Walnut: using NUTSS to harden services against DDOS attacks

Ari Rabkin

March 7, 2007

Abstract

Protecting the bottleneck link of an internet services against denial of service attacks is a difficult problem. The NUTSS architecture can be used to protect the bottleneck link for private services whose authentication can be replicated, provided that a NAT can be installed at the upstream end of this link. This paper analyzes the proposed defense and argues that it has a low run-time cost and offers substantial security benefits.

1 Introduction

1.1 The Problem

Distributed Denial of Service (DDOS) attacks are a serious problem on the internet today. In a DDOS attack, large numbers of compromised nodes make malicious connections to a service in order to consume some resource, and exclude legitimate users. DDOS attacks can either seek to consume network bandwidth, server computational resources, or OS resources (such as sockets) on the server. Distributed denial-of-service attacks that seek to overload the bottleneck link to a server are particularly difficult to stop, since the damage has already been done before the malicious traffic even arrives at the target server. Whereas OS limitations can often be avoided by cleverer systems programming, and while cryptographic primitives allow a server to impose an arbitrary computational burden on clients, network limitations are not easy to fix at the end host. Therefore, to resist such an attack, malicious traffic needs to be stopped before it reaches the bottleneck link, and thus some distance away from the target server.

This means that in order to protect the bottleneck link, there must be some sort of filter that can distinguish legitimate connections from malicious ones, and it must be possible to implement this filter on the "internet" side of the bottleneck link: typically this means that unwanted packets must be filtered at the ISP end of the connection. This is not possible in today's internet: The internet's guiding philosophy is that services should be end-to-end [14], but stopping a network flow before the bottleneck link by definition requires that the interception be done by a service inside the network, a service that doesn't fit in an end-to-end framework.

Other network architectures might, however, be more accommodating, and in recent years, there has been growing enthusiasm for non end-to-end architectures [16]. This paper presents a technique for handling DDOS attacks in the NUTSS framework, a novel service architecture developed at Cornell over the last several years [9], and described later in the paper. This paper will first discuss NUTSS, will then discuss related work and will then explain how NUTSS allows superior defense against some forms of denial of service attacks. The defense will be evaluated in terms of its scope of applicability, effectiveness, and cost.

1.2 NUTSS, NAT, and All That

Network Address Translation is now a ubiquitous internet technology. NAT extends the internet's address space, by allowing so-called NAT boxes or NATs to multiplex traffic from many hosts with distinct private IP addresses through a single public address.

When a connection is made outward through the NAT, the NAT box is able to maintain enough state to match incoming packets with the original outbound connection, and so the NAT can forward the response to the appropriate internal address. However, since the internal addresses are private, it is not possible for internet hosts to transparently route to the hidden host.

It is possible for two hosts behind separate NATs to communicate, via what is called NAT traversal. NAT traversal requires both endpoints to send a sequence of packets carefully orchestrated to put both NATs into a suitable state, at which point traffic will flow transparently. There are a number of traversal techniques, and on well-behaved NAT hardware, they work efficiently and reliably [7] [4]. Doing the traversal, of course, requires that the hosts that wish to connect have some sort of signaling channel to coordinate the traversal. This channel need only be used to set up the connection, at which point data can flow directly. As a result, using a relay or rendezvous server is appropriate.

NUTSS is not a particular technology, rather, it is as an architecture for constructing services [9]. The acronym NUTSS stands for "Network Address Translation", "URIs", "Tunnels", "SIP", and "STUN[T]". NAT is used to shield end hosts from unwanted connections. URIs are used to label endpoints, since in the presence of NAT, IP addresses are not unique. Tunnels are needed to allow protocols like TCP and IPSec to pass through NATs that might otherwise disrupt them. SIP (the Session Initiation Protocol) is a general-purpose signaling protocol, first designed for IP telephony, but now being used for many other purposes [12]. Using SIP for signaling allows deployed infrastructure, and a large body of research and engineering, to be reused. STUNT is a technique for TCP NAT traversal; STUN is the UDP equivalent [7] [13]. NUTSS is the architecture that combines these components to build services able to cope with a hostile network.

NAT has often been thought of as an inconvenience, since hosts behind NAT do not have publicly routable addresses, and thus traditional services can not be hosted behind a NAT box. The NUTSS project, however, aims to turn NAT into a good thing, by using it to build services with stronger security guarantees than have been previously possible. In NUTSS, services are actually hosted behind a NAT; when clients wish to connect, they signal this to the server via SIP. If the service's policy allows the connection to go forward, an affirmative response is sent via SIP, and the two hosts build a TCP (or UDP) connection through the NAT. A service that was intentionally built for NUTSS will of course choose a NAT that can be easily traversed, and so we expect that traversal would succeed essentially always. Since NAT traversal is fundamentally cooperative, no packets can be received by the service unless it explicitly approved a connection to the origin IP address.

