DAM: a DoS Attack Mitigation Infrastructure [*]

Byung-Gon Chun, Rodrigo Fonseca and Puneet Mehra
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
{bgchun,rfonseca,pmehra}@eecs.berkeley.edu

ABSTRACT
Denial-of-Service (DoS) attacks represent a serious and growing threat to the firms utilizing the Internet. Defenses against DoS that rely upon identification and isolation of the attack stream are difficult due to the distributed nature of most attacks and source IP spoofing. Stopping an attacking host often requires human intervention and cooperation across administrative boundaries, resulting in costly downtime for victim sites. An alternative safeguard against DoS attacks is the use of load-balancing techniques to simply diffuse the attack load across replicated content servers. However, current Content Distribution Networks (CDNs) do not provide adequate support for this protection mechanism since they provide direct IP access to the servers and generally rely on vulnerable redirection mechanisms. This provides attackers with the opportunity to either directly attack the server, or the redirection mechanism itself. In this paper we present DAM, a novel infrastructure for DoS Attack Mitigation, designed to address these deficiencies in current CDNs. Our basic approach is to use an additional layer of indirection to prevent direct IP access to the content servers, and to use a robust overlay protocol, Chord, as the basis for our redirection mechanism. The robust load-balancing mechanism evenly distributes service requests, including potential attacks, across servers, while the additional level of indirection protects content servers from network-saturating flooding attacks. Various simulations using the NS-2 simulator demonstrate the efficacy of our approach.

1. INTRODUCTION
Denial of Service (DoS) attacks arguably represent the most serious security threat facing the Internet community today. A recent empirical study of Internet DoS attacks observed 12,805 attacks on over 5,000 distinct hosts over a three-week period [1]. Websites are a prime candidate for such attacks, and a series of successful DoS attacks on such high-profile targets as eBay, Yahoo, CNN.com, and buy.com in February 2000 prompted attention from the Attorney General and an official FBI investigation into the source of the attacks [2]. These attacks have had a significant financial impact on companies, with the firms losing an estimated $1.2 billion in lost revenues during the short attack period noted above [3]. Even more alarming than these episodes is the possibility that DoS attacks may actually be on the rise [4], representing a growing threat to business utilizing the Internet.

Defenses against DoS which rely upon identification and isolation of an attacking stream have several drawbacks. The identification of an attacking stream is often difficult since attackers usually forge source IP addresses (IP spoofing) or use machines in many different domains to conduct a coordinated attack. It may also be difficult to differentiate a DoS attack from a flash crowd [5], potentially resulting in denying access to legitimate users. Furthermore even after sucessful identification of attacking hosts, it may be difficult to get cooperation from different network providers to contain the attack. In any case, such cooperation usually involves human intervention and consequently takes on the order of several hours, resulting in costly downtime for the victim site.

In this work we are mainly concerned with DoS flooding attacks. These attacks have two distinct flavors. First, they may involve overwhelming the network link capacity of the content server via brute force packet-flooding, which is representative of the previously mentioned attacks on websites. Secondly, the attacks may involve saturating a server's processing ability, i.e. kernel data structures or CPU cycles, through seemingly-legitimate HTTP requests.

Given the threat model, and the fact that many content sites already use replicated servers and geographic mirroring of content for robustness and performance enhancements, an alternative defense against DoS attacks is to evenly distribute the offered attack load over these replicated content servers. Current Content Delivery Networks (CDNs), which provide geographic mirroring and access to replicated servers, do not necessarily provide enough protection from DoS attacks, for two reasons. First, unless the CDN hosts all of the content on its own servers, it usually provides a redirection mechanism for clients to find a "good" content server with the requested data. In such cases an attacker,

[*]DAM is a joint project of Byung-Gon Chun and Puneet Mehra in cs261 and Byung-Gon Chun and Rodrigo Fonseca in cs262A.

after obtaining the server IP, can directly flood packets to this IP address, bypassing any redirection provided by the CDN. A second disadvantage of CDNs is that the redirection schemes generally rely upon DNS, or load-balancing hardware appliances, which represent a single point of attack.

In this paper we present DAM, a novel infrastructure for DoS Attack Mitigation. Our proposed infrastructure provides robust load-balancing to protect against DoS attacks, and addresses the deficiencies of the CDN systems outlined above. First, our infrastructure uses an additional level of indirection to deny clients direct IP access to content servers. Since all communication must go through a node in our infrastructre, flooding attacks seeking to saturate network resources only affect the path to the infrastructure node and not the content server. Secondly, our infrastructure uses a robust overlay protocol, Chord [6], to provide a redirection mechanism which is not vulnerable to DoS attacks.

