Steps Towards a DoS-resistant Internet Architecture

Mark Handley, Adam Greenhalgh
University College London
{M.Handley, A.Greenhalgh}@cs.ucl.ac.uk

ABSTRACT

Defending against DoS attacks is extremely difficult; effective solutions probably require significant changes to the Internet architecture. We present a series of architectural changes aimed at preventing most flooding DoS attacks, and making the remaining attacks easier to defend against. The goal is to stimulate a debate on trade-offs between the flexibility needed for future Internet evolution and the need to be robust to attack.

Categories and Subject Descriptors

C.2.0 [**COMPUTER-COMMUNICATION NETWORKS**]: General - Security and Protection

General Terms

Design, Security

Keywords

Internet, Denial-of-Service, Security, Network Architecture

1. INTRODUCTION

Denial-of-Service (DoS) attacks are one of the most significant problems currently facing the Internet and its users. For many users these attacks are merely an irritation - even if they understand the reason for the poor performance they occasionally observe. However, for the Internet to achieve its full potential, it has to be able to offer highly reliable service, even in the face of hostility.

We will examine some significant changes to the Internet architecture aimed at making the Internet more robust. Such changes should not be made lightly - any widespread change has real costs associated with it. In writing this paper we are only too aware that the problem of DoS attacks can not be *completely* solved by the architecture we propose. The problem needs to be tackled on many fronts simultaneously. However we do believe that architectural changes are necessary. The only question is what form those changes must take? In this paper we will take a fairly radical position, with the aim of stimulating this debate.

SIGCOMM'04 Workshops, Aug. 30+Sept. 3, 2004, Portland, Oregon, USA.

2. THE NATURE OF THE PROBLEM

The first step in considering a security problem is to consider the nature of the threat. In [9], the Internet Architecture Board provides a detailed discussion of the nature of DoS attacks on Internet systems, and we strongly recommend this document. Basically all systems are vulnerable to some form of attack, be they clients, servers, firewalls, routers or links. Attacks can attempt to exhaust processing power, memory, bandwidth, quotas, disk-space, and pretty much any other "consumable" that a system requires to perform its job.

One modern PC connected to a high-speed network can source around 1Gb/s of traffic, which is enough to saturate many network links and, if the traffic is carefully crafted, enough to overload many large servers. However, traffic from a single machine is relatively easily filtered. Although automated mechanisms to *push-back* such filters towards the source are not widely deployed, there are few technical problems in doing so[13][10].

Unfortunately *source-address spoofing* makes it harder to push-back filters without causing collateral damage. Further, many DoS attacks are *reflection* attacks[16], where the attacker sends traffic to a third party, spoofing the source address of the victim. The third party then replies to the victim, overwhelming them. The attacker can then use many third parties to spread his attack, so now the traffic is "originating" from all over the Internet. In addition, some reflection attacks manage to *amplify* the original attack because the responses sent by the third party are larger or more numerous[6][9] than the original messages sent by the attacker.

The biggest DoS problem is caused by *distributed* denial of service (DDoS) attacks, where the attacker compromises a large number of systems and then uses these "zombie" systems to attack the victim. DDoS attacks of sufficient scale provide the firepower needed to overwhelm almost all victims. They can also be combined with spoofing or reflection to make the attack even more difficult to defend against. Currently most DDoS attacks do not bother to spoof the source addresses because, as no automatic push-back mechanism is widely deployed, it takes so long to shut down each zombie that there is no need to hide their identity.

DDoS is principally an issue due to widespread exploitation of software vulnerabilities, which permit the control of large numbers of compromised systems. To gain sufficient scale, such exploitation is typically automated using worms, viruses, or automated scanning from already-compromised hosts (so called "bots"). Fast-spreading worms are extremely hard to combat in the current Internet Architecture, so these are a particular concern[14]. Although viruses and bots are a serious issue, their spread rate is slower, which permits a wider range of defense options.

3. DEFENSE

It is important to begin by recognizing that it is not possible to completely protect all servers against all DDoS attacks. If a sufficiently subtle attacker with sufficiently many compromised hosts at his disposal can mimic legitimate traffic well enough that the victim cannot tell good from bad, there is little that can be done beyond load-shedding, adding server resources, and minimizing collateral damage. However, our ultimate goal is that this is the *only* way to persistently DoS-attack a server, and that routers and client systems are invulnerable to DoS attacks.