Perhaps surprisingly, NUTSS works for legacy services and applications. The socket calls made by an application can be intercepted by a userspace library, and redirected into NUTSS operations in much the same way that a SOCKS proxy such as Dante [10] intercepts connections. A modified version of Dante with support for NUTSS socket interception is in development. The whole process can be made almost entirely transparent to end users and user applications; the application does its network operations normally, and they are silently translated into the relevant authentication and session creation operations. The interception can be specified by a single configuration file. As a result, an organization can switch to the NUTSS model without having to do extensive rewrites. This makes NUTSS useful for protecting legacy services such as corporate email or internal websites: administrators can allow clients on the broad internet to connect, but can require authentication before ever allowing packets from the outside internet to ever reach these services.

The expense of NUTSS is fairly small, since only a handful of signaling messages need to be exchanged before a direct channel can be constructed through the NAT or NATs between client and server. After the direct connection is established, the flow of traffic is identical to that for a traditional publicly routable service.

NUTSS strengthens system security in a number of ways, since it allows fine-grained policies to be enforced both by the service, and by the SIP server [8]. It can be thought of as a form of security through

naming, where untrusted hosts cannot address packets to the private address of the service. Further, NUTSS allows many different services to share the same SIP front end. Since SIP is a widely used and fairly simple protocol, the problem of building a secure SIP server is likely much more approachable than building a separate secure front end to many different services. Lastly, SIP can resolve URIs directly to IP addresses, without the need for a separate DNS lookup. This means that NUTSS services avoid certain DNS-related problems that traditional services suffer from; for instance, the services can change their [public] IP addresses at will, while the SIP servers can use long-lasting name records.

NUTSS, then, helps improve the security guarantees of services, and allows more flexibility in their placement. The extent to which NUTSS helps protect against network saturation attacks, and the proper way of deploying NUTSS in order to prevent such attacks, however, have not yet been carefully analyzed. It is the purpose of this paper to perform such an analysis.

1.3 Related work

NUTSS is not the only effort to fundamentally reshape services on the internet. A number of researchers have proposed supplanting end-to-end architecture with what they term "delegation oriented architecture" [16]. In Delegation Oriented Architecture, nodes resolve service identifiers not directly to the IP address of the service in question, but to the address of the delegate. NUTSS and delegation oriented architecture both seek to allow internet hosts to have control over the path that data takes to reach them. Delegation Oriented Architecture per se is implemented above the network level, and so an attacker able to discover the actual IP address of a protected service would be able to overload it; the authors propose a lightweight packet tagging and filtering scheme to address this. With this addition, DOA looks fairly similar to NUTSS, except that NUTSS does the negotiation via SIP, interactively, rather than using a static lookup service, and that NUTSS uses NAT boxes to filter traffic, rather than specialized routers.

There are a number of known techniques for hardening services against DOS attacks that seek to exhaust computational resources. The most familiar of these is the use of cryptographic puzzles, which force an attacker to do exponentially more work than the service requesting the puzzle [17]. Efforts to use such puzzles to protect network resources, by, e.g., including per-packet puzzles have largely failed; such efforts typically require the network infrastructure to verify puzzle solutions which poses serious deployment problems.

There have been attempts to track denial of service attacks backwards, and configure routers as close to the attacker as possible to drop the offending traffic [1]. It has been suggested to have routers drop packets by default, rather than forwarding them [3]. However, authenticating incoming connections, in order to know which connections ought to be forwarded, is an open problem. This paper discusses the authentication architecture necessary to support this "default off" model.

Badishi et al have observed that network-level filtering (using filters that operate on packet headers) is very efficient, and that if applications can control these low-level mechanisms, this allows for efficient enforcement of policy [2]. More concretely, they propose building a secure two-party channel by having endpoints shift which port numbers to read from unpredictably, according to a secret pseudorandom function. Only the correct endpoint has access to the pseudorandom function, and is able to route traffic to the open port. The system works analogously to frequency-hopping radios.

Of course, the most standard approach is to replicate the service, or at least critical front-end parts of it. One way to do this replication is with IP anycast [11]. Another is to have the DNS server load-balance by dispatching requests to various servers [6]. This DNS-based approach has the drawback that malicious clients can submit requests to machines via their numerical IP address, thus circumventing the load balance.

1.4 Restricting the problem

Any sort of replication, of course, only makes sense if the service can be conveniently replicated. Many services require substantial resources to replicate, or require some sort of consistency semantics, and so replicating the entire service is impractical.

Protecting public services is difficult, because the service's policy by definition allows any host on the internet to connect; there is no a priori notion of an invalid client. Particularly if an attacker can spoof the source address on packets (so the attacker can use an unbounded number of identities), it becomes difficult to distinguish malicious traffic from legitimate use.

However, many services can do some sort of authentication before processing connections. For instance, private web servers, the central servers of massively-multiplayer online games, and most email servers have a strong notion of valid and invalid clients. Therefore, for these services it makes sense to isolate and replicate only the "authentication" component that filters out malicious traffic, and forwards legitimate traffic to the service. This paper only focuses on defenses for such access-controlled servers.

