, since router G2 knows ISP F to be a legacy ISP.

Thanks to the border patrols and the true source bit, a victim can now know whether the source IP address in a packet is valid or not. Naturally, the server will only be able to trust the source IP address field for packets that traversed a Terminus ISP-only path, but as deployment progresses this will become the common case.

2.2 Filtering Requests

We now have control points (border patrols) deployed on outgoing paths to the rest of the Internet, and a way for a victim to determine where a packet came from (the combination of the true source bit and the IP source address field). When an attack is detected, the next step is to send filtering requests.

Although the IDS or server acting as the detector knows what it wants to filter, it does not know where to send the request. To avoid burdening an already busy system, we offload this to another box, which we will call a *filter manager* (FM). The IDS is simply configured with the address of its local FM, and sends all its filtering requests there (Figure 1).

An FM needs to map an IP address to be filtered to the address of the border patrol handling traffic from that IP address. There are many ways to do this, but our preferred solution is to use a peer-to-peer flooding protocol to distribute digitally signed bindings to all FMs worldwide. This is essentially the same robust flooding mechanism as was proposed for Push DNS[**?**], and is extremely resilient to attack.

The size of this "routing" table would certainly be manageable: only one entry would be required per AS, or about 20,000 entries in the current Internet. Each entry could con-

[1]Our earlier work[**?**] used a similar concept to the true source bit; however the rest of the Terminus architecture is completely different and considerably simpler.

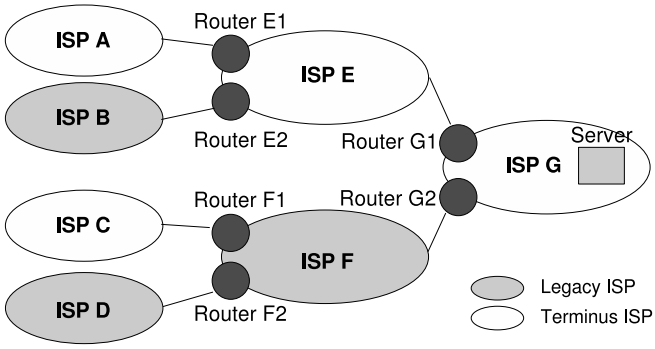


Figure 2: True source bit scenarios.

sist of an IP address and a set of prefixes, aggregated as much as possible, representing the clients of the ISP that sit behind border patrols.

Using this mapping table, the FM determines the address of the host at the remote ISP from which it needs to request a filter. Such a host, called a *border manager* (BM), needs to forward the filtering request to the appropriate border patrol (in a small deployment, the BM could be the same host as the BP; the architecture places no requirements on how this should be implemented). To accomplish this, the ISP could install the necessary mappings of source IP addresses to border patrols, and update them should these change.

Once filters are installed we also need a way of removing them. Simplest is to include, along with the filtering request, information about how a filter should expire. The BP then removes the filter when the criteria are met; the expiration could be time-based or even rate-based. When filters are installed and attack traffic subsides the victim has no obvious way to know if the attack has actually ceased or if it is the filtering mechanism that is being effective. The IDS could of course request the filter be removed and measure the effects, but perhaps a better solution is to provide a way to retrieve filter traffic statistics from BPs. This allows the IDS to explicitly remove unneeded filters, and provides a more flexible tool for the IDS to use as it sees fit. The filtering protocol described in Section 4.2 supports all these approaches.

We now have all the basic elements needed to filter a DDoS attack. Traffic from sources traverses border patrols and arrives at the server, where a nearby IDS detects the attack, determines the malicious sources, and sends a filtering request to its local filter manager. The FM, in turn, uses the mapping of source IP address to border manager obtained via the peer-to-peer network to send the necessary filtering requests to the appropriate border managers. These, in turn, ensure that the requests arrive at the appropriate border patrols that the traffic is actually going through, where the malicious traffic is finally filtered.