A basic premise of our approach is that DoS attacks may be viewed as a resource competition between attackers and content providers; the side with greater resources wins the match. As such, we envision DAM as a large, shared infrastructure which allows different content service providers to pool resources to obtain greater protection from DoS attacks than otherwise possible due to their individual economic constraints. Specifically, having a large number of nodes in our infrastructure is more economically and technically feasible than having an equal number of content servers. Since our infrastructure does not actually store any content, it has more modest computational and storage requirements than the servers hosting the content, and hence each node in the infrastructure may support multiple content providers simultaneously. Simply having a large number of replicated servers, aside from being more costly, also incurs additional complexity to ensure that all replicas have the latest content. DAM avoids this complexity since it does not store any content. We demonstrate the efficacy of our approach through a variety of different simulations involving the NS-2 network simulator [7].

The rest of the paper is organized as follows. In Section 2 we discuss related work. We provide a detailed discussion of the architecture of DAM in Section 3. In Section 4 we describe the simulations, and their results, which validate our proposed architecture. Finally, in Section 5 we provide some directions for future work and conclude this paper.

2. RELATED WORK

There has been much work done on detecting attackers and isolating attack streams, including filtering, traceback and pushback [8, 9]. To avoid the difficulties of detecting and isolating DoS attacks, our approach focuses on evenly distributing the offered load over server replicas in order to mitigate DoS attacks. Identification and punishment of attacking streams is orthogonal to our approach, and can in fact be combined with our infrastructure. Furthermore, our infrastructure may provide useful information about the source of attacks if proximity-based load-balancing is used, thus providing more accurate detection. Y. Chen et al. investigated the resilience of object location services, such as Tapestry to DoS attacks [10]. Their work focused on evaluating the object location services themselves, while our work proposes a new architecture to protect general web services. A. Keromytis, V. Misra, and D. Rubenstein proposed a Secure Overlay Service (SOS) to protect critical emergency services from DoS attack [11]. The goal of the SOS is to allow secure communication between a small number of preapproved sources and a particular destination. The SOS protects the destination by allowing only secret servlets to contact the destination. To provide protection, every packet must traverse the secure overlay, resulting in a substantial increase in communication latency. Our work differs because we must allow communication between a large number of untrusted clients and the server with the desired content. Furthermore, to accomodate the needs of the target application, namely web services, our infrastructure must provide security with minor performance impact. Hence, after the service name resolution step, all communication is done with only one overlay hop, reducing the communication latency.

3. DAM ARCHITECTURE

3.1 Goals and Assumptions

The goal of DAM is to provide protection to web services from the DoS flooding attacks descibed in Section 1. DAM aims to protect content servers from many, untrusted clients. It is a lightweight mechanism designed to support a large volume of web requests and responses with minimal overhead. As discussed in Section 1, since DAM is shared by multiple services, it is possible to dedicate more resources to this infrastructure than that possessed by a single service provider. Consequently, DAM requires greater effort from attackers to launch a successful attack against a service provider.

We now state the assumptions which have influenced the design of our infrastructure. We assume that infrastructure operations and node IP addresses are public information. Hence, we cannot rely on secrecy of algorithms or infrastructure node addresses to provide any protection. We assume a trustworthy infrastructure. Specifically, we do not consider the problem of malicious infrastructure nodes. This issue has already been addressed in other work [12]. Finally, we assume that content servers are only accessible through DAM.

3.2 Architecture Overview

We now provide an overview of DAM, and the key differences between DAM and traditional web-browsing using a CDN. Traditional access of web servers consists of two steps. First, a client interested in accessing a particular web service such as www.cnn.com must perform a name service resolution to obtain an IP address for that service. Specifically, a client does a DNS lookup for the service name and obtains a server IP address. After the resolution step is complete, the client makes request(s) directly to the IP address to access the desired content. As shown in Figure 1, these two steps are still present in DAM, though each step has been modified to facilitate the desired security goals of our infrastructure.

