gore: Routing-Assisted Defense Against DDoS Attacks

Stephen T. Chou[1], Angelos Stavrou[1], John Ioannidis[2], and Angelos D. Keromytis[1]

[1] Department of Computer Science, Columbia University, New York, NY
[2] Center for Computational Learning Systems, Columbia University, New York, NY
{schou,angel,ji,angelos}@cs.columbia.edu

Abstract. We present *gore*, a routing-assisted defense architecture against distributed denial of service (DDoS) attacks that provides guaranteed levels of access to a network under attack. Our approach uses routing to redirect all traffic destined to a customer under attack to strategically-located *gore* proxies, where servers filter out attack traffic and forward authorized traffic toward its intended destination.
Our architecture can be deployed incrementally by individual ISPs, does not require any collaboration between ISPs, and requires no modifications to either server- or client- software. Clients can be authorized through a web interface that screens legitimate users from outsiders or automated zombies. Authenticated clients are granted limited-time access to the network under attack. The *gore* architecture allows ISPs to offer DDoS defenses as a value-added service, providing necessary incentives for the deployment of such defenses. We constructed a PC-based testbed to evaluate the performance and scalability of *gore*. Our preliminary results show that *gore* is a viable approach, as its impact on the filtered traffic is minimal, in terms of both end-to-end latency and effective throughput. Furthermore, *gore* can easily be scaled up as needed to support larger numbers of clients and customers using inexpensive commodity PCs.

1 Introduction

Denial-of-Service (DoS) attacks can take many forms, depending on the resource the attacker is trying to exhaust. For example, an attacker may cause a web server to perform excessive computation, or exhaust all available bandwidth to and from that server. In all forms, the attacker's goal is to deny use of the service to other users. Apart from the annoyance factor, such an attack can prove particularly damaging for time- or life-critical services, or when the attack persists over several days: in one instance of a persistent DoS attack, a British ISP was forced out of business because it could not provide service to its customers. Of particular interest are *link congestion* attacks, whereby attackers identify "pinch points" in the communications infrastructure and render them inoperable by flooding them with large volumes of traffic. We concentrate our interests on this form of attacks because there is little, if anything, the victim can do to protect itself; what is being attacked is not any particular vulnerability of the target, but rather the very fact that said target is connected to the network.

There are many reasons why, despite extensive research work on the subject, we have seen very little deployment of effective anti-DDoS technology by Internet Service Providers. An important one is the lack of financial incentives for ISPs to deploy such

services: they cannot easily sell a premium service to high-value customers whereby these customers are better protected. However, it is precisely these high-volume, high-value customers who often attract the more serious DDoS attacks, and whom the ISP would want to keep better protected, either by charging more, or by considering the expense of the extra protection as the cost of attracting these high-value customers (or even protecting their own network from the attacks these customers would attract).

Many previous approaches that address the general network DoS problem ([1–3]) are reactive: they monitor traffic at a target location, waiting for an attack to occur. Once the attack is identified, typically via analysis of traffic patterns and packet headers, filters may be established in an attempt to block the offenders. The two main problems with this approach are the accuracy with which legitimate traffic can be distinguished from the DoS traffic, and the robustness of the mechanism for establishing filters deep enough in the network so that the effects of the attack are minimized. Approaches such as WebSOS [4, 5] protect particular kinds of services (web traffic in this case) by introducing additional processing elements into the network infrastructure and introducing ways of identifying legitimate, human-originated web sessions and only processing those in times of heavy attack.

We introduce *gore*, an architecture that individual ISPs can use to protect customers under attack. Some prior architectures assume that ISPs collaborate in order to quench DDoS attacks. This appears to be an unrealistic approach, since the security and policy problems that crop up far outweigh the putative benefits of quenching attacks in that way. In our approach, when an attack against a particular customer is detected, all traffic to that customer's IP address prefix is redirected to strategically-located *gore* proxies inside the ISP's network. This redirection is accomplished by properly advertising the customer's prefix from the appropriate *gore* proxy over the ISP's Intradomain Routing Protocol (OSPF, IS-IS, *etc.*).

Such a proxy is not necessarily a single computer; it can be a cluster, and there can be many such clusters throughout the ISP's network, subject to cost constraints. However, it is possible to take advantage of a form of statistical multiplexing: since only a very small fraction of an ISP's customers are typically attacked at any particular time, the ISP need only provide proxies and capacity to handle this smaller set of attacks.

gore proxies use some method for differentiating real traffic from attack traffic. The specific approach we use involves *Graphical Turing Tests* (GTTs) [6] if no prior agreements between the customer and its potential clients exist; authentication based on customer-provided credentials to the users may be used instead, or in addition to GTTs. Traffic that is characterized as legitimate is tunneled to the customer's access router(s) over a GRE [7] tunnel; all other traffic is dropped. Return traffic from the customer to its clients is simply routed back to the client without passing through *gore*.

As *gore* centers are not normally addressable from outside the ISP (and, presumably, a well-managed ISP can detect and quench portions of an attack that originate within its own network), they cannot be independently attacked. The only times that traffic from outside the ISP reaches the *gore* proxies is when a customer is under attack. Naturally, the proxies are located where there is a lot of link capacity, and must be provisioned to handle at least as much raw traffic as the customer's access link.