2.3 Defending Against Bots at Legacy ISPs

If Terminus were fully deployed, it would be possible to filter large DoS attacks even if there still remained a few legacy ISPs. However, during initial deployment the story is likely to be different, with legacy ISPs being the norm rather than the exception. As a result, we need to provide some level of protection for a victim being attacked by sources hosted by legacy ISPs.

To this end, we can make use of the true source bit already described. This bit not only denotes that the IP source address is valid, but it also says that the packet originated at and has traversed Terminus ISPs. It makes sense for a Terminus ISP to reward other Terminus ISPs, and so we can install diffserv classifiers at the edge routers of the destination ISP (or indeed other Terminus ISPs on the path if they wish to do so), sending packets that have the true source bit set to a higher priority queue. As a result, packets from legacy ISPs will always get lower priority and, during an attack, potentially little or no service.

3 Protecting the Architecture

With such a powerful mechanism in place, care must taken to make sure that the architecture is not used as a DoS tool in its own right; protecting it from such misuse and dealing with other forms of attacks is the topic of this section.

3.1 Validating Filtering Requests

As described so far, an attacker could contact a border manager and request a malicious filter. A nonce exchange suffices to avoid this: on receipt of a filter request, the border manager sends a random nonce back to the filter manager, and only installs the filter when it gets the nonce echoed back. This serves to validate that the IP address of the FM is not spoofed (of course this is not strictly necessary if the true source bit is set in the request, but as Internet paths are asymmetric we cannot count on this being the case, and the extra validation is cheap in any event).

Validating the FM's IP address is not sufficient though: it is also necessary to validate that this particular FM is authorized to request a filter for this particular destination IP address. In essence we need the reverse mapping table from that by which the FM discovers the BM's address, and the same peer-to-peer distribution of digitally signed mappings can be used to distribute it. Likely the CAs for these signatures will be the RIRs, as they already handle IP address allocations to ISPs. It is worth noting that nonce exchange will not stop an attacker on the path between the border manager and the filter manager; however the additional risks are minimal as a compromised router can already filter the traffic by simply dropping it.

With this in place, upon receiving a filtering request the border manager will inspect its source IP address. If the mapping between this address and the destination address of the actual filter exists in the set of mappings distributed using the peer-to-peer network, then the BM will issue a nonce. This nonce will reach the FM, which will echo it if it had, in fact, issued a filtering request. Finally, the BM will contact the appropriate border patrols to block the unwanted traffic.

3.2 Triggering Requests Through Spoofing

Although the architecture ensures that filter requests come from legitimate parties, it might still be possible for an attacker to spoof client traffic to trigger a filter against traffic from an unsuspecting legitimate client. The list of possible attack scenarios have to do with the location of the attacker with regards to the victim. Only five such scenarios exist:

1. The attacker is in a legacy ISP that permits spoofing.
2. The attacker is in a legacy ISP that performs ingress filtering.
3. The attacker is in a different Terminus ISP from the legitimate client.
4. The attacker is in the same Terminus ISP as the legitimate client but behind a different BP.
5. The attacker is behind the same BP as the legitimate client.

In the first scenario, the attacker can spoof the client's address. However, the attacker's packets will arrive with their true source bit unset and, consequently, the filter manager will know not to issue a filtering request. The next two scenarios are impossible: an attacker from an ISP that performs ingress filtering simply cannot spoof the address of a client in a different ISP . The attack described in the fourth scenario is easily preventable by either performing ingress filtering at the border patrols or by ensuring that the ISP uses the true source bit internally. In the last scenario, the border patrol cannot tell traffic from the attacker and the victim apart. In essence, the problem is a local one, and the ISP can use ingress filtering or other local measures to combat it.

3.3 Protecting BPs, BMs and FMs

For Terminus to be successful, all of its components must be robust against attack. Attacking border patrols, for instance, could deny traffic from clients from reaching a server. Under full deployment, there will be a significant number of BPs, and so DoSing a server by stopping client traffic would prove very difficult at best. During initial deployment, however, there might only be a few BPs deployed and the attack might be effective. The solution is simple: do not advertise the BPs' address prefixes externally via BGP. If they are not externally reachable they are not susceptible to attack.