Viewing the problem from a high level, there are many tracks we can take to make a network architecture that is more resilient to DoS than the current Internet:

- Significant improvements in end-system software security will reduce the ease with which systems are compromised, and hence make the construction of zombie-armies more difficult. However, we do not expect this to be sufficient by itself.
- Reducing the ability of worms and viruses to spread quickly will reduce the threat of very large scale DoS attacks. Fast spreading worms are a particular threat, because they outpace the speed of any possible human mediated response.
- Preventing source-address spoofing will aid the shutdown of attacks that do occur using push-back mechanisms.
- Preventing reflection attacks will also aid the shutdown of attacks and prevent innocent third parties from being implicated or harmed by reactive defenses.
- Without source-address spoofing and reflection attacks, automated push-back mechanisms would be much more accurate and effective.
- Large-scale wide-area distribution of key services such as DNS would localize attacks, reducing the attacker's advantage and minimizing collateral damage.
- Hosts not wishing to receive incoming connections attempts from the public Internet should not have do so.
- Router-to-router traffic should be effectively isolated from all other traffic to reduce DoS threats to routers. Although this is at least as important as the points above, it is not the main focus of this paper, so we will not discuss it further.

In this paper we do not discuss the protocol and implementation *details* needed to ensure that transport protocols and applications are robust to DoS. Instead we concentrate on architectural changes that more widely restrict an attacker's freedom and permit more effective defense.

3.1 Advertised Service

In the Internet architecture, a route advertisement is effectively a service advertisement for all the hosts on that subnet, saying "route packets to these hosts". However, such a service advertisement is rather more liberal than is generally desired. What is mostly desired is the ability to route service requests to a server for that service, and to route responses back to the client *and nothing else*. This is of course an over-generalization, but we shall use this as a starting point for a revised service model. Such a service model would have immediate implications for DoS:

- A client could not send any request to another client, which prevents a whole category of DoS attack, and also prevents client-to-client worms.
- If the server to which a client initiates communication is the only host that can send packets back to that client, then no other server can DoS that client. This also prevents many reflection attacks.

4. TOWARDS A DOS-RESISTANT ARCHITECTURE

How might we implement such desired restrictions without encountering scalability issues? In this section we will propose a set of architectural changes that, taken together would greatly limit the scope for attack. The aim is to allow all the desired modes of interaction between systems, but greatly restrict everything else.

Step 1: Separate Client and Server Addresses

The IP address space[1] can be divided into a set of client addresses and a set of server addresses. The aim is to allow clients to initiate connections to servers, but not to allow clients to initiate connections to clients, nor servers to initiate connections to servers.

The benefits of this depend on where the restriction is enforced. Even if it were only enforced by the recipient, this would immediately reduce the threat from worms. A worm would have to spread from client→server→client, exploiting two separate vulnerabilities. This greatly slows the worm because the server→client phase depends on clients choosing to contact an infected server. Very fast spreading worms would no longer be possible. Worms that attempt to spread via contagion are still possible, but are significantly more likely to be spotted in the client→server phase in time to react by the sort of organized honeypots proposed in [20].[2]

In addition, many reflection DoS attacks on servers (such as bang.c[9]) are prevented. A reflection attack on a server would require server→client→server communication, and most clients are not going to respond to a new connection request from a server[3].

In the current Internet, some of these benefits arise from the use of Network Address Translators. However, NATs only benefit those clients who chose to use them; attackers simply choose to attack from hosts that are not behind NATs. By formalizing the asymmetry and accepting it as a key part of the architecture, the benefits can be much more widespread, as we will show below. In addition the downside of NATs not being a consistent part of the architecture can be avoided.

Clearly if many hosts have both client and server addresses then some of the benefits are lost; we hope that few hosts will need both *globally-reachable* client and globally-reachable server addresses. Typically a server should not be initiating wide-area outgoing connections, and most clients only need to accept incoming connections to permit local management. Obvious exceptions are peer-to-peer applications, and telephony-style applications. We will discuss these special cases in Section 5.1.

Step 2: Non-global Client Addresses

A client address does not need to have any global significance. It only needs to have significance along the path between the client and the server, so that packets from the server can be returned to the client. In fact, a client *wants* its address to not have global significance - this prevents distributed DoS attacks on the client

[1]Typically we are thinking about IPv6 addresses here due to the additional flexibility available from longer addresses.

[2]The state of the art in honeypots still needs to advance somewhat before this could be considered a solved problem.

[3]The exceptions would be stateless reflection such as responding to an ICMP echo request.

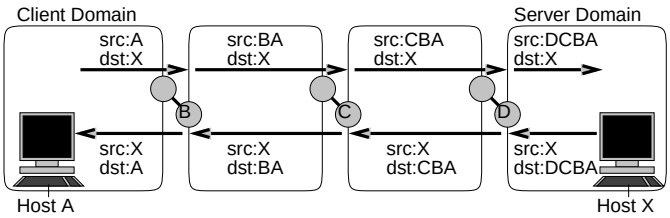

FIGURE 1—Path-based Addressing

or its access network unless each attacking host can figure out a workable address to route traffic towards the client.