The contributions of our work are threefold. First, we present a novel architecture, *gore*, that significantly extends and improves best current practices currently used by ISPs (blackholing, as discussed in Section 4) to maintain connectivity in the face of large DDoS attacks. Second, contrary to other proposed work that does not allow ISPs to recoup the costs associated with installing, enabling, and managing DDoS defenses, *gore* can naturally be offered as a value-added service to customers. Third, we characterize the impact on end-to-end latency and throughput that *gore* imposes on communication flows that traverse it, which we determine to be less than 2% in either case for experiments involving up to 2,000 clients. It is important to note that these overheads are only incurred when an attack is taking place; otherwise, *gore* does not have any impact on network traffic. Furthermore, communications would be otherwise halted when a DDoS attack occurs. Thus, we believe *gore* offers a particularly attractive mechanism for ISPs to counter the increasing threat of denial of service attacks.

The remainder of this paper is organized as follows: Section 2 describes the *gore* architecture in detail. Section 3 gives the details of an actual implementation of the architecture, along with performance results over a simple testbed. We conclude with related work in Section 4 and a summary directions for future work in Section 5.

2 Architecture

We propose an architecture that provides a scalable router- (and routing-) assisted mechanism to protect ISP customers from DDoS attacks. The architecture is transparent, in the sense that no additional software needs to be deployed on either the customer web servers or web clients. Our DDoS defense is *reactive* and is enabled only when customers are under attack, and then only for *those* customers. Our scheme does not affect any transit traffic through the ISP, nor does it affect the way the ISP advertises its customers' prefixes over BGP. Since the mechanism works entirely within an ISP's network, it allows the ISP to retain full control of its defense policies, for example, turning them on only for specific customers, *e.g.,* those who have subscribed to a hypothetical "DDoS Protection" plan.

Central to our architecture is a *gore center*, in which two pieces of functionality are present: a *firewall/forwarder*, and a *proxy*. We shall limit this discussion to showing how to protect web traffic, although nothing precludes generalizing our techniques to other kinds of identifiable traffic. We also assume that the ISP has the ability to detect a DDoS attack and report it to some management agent. Such ability is common, but it can even be as crude as the customer noticing the attack and calling up the ISP's Network Operations Center. Once the attack is detected, it is communicated by the NOC (or some automatic mechanism) to one or more *gore centers*.

Figure 1 illustrates a customer network under DDoS attack. Attack traffic converges from all over the Internet, overwhelms the customer network's access links, and legitimate clients are not able to communicate with the (web) servers in the network under attack. Furthermore, if the attack is severe enough, the links from the ISP's backbone to the access router where the customer connects may get congested, or the access router itself may be overloaded, causing other customers who are not themselves under attack

to suffer. For this reason, it is common when one customer is under attack to *blackhole*[3] that customer's IP prefix at the ISP's border routers so that attack traffic gets dropped before it enters the ISP's network. While this practice protects the innocent bystanders, it also means that the customer is not getting *any* connectivity to the Internet while the attack lasts, rendering the attack even more effective.

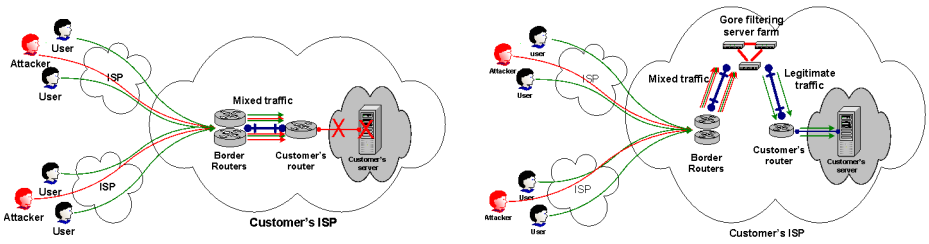

Fig. 1. (Left) DDoS attacks on an ISP customer's network: the attackers can render customer's the low bandwith connection and its servers unusable. (Right) DDoS Attacks when *gore* gets activated: customer's traffic is redirected and filtered through the *gore* servers.

Instead of indiscriminately blackholing all traffic to the customer, we want to instead *"whitehole"* traffic we know to be good. As soon as an attack on a prefix is reported, a *gore* center with farm of dedicated *gore* servers start handling all traffic to that prefix. *gore* centers participate in the ISP's interior routing protocol (for example, OSPF), and when they decide to "take over" a prefix, they advertise the two more-specific-by-one-bit prefixes over the routing protocol. For example, if the customer's prefix is *135.207.0.0/16*, the *gore* centers will advertise *135.207.0.0/17* and *135.207.128.0/17*. Because routers forward based on longest-match, the *gore* center will receive traffic for *135.207.0.0/16*, regardless of how close or far to the access router such traffic enters the ISP's network. In this case, the access router must be configured to filter out such more-specifics for a prefix it knows it handles[4]. Furthermore, peering routers are configured to not announce these more-specifics over BGP, as there is no change in the way outside traffic should reach the ISP[5].

The *gore* center does not use addresses that are routable outside the ISP, and thus cannot be directly targeted. The reason is that, although the center has enough capacity to handle a worst-case scenario attack, individual servers (if they can be identified and targeted as such) can be overwhelmed; thus, an attacker that could somehow determine

[3] In a nutshell, *blackholing* means that border routers are told to drop all traffic destined to the blackholed prefix rather than forwarding it to the next-hop router. This is typically accomplished by including a routing entry for the blackholed prefix pointing to the *null* interface.