2 Using NUTSS to stop DDOS

To prevent the bottleneck link to the non-replicated component of a service from being overloaded there needs to be some sort of firewall or filter that keeps out packets from attackers. Moreover, this filter itself needs to be at least as hard to saturate as the underlying network. To give an analogy: If we consider the internet as a "series of tubes", [18] then a bandwidth exhaustion attack is an attempt to fill up the narrowest pipe with debris to block the flow of legitimate traffic. The goal of a defense is to put a filter at the junction between this narrowest pipe and the wider pipe beyond. Moreover, the filter must be implemented in such a way that the filter itself cannot be clogged.

In the NUTSS architecture, the filtering or authorization mechanism is a NAT, and access through the NAT is mediated by a SIP server and some sort of authentication service. However this approach raises a number of issues which the remainder of this paper is devoted to exploring. Where appropriate, options will be pointed out and analyzed. The most important questions are: How well does NAT work for filtering? How best to authenticate clients? Which of the many possible SIP server configurations is optimal? How secure is it all? How scalable?

NATs are very effective filters. They typically maintain a small state machine for each ongoing connection, and will reject all incoming packets not associated with an active connection. If the underlying connection through the NAT is TCP, which is likely to be the most common case, then it is very difficult for a third-party to insert packets into an ongoing stream. TCP packets have per-byte acknowledgments, and so the only way an attacker can insert a packet is for the attacker's packet to show up with the appropriate sequence number. These numbers change rapidly and unpredictably, and so for an attacker to forge them, the attacker needs to have very precise knowledge as to the state of the TCP stream, and the ability to respond very quickly to events. Thus only a very powerful attacker can forge TCP packets. A NAT can detect invalid sequence numbers, and so is able to detect spurious packets with high probability. Thus, a NAT is able to efficiently detect and filter bad TCP traffic. Preventing UDP packets with forged sender addresses from being routed through the NAT is harder, and is discussed below.

For NUTSS to protect a network link from saturation, that link needs to be behind the NAT or firewall so that traffic can be filtered upstream. This typically means installing a NAT at the ISP end of a link, rather than at the service end. While such NATs are not widely deployed today, their deployment poses no major organizational or technical problems, and any organization that would make a good target for a DDOS attack could afford the cost of the hardware. Any ISP able to host substantial services would have the technical capacity to host such a NAT.

In any NUTSS deployment, the SIP service used to negotiate the traversal must be outside the NAT. Denial of service resistance adds an additional requirement: the SIP service must be able to deny invalid

connections without contacting the service being protected, or any other service at the far end of the bottleneck link. Otherwise, an attacker could overload the link by inducing the SIP service to send a large number of bogus connection requests. Rate-limiting is an imperfect defense, since an attacker able to exceed the rate-limiting threshold will cause legitimate connections to time out or be dropped.

SIP is likely to be the practical bottleneck for most NUTSS deployments, and so the performance evaluations of this paper largely focus on SIP performance. NATs can handle packets associated with an existing route quickly, and can simply drop packets without a valid translation; NAT can therefore handle large volumes of inbound traffic. If the bottleneck is the network "in front of" the NAT, then NUTSS is unhelpful: there is always a network bottleneck somewhere.

2.1 Requirements

The requirements for deploying the DDOS defense presented in this paper are believed to be reasonable. The required placement of NAT boxes and SIP servers beyond a service's bottleneck link poses no technical problems. In addition, the proposed defense only relies on a subset of the features supplied by NUTSS. No assumptions are made about the details of NAT, URIs, or STUNT; the key requirement is actually that end hosts are behind some sort of middle box that controls network flows, and that passage through the middle box is negotiated over SIP. NAT is useful because it forces connections to machines with private addresses to pass through a NAT box, but conventional firewall deployments can also have this property. The analysis of this paper applies equally well to a service architecture using firewalls and IP addresses, instead of NATs and URIs ("FITS", as it were).

2.2 Scope and Security

NUTSS uses NATs to do authorization, meaning that the authorization token in each packet is the IP packet headers, plus (in the case of TCP) a sequence number. An optional per-packet MAC might also be employed. Packet headers can of course be forged if the attacker knows the port on the NAT which was opened for the session. This can be determined by intercepting at least one legitimate packet.

Note that the security guarantees provided by TCP are maintained the presence of a NAT, and injecting packets into ongoing TCP connections requires an adversary powerful enough to intercept legitimate packets and send new packets with very precise timing constraints. This threat is considered unimportant for most internet services. For services needing stronger guarantees or using UDP traffic, per-packet MACs are appropriate; this option is discussed in a subsequent section.