One way to allocate client addresses in a non-global manner would be for the source address to be constructed domain-by-domain as the packets travel from client to server. This is illustrated in figure 1. On entry to a domain, a local routing ID (indicating the next domain toward the client) would be pre-pended to the source address. Packets being forwarded back towards the client would then be forwarded by the usual longest-prefix-match forwarding mechanisms within each domain (in this case the prefix is the local ID for the next domain). As the packet leaves each domain heading back towards the client, the local ID is removed.

It is likely that such address prepending can be done at line-speed in some of todays backbone routers with only firmware upgrades.

The goal is that even if one malicious server in the Internet knows a client's address, no-one elsewhere in the Internet who is given that address can easily deduce a workable client address to reach the client from their location. The simple prepending scheme above provides this protection to some degree, but it is possible that more security is required. This can be achieved at the expense of additional forwarding cost by encrypting the client address before prepending the local ID in the client-to-server path, and decrypting it in the return path after removing the local ID. For example, the router with interface C in figure 1 would receive client→server packets with source address BA and forward then on with source address $Cf(BA)$ where $f(x)$ is a shared key encryption function[4]. Whether such additional security is really necessary is an open question, but the architecture would permit it as a matter of local policy. In practice it is likely that only a few core ISPs need to do this to gain most of the benefits.

Such path-based client addresses have some useful properties:

- The make complete source-address spoofing impossible for clients. A client address will always reveal the path back to the origin domain, although it may not reveal the precise host at that domain. This should allow pushback mechanisms to function effectively against traffic sourced by clients.

- In contrast to the current Internet, the path between client and server would be symmetric at the domain level (although not at the router level). The economic implications of this are not completely clear to us. However, at the very least it should simplify many network monitoring functions for transit ISPs.

  Perhaps more importantly, if an ISP monitors that there is a large amount of traffic in one direction but no appropriate responses in the reverse path, then the ISP can conclude by itself that this traffic is malicious. This may allow link-saturation attacks to be identified and shut down. However, such a deduction may require collating information from multiple routers within the ISP's domain, so may not be easy without substantially improved network monitoring tools.

[4] All the border routers of a domain share the same key

- All reflection attacks against remote client targets are prevented. A reflection attack would need to go client→server→client, but path-based addresses do not allow the spoofing of a remote victim's client address.

- Many routing DoS attacks on client systems, such as announcing bogus routes, are prevented as client routes are simply not announced inter-domain.

It is important to note that such client addresses are inherently changeable. If the client moves location, or the inter-domain routing between client and server changes, then the client's address as seen by the server will change. This mechanism therefore requires that transport connections have access to a more stable ID above the IP layer. Proposals such as HIP[15] would provide such a suitable ID, although other simpler forms of ID might also suffice.

Another implication for transport protocols becomes clear if we consider what happens when routing changes in such a way that the original server→client inter-domain path becomes unusable. The server has no way to reach the client until it next receives a packet from the client indicating a new valid path. This is primarily a problem if a connection is idle when a change occurs, and the server then wishes to re-start communication. Periodic "keepalives" are one way to solve this. Alternatively a client can passively monitor the inter-domain route for this server and deduce when it needs to send a keepalive to update the server's knowledge of the its address. To do this a client would need access to the BGP AS Path for the route used to reach the server. For many practical reasons with BGP as it is currently deployed, this is not quite as simple as it might seem, but the general idea is feasible.

Step 3: RPF Checking of Server Addresses

Using path-based client addresses severely restricts source-address spoofing by a client, but it does not restrict spoofing by servers. However, the domain-level symmetry that emerges from using client-based addresses means that packets traveling from server→client follow the reverse client→server inter-domain path. The allows domain boundary routers to perform a reverse-path forwarding (RPF) check on the source address of server→client packets. This check largely prevents a server from spoofing the address of a server in a different domain.

When combined with path-based client-addresses, one effect of this is to make it much harder to launch blind DoS attacks on ongoing communications, such as injecting a TCP Reset into a connection whose existence can be inferred.

Step 4: State Setup Bit

Not all packets are equal. Packets that require the recipient to set up new state are more risky from a DoS point of view than those that don't. It is useful to single these packets out for special handling in a manner that is independent of the upper layer protocol.

Packets that will cause transport communication state to be set up (especially connection setup) should set a new state-setup bit in the IP header. Other packets would leave this bit unset. This serves a number of purposes:

- It provides a generic protocol-independent way to identify packets that need special validation. A server receiving a connection setup request with this bit not set would simply discard the packet.

- Stateful firewalls can validate packets with this bit set before instantiating state. We will discuss what form this validation might take below.

- Packets without this bit set can be safely dropped by stateful firewalls if no matching state is found. This provides a way for firewalls to permit evolution of network protocols without always needing to know the protocol semantics, and provides a way for firewalls to do some degree of transport-independent validation of encrypted traffic such as IPsec-protected connections.