[4] We ignore the limit case of traffic entering the ISP's network from the same access router that the customer under attack is connected to. Access routers are almost never peering routers. Traffic from another customer, even if it is attack traffic, is probably negligible.

[5] This practice may lead to suboptimal paths to be taken inside the ISP, but we consider this a second-order effect; how it should be handled is beyond the scope of this paper.

that a particular *gore* server happened to carry legitimate users' traffic, would be able to direct an attack against that server and disrupt client-customer traffic.

To balance the load among *gore* servers, the *gore* center dynamically assigns each server a specific range of source addresses of outside traffic. Since the origin of attack traffic spread evenly in the IP address space, the dynamic assignment prevents any individual server from overwhelmed by the attack traffic for an extended period of time.

Most traffic entering the *gore* center at the firewall/forwarder will get dropped. The first exception is connection attempts to TCP ports 80 and 443 (web traffic). These connections are passed on to a *gore* proxy, much like the proxy in WebSOS [5], whose purpose is to differentiate between human users and automated processes (such as DDoS zombies), or to identify legitimate users that are provisioned with authentication material (*e.g.,* a username/password or a public key certificate) by the customer. The human/process separation is carried out by using a test that is easy for human users to answer, but would be difficult for a computer. For a brief description of these tests, see Section 2.1. If necessary, *gore* can ask additional questions to validate the client's identity and authorization before granting a transit through the *gore* center.

Once the client has passed the test, the proxy installs a firewall rule on the firewall/forwarder that allows all traffic from the source IP address of the client that passed the authentication to reach the customer's servers. In order for that to happen, the *gore* firewall/forwarder maintains a Generic Routing Encapsulation (GRE) [7] tunnel, typically created in advance with the access routers, over which it forwards all traffic from the authenticated clients. The tunnel creates a transparent virtual link between a *gore* firewall/forwarder and an access router such that traffic routed through the tunnel will be unaffected by route redirections. These firewall rules are set to expire after either a fixed amount of time, or after a period of inactivity. Note that the firewall/forwarder only sees traffic from the client to the server; return traffic is independently routed and never goes through the *gore*, as shown in Figure 2. In essence, we have what is usually referred to as triangular routing: when the defense mechanism is enabled, traffic to customer servers is first routed to *gore* centers; authorized traffic is then passed on to its intended destination; return traffic travels along the path that it would be travelling before the attack.

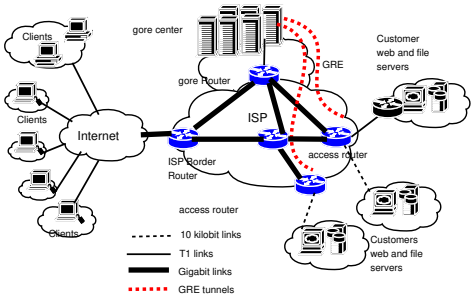

Fig. 2. Details of *gore* architecture.

The *gore* router and the various customer routers need not be directly connected to each other; since authorized traffic from the *gore* router to the customer router is tunneled, they can be anywhere in the ISP's network. Also, an ISP with multiple *gore* servers and with multiple customer networks is possible, and in fact should be common. Ingress traffic destined to customer under attack will simply be routed to the near *gore* center from an ISP border router. In this configuration, the ISP will need to set up tunnels between every *gore* server and every customer access router. Although such tunnels can also be constructed as needed, the resources needed for "dormant" tunnels are so limited that it may be simpler to establish them in advance.

One limitation of our approach is that attack traffic is carried over the ISP's network to the *gore* center. Thus, it is conceivable that legitimate users' traffic that happens to use some of the same links will experience degraded performance, if the attack volume is high enough. However, the vast majority of attacks we have seen to date do not cause problems in the major ISPs' backbone networks. Thus, we believe that the impact on legitimate traffic of routing attack traffic to the *gore* center would be relatively small.

2.1 Client Legitimacy Tests

In order to prevent automated attacks from going past the *gore* center, we need a mechanism with which to differentiate between legitimate users and (potential) attacks. One obvious way of doing this is via authentication (*e.g.,* client-side certificates). The *gore* center would use RADIUS [8] or a similar protocol to connect to the customer's authentication server and verify the validity of the client's authentication credentials. This traffic would be carried over the GRE tunnel, and thus would not be subject to the routing-based redirection.

In many cases, however, customers may not have a well-defined client base (*i.e.,* one that can be identified through traditional network-based authentication), or may simply want to provide service to all users. Fortunately, there exist mechanisms to differentiate between human users and unsupervised programs, which under a DDoS attack can be presumed to be zombies. Although this would prevent legitimate automated processes (*e.g.,* a web-indexing "spider") from accessing the customer's network, this may be a price that the customer is willing to pay, when a DDoS attack is in progress. If these automated processes are known *a priori*, then it is possible to supply them with cryptographic credentials that allow them to bypass any human-legitimacy tests (see previous paragraph).