NUTSS is a feasible defense even when a limited number of authorized hosts have been compromised. Any particular host can only send a limited amount of traffic, preventing that host from mounting an effective denial-of-service attack. Moreover, if an attacker attempts to overload the service using the path through the NAT arranged for the compromised host, then the service can easily determine which machine was compromised, and can then close that connection.

NUTSS trades the problem of building secure replicated services for the problem of building secure replicated SIP servers. At a high level of abstraction, it is better to build one secure high performance signaling solution than N secure services. Further, SIP is a lightweight and common protocol and so the security and performance problems are well understood and easy to solve. As a result, there is a great deal of industry experience with SIP servers, and a great deal of effort has gone into building secure servers. Due to the simplicity of SIP, replication is straightforward, with [5] an example of a workable approach. As a result, this trade is advantageous.

3 Design Options

This bare-bones description leaves a number of open design choices. This section will discuss options for how to authenticate clients, for where to authenticate clients, for what transport protocol to use for the signaling plane, and for protecting the data plane

against tampering.

3.1 Authenticating Clients

Since authentication requests cannot be funneled through the bottleneck link to the service, they must be handled by some authentication service located on the same side of the bottleneck link as the SIP server. The message exchange required for authentication can be done cleanly within the standard SIP protocol. SIP was originally designed for voice-over-IP telephony, and VOIP providers needed strong authentication in order to bill clients. As a result, the SIP specification and commercial SIP software have ample provision for authentication. The SIP specification includes a simple challenge-response protocol, and allows various other authentication techniques to be used as well.

The simplest way to do authentication is to maintain some sort of user database that maps from user ID to a secret shared between that user and the service. To authenticate a connection, some cryptographic operations must be performed to prove that the client knows the relevant secret. Even with a million users, the user database can be easily stored in main memory.

Updates to the authentication database can be distributed reliably and efficiently, provided the rate of change is not too large (not too many users added and removed per unit time). Each authentication server by assumption has ample bandwidth, and so can receive incoming connections from the protected service. These updates can be signed to prove their authenticity. As a result, any standard reliable-multicast protocol can be used to distribute these updates. If a brief window of un-availablity is acceptable, then only minimal synchronization is needed. The protected service does, however, need to be able to propagate updates to any servers that are performing authentication.

This approach has one important weakness, which is that the protected service has offloaded the complete user database to several authentication proxies, making the service vulnerable to a compromise at any of the proxies. It is tempting to seek a protocol where the authentication services do not actually hold secrets, but merely have some sort of oracle access to an authentication function. However, this is not possible, since if an authentication server is compromised, the attacker will learn the authentication tokens of a legitimate user when that user logs in. The best that can be done is to rotate the SK_c values; this can be done by having the clients and the protected service arrange some sort of key rotation scheme (so SK_c is replaced by $SK_{ct} = H(SK_c|t)$; the service can then distribute the rotated keys to the authentication servers on a schedule. This way, a compromised authentication server can only leak short-term keys.

This authentication can be done inside SIP. The SIP specification includes a "digest authentication" mechanism very similar to that employed by HTTP 1.1 [**?**]. The protocol in essence is as follows. Let c be the ID of the client, t be a nonce, and K_S a symmetric encryption key known only to the replicated SIP servers. SK_c is a secret shared between the SIP server and this particular client for authentication. This can be either a password or the salted hash of a password. H is a secure hash function, typically MD5.

Client	$\rightarrow$	Serv	INVITE $[c]$	
Serv	$\rightarrow$	Client	407 Auth Required t	
Client	$\rightarrow$	Serv	INVITE	$[c]$
			$t, H(t, c, SK_c)$	

The protocol can be made stateless; servers can generate the nonce t from a timestamp, and all the replicated servers can accept any timestamp that is sufficiently recent. The protocol completes successfully provided that the client-supplied hash matches the server's expectation. So long as H is not invertible, the scheme is secure. Only a user able to receive packets addressed to c and forge packets from c will be authorized. (An attacker able to do so can mount a man-in-the-middle attack by allowing the legitimate user to authenticate and then taking over the session).

An attacker able to intercept a single legitimate third message is, however, able to replay it. This will force the server to repeat the authorization step, and then to pass the authorization on to the protected service, thus perhaps overloading the protected service or the bottleneck network link. One fix for this is to

have the SIP server keep track of recently used nonce values, and allow only one request for the nonce. In a replicated setting, each SIP server can track this independently; this allows a small constant number of replays equal to the number of servers.

This protocol has an additional benefit: $K_S\{t, cIP\}||SK_c$ is a shared secret held by the authentication service and the user that will never be reused. It can thus be used as a seed for further cryptographic protocols, such as requiring a MAC on every packet. This approach can be useful in some circumstances, as will be discussed in a later section of this paper.

Alternatively, users might be issued with public-private key pairs and certificates signed by the server. In this case, the storage burden is lessened since the server need not hold secrets, but the computational cost of authentication rises significantly since authentication requires comparatively expensive public-key operations. This avoids the cost of a database of passwords, but has a higher cost per-authentication, since public-key cryptographic operations are substantially more expensive than hash functions. In addition, it requires users to maintain secret keys, rather than remember human-intelligible passwords. On the other hand, using public-key methods avoids the security risks in maintaining a replicated database of per-user secrets, making it attractive if the SIP servers are untrusted.