- Server addresses cannot send packets with the state-setup bit set - all routers would drop such packets. This prevents all unsolicited state-holding DoS attacks initiated from server addresses.

- Sites might rate-limit at their outgoing firewall the number of state-setup packets per second sent by some clients.

Step 5: Nonce Exchange and Puzzles

We need mechanisms to validate a client and to add asymmetric costs to communication so as to tilt the balance of control towards the server. Doing this under heavy load on the server itself is hard, so we need to off-load some of this checking. As the validation point needs to be upstream of links that are vulnerable to saturation attacks, this implies the use of external firewalls to validate flows.

The simplest form of validation a firewall could perform would be to validate that the client's address is valid. Although complete client source-address spoofing is not possible with the path-based addresses described above, the client still has some ability to obfuscate the true origin or to launch reflection DoS attacks on clients in the same domain as itself. Thus pushback mechanisms might inflict some collateral damage on clients co-located with an attacker.

One feasible mechanism to do address validation would be a simple nonce-echo. On receiving a state-setup packet, a firewall could bounce the packet back to the sender with a nonce appended. The sender would then echo the nonce-packet back, whereupon the firewall would check the nonce, strip it off, instantiate any appropriate state, and forward the packet on towards the intended destination.

Such a simple validation could be performed by an edge site's outgoing stateful firewall to validate the source address on an outgoing request. This is in the site's own interest because it would prevent the other clients at that site from suffering collateral damage from any pushback that a DoS attack launched by that client might incur.

Going beyond source-validation, we can also add transport protocol independent mechanisms to push cost onto the client to limit the sustained attack rate that a client can maintain. CPU puzzles are one such mechanism; the goal is to make the client perform a CPU-intensive task before connection setup is permitted. The hope is that a normal client will not be excessively delayed in solving the puzzle, but that an attacker (who wants to initiate a great many connections and so needs to solve many puzzles) will be CPU limited, and so his rate of attack will be much lower. An example of such a puzzle would be to provide a random bit-string to the client, and require the client to find a text where the first n bits of the MD5 hash match the bit-string. Such a puzzle is expensive to solve, but cheap to check. There are also puzzles limited by other resources such as memory bandwidth, which might be more appropriate in some cases.

Just as firewalls can validate source addresses, they can also issue puzzles to impose cost on the clients before allowing traffic through. The idea of IP-layer puzzles is not new[8], but our architecture provides the right framework to make puzzle deployment safe and effective. In particular, the state-setup bit would provide an appropriate signal to issue a puzzle, and the addressing architecture makes it hard to use malicious puzzle requests or spoofed responses as a DoS attack in their own right.

Puzzles at the IP/transport level do not solve application-level DoS problems or complexity attacks, but they do at least constrain the rate of incoming incoming connections. The application may well need to do its own DoS prevention; what form this might take depends on the nature of the application itself.

Such puzzles also do not prevent *massively* distributed DoS attacks because the connection setup rate from each attacking zombie is so low as to not be CPU constrained. Our hope is to make it much harder to compromise so many client hosts in the first place.

Step 6: Middlewalls

Firewalls are normally deployed at site borders, with the goal of limiting the types of traffic allowed into or out of a site to protect the hosts inside. However, in the case of DoS, traditional site firewalls may well be too close to the destination to provide sufficient protection.

Pushback mechanisms can provide reactive protection back into the core of the network when an end-system or network link discovers it is under attack. Unfortunately to pushback each source in a large distributed DoS attack is likely to be relatively expensive in terms of network state, and requires each source to be identified and pushed back individually. Such identification may be difficult - it may be hard to tell good traffic from bad if an attack is not too blatant. Non-source-specific pushback is still feasible, but pushes back good and bad traffic alike.

We do not wish to re-invent virtual circuits in the Internet, but the only way to throttle requests through a bottleneck is to have some form of access control upstream of that bottleneck. We envisage very simple special-purpose high-speed firewalls being deployed in the core of the Internet at inter-domain boundaries to serve this purpose. Such *middlewalls* would not normally filter traffic, but a server under stress may issue a middlewall solicitation message to request the assistance of one or more middlewalls. Such a request could be specific - sent along the path back towards a source (another benefit of routing symmetry), or it could be non-specific, flooded in the inter-domain routing information. On receipt of such a request for help, the middlewall would start performing source-validation, and issuing puzzles on behalf of the server.

The use of middlewalls to perform source-validation or issue puzzles opens up the question of how multiple firewalls in a path should interact. We would prefer to avoid adding multiple round-trip times to connection setup, but at this stage this seems simplest. In general though, unlike a site firewall, a middlewall should be transparent unless its help is actively solicited by a server, so the additional delay would not normally be incurred.