During the resolution step, a client using DAM resolves a service name to an IP address, using our infrastructure instead of traditional DNS. However, this IP address is the address of a *gate* node, which serves as a level of indirection for the content server. During the second step of the browsing process, a client issues requests and receives responses

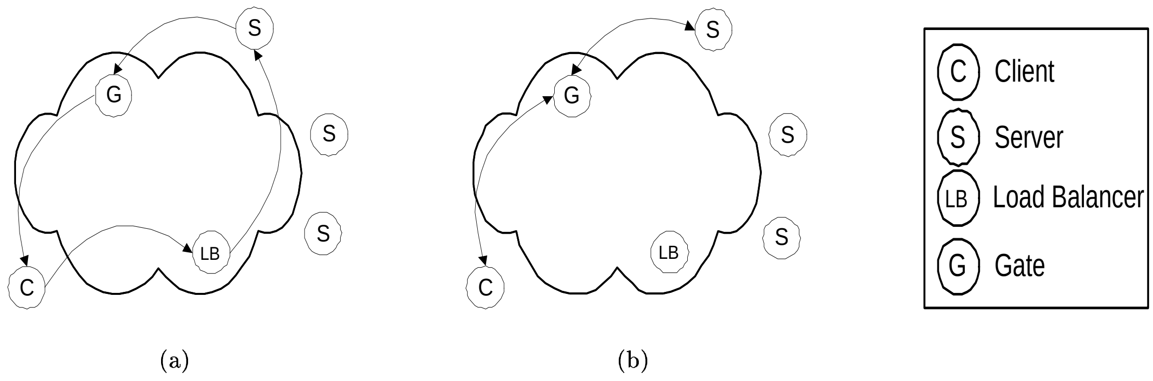

Figure 1: Web service access (a) The client resolves a host name to a gate IP address. In the resolution, the infrastructure determines a server to access through the gate. (b) The client communicates with the chosen server through the gate.

from the gate instead of directly communicating with the server. We now provide more details about the resolution and communication steps.

The client sends its initial name resolution request to an *adapter* node, which serves as the interface to our infrastructure. This request is then transmitted to an authoritative *load balancer* node for that service using a distributed hash table (DHT) for routing the message from the adapter to the load balancer. We choose to use Chord for its simplicity and robustness properties, but DAM can use any recently proposed DHT [13],[14],[15] as the underlying communication mechanism. The load balancer selects a particular content server based on its load-balancing policy, which considers network proximity and server load in its selection decision. The resolution request is then forwarded to the chosen server. The server is responsible for selecting a gate from among multiple gates, based on proximity and load information obtained from these gates. Finally, the gate receives the name resolution request, and sends a resolution reply with its IP address to the client. At this point, the client communicates with the chosen server through the gate, which transparently performs network address translation (NAT) in forwarding packets between the client and server.

In essence, DAM performs load balancing across servers to evenly distribute HTTP requests, while a server distributes these HTTP requests across different gates to handle the possibility of malicious clients launching a flooding attack against a given gate. Furthermore, only the dedicated gate(s) can communicate with a given content server, and each server selects a random subset of nodes in the infrastructure for its dedicated gates.

We now discuss a few important details regarding client access process. Each resolution reply has a Time-To-Live (TTL) value. A client can initiate connections to the gate for a given service until the resolution mapping the service name to the gate expires. The infrastructure uses this mapping cache at the client to avoid resolution latency for every HTTP request. After the resolution step is complete, the client communicates with the server through only one overlay hop. If the gate used for communication is appropriately selected based on proximity to the client, the additional latency of using this indirection point will be minimal.

There are several robustness and scalability issues in our infrastructure which must be addressed. To avoid a single point of attack or failure in the architecture, DAM maintains replicated load balancers. In addition, each server is aware of multiple random gates in the infrastructure. If gates are attacked, a server can simply choose other gates, which are not likely to be under attack since they are randomly chosen, for communicating with clients. DAM assigns the task of server selection to load balancers and the task of gate selection to servers to minimize the amount of state which must be maintained at either load balancer or server nodes. Furthermore, packet filters may be employed at the server to only allow communication with the dedicated gates.

3.3 Adapter

As previously mentioned, an adapter node serves as the interface between clients and our infrastructure. Clients send name resolution requests to a local adapter in the infrastructure via IP on a well known port. The local adapter hashes the service name (e.g., www.cnn.com) to an identifier in Chord with the SHA-1 hash function. The adapter then forwards the name resolution requests to the authoritative load balancer for that service. This load balancer is simply the node that is responsible for the identifer in the Chord identifer space.