Border managers, on the other hand, do have to be externally visible in order to receive filtering requests. However, these boxes are not on the fast path, and so can devote all their resources to the control protocol. The BM implementation described in Section 4.2 can not only service requests at a fast rate, but also ensures that no state is held for a client before it has responded to a nonce. In the end, overloading a BM with requests only prevents filter installation. To allow a bot behind a BP managed by such a BM to continue an attack requires many bots to DoS the BM; this simply is not a good return on investment for the attacker.

One final element of the architecture that could be attacked is the filter manager. Again, this box is not on a fast path, and so it can devote all its resources to filter requests. More importantly, its traffic is constrained: there are a limited number of IDS systems from which it should accept requests, and the path for nonce requests from BM to FM will always be Terminus-enabled (or we would not have requested the filter in the first place).

4 Implementation and Results

Ideally we would like the mechanisms described so far to be implemented in hardware, perhaps as part of a router platform. In reality, however, and especially during the early deployment stages, it is unlikely that commercial vendors will adopt the approach without having seen some level of real-world deployment. Consequently, we have opted to implement the solution using off-the-shelf hardware to show its feasibility.

As with any performance evaluation, repeatability is key, and so using a controlled network testbed is a logical step. However, creating realistic attack scenarios in a testbed is problematic, since whatever scenarios are chosen, they could never be general enough to reflect real-world diversity. Despite this, it is possible to test each of the components of the architecture individually to see how they behave under heavy load and pathological cases, and thus provide some level of confidence with regards to the architecture's overall performance.

The following sections describe the implementation and performance results of the various components of the architecture. Section 4.1 discusses the testbed, including the hardware and software used to derive the results; section 4.2 illustrates the implementation of the control plane elements, giving figures for how quickly filters can be installed; section 4.3 discusses baseline as well as forwarding and filtering figures for the border patrol; finally, section 4.4 discusses the effects on performance of combining the control and forwarding planes at the border patrol.

4.1 Setup

The computers used to obtain the results were inexpensive 1U servers with two dual-core Intel Xeon 5150 processors running at 2.66GHz. These CPUs have 64KB of L1 cache on each core, with a 4MB L2 cache shared between the two cores. The systems had two dual-port Intel Gigabit Ethernet cards on 8x PCI Express slots[2]. We implemented the control-plane elements in C++ and for the forwarding plane we used Linux 2.6.16.13, version 1.5 of the Click modular router [**?**] and version 6 of the e1000 driver with polling extensions. The kernel used was generally uni-processor, except where otherwise stated.

4.2 Control Plane Performance

To transmit requests between the components of the control plane we designed and implemented the *Internet Filtering Protocol* (IFP). While the protocol supports various operations such as filter removal and retrieving attack traffic statis-

[2]Results on older machines with PCI-X buses turn out to be bus-limited

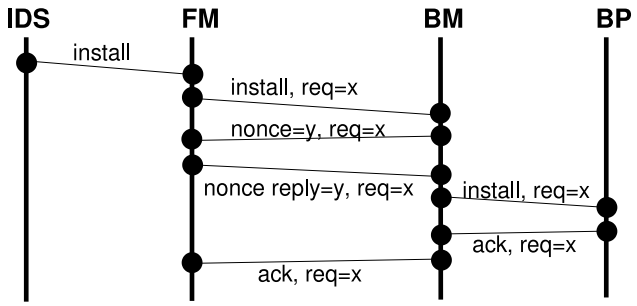


Figure 3: Filter installation request using the Internet Filtering Protocol.

tics, the results presented here focus on the most important operation, filter installation. To test the worst case, we used fine-granularity filters consisting of a source and destination IP address pair (though the protocol supports prefix-based and destination-only filters), and each IFP packet contained a request with a single filter.