The economics of middlewalls are not completely clear, but they seem tractable. In the case of a specific request, the middlewall might charge the end-system for the service. Even in the case of a non-specific request, it might still be in a transit ISP's interest for its middlewall to help out because the middlewall may allow an ISP to avoid forwarding a flood of attack traffic that might disrupt other customers.

Step 7: Multicast

Traditional IP multicast[7] (so called *ASM*) has no wide-area role in a DoS-resistant Internet. The ability for any host anywhere in the Internet to simply start sending and cause routers worldwide to instantiate forwarding state presents an insuperable DoS problem. This is made even worse by the ability of a sender to send to any existing multicast group without the consent of receivers or the other senders.

In contrast, Source-Specific Multicast (SSM [4]) is relatively robust against DoS attacks. With SSM, a receiver joins explicitly to a specific source address to receive a specific multicast group. This means that there is no way for a multicast sender to be malicious without the active participation of the multicast receivers. However, there are two remaining DoS vulnerabilities with SSM:

- Receivers can join many non-existent multicast groups on many potential sources. This can be a state-holding attack on the routers along the path to these non-sources.
- A receiver can join many high-bitrate groups causing high packet loss on network links towards the receiver.

The latter is really a self-DoS. However, even this can be prevented to a reasonable extent if the router at the congested link simply unsubscribes groups that do not appear to be utilizing congestion control.

The former problem is really two separate issues. First, receivers should not be able to join a group where the sender is never going to be active. This can be solved using cryptographically generated addresses (CGA [2][5]) where a router can validate that the group address must have been chosen by the owner of the sender's key. However, CGA might be too heavyweight and a DoS risk in its own right.

Alternatively, a two-pass multicast join mechanism can first validate group liveness (or the sender's intent to send), then on the second pass actually instantiate forwarding state. This would be done by allowing a join-probe to propagate upstream towards the sender without instantiating any state until it reaches the sender's subnet, or until it reaches a router with active multicast join state. The join-probe would accumulate the entire hop-by-hop path, and an affirmative join response travelling the reverse path could be used to instantiate the join state. The same domain-border encryption mechanism used for client addresses could be used to prevent spoofed join responses.

Second, we want to make it hard for a receiver to join so many low-rate multicast groups that the routers can't hold the forwarding state. This is mainly an issue close to the receiver itself, because the backbone already needs to transit most of these groups in the absence of any DoS attack. A partial solution is for routers to have a threshold for the number of groups that can be joined. If this threshold is exceeded, then a join-failure message is propagated downstream for all groups. A site firewall or router close to a likely malicious receiver can then use this to trigger a limiting mechanism that reduces the large number of groups that any single receiver has joined. Receivers that have not joined many groups would be unaffected.

With our addressing architecture, only server addresses can send multicast traffic and only client addresses can receive it. We also note that multicast sender need not be reachable via unicast - this is already possible today using a multicast-only SAFI in BGP4+[3].

With these mechanisms in place, SSM should be rather resistant to DoS attack. This makes it a useful building block for wide-area information services such as DNS which would otherwise be high-profile targets for DoS attack.

4.1 Architectural Summary

The seven steps above, taken together, provide an enhanced Internet Architecture that is much more robust to DoS attack than the current Internet. The changes are largely independent of applications, so should permit the Internet to continue to evolve successfully, and the benefits are many. Worms cannot spread rapidly. Reflection DoS attacks are almost completely eliminated, and where they remain feasible, they are only against local targets. Clients are almost completely protected from direct attack and compromise[5], and they are protected from non-local DoS attacks except by servers with whom they have chosen to communicate. Servers are protected against DoS attack from other servers - as clients now become hard to directly compromise, this provides not insignificant protection in its own right. The routing architecture means that complete source-address spoofing by clients or servers is not possible, and this in turn means that pushback mechanisms become easier to deploy against known attackers. Client address validation using nonces further reduces the chance of collateral damage from such pushback. The use of middlewalls to validate CPU-puzzles can make the remaining DoS attacks impossible to sustain except through overwhelming numbers of attacking zombie hosts. Even then, if the attacking hosts are readily identified, it should be feasible for middlewalls to do client-address validation, and then use pushback to shut them down. We note that with a huge attack this may entail too much filter state in the network, but this architecture would permit the deployment of specialist high-speed filter gateways that could do this filtering.

The remaining problems are primarily those in which huge numbers of attacking client zombies cannot be distinguished from legitimate clients. In such cases we have succeeded in our design goal which is to force the attacker to be indistinguishable from a flash crowd. In fact we have gone rather further than this by removing a number of key vectors for infection that can currently be used to create large DDoS zombie armies in the first place.