3.2 Placement of the Authentication Service

The use of SIP, and the choice of a cryptographic protocol to use over SIP, do not constrain the actual placement of the authentication database or service. Handling requests locally is somewhat more efficient than network communication. However, if SIP servers are administered separately from the underlying service, the SIP administrators might not wish to have fairly expensive processes running on their servers on behalf of other organizations, and might prefer to forward authentication requests to a separate service. This separate service would then need to be replicated; database replication is straightforward when the frequency of writes is small and so this poses no technical problems. The cost difference between a local and a remote procedure call is small, however, and so ultimately, the optimal placement of the authentication service likely depends more on administrative concerns than technical ones.

3.3 Transporting the Signaling Plane

SIP is a highly configurable protocol, and among other things, allows a choice of TCP or UDP as the transport protocol. UDP is more common, since most SIP messages fit comfortably in a single packet, and since the SIP protocol is fundamentally message, rather than stream, oriented TCP provides no security benefit over digest authentication; both make it impossible for an adversary to insert packets into an ongoing connection.

What is worse, there are significant performance costs to using TCP. First, the TCP handshake requires additional message exchanges, increasing bandwidth consumption and session initiation time. Second, TCP results in much less graceful handling of overload conditions. While the SIP server can drop UDP packets if it is overloaded, the operating system manages sockets, and so an overloaded SIP server can consume all available sockets, and will then stop accepting connections entirely. This means in particular that an attacker mounting a denial-of-service attack could attempt to consume all available sockets on the server. Since the sockets are created before the client is authenticated, the server will be forced to deny incoming connections at random until the load subsides; this denies service to legitimate users. In contrast, with UDP, much less work happens before the authentication step and no OS resources need be allocated.

It is also possible to use TLS over TCP, providing a very high measure of security, at the cost of a significant added computational burden. This has the advantage of making the entire SIP transaction opaque, and in particular, of concealing the "To" header of the INVITE request that initiates the session. However, the actual privacy gain is minimal. A passive adversary can learn the endpoints of a connection by examining the data plane connection instead of by intercepting the signaling messages. TLS

has a substantial cost however, since it requires several more message exchanges before the connection is initiated. Moreover, TLS seems to be poorly supported by current SIP software; we discovered that OpenSER, the widely-deployed SIP server that we used for performance evaluation, crashes frequently when using TLS.

Thus, it seems as though the signaling plane by which NUTSS hosts communicate ought to be carried over UDP if possible, since it has much better performance characteristics for equal security. The only circumstance in which it makes sense to use TCP is if UDP is not an option, for instance if the client is behind a firewall that will not forward UDP data.

3.4 Packet tagging

For UDP streams a low-cost packet tagging approach is needed to prevent an unauthorized adversary able to intercept at least one legitimate packet from sending data to the protected service. In such a scheme, each packet carries a message authentication code (a MAC) generated by a secret key shared between the NAT and the client. The NAT would drop packets without the appropriate MAC. Without such MACs, an attacker who can read the headers on a legitimate packet can forge source and destination IPs and ports in order to send packets to a host behind the NAT, possibly congesting the bottleneck link. Note that this technique requires a customized NAT box, rather than one that is available "off-the-shelf" today. Despite this cost, using per-packet MACs is an attractive and well-understood technique; it has been proposed in a very similar context by [16] and [2]. The cost of generating and evaluating the MAC can be a single hash function evaluation; this cost is fairly small for modern systems at reasonable load levels.

it is difficult for any plausible passive observer to inject traffic into a legitimate TCP connection through a NAT, because successfully forging TCP sequence numbers requires the attacker to know what the state of the channel at the instant the forged packet arrives. Moreover, adding MACs to TCP packets is difficult; most operating systems do not expose the boundaries between packets to applications, making it difficult to place or require a MAC on each packet. Preventing an adversary able to effectively forge TCP headers from routing TCP traffic to a protected service is an unresolved problem. TLS or other encryption prevents such traffic from being delivered to the end-application; however it is filtered out at the end-host, and thus if the threat is bandwidth exhaustion, TLS is no help.

4 Performance Evaluation

As was discussed in previous sections of this paper, SIP is a very flexible protocol. SIP is the public interface to a NUTSS service, and so the scalability and performance of a NUTSS service will likely be driven by the behavior of the SIP server. One of the key advantages of SIP is the opportunity to use existing hardware and software, and so we evaluated the behavior of current industry-standard SIP software. In particular, we attempted to quantify the performance costs of different options by measuring the performance of OpenSER [15], a popular open source SIP server, and then comparing the baseline behavior against a number of alternate request-handling strategies.

4.1 Experimental Set-up

We tested SIP server performance with a simple three-machine loop, comprising an initiator, a respondent, and a SIP server. We used SIPP, a SIP analyzer and load generator, on the initiator and respondent. The sample workload we used for evaluation was to have the initiator create a call (the basic transaction in SIP) to the respondent, and then terminate the call as soon as the respondent returned success. This represents the typical workflow when SIP is used to coordinate NAT traversal.

All our measurements were repeated on two different machines, one fast, one slow. The "slow" machine was a 2001 Dell, with a 1.7 GHz P4 processor and 512 MB of Ram. The network link was a shared 100 MBit ethernet. The "fast" machine (oslo) was a 2006 Alienware workstation, with an AMD Athlon 64 X2 (dual-core) running at 2.6 Ghz and provisioned with 2 GB of Ram. For all measurements, the initiator

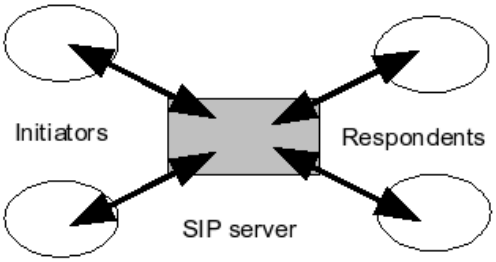

Figure 1: Experimental Setup

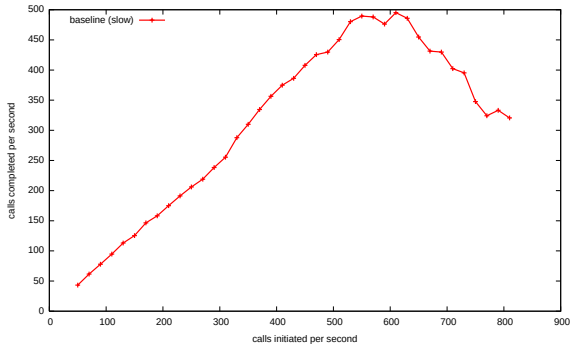

Figure 2: Baseline OpenSER Performance (slow)

(erie) was another Alienware system, identical to the "fast" server. The respondent (hathor) was a server with two dual-core Opteron 275 chips, running at 2.2 GHz, and provisioned with 4 GB of RAM. All machines ran Fedora Core 5 Linux.

In order to add load to the fast SIP server without saturating the endpoints, we measured performance of the "fast server" with two senders and two receivers operating in rough synchrony. Since our statistics were taken from the middle of each burst of activity, minor drift between clocks was not an issue. Final results were calculated by averaging the two.

4.2 Baseline SIP performance

We started by measuring the performance of a baseline SIP server on our fast and slow test platforms. The slow machine saturated when receiving between 500 and 600 calls per second (cps); as the call rate

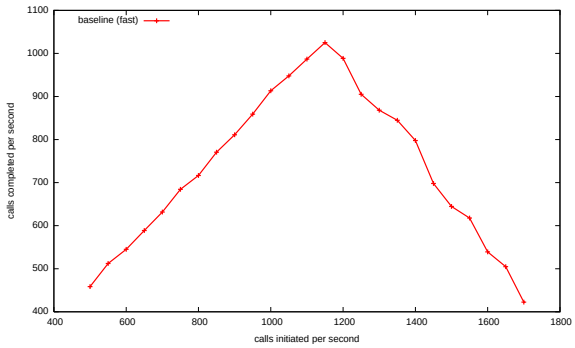

Figure 3: Baseline OpenSER Performance (fast)

increased beyond that point, performance started to fall off. Examination of detailed statistics from the experimental run showed that as the load increased beyond that range, the server started to drop a large fraction of packets; the resulting retransmission overhead is one of the chief reasons for the decline in performance beyond that point. Qualitatively similar behavior was observed when the server was run on the fast machine, although in this case, the maximum possible throughput was in the 1200 cps range.

4.3 Filtering costs

The quantity of malicious traffic that a SIP server can absorb while still handling legitimate requests is is irrelevant in a sense. Malicious requests need to be filtered somewhere, and the computational cost of filtering them is not substantially different if the filtering is done by the end service rather than by a SIP server some distance away in the network. Even so, for practical purposes, the degree of SIP server replication required to handle a certain level of malicious traffic is worth knowing. The precise cost of handling a malicious packet depends on the details of the authentication mechanism in use. However, the cost of receiving and parsing a SIP request gives a minimum bound on the computational cost of denying a request.

We modified OpenSER to automatically reject requests whose sender matched a particular seven-character string; the check was performed after a par-

tial parse of the packet. (The first line and the Via headers were parsed). We put a load of 300 valid calls per second on our slow testbed. We then compared the call completion rate with and without bad traffic of 25000 calls per second. To put that number in perspective, we observed that 30,000 SIP requests per second saturated the fast ethernet network connecting the hosts in the experiment. Thus, this experiment compares call completion rate under moderate load, with call completion rate under close to the maximum input rate the network could support. The standard deviation in the table below indicates the deviation between the call completion rate from minute to minute.

Bad rate	Goodput	Std. Dev
0	266	4.69
25000	238	7.96

Our results indicate that even at close to the maximum network bandwidth, and even on a comparatively slow machine, receiving and parsing packets is not the bottleneck limiting SIP server performance. This suggests strongly that a modern computer has the CPU capacity to filter requests even at close to the maximum load tolerated by the network.

4.4 Digest Authentication

Digest authentication is the standard technique for authenticating SIP connections. On both fast and slow hardware, digest authentication turned out to decrease throughput by no more than 10%. This number is reasonable, since the number of messages per call increases by approximately 10%, and the additional computation is a single MD5 sum.

Our tests were conducted with passwords stored in an in-memory database in the SIP server process. We believe that this is reasonable, since the memory cost of a password table is small, and would be insignificant for deployments of even tens of thousands of users. If an SQL database were used, there would be an additional overhead from the database lookup. Estimating the cost of this lookup for a typical deployment is outside the scope of this paper.

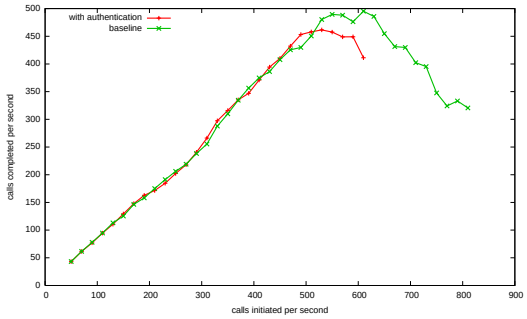

Figure 4: Digest Authentication: slow server

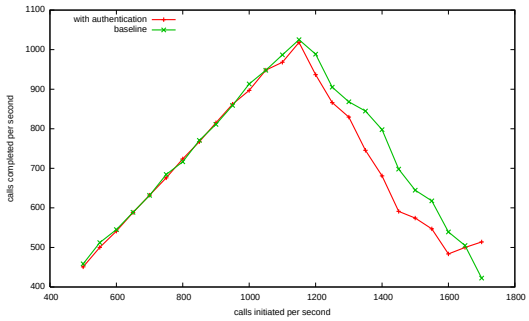

Figure 5: Digest Authentication: fast server

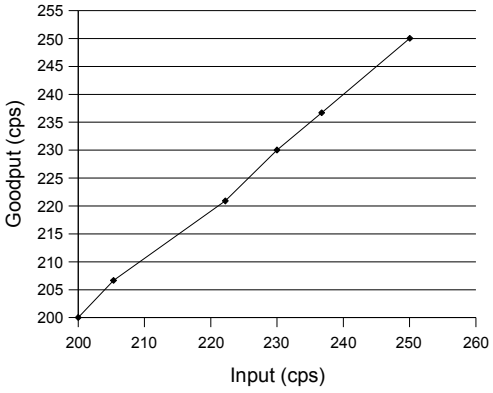

Figure 6: SIP over TCP: slow server

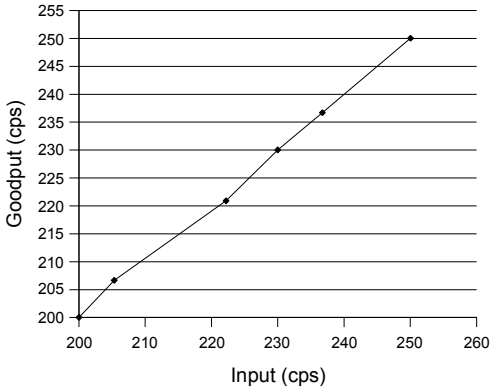

Figure 7: SIP over TCP: fast server

4.5 SIP over TCP

We attempted to repeat our UDP baseline measurements using TCP. TCP was used on both the link from the initiator to the SIP server and the link from the server to the respondent. While only one socket was used for the latter link, a new socket was created for each connection from the initiator to the SIP server. This was so that we could measure precisely the number of sockets being created per unit time by tracking them at the initiator, only. In addition, this accurately models the likely deployment scenario, with a long-lived connection between the SIP server and the private service, and clients connecting, and each client creating a socket for the connection.

One of our significant results is that SIP over TCP fails very differently from SIP over UDP. When SIP is run over UDP, recall, requests that cannot be handled are dropped at the application level, and so performance degrades gradually as the load increases beyond what the server can handle. When using TCP though, the server fails much more drastically.

Below the critical threshold, we discovered that the server's throughput is precisely equal to the load; in other words TCP's reliability properties are sufficient to guarantee correct handling of every request. However, once the load exceeds the server's computational resources, open sockets start to accumulate. Since the operating system has higher priority than the application, the OS is able to establish the socket correctly. However the SIP server is unable to finish handling the call, and so the socket remains open. As a result, the number of open sockets surges, until a limit is hit, at which point errors start being returned. The client sees these errors as connection-refused packets, and the server application sees them as failed calls to accept().

OpenSER and SIPP handle these conditions badly, typically aborting entirely. This condition could be handled more gracefully by better programming, but the application will still be subject to the limitations of the operating system's TCP stack.

The critical threshold at which errors started appearing for the slow machine was around 260 connections per second, and on the fast machine, around 500 cps. These numbers vary significantly between different experimental runs, and we suspect that this threshold is very sensitive to the underlying hardware, and so they should not be treated as anything other than a very rough benchmark.

5 Conclusion

Placing a NAT at the upstream end of a bottleneck connection, and using NUTSS to deploy services behind the NAT, makes the services far more resilient to bandwidth saturation attacks, by moving the bottleneck from the service to the (better-provisioned) ISP. Moreover, SIP signaling provides a flexible and efficient platform for authenticating connections.

Every service will always have some bottleneck that is vulnerable to saturation. However, all bottlenecks are not alike. Computational resources can be allocated on-demand, and thus shared between services; network bandwidth is intrinsically less sharable and cannot be expanded on demand. Hence, shifting a service's bottleneck from the network connection to computational resources can be a significant gain.

This paper argues that authenticating connections in the middle of the network, rather than at the endpoints, allows just such a gain. The SIP servers and other authentication systems required can typically be distributed widely and capacity can be shared in an adaptive way between different services. One can

envision a very large pool of SIP servers, allocated between many different private services as needed. As a result, the "front end" of a service can be made highly resistant to DOS attack without having to replicate the entire service. While NUTSS is not yet suitable for wide-scale deployment, it is maturing rapidly. We hope that it will be possible to deploy NUTSS-hardened services within the year.

More work, of course, remains to be done. An obvious next step is a comprehensive examination of the performance implications of authentication. Another step would be to repeat the performance experiments for more network and server configurations to better understand how current SIP implementations scale. Perhaps most importantly, the defense proposed in this paper does not apply in any straightforward way to public services; protecting public services from denial-of-service remains an unresolved problem.

6 Acknowledgments

We would like to thank Paul Francis for suggesting this line of work. Saikat Guha supplied extensive and invaluable advice, encouragement, and guidance. We also appreciate the comments of the anonymous reviewers. Lastly, we appreciate the patience and understanding of the Cornell Systems Lab community, who tolerated the network disruption from our test runs.

References

[1] K. Argyraki and D. Cheriton. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. *USENIX 2005*, 2005.

[2] G. Badishi, A. Herzberg, and I. Keidar. Keeping Denial-of-Service Attackers in the Dark. *LECTURE NOTES IN COMPUTER SCIENCE*, 3724:18, 2005.

[3] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by Default! In *Proceedings of HotNets '05*, College Park, MD, November 2005.

[4] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig. NATBLASTER: Establishing TCP connections between hosts behind NATs. In *SIGCOMM Asia Workshop*, 2005.

[5] M. Bozinovski, T. Renier, HP Schwefel, and R. Prasad. Transaction Consistency in Replicated SIP Call Control Systems. *4th International Conference on Information, Communications & Signal Processing and Fourth Pacific-Rim Conference on Multimedia (ICICS-PCM 2003), December*, 2003.

[6] V. Cardellini, M. Colajanni, and PS Yu. Dynamic load balancing on Web-server systems. *Internet Computing, IEEE*, 3(3):28–39, 1999.

[7] Saikat Guha and Paul Francis. Characterization and measurement of TCP traversal through NATs and firewalls. In *IMC*, 2005.

[8] Saikat Guha and Paul Francis. Towards a secure internet architecture through signaling. Under Submission, April 2005.

[9] Saikat Guha, Yutaka Takeda, and Paul Francis. NUTSS: a SIP-based approach to UDP and TCP network connectivity. In *FDNA '04: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 43–48, 2004.

[10] Inferno Nettverk. Dante: A Free Socks Implementation. http://www.inet.no/dante/.

[11] D. Katabi and J. Wroclawski. A framework for scalable global IP-anycast (GIA). *ACM SIGCOMM Computer Communication Review*, 31(2 supplement):186–219, 2001.

[12] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. RFC 3261: SIP: Session Initiation Protocol. June 2002.

[13] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. RFC 3489: STUN - simple traversal of UDP through network address translators (NATs). March 2003.

[14] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[15] The OpenSER Project. OpenSER SIP Server. http://www.openser.org/.

[16] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes No Longer Considered Harmful. In *OSDI 04*, San Francisco, CA, December 2004.

[17] X. Wang and M. Reiter. Defending against denial-of-service attacks with puzzle auctions. *Proceedings of IEEE Symposium on Security and Privacy*, 2003.

[18] Wikipedia. "series of tubes". Online. http://en.wikipedia.org/wiki/Series_of_tubes, 2006.