We tested the performance of each of component individually. The first of these, the filter manager, listens to requests from an IDS or server. On receiving a filter install request, it assigns a random request number to it, looks up the mapping between the source address of the filter and the appropriate border manager, and forwards the request to that BM. When it receives a nonce from the BM, it echoes it along with the filter spec, and waits for the final installation acknowledgement from the BM (see figure 3). To test the FM's performance, the other components that it communicates with must not become a bottleneck. To achieve this, we implemented dummy versions of the IDS client and the BM which do the bare minimum. With this setup, the FM was able to sustain a rate of 75,000 requests/second. To put this in perspective, the largest botnet currently reported in the media contains around 1.5 million hosts [**?**], although not all hosts in such a large botnet may be used in any attack. Even for such a large botnet and using fine-granularity src/dst IP address filters, the FM would be able to filter all these bots in only 20 seconds.

The second component to test in the control plane is the border manager. The BM listens to requests from FMs. Upon receiving a filter request from one of these, it sends back a nonce which is generated from the FM's address, the request number, the filters and a secret. When it receives a nonce reply, the BM ensures that the FM has the authority to request filters for the given destination IP addresses (using mappings distributed by the peer-to-peer mechanism) and that it knows about a BP that can filter the given source address. If both of these checks succeed, the BM forwards the filter install request to the relevant BP(s), waits for an ack and forwards it to the requesting FM. To test the performance of the border manager, we constructed dummy versions of the filter manager and the border patrol. With this test framework the real border manager was able to sustain a rate of 87,000 requests/second. Again, this is sufficient to filter even the largest botnets in a matter of seconds.

The last control plane element is the border patrol. The BP simply receives filter installation requests, installs them in the filtering element of the forwarding plane, and sends and acknowledgement back to the requesting BM. Once again, we used a dummy version for this BM which always sends the same request. It is worth noting that for this experiment we set up the BP to use an SMP kernel, so that the control plane process and the process representing the forwarding plane were executed on separate processors. No packets were forwarded for this experiment, since the aim was to test the performance of the control plane part of the border patrol. Installing filters into the filtering element of the Click router consists of writing to a '/proc' entry. To minimize the impact from this operation we installed filters in batches of 100. With this in place, the BP was able to service 354,000 requests/second. Clearly this is more than sufficient to filter any malicious sources sitting behind the BP in very little time.

To sum up, the filter manager, border manager and border patrol are able to handle requests at rates of 75,000, 87,000 and 345,000 requests per second respectively. While we did not particularly optimize the performance of any of these components nor did we take advantage of multiple processors for the control plane, these results clearly show that the control plane of the architecture would be able to filter even the largest botnets in a matter of seconds.

4.3 Forwarding Plane

We implemented the border patrol's forwarding plane using Click and added some custom elements of our own. Before testing our elements, we conducted experiments to establish baseline performance figures for Click. Throughout this discussion, it is worth keeping in mind that we focused on minimum-sized packets, since these put the biggest strain on forwarding. For reference, the theoretical maximum throughput for gigabit Ethernet using minimum-sized packets is 1,488,095 packets per second, or 681 Mbps.

4.3.1 Baseline Performance

Before conducting any filtering tests we had to baseline the system's capabilities to have a better understanding of where the bottlenecks might be. To accomplish this, we ran three sets of experiments aimed at measuring Click's performance when generating, counting and forwarding packets.

In the case of packet generation, the topology consisted of a single host sending packets out of multiple interfaces, each connected to a host also using Click to count the packets. Using minimum-sized packets, we were able to simultaneously saturate two interfaces, for a combined rate of about 1,362 Mbps. With three interfaces we could not quite saturate the links, seeing an aggregate of 1,862 Mbps, equivalent to 91% of the theoretical maximum. Adding a fourth interface resulted only in a minor increase in throughput (1,905 Mbps), suggesting we were CPU-limited. Using an SMP kernel and multi-threaded Click with each generating interface assigned

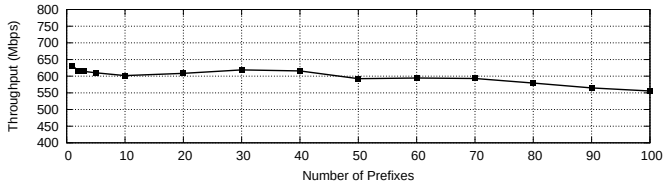


Figure 4: IngressFilter element performance.

to a separate processor confirmed this: using four interfaces we were able to generate at a rate of 2,482 Mbps, or 91% of the maximum.