In our system, we decided to use Graphic Turing Tests (GTTs) to identify traffic that is under direct human supervision. A CAPTCHA [6] visual test is implemented when a web connection is attempted in order to verify the presence of a human user. CAPTCHA (Completely Automated Public Turing test to Tell Computers and Humans Apart) is a program that can generate and grade tests that most humans can pass, but automated programs cannot. The particular CAPTCHA implementation we use is GIMPY, which concatenates an arbitrary sequence of letters to form a word and renders a distorted image of the word. GIMPY relies on the fact that humans can read the words within the distorted image and current automated tools cannot. Humans authenticate themselves by entering as ASCII text the same sequence of letters as what appears in the

image. Updating the GIMPY interface can be performed without modifying the other architectural components.

Although recent advances in visual pattern recognition [9] can defeat GIMPY, there is no solution to date that can recognize complicated images or relation between images like Animal-PIX. Although for demonstration purposes in our prototype we use GIMPY, we can easily substitute it with any other instance of Graphical Turing Test.

2.2 *gore* Center Details

As we have already explained, a *gore center* consist of a *gore* router and one or more *gore* servers. The purpose of the router is to participate in the OSPF process of the ISP and announce the customer prefix(es) to protect when called to do so, and also to distribute arriving traffic to the *gore* servers as evenly as possible. The *gore* server, in turn, consists of a firewall/forwarder and a proxy. The firewall/forwarder accepts incoming traffic sent to it by the *gore* router; if it is from a previously unseen source, it passes it on to the proxy so it can be authenticated. Otherwise, it is either attack traffic, in which case it is blocked, or it is good traffic, in which case it is tunneled to the appropriate customer's access router. These two functions could be implemented on different boxes, but since each modifies the other's behavior, we prefer to implement them on the same box, namely a commodity x86 PC. While a high-end router can filter and forward packets more efficiently than a commodity PC, the latter are much cheaper. Also, unlike typical firewall operations, the rules in a *gore* firewall/forwarder need not be traversed in a linear manner — a hash table or a trie, or even a simple bitmap, can be used instead for much faster matching. Also, the only functions that the firewall/forwarder performs are inspecting the protocol field, source and destination IP addresses, and the destination TCP port; there is no stateful packet inspection, or per-connection state to maintain (which would be impossible to do anyway since the firewall never sees the return traffic).

gore servers run two sets of packet filtering rules. The first set has network address translation (NAT) rules that redirect web traffic to the proxy function, which administers the GTT. The second set contains rules to forward traffic from authorized sources to the corresponding customer's network. At initialization, the NAT rules redirect all arriving web traffic to the *gore* proxy; forwarding rules deny any transit through a *gore* server. A client needs to pass a challenge before it is granted access to the customer network. Once a source has passed the GIMPY challenge, the *gore* server disables NAT redirection and enables the forwarding for all traffic with the specific source address. This enables web traffic, as well as other traffic from that source, to reach the customer's network through a *gore* center without further redirection. Traffic from unauthorized sources will be dropped by *gore* servers. This approach is similar in nature to what most commercial pay-per-use 802.11 networks and hotel room networks do: when the user first attempts to connect to anything, the request is redirected to a local authenticating web proxy; once a credit card number or other authentication mechanism is entered, the user's IP address (or, in some cases, the MAC address) is allowed to connect to the Internet.

To reduce the possibility of unauthorized exploits of known authorized hosts by spoofers, *gore* servers limit the duration of access to customer network from any autho-

rized source. This is achieved by running a periodic process to purge the installed NAT and forwarding rules for each timed-out client. Clients that wish to continue access can seek a re-authorization by repeating the authentication procedure. Even if the attacker can monitor communication between the customer server and authorized clients by sniffing network traffic, time-limited access can curtail the duration of an attack.

Given the limited number of authorized sources admitted by *gore*, the attacker's chances of making a good pick are slim. Time-limited authorization will reduce the probability of randomly succeeding to attack (by guessing an authorized source address) even futher. It is conceivable that an attacker could first connect as a legitimate client, then communicate his source IP address to his zombies, who would then all spoof their source IP address to be the authorized one. As more ISPs are finally obeying RFC-2267 (making sure that their customers only send packets from IP addresses they own), this may not turn out to be a big concern. If this indeed is a concern, stronger authentication methods than just checking the source IP address may be used, *e.g.,* establishing IPsec tunnels between the clients and the *gore* nodes. Furthermore, since traffic is naturally aggregated at the *gore* center, it is fairly easy to rate-limit all traffic flows that traverse *gore* toward a customer. Thus, attackers that have guessed or acquired an authorized address can do limited damage.

However, a single computer, no matter how powerful, cannot handle all attack traffic. Fortunately, the *gore* architecture scales in two ways: multiple *gore* centers can be deployed around an ISP's network, and each *gore* center can employ many individual computers to perform the firewall/forwarder function and the authentication fuction. No state-sharing is necessary between *gore* centers. An issue that arises when multiple *gore* centers are used is that traffic from a particular source is not *guaranteed* to always follow the same path through an ISP, and thus may not always go through the same *gore* center. There would be two reasons why this may happen; either because traffic from a particular source enters the ISP through more than one border router, or different paths are followed inside the ISP itself. The latter is not a concern; paths change only when links change state, or when traffic-engineering decisions change link weights. Neither is a frequent event, and is something that is easily tolerated. The former could be a concern if it were a persistent situation, but packets that are part of the same short-lived flow almost always take the same path. If a major BGP instability causes this path to change, the user may need to re-authenticate, but this is an acceptable price to pay in order to provide service during DDoS attacks. In either case, this is only a problem during an attack, and we assume that most clients will not be affected by such problems.