3.4 Load balancer

A load balancer is responsible for choosing a server for a name resolution request. It uses a load balancing policy, utilizing server proximity to the client and server load, to make this decision. After selecting a server, it forwards the name resolution request to the chosen server with a particular TTL. This TTL value is used by the client name cache, as previously discussed.

DAM uses landmarks to decide the network proximity between two hosts. We choose the landmark scheme since it has been well explored in overlay network research [16]. Another alternative method of deciding proximity is to maintain the AS-level map of the Internet to determine distance between two hosts. We map each host to a coordinate in the Euclidean geometric space. The round trip time to a landmark determines the magnitude for a given axis in this space. The distance between two hosts is calculated by the cosine similarity of two network vectors defined in the equation (1) [17]. When the distance is smaller, the two hosts are more proximate in the network.

$$Distance(\vec{h1}, \vec{h2}) = 1 - \frac{\vec{h1} \bullet \vec{h2}}{|\vec{h1}| \times |\vec{h2}|} \tag{1}$$

where $\vec{h1}$ is a network vector of host 1 and $\vec{h2}$ is a network vector of host 2.

For server load information, we currently use the utilization of the server during an update interval. Other metrics may include average queue length in the server and the number of active connections. When a site is composed of servers in a cluster, we assume that the site exposes the aggregated load information and the load balancing switch IP address to our load balancer. This hierarchy decreases the amount of load information to update.

We propose a novel load balancing policy – the dynamic proximity load policy. This policy tries to exploit proximity and avoid overloaded servers. It tries to achieve the best of the policy based on only proximity and the policy based on only load. It also overcomes the inflexibility of static proximity load policy. In the static proximity load policy, the load balancer chooses k servers among m servers ($k \leqq m$) with proximity and chooses one server among k servers with the smallest load. If the client population is skewed in the network, static policy will assign clients to heavily-loaded servers, which may result in poor performance, especially when the closest server is under attack. The dynamic proximity load balancing policy exploits proximity information, while avoiding over-loaded servers. The pseudocode for this algorithm is presented below:

```
Algorithm Dynamic Proximity Load Algorithm
1.  SortedServerList = List of servers sorted by distance to
    the client in ascending order
2.  for each server S ∈ SortedServerList
3.          if satisfy_load_condition(S)
4.             then return S

6.  // no server matches the load condition
7.  return least_loaded_server()
```

In the algorithm, the satisfy_load_condition function checks whether the load of a server is below a certain threshold value. The least_loaded_server function finds a server with the smallest load. Further discussion of different load balancing polices may be found in [18].

The load balancer uses secure authentication in server registration and de-registration to protect from DoS attacks involving fake servers. Replication of load balancers is used to protect against flooding directed at the load balancers themselves. The authoritative load balancer is the primary load balancer. Each primary load balancer pushes the collected load information to secondary load balancers periodically. If attackers shut down the primary load balancer, the Chord protocol handles the failure through its re-stabilizing procedure. The identifier space is delegated to the new closest successor node, which will become the new primary load balancer.

3.5 Server add-on

When the server receives the resolution request, the server selects a gate to allow communication with the client. Since the server knows exactly which gate can forward packets, it can set up packet filtering for additional defense. Currently the server uses only proximity information to select a gate. A gate selection policy based on bandwdith usage and gate processor load is an area of further investigation.

The server also sends periodic load updates to the authoritative load balancer. Currently this update interval is fixed, and investigating an adaptive scheme is a direction for future work. Such an adaptive technique would send more frequent load updates if the load was changing often during the interval, and would send updates less frequently if the amount of server load was relatively steady.

3.6 Gate

A gate node perform network address translation (NAT) to forward packets between clients and servers. When the gate receives the resolution request, it puts an entry into its *connection ready* table with the timeout of the TTL given in the resolution message. If the client issues a connection request within the TTL, the gate sets up a mapping in *NAT* table in both directions, i.e. from the client to the server and from the server to the client. Each entry in the NAT table maintains the client and server IP addresses, port numbers and associated timeout value. Whenever a packet traverses the gate, the matching entry is refreshed. If there is no valid mapping for the packet in the table, the gate may send a RST packet to the sender, or silently drop the packet, based on the desired semantics. When there is no packet flow for a given connection for the timeout duration, a garbage collector deletes the NAT entry to reclaim the memory space. The garbage collection can be implemented efficiently with the second-chance clock algorithm used in operating systems.