For the next set of baseline tests we concentrated on packet counting at the traffic sink. The topology consisted of four source hosts, each sending to one of the four interfaces on the counting host. As with sending, using two interfaces Click is able to count at the maximum rate. Adding a third and fourth interface did increase this rate, but switching to an SMP kernel and multi-threaded Click with each interface handled by a separate processor allowed us to count at a rate of 2,724 Mbps, the theoretical maximum.

What about forwarding performance? The border patrol acts more as a filtering device rather than a router, and so it will generally have a single outgoing interface connected to the next-hop router. Taking this into account, and using one, two and three incoming interfaces with a standards-compliant IP router Click configuration we were able to saturate the outgoing link even for minimum-sized packets (681 Mbps). Adding more interfaces does not increase the rate; with two incoming and two outgoing interfaces, the aggregated throughput on the outgoing links was about 680 Mbps, showing a performance bottleneck on the Click router.

As in the previous experiments, we then switched to an SMP kernel and multi-threaded Click to see if we were CPU-limited, and if so, how far we could improve this rate. When using multi-threaded Click and a router configuration, care must be taken in the way that threads (essentially CPU cores) are assigned to portions of the router. Maximizing the utilization of the four CPU cores does not necessarily maximize throughput; to have good cache performance it is important to avoid packets switching CPUs as they travel through the forwarding path. Since, as mentioned earlier, the border patrol acts as a filtering box more than a router, we can force all traffic arriving at one interface to leave on the same outgoing interface. With this in mind, we assign each of the two forwarding paths to separate processors. Testing this configuration and using some of Click's optimization tools resulted in a combined rate of 1,362 Mbps for minimum-sized packets, or 86% of the theoretical maximum. We were able to forward 100-byte packets at line-rate. Clearly this shows that adding CPU cores increases the forwarding performance; adding more interfaces to the router and seeing how these rates continue to increase is future work.

4.3.2 Border Patrol Performance

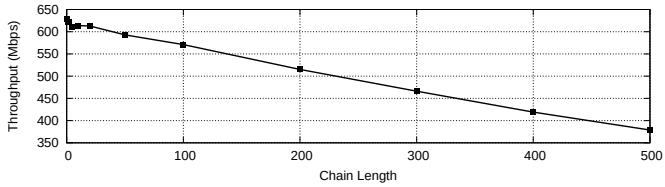
Besides forwarding packets, the BP has essentially two functions: ingress filtering and filtering based on requests from filter managers. To this end, we built two Click elements, *IngressFilter* and *HashFilter*, testing their performance individually at first and then combining them.

The IngressFilter element is quite simple, dropping any packets that do not match any of the allowed prefixes and setting a diffserv code point representing the true-source bit on any packets that do. We implemented the list of prefixes using a vector, and so we wanted to see how the element would perform as this list got longer. For these tests we used a router with a single incoming and single outgoing interface, where the last entry in the list was the prefix allowing the test packets to be forwarded so that each packet forced a full list traversal. As shown in Figure 4, even for a large number of prefixes (50), we were able to forward packets at 87% of the theoretical maximum for minimum-sized packets. The curve of the graph remains relatively flat right until the 70 prefixes mark or so: we hypothesize that before this point memory accesses are being serviced from the L2 cache, and so additional prefixes result in a negligible performance hit; beyond this point, the curve begins to decline, suggesting having to access main memory more frequently.