To fully utilize multiple packet filtering servers, we need a router (or switch) that can fairly evenly distribute the traffic among them. Since we have no way of finding out attackers in advance, we assume that the attackers are evenly spread among the IPv4 address space. Each *gore* is responsible for the defense against attacks originating from its allotment. The access router in front of a cluster of *gore* machines is responsible for this load-balancing; the details on how to achieve it are router-architecture-specific, but are efficiently implemented in most modern routers. Various methods of farming out traffic to individual forwarders or proxies can be used, but the details are not of particular importance to the system architecture.

3 Experimental Evaluation

Our goal is to evaluate the effectiveness and scalability of the *gore* architecture. In particular, we want to know the highest attack intensity we can defend against using the *gore* architecture when implemented on inexpensive commodity hardware (*e.g.,* x86 boxes running a Unix clone). This will allow us to directly calculate the deployment and management costs necessary to defend against a DoS attack of specific size and intensity. Additionally, we would like to estimate our system's service capacity in terms of legitimate client requests when under attack. Most of all, we want to identify possible resource bottlenecks, if any, that limit the scalability of our system. Answers to these questions are crucial for judiciously deploying defenses against DDoS attacks.

3.1 Testbed

To evalute the overall system architecture, we assembled a testbed that resembles a simplified ISP using *gore* system for a single customer as shown in Figure 3. The ISP has a border router connected to the "Internet" where clients reside. This border router is also connected to a customer access router, serving a customer network that, for simplicity, contains only a web/file server. Furthermore, the border router is connected to a protected network where a *gore* center consisting of one or more units resides. When the NOC detects an attack on the customer's network, traffic from the border router to the customer is redirected to the *gore* network. There, the *gore* farm admits authorized traffic and rejects the rest.

Initially, we used a single server configuration to test limitation of our system. To investigate scalability of our architecture, we proceeded with a testbed of multiple *gore* servers. Each *gore* server handles its own range of source IP addresses. when an attack is initially, the traffic is evenly distributed to all *gore* servers using the load-balancing aspect of the *gore* router. Thus, traffic destined to the customer's network will be appropriately filtered and forwarded by the *gore* servers. This works efficiently when we employ load-balancing based on the source ip address and per-flow, not per-packet. [6] For Linux, this is the default definition of a flow whereas in commercial routers is a configurable parameter.

We conducted experiments in both single server and multi-server testbed configurations. The focus of the single-server experiments was to measure the performance and to identify possible bottlenecks. Then, we investigated the load-balancing on the multi-server testbed and how the capacity of our system scales as we vary both the number of legitimate clients and the attack intensity.

For the *gore* server farm, we used Dell 750 servers with 2.4GHz Pentium4 processors and 512MB of memory running Debian Linux with the 2.4 kernel. These machines were equipped with 1 Gbps Ethernet interfaces and interconnected with a gigabit switch. Both attack and legitimate traffic were generated by machines residing outside our testbed, connected to a border router. We used two different metrics to measure the impact of the attacking traffic to a legitimate client: throughput and end-to-end latency.

[6] A flow in this case is defined as all packets with the same protocol, source and destination IP addresses. In some routers the definition of a flow includes the TCP or UDP port numbers.

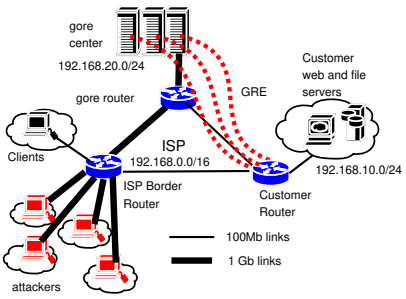

Fig. 3. *gore* experimental testbed activated for a single customer.

These two metrics capture the characteristics of a link for both interactive and time critical applications. They also quantify the effective capacity of the link when under attack.

The internal ISP network used OSPF to maintain its routing paths among the three routers: the border router, the customer's access router and the *gore* router. All routers were configured as a single OSPF area. While the routing mechanism would also work with other interior routing protocols, the use of a link-state protocol helps reduce convergence time when the routing information changes. For the customer's access router, we used a PC-based router running *Zebra 0.94* with *ospfd*. In addition, we used the *iproute* Linux kernel package and corresponding utilities to create a GRE [7] tunnel between the *gore* machine and the customer's router. Each of the *gore* servers has two role: it acts both as firewall/forwarder but also as a web server authenticating users for a limited time. The Linux kernel's *netfilter* facility is used for packet filtering, Network Address Translation (necessary to communicate with the proxy) and other packet processing. *Iptables* provides the user-level utility to install and remove firewall rules from netfilter. By default, all web traffic passing through a *gore* server is directed to its own web server for the graphical turing test. All other traffic is considered malicious and is discarded.

Deploying *gore* farms in different network locations inside the same ISP requires a mechanism to redirect traffic destined for the customer's network not to one but to many *gore* routers. This is handled easily by having the *gore* routers advertise the same most specific address prefix and letting the routing protocol decide where to redirect the traffic based on shortest path routing.