To get this far though, we have sacrificed a significant amount of flexibility that was permitted in the original Internet architecture. We believe that some such medicine is necessary, and the debate should not be *whether* we change the architecture, but rather *how* it should be changed?

In any event, it is important to examine what we have given up in such an architecture, and we discuss this next.

5. LOSS OF SYMMETRY

The most significant effect of our architectural changes is the loss of symmetry between clients and servers. For the most part, this is beneficial to both clients and servers, but a number of applications require special consideration.

5.1 Peer-to-Peer Applications

Peer-to-peer applications are possible because the peers can be both clients and servers, forming a mesh of connectivity that can support all manner of functionality. Some people might make value judgements about the importance of such applications. The truth is no-one knows how Internet applications will need to evolve in future, so it is important to examine options for supporting all applications.

There is no technical reason why hosts that need both client and server functionality with global scope should not have both client and server global-scope addresses. However, if most hosts have both client and server addresses, then many of the architectural benefits are lost, so it behooves us to examine alternative ways to support such applications.

Peer-to-peer applications all need some form of rendezvous to discover the addresses of peers before any direct peer-to-peer connections can be established. For scalability reasons the solution of relaying all communication through the rendezvous agent is not viable. If we assume the peer-to-peer mechanism provides suffi-

[5]They are however still vulnerable to viruses and trojan horses, which will require alternative means of protection.

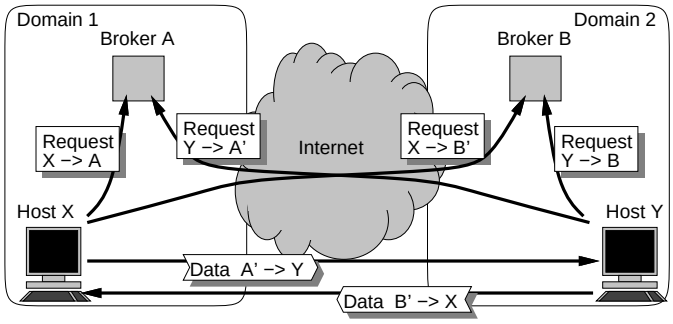

FIGURE 2—Using a broker to setup a peer-to-peer session

cient bootstrap capability to allow indirect signaling between two peers, then we can use this to bootstrap a client-to-client connection. From the point of view of preventing DoS, the key property of such connections is that they are simultaneously set up from both ends.

Our forwarding rules do not permit packets with client addresses as both source and destination. We could relax these rules, but there are a number of ways we can achieve client-to-client connections without doing so. We shall present one such mechanism as a proof of concept.

To be valid, all packets require a server address as source or destination. We can use a broker (Fig. 2) to bootstrap this:

1. Each client connects to a local broker on its local server address[6], sends its HIP ID[7], and obtains a global address from the broker's address pool.

2. Over the peer-to-peer signaling channel, each client informs the other client of its own HIP ID and the pool address of its broker.

3. Each client sends the remote client's HIP ID to the local broker.

4. Each client connects to the remote broker on its pool address and provides both its own HIP ID and that of the remote client.

5. Each broker now returns the remote client's address to the local client.

At this point the peers now know enough to communicate. Data packets are sent using the pool server address as the source, and the remote client address as the destination.

This should be acceptable to middlewalls, as state is setup in the initial request to the broker. However, the traffic does not appear to be bi-directional to an observer, because neither addresses or paths are symmetric. In addition, if the original inter-domain route fails, the clients will need to send via the broker to re-establish communication.

It might seem that this re-introduces DoS vulnerabilities that we tried hard to remove, but this is mostly not so. As packets reach each client sourced from a server address (the broker's pool address) the state-setup bit cannot be set in them, so this mechanism cannot set up any incoming connection state. At the application level, it is up to both parties to agree to communicate - if either does not wish to communicate, then no DoS attack can be performed.

An alternative mechanism might be for one of the clients to obtain a temporary use "server" address on a short-term lease from its broker. As above, the broker can be used to discover the remote client's address, so only a connection setup from the remote client would be accepted on the leased address. The lease would need to be renewed frequently, or it would cease to be locally routed, giving fail-safe functionality in the event of a DoS attack.

Unfortunately, whatever the addressing solution, one vulnerability is re-introduced; it appears to be possible for worms to spread rapidly through peer-to-peer networks. This seems to be an inherent problem with the peer-to-peer model rather than a deficiency in our architecture. If people ever become serious about preventing the spread of worms, then peer-to-peer architectures may be an inevitable casualty.

5.2 Internet Telephony

Internet telephony has similar issues to peer-to-peer applications, but we must consider the signaling pathway and the media-data pathway separately.

The media-data pathway is similar to the peer-to-peer case; once call-setup signaling has completed, both parties wish to setup a direct connection carrying, for example, RTP[18], over DCCP[11]. The same broker mechanism used for peer-to-peer can be used for the media data pathway, allowing audiovisual data to flow end-to-end with minimal delay.