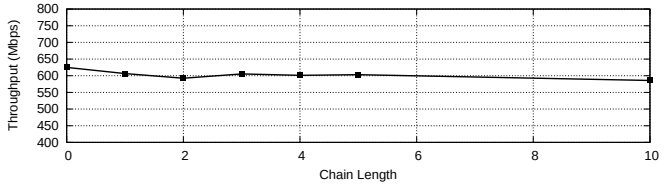
Using two incoming and outgoing interfaces and using an SMP kernel with multi-threaded Click resulted in a combined throughput of 1,017 Mbps, or 75% of the theoretical maximum. Even though this figure could be improved upon using some of Click's optimization tools or by changing the way that tasks are assigned to CPU cores, it shows that the border patrol is still able to process over a gigabit of minimum-sized packets even for a large number of prefixes.

We then tested the performance of the HashFilter element. As the name suggests, the element uses a hash to store the filters, and so its performance is essentially bound to that of the hash. First we decided to see the pathological effect on forwarding of having long chains in the hash. To do so, we tweaked the hash function so that all filters were installed in the same chain, and each packet being forwarded forced a full traversal of this chain (see Figure 5(a)). Even for a very long chain of 100 filters, the border patrol was able to forward at a rate of 571 Mbps for minimum-sized packets, or 84% of the maximum. Adding an incoming and an outgoing interface and using an SMP kernel with multi-threaded Click resulted in a rate 946 Mbps, showing that performance scales well with additional interfaces and processors.

Next we wanted to see the effects that different source/destination IP pairs in the incoming packets would have on performance. For this experiment we used a hash in which all chains were of equal length and each of these contained different filters. We modified the hash function so that each incoming packet hit a different chain to force bad cache locality. Figure 5(b) shows the results. Even for an over-populated hash table containing chains of 10 filters each, the border patrol forwarded packets at 586 Mbps, 86% of the maximum. This shows good performance even in the face of a poor hash function or a heavily-loaded hash. In the SMP case and using four interfaces we obtained 976 Mbps,



(a) Worst-case scenario, filters are in one chain and packets hash to it.



(b) All chains are equal length and each packet hashes to a different one.

Figure 5: HashFilter element performance.

showing once again scalability with regards to number of interfaces and processors.

The final set of forwarding tests focused on testing the border patrol's performance when using the IngressFilter and HashFilter elements at the same time. Using a list of 20 prefixes, 10-filter long chains on all of the hash's buckets and forcing each incoming packet to hit a different bucket yielded a rate of 585 Mbps. Even in such an extreme scenario, the border patrol is able to forward minimum-sized packets at 86% of the theoretical maximum. In the SMP case, we were able to bump this rate up to 922 Mbps.

These results show that even for a large number of prefixes in the IngressFilter element and a heavily loaded hash in the HashFilter element, the border patrol is still able to forward minimum-sized packets at very high rates. Further, we have shown that these figures scale with the number of interfaces and processors, giving evidence about the feasibility of building a well-performing border patrol with off-the-shelf hardware, and demonstrating the ability to utilize tomorrow's many-core CPUs effectively.

4.4 Combining the Two Planes

For the final test, we wanted to determine how filter installation would impact the border patrol's forwarding performance. To do so, we used an SMP kernel but single-threaded Click. Using batches of 100 filters we were able to install filters at a rate of about 354,000 filters/second, while sustaining a forwarding rate of 568 Mbps, 83% of the maximum. This shows that the control plane has very little impact on the forwarding plane because they run on separate cores. Indeed, conducting the same test on a uni-processor kernel causes the forwarding rates to plummet to 78 Mbps (the filter insertion rate remained the same). These figures clearly show that the border patrol can install filters at a very high rate while leaving the forwarding rate largely unaffected.

5 Deployment Incentives

Under full deployment, Terminus would clearly provide significant protection against DoS attacks for all hosts. However, this "common good" argument is not enough on its own to motivate early adopters to embrace our architecture, since entities on the Internet generally act only in their own self-interest. To bring about change, a solution must provide incentives even for those early adopters, or it will never see any important level of deployment.