In the multiserver configuration, each *gore* server maintains its own set distinct set of admissible clients. In an idea scenario, each server gets an equal share of incoming traffic. Since the origins of incoming traffic change from time to time, static address block assignments are unlikely to divide properly incoming traffic among available servers. This calls for a dynamic address assignment scheme to reduce load imbalances between the servers. This calls for frequent reconfiguration of the *gore* router, which can be achieved by simply loading a new configuration file.

We can thus assume each *gore* servers gets an equal share of incoming traffic and equal shares of burden of legitimate clients. The file containing the firewall rules is kept on an a shared file system to keep the *gore* servers in sync. Each server polls the

rules file periodically to get the latest set of filtering rules and apply only rules assigned to it. To prevent simultaneous modification of the rules file, a server has to acquire and release a lock file before and after making changes to the rules file. This approach works reasonable well with a small number of *gore* servers. We did not observe any problems related to lock contention, but keeping synchronized copies of data across a network is a solved problem, and we did not worry about this part of the implementation too much.

Since different *gore* servers process different sets of legitimate clients, *gore* server must be deterministically associated with incoming traffic from a specific source address. To achieve this, we use a Cisco router with policy-based routing (PBR) on source address prefixes to forward incoming traffic to *gore* servers. Using fast-switch PBR, the Cisco router can forward at line rate.

3.2 Experimental Results

Our first goal after deploying our testbed was to quantify our system's capacity and performace under normal (non-attack) conditions. To that end, we measured latency and throughput from legitimate clients outside the ISP network, to a server running inside the customer's network. We used *Iperf* to measure the capacity of the line, *i.e.,* the maximum TCP throughput between a client and the server. Furthermore, we computed the round-trip delay using a combination of *traceroute* and *ping*. The term "round-trip" is somewhat misleading, because traffic originated from the client is routed through *gore* when redirection is turned on, while the reply traffic uses a direct path. As we expected, there was no measurable impact on the tcp throughput observed between the direct connection and when we enabled *gore*. Moreover, we measured a minimal increase of 0.2ms in latency due to the addition of GRE tunnel. The effects of non-optimal routing were below our measurement threshold.

Next, we measured the performance of our system under attack, with multiple clients trying to access the customer's server. Figure 4 shows the measured throughput and round-trip latency as we increased the number of firewall rules. The change in throughput and latency between non-redirected and redirected traffic is mostly attributable to the overhead of delivering packets through the GRE tunnel. Each admitted legitimate client adds a NAT "prerouting" rule and a "forward" rule to *netfilter*. This implies two additional rules are evaluated per packet arrival for each admitted client. To ensure that we are measuring worst-case performance, the source address of the *legitimate* traffic is added at end of iptables chain to ensure traversal of the entire set of packet filter rules. Even when two chains of over 2,000 rules were each added to the system, *gore* was able to maintain a throughput almost identical to effective line capacity. The drop in TCP throughput on a *gore* with 10,000 rules indicated a CPU overload on the *gore* server. This overload was due to fact that netfilter stores the rules in a linked list requiring linear time to search for a matching firewall rule. As a consequence, when we increase the size of the firewall rules we also increase the amount of time required to process each packet. Given the size of the filtering rules, we can compute the maximum threashold of packets a machine can process per second. A hash- or trie-based implementation would have an almost constant access time regardless of the number of sources, and should be used in a production system, as demonstrated by Hartmeier

[10]. Thus, our results should be viewed as a lower bound. Although we cannot measure latency directly, we could infer from the round-trip measurements that it increased linearly.

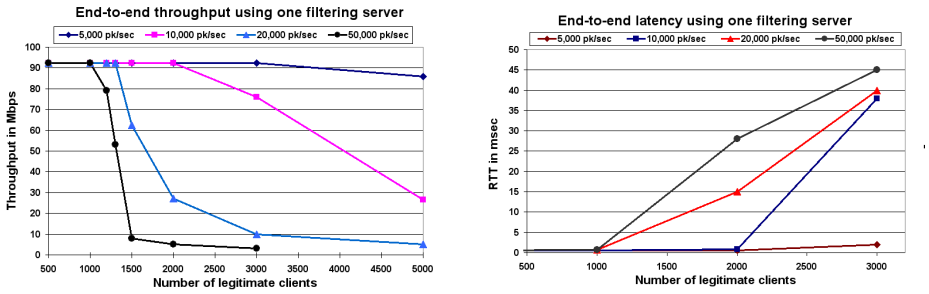

Fig. 4. (Left) Throughput of legitimate traffic with an average DDoS attack packet size of 1024 bytes *vs.* the number of legitimate clients. (Right) Round-trip time of traffic with an average DDoS attack packet size of 1024 bytes for different numbers of clients.

Next, we measured the throughput and latency to the server when the customer is under DDoS attack. We use the traffic generator *tg*[7] from ISI to create attack traffic. We measured performance by varying the arrival rate of the attack traffic. We set up *tg* to generate CBR traffic at different rates.

Figures 4 shows the measured throughput and latency for legitimate client traffic of a DDoS attack. In this case, the attacker uses an average packet size of 1024. The figures show a scenario where the performance is mostly CPU-bound instead of network-bound. An ISP's internal links have enough capacity to carry a large amount of both attack and legitmate traffic. DDoS traffic pushes the legitimate client traffic aside and introduces a precipitous drop in throughput. The legitimate client traffic with 1,500 clients is on the verge of overload when the DoS traffic arrives at a rate of 50,000 packets per second (pps). At lower packet rate, a *gore* server can service many more clients before it gets overloaded. Of course, without activating the *gore* system the attack traffic would have congested the customer's network completely. With *gore* only the filtered traffic is allowed to pass through and thus only authorized clients are allowed to exchange data with the customer's network.