The signaling channel is somewhat different. SIP[17] generally assumes that a SIP terminal device can accept incoming connections to make the "phone" ring. With our architecture, no client-only host can do this. However, SIP also makes extensive use of proxies for call routing, personal mobility, device mobility, and so on. Although most current devices don't take advantage of it, SIP largely separates the process of sending and responding to a SIP message from the process of setting up the transport level connections to carry those messages. Thus a SIP terminal can simply register with a proxy and leave the signaling transport connection open. Incoming calls will come into this proxy, and the existing connection can carry them to the client host.

In this scenario, the SIP proxy is an obvious DoS target, but this is no different from its current role, and our architecture helps with its defense. We note that SIP proxies are themselves one of the class of devices that will need both client and server addresses.

6. TRANSITION AND APPLICATIONS

Any significant change to the Internet architecture cannot be successful unless there is both an incremental transition plan and economic incentive for change. In our case we presuppose an IPv6 Internet to allow sufficient space for the client-address path-encoding. There are two possible transition plans - deploy IPv6 and transition from there, or skip the regular IPv6 addressing stage. In practice it does not matter greatly. The only difference is that in the former there are three classes of unicast address - *client*, *server*, and *unrestricted*. Unrestricted addresses obviously allow an attacker some freedom, but in the face of attack, a server or firewall could preferentially drop service to unrestricted addresses.

IPv6 itself faces significant hurdles to deployment, despite now being supported on most current operating systems and routers. For the most part this appears to be because there are no short-term benefits from deployment and significant short-term costs. A DoS-resistant architecture utilizing IPv6 can use the same routers and routing protocol implementations as vanilla IPv6. End-systems would require HIP or a similar persistent identification mechanism

[6]How it learns this address is not critical - we can assume it learns this in a similar manner to how it learns about DNS servers and default routers.

[7]Other forms of persistent ID might be possible in place of a HIP ID

to be deployed, but such a mechanism is likely to be needed in any event for mobile systems. The additional benefits of a DoS-resistant IPv6 architecture should be much greater in terms of robustness, uptime, and the associated lower network management costs than the current IPv4 Internet.

6.1 DNS

No discussion of DoS is complete without mentioning the Domain Name System. DNS is central to the operation of the Internet, but it is also a centralized architecture which provides a central point of attack - deny service to the root nameservers or the GTLD servers for any significant duration, and the DoS effects will be serious. Current attempts to replicate the root nameservers using BGP anycast[1] are a step in the right direction, but even so the root nameservers would be vulnerable to a sufficiently large attack.

A viable solution might use IP multicast. The DNS zone files for the root and top-level domains contain perhaps tens of millions of entries - a significant amount, but a single PC is capable of holding the entire database. If signed zone files were continuously multicast from the master root nameserver and, as appropriate from the zone masters for the top-level domains, every well-connected site around the world could simply cache the entries. The master servers need not be unicast-reachable, so would be protected from DoS attack. This would not replace the request/response mechanism, but it would become the principal way most nameservers would receive the top part of the DNS tree. Nameservers which know they have incomplete data (due to packet loss or server restart) could still query directly as they do today, but the consequences of a successful DoS attack on the request/response servers would be minimized.

We also note that the inability to spoof server addresses largely prevents DNS cache poisoning DoS attacks.

With current DNS deployments, the normal mode of operation is for clients to query a client-side DNS server that relays requests to remote servers in search of an answer. Clearly such client-side DNS servers need both a client address and a server address. We have no problem with this for relays that need such functionality, but we note that often client-side DNS servers do not need their server address to be globally-reachable. In addition, DNS deployment guidelines recommend disabling relaying on authoritative servers, so these will not need a globally routable client address.

6.2 SMTP, NNTP and SIP

A number of protocols explicitly build in the capability to relay at the application level. The addressing requirements of mail transfer agents, SIP proxies, and NNTP relays are similar to those of DNS servers. Sometimes they need both globally routable client and server addresses; in many cases either the client or server address could be local. Obviously the potential for attack is reduced if as many relays as possible do not have both client and server addresses that are globally routable. To avoid ambiguity, we recommend that non-globally-*routable* server addresses are still globally *unique*.

7. RELATED WORK

This work builds upon the pushback mechanism[13], but we believe it provides a framework in which pushback can be deployed more effectively and safely.

A number of other mechanisms for defending against DoS attacks have been proposed. The use of CPU puzzles at the TCP/IP layer has been suggested by several researchers [8]. We believe that puzzles have a definite role to play, and our architecture provides a clean framework for their deployment. However, they are probably too difficult to deploy without such a framework, because they potentially open up new DoS possibilities.