ISPs hosting potential victims have the clearest incentive, since they can charge for the protection they provide. Alternatively, such an ISP could provide this protection for free, attracting customers from ISPs that do not provide this service. Deploying our solution at the ISP is simple: set up a box to act as the filter manager, configure it to receive the source IP prefix-to-BM mappings from the peer-to-peer network, obtain a certificate from the local RIR to sign its prefix-to-FM mapping, and install a diffserv rule at the edge routers so that packets with their true source bit set receive higher priority. Evidently we would expect ISPs deploying these mechanisms to also deploy border managers and border patrols, but it is not required.

A source ISP has less incentive, since it is not directly affected by attacks. However, deploying Terminus will result in its customers receiving priority in the destination ISP, avoiding delays during normal operation and actually receiving service during an attack. Perhaps more importantly, deployment might reduce technical support costs. Since the filtering mechanism has no false positives (traffic is only filtered if the receiver does not want it), the ISP can rely on the automatic filtering mechanism rather than having to handle this manually, as is currently the case. The actual deployment would entail installing a border manager and a border patrol, setting the BM to receive the mappings between destination address and filter managers, and obtaining a certificate from the local RIR to sign its prefix-to-BM mapping.

What about transit ISPs? They have the weakest incentive, but may be persuaded to deploy by a client ISP (either a source or destination ISP) that has deployed Terminus. The transit ISP's reputation might also motivate it to implement the scheme. Fortunately the changes needed are minimal requiring no additional hardware: just configure border routers to set or unset the true source diffserv code point depending on whether a packet came from a Terminus or legacy ISP.

6 Related Work

The rise in DoS attack activity in recent years has resulted in many proposed solutions from the research community. One type of approach relies on building an overlay of nodes to protect victims [**?**, **?**, **?**]. While they have their merits, these solutions generally operate above the network layer, and so other mechanisms are needed to protect this layer. Other approaches rely on so-called *capabilities* [**?**, **?**, **?**], whereby a host must ask permission to send from the receiver before actually sending any traffic, and include a token in subsequent packets. These solutions tend to rely on the network to police packets so that only those with valid tokens are allowed through, presenting a difficult deployment hurdle. Pushback

[**?**] aims at filtering aggregates by having routers ask upstream ones to filter traffic. However, in order to be effective, it needs paths where every single router has the scheme deployed, and it can only defend against large attacks if these deployed routers are close to the sources of malicious traffic. Traceback solutions [**?**, **?**, **?**, **?**], as their name suggest, focus on determining where packets come from. The true-source bit described in this paper solves this problem with much less mechanism.

One of the more recent solutions [**?**] provides application-layer protection against DoS, and so it would nicely complement the solution we presented. Another proposal called *dFence* [**?**] mitigates DoS attacks at the network layer by dynamically inserting middlebox devices in front of a victim when an attack takes place. However, these boxes are deployed near the victim, and so it is unclear how well they would be able to cope with large attacks. In *PRIMED* [**?**], the authors present a proactive approach to mitigation based on communities of interest (COIs), using them to capture the collective past behavior of remote network entities and to predict future behavior. Despite its merits, attackers will eventually outsmart the heuristics used in the solution. In addition, PRIMED's analysis may be vulnerable to spoofing attacks trying to incriminate members of a good COI.

Like the solution presented in this paper, there have been others put forth that aimed at filtering malicious traffic near its sources. *AITF* [**?**] is one such approach, but it faced deployment hurdles because of changes required to core network nodes: it relies upon the core of the network to perform the filtering during the initial deployment stages, and uses a variant of IP route record to mark where packets came from. In *Firebreak* [**?**], the authors also place filtering boxes near the edges. However, it suffers from several shortcomings, including complications arising from the fact that these boxes use IP anycast to advertise their addresses: not only is large-scale IP anycast not well understood nor widely deployed, but advertising in this manner is likely to present scalability problems. Another solution [**?**] aims at filtering traffic on a per-customer basis, but relies on BGP to relay filtering requests from AS to AS and focuses on the economic feasibility of its deployment.