For our multiserver study, we focused on the scalability of the system. We generated legitmate traffic along with DDoS traffic from multiple sources. Since the *gore* server selection is based on the source address, we assigned source addresses of attackers such that the attack traffic spread evenly among *gore* servers. The legitmate traffic gets the remaining capacity through one of the *gore* server.

Figure 5 left shows throughput of legitimate traffic under an attack rate of 50,000 pps. We ran the experiments with 1, 2, 4, and 8 *gore* servers. Assuming the legitmate client traffic is is a small percentage of overall incoming traffic, the legitimate client's

[7] `www.ip-measurement.org/tools/trag.html`

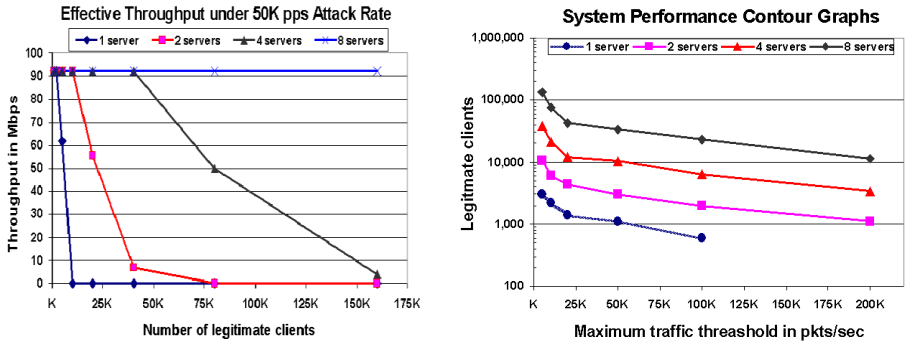

Fig. 5. (Left) Throughput of legitimate traffic under DDoS arrival rate of 50,000 packets per second. (Right) System performance contour graphs: we measured the maximum traffic threashold for different number of legitimate clients as we increase the amount of servers in a *gore* farm.

bandwidth with two servers is roughly equivalent to the bandwidth of a single server at half of the attack rate, yet twice as many clients are protected. Doubling the number of servers protected roughly four times as many legitimate clients for the same traffic rate. We repeated these experiments under different attack rates and observed similar scaling factors.

Figures 5 right shows number of legitimate clients that can be supported for a given attack traffic rate and number of servers. The experiement demonstrated the scalability of our *gore* solution to multiple servers. As long as internal link capacity of the ISP is large enough to handle attack traffic, the ISP can always add more *gore* servers and centers throughout its network to handle DDoS attacks.

The experiments with DDoS traffic demonstrate that performance is mostly CPU-bound until the network becomes saturated. To determine the number of *gore* centers to deploy in the system, we need to know the expected arrival rate of attack as well as desirable number of legitimate clients. These can be provisioned in advance, using measurements done by either the customer or the ISP under normal load conditions.

4 Related Work

The need to protect against or mitigate the effects of DoS attacks has been recognized by both the commercial and research world. Some work has been done toward achieving these goals, *e.g.,* [1, 11, 3, 12, 2, 13]. These mechanisms focus on detecting the source of DoS attacks in progress and then countering them, typically by "pushing" some filtering rules on routers as far away from the target of the attack (and close to the sources) as possible. The motivation behind such approaches has been twofold: first, it is conceptually simple to introduce a protocol that will be used by a relatively small subset of the nodes on the Internet (*i.e.,* ISP routers), as opposed to requiring the introduction of new protocols that must be deployed and used by end-systems. Second, these mechanisms are fairly transparent to protocols, applications, and legitimate users. Unfortunately, these approaches by themselves are not always adequate.

The approach most similar to ours is a commercial offering by Riverhead [14]. It tries to characterize and detect bad traffic, and "scrub" it before forwarding the clean traffic to the customer. Also, it employs MPLS rather than a combination of OSPF and GRE to redirect traffic,

Blackhole filtering is a popular technique against DDoS attacks and is employed by many ISPs. The scheme sets up a redirection to a pseudo-interface *null0* by advertising routes for hosts or networks under attack. The techique avoids the use of packet filtering through access lists, which could impact the performance of router. The scheme requires deployment of a network intrusion detection system to activate the routing change. The main concern with the approach is that it effectively disconnects the network it is trying to protect from the rest of the Internet, essentially achieving what the DDoS attackers try to achieve in the first place. In addition, the scheme does not support filtering of packets at layer 4 or above.

The NetBouncer project [15] considers the use of client-legitimacy tests for filtering attack traffic. Such tests include packet-validity tests (*e.g.,* source address validation), flow-behavior analysis, and application-specific tests, including Graphic Turing Tests. However, since their solution is end-point based, it is susceptible to large link-congestion attacks.