3.7 What does DAM provide?

In this section, we discuss the protection DAM provides for different forms of DoS attacks. For network link saturation attacks, DAM protects the network links to servers. In contrast, in current CDNs attackers may directly attack servers after the resolution step. In DAM, attackers can only flood gate nodes, and hence servers are not affected by such attacks. The flooding packets will simply be dropped at gates since there will not be a valid NAT entry for these packets.

Attacks seeking to saturate server processing capabilities are difficult to mount in DAM, since DAM spreads HTTP sessions to different content servers based on proximity and load, thus denying attackers the ability to target a particular content server. Specifically, every communication with a content server must go through a gate for that server. To get a valid mapping at a gate, an attacker must perform the resolution step. Since attackers cannot control the server selected by load balancer, it is difficult to focus the attack volume on a given content server. Hence, with DAM, attackers need more resources to attack a service since they cannot mount a sequential attack that overwhelms content servers on an individual basis. Instead, attackers must have enough resources to issue HTTP requests overwhelm *all* content servers, or launch a flooding attack which overwhelms *all* gates that are in use by any content server for the victim service. Thus, with DAM, adding an additional content server for a given service increases the protection level of every other server, since attack volume is spread across all servers. Although the current CDNs increase scalability for client requests, they do not increase the protection level for other content servers in a similar manner to DAM.

Finally, using the DHT makes DAM highly resistant to DoS attacks. Since the responsibility of resolutions of names is distributed to geographically distributed nodes in the network, attackers need to shut down the whole infrastructure to shut down the whole name space. In addition, the replication of load balancers provides additional protection against DoS attacks that target a portion of the namespace, i.e. the namespace for a given service.

4. SIMULATIONS

4.1 Simulation Framework

We have used the NS-2 network simulator to validate our proposed infrastructure. Our simulations include all of the salient aspects of the problem we are investigating, including web clients, servers, attackers, and the different components of our infrastructure discussed in Section 3. We test the performance of our architecture against a competing CDN approach, which utilizes load-balancing among content servers, but provides clients with a direct IP connection to a "good" server containing the desired content.

We have tested scenarios involving 200, 400 and 1000 node transit-stub network topolgies generated by the GT-ITM [19] library. The links in the topologies were assigned the following bandwidths: 100Mbps for intra-stub domain links, 1.5Mbps for stub-transit links, and 45Mbps for intra-transit domain links. These bandwidth values have been previously used in other simulation studies [10],[20]. In addition, the links in the topologies were assigned the following delays, as recommended in [16]: 1ms for intra-stub domain links, 10ms for stub-transit links, and 100ms for intra-transit domain links. Landmarks are randomly placed in the network. DAM nodes, servers, clients, and attackers are placed randomly in stub domains.

We ran each simulation for 300 seconds. For each data point we averaged simulation results over ten different client traffic generation scenarios. We now discuss the various elements of our simulations in more detail.

4.1.1 Client Model

We have tested two types of client workloads – a naive client and a more sophisticated HTTP client. In both workloads, if a client cannot find a valid mapping from a service name to an IP address in the name cache, it issues a resolution request to DAM. The current TTL of a cache entry is 30 seconds. A naive client requests an object and waits for a response. After getting a response it can issue another request. The maximum request sending rate allowed is 2 requests/second. The size of the server responses are uniformly distributed in $[1KB, 50KB]$.

Component	Shape (α)	Scale (k)
Page Size	1.1	1k
Embedded Obj. Size	1.1	10k
Num. Embedded Objs.	2.43	1.0
OFF Time (s.)	1.5	1.0

Table 1: Parameters for the distributions used in the HTTP Client model

The HTTP client tries to more accurately model the behavior of human clients. In particular, we attempt to generate traffic that is bursty and self-similar, presenting high variability over different time-scales. Our model is based on the SURGE web workload generator [21], which generates HTTP traffic matching empirical observations. We use a simplified model, in which objects are represented simply by their size, since we do not model caching at any point.

One important characteristic that we preserve is the concept of user equivalents, and that of an ON-OFF process for request generation. Each client alternates between two phases, the ON and OFF periods, both with durations following heavy-tailed distributions. The client operates according to HTTP/1.1, using persistent and pipelined connections [22]. During the ON period, the client issues several requests. The first request is considered to be an html page, which references a number of embedded files. The number of embedded files per page, as well as the sizes of the files returned follow Pareto distributions. The client waits for the response to the first requested object to arrive, and then