In [**?**] and in previous work [**?**] a diffserv code point was used to differentiate traffic originating from well-managed sites and from sites that supported the proposed filtering scheme, respectively. The solution we presented in this paper also uses a code point, but to signal the validity of the source IP address field in the packet.

7 Conclusions

We presented Terminus, a new architecture that can filter even the largest Denial-of-Service attacks while providing clear incentives to early adopters. Terminus is well suited to being implemented in fast routing hardware, but in the early stages of deployment this is unlikely to be the case. Because of this, we have shown that these elements can also be built using inexpensive off-the-shelf hardware while still being able to cope with the largest attacks, both in terms of filter installation rates and forwarding rates even for minimum-sized packets. Optimizing the current implementation and taking full advantage of all the processors in the hardware used would improve these figures even further.

Despite the title of this paper, Terminus is actually named after the Roman god of boundary markers.

References

[1] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with Capabilities. In *Proc. ACM SIGCOMM 2nd Workshop on Hot Topics in Networks*, November 2003.

[2] K. Argyraki and D. Cheriton. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *Usenix Annual Techical Conference*. Usenix, April 2005.

[3] S. Bellovin, M. Leech, and T. Taylor. The ICMP traceback message, Oct 2001.

[4] CNET. Bot herders may have controlled 1.5 million pcs, October 2005.

[5] S. Crocker. Protecting the internet from distributed denial-of-service attacks: a proposal. *Proceedings of the IEEE*, 92(9), 2004.

[6] Paul Francis. Firebreak: An IP Perimeter Defense Architecture. 2004.

[7] R. Govindan, A. Hussain, R. Lindell, J. Mehringer, and C. Papadopoulos. COSSACK: Coordinated suppression of simultaneous attacks. In *Proceedings of IEEE DISCEX Conference*, April 2003.

[8] Mark Handley and Adam Greenhalgh. The case for pushing DNS. In *Proc. ACM HotNets IV*, November 2005.

[9] Felipe Huici and Mark Handley. An edge-to-edge filtering architecture against DoS. *SIGCOMM Comput. Commun. Rev.*, 37(2):39–50, 2007.

[10] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proc. Network and Distributed System Security Symposium, San Diego*. ISOC, Reston, VA., February 2002.

[11] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of ACM SIGCOMM*, August 2002.

[12] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek. The Click modular router. *ACM Trans. on Computer Systems*, 18(3):263–297, August 2000.

[13] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. Taming IP packet flooding attacks. In *Proceedings of the 2005 HotNets Workshop*, 2003.

[14] W. Lee and J. Xu. Sustaining availability of web services under distributed denial of service attacks. *IEEE Transactions on Computers*, 52(3):195–208, 2003.

[15] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang. dFence: Transparent Network-based Deniall of Service Mitigation. In *4th USENIX NSDI*, Cambridge, MA, Apr 2007.

[16] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. *SIGCOMM Comput. Commun. Rev.*, 30(4):295–306, 2000.

[17] D. Simon, S. Agarwal, and D. Maltz. AS-Based Accountability as a Cost-effective DDoS Defense. In *USENIX First Hotbots Workshop*, Cambridge, MA, April 2007.

[18] A. Snoeren, C. Partridge, A. Sanchez, Jones C., F. Tchakountio, S. Kent, and T. Strayer. Hash-based IP traceback. In *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001.

[19] P. Verkaik, O. Spatscheck, J. Van der Merwe, and A. Snoeren. PRIMED: Community-of-Interest-Based DDoS Mitigation. In *ACM SIGCOMM LSAD Workshop*, Pisa, Italy, September 2006.

[20] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically, 2004.

[21] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. DDoS Defense by Offense. In *ACM SIGCOMM 2006*, Pisa, Italy, Sept 2006.

[22] Wikiquote. Voltaire. http://en.wikiquote.org/wiki/Voltaire.

[23] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against DDoS attacks. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 93. IEEE Computer Society, 2003.

[24] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *Proceedings of the IEEE Security and Privacy Symposium*, May 2004.