The SOS architecture [16, 17] combines the notions of a distributed firewall [18] inside the network, overlay routing, and aggressive packet filtering near the target of the attack to only allow traffic from "good" sources to reach the protected site. Traffic from legitimate users, who can be authenticated by any of the overlay nodes, is routed over the overlay to a specific node that is allowed to forward traffic through the filtering router(s). WebSOS [4] is a specific instantiation of the SOS architecture for web services, and uses Graphic Turing Tests [5] to discriminate between zombies and human-directed accesses to a web server. In *gore* we use Graphic Turing Tests to enable access to the attacked site for all types of traffic (not just web traffic). Unlike WebSOS, *gore* uses a centralized approach; while deployment in a piece-meal fashion without the ISP's collaboration (as was the goal with SOS) becomes impossible, it offers a natural model for a service offered by an ISP that has control over their network topology and internal routing.

5 Concluding Remarks

We presented *gore*, a routing-assisted defense architecture against distributed denial of service (DDoS) attacks. The goal of our system is to provide guaranteed access to a network under attack. *gore* routes all traffic destined to the network under attack to pre-constructed, ISP-controlled *gore* proxies, where servers filter out attack traffic and pass authorized traffic onward. We use web-based client legitimacy tests to identify legitimate users, where the definition of legitimacy is left to the customer; once the test is passed, *gore* transparently redirects all traffic from the user (not just web traffic) to the network under attack using GRE tunnels. In this manner, our approach is similar to the way mobile users currently access commercial wireless networks.

Our experimental results using a PC-based testbed show that *gore* is a viable approach, as its impact on the filtered traffic is minimal, in terms of both latency and

throughput. *gore* can be scaled up as needed to support larger numbers of clients and customers. Our architecture can be deployed incrementally by individual ISPs, does not require any collaboration between ISPs, and requires no modifications to either server- or client- software. Furthermore, our system allows an ISP to offer DDoS defense as a value-added service, providing an incentive missing from other proposed mechanisms.

References

1. Dean, D., Franklin, M., Stubblefield, A.: An Algebraic Approach to IP Traceback. In: Proceedings of ISOC NDSS. (2001) 3–12
2. Savage, S., Wetherall, D., Karlin, A., Anderson, T.: Network Support for IP Traceback. ACM/IEEE Transactions on Networking **9** (2001) 226–237
3. Ioannidis, J., Bellovin, S.M.: Implementing Pushback: Router-Based Defense Against DDoS Attacks. In: Proceedings of ISOC NDSS. (2002)
4. Cook, D.L., Morein, W.G., Keromytis, A.D., Misra, V., Rubenstein, D.: WebSOS: Protecting Web Servers From DDoS Attacks. In: Proceedings of the 11th IEEE International Conference on Networks (ICON). (2003) 455–460
5. Morein, W.G., Stavrou, A., Cook, D.L., Keromytis, A.D., Misra, V., Rubenstein, D.: Using Graphic Turing Tests to Counter Automated DDoS Attacks Against Web Servers. In: Proceedings of the 10th ACM International Conference on Computer and Communications Security (CCS). (2003) 8–19
6. von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: CAPTCHA: Using Hard AI Problems For Security. In: Proceedings of EUROCRYPT. (2003)
7. Farinacci, D., Li, T., Hanks, S., Meyer, D., Traina, P.: Generic Routing Encapsulation (GRE). RFC 2784 (2000)
8. Rigney, C., Rubens, A., Simpson, W., Willens, S.: Remote Authentication Dial In User Service (RADIUS). Request for Comments (Proposed Standard) 2138, IETF (1997)
9. Mori, G., Malik, J.: Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. In: Computer Vision and Pattern Recognition CVPR'03. (2003)
10. Hartmeier, D.: Design and Performance of the OpenBSD Stateful Packet Filter (pf). In: Proceedings of the USENIX Technical Conference, Freenix Track. (2002)
11. Goodrich, M.T.: Efficient Packet Marking for Large-Scale IP Traceback. In: Proceedings of ACM CCS. (2002) 117–126
12. Li, J., Sung, M., Xu, J., Li, L.: Large-Scale IP Traceback in High-Speed Internet: Practical Techniques and Theoretical Foundation. In: Proceedings of the IEEE Symposium on Security and Privacy. (2004)
13. Snoeren, A., Partridge, C., Sanchez, L., Jones, C., Tchakountio, F., Kent, S., Strayer, W.: Hash-Based IP Traceback. In: Proceedings of ACM SIGCOMM. (2001)
14. Riverhead Networks, Inc.: Centralized Protection — Riverhead Long Diversion Method Using MPLS LSP. (`http://www.riverhead.com/re/cprotection.pdf`)
15. Thomas, R., Mark, B., Johnson, T., Croall, J.: NetBouncer: Client-legitimacy-based High-performance DDoS Filtering. In: Proceedings of DISCEX III. (2003) 14–25
16. Keromytis, A.D., Misra, V., Rubenstein, D.: SOS: Secure Overlay Services. In: Proceedings of ACM SIGCOMM. (2002) 61–72
17. Keromytis, A.D., Misra, V., Rubenstein, D.: SOS: An Architecture For Mitigating DDoS Attacks. IEEE Journal on Selected Areas of Communications (JSAC) **33** (2004) 413–426
18. Ioannidis, S., Keromytis, A., Bellovin, S., Smith, J.: Implementing a Distributed Firewall. In: Proceedings of Computer and Communications Security (CCS). (2000) 190–199