The use of capabilities to give explicit permission to send was suggested in [19]. Such a mechanism is viable, but is somewhat heavyweight to be the first line of defense, especially as a defense against attacks on clients. If necessary though, capabilities could be deployed within our framework, to aid in the defense of servers, as an alternative to CPU puzzles.

The Internet Indirection Infrastructure (i3) has also been suggested as a mechanism to aid in the defense of servers from DoS attack[12]. i3 does this by obfuscating the true address of a server, and ensuring that access is routed through large numbers of "triggers" which are disposable, and hence can be changed to shut down specific attackers or reduce overall server load. This is in fact a much more radical departure from the current Internet architecture than our proposal, and it is probably too early to tell if non-technical issues such as trust between multiple competing i3 providers can be solved.

There are also a number of commercial products that advertise protection against DoS attack, but publicly available information is somewhat sketchy on how these actually work.

8. CONCLUSIONS

We have outlined a set of changes to the Internet architecture, including the explicit separation of the IP address space into client and server addresses, along with associated rules restricting how those addresses can be used. These changes significantly limit the modes of interaction between Internet systems, in such a way that the vast majority of the desirable interactions are allowed, and a large number of undesirable interactions are disallowed. Taken together, these changes significantly improve the ability to defend against DoS attack. They also increase the difficulty involved in building large DDoS attack networks, and the hope is that the remaining ways to compromise systems are more easily detected or prevented.

We do not expect these changes to be popular at first, as people have become rather used to the flexibility inherent in the current architecture. However, we do believe that the problems are severe enough to merit such solutions. Without changes on this scale, the Internet cannot fulfil our aspirations for digital convergence, because it simply will not be robust enough to use for many serious applications. As anyone responsible for running a high-profile network understands, there are no easy solutions. It is time to take a hard look at the architecture itself and question what we got right and what we cannot live with anymore.

9. ACKNOWLEDGMENTS

We are grateful to Vern Paxson at ICSI and the members of the Networks Research Group at UCL for valuable feedback. The idea that a server should not be able to make outgoing connections to protect against worms first came from Nick Weaver. The state-setup bit was suggested by Dave Clark during an End-to-end Research Group meeting.

10. REFERENCES

[1] APNIC. Root server trial FAQ. http://www.apnic.net/info/faq/rootserver-faq.html.

[2] T. Aura. Cryptographically Generated Addresses (CGA). draft-ietf-send-cga-05.txt, work-in-progress, IETF, Feb. 2004.

[3] T. Bates, Y. Rekhter, R. Chandra, and D. Katz. Multiprotocol extensions for BGP-4. RFC 2858, June 2000.

[4] S. Bhattacharyya (editor). An overview of source-specific multicast (SSM). RFC 3569, IETF, July 2003.

[5] C. Castelluccia and G. Montenegro. Securing group management in IPv6 with CGA (Cryptographically Generated Addresses). presentation to the IRTF GSEC Research Group, February, 2002.

[6] CERT Advisory CA-1998-01. Smurf IP denial-of-service attacks, Jan. 1998. http://www.cert.org/advisories/CA-1998-01.html.

[7] S. Deering. Host extensions for IP multicasting. RFC 1112, IETF, Aug. 1989.

[8] W. Feng. The case for TCP/IP puzzles. In *Proc. ACM SIGCOMM workshop on Future Directions in Network Architecture*, pages 322–327. ACM Press, 2003.

[9] IAB, M. Handley (editor). Internet denial of service considerations. draft-iab-dos-00.txt, work-in-progress, IETF, Jan. 2004.

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

[11] E. Kohler, M. Handley, and S. Floyd. Datagram congestion control protocol (dccp). draft-ietf-dccp-spec-06.txt, work-in-progress, IETF, Feb. 2004.

[12] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. Taming IP packet flooding attacks. In *Proc. ACM SIGCOMM 2nd Workshop on Hot Topics in Networks*, Nov. 2003.

[13] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM Computer Communication Review*, 32(3), July 2002.

[14] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code, Apr. 2003.

[15] R. Moskowitz, P. Nikander, and P. Jokela. Host Identity Protocol. draft-moskowitz-hip-09.txt, work-in-progress, IETF, Feb. 2004.

[16] V. Paxson. An analysis of using reflectors for distributed denial-of-service attacks. In *ACM Computer Communication Review 31(3)*, July 2001.

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

[18] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, IETF, July 2003.

[19] D. W. Tom Anderson, Timothy Roscoe. Preventing internet denial-of-service with capabilities. In *Proc. ACM SIGCOMM 2nd Workshop on Hot Topics in Networks*, Nov. 2003.

[20] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. Large scale malicious code: A research agenda. Darpa sponsored report. http://www.cs.berkeley.edu/~nweaver/ /large_scale_malicious_code.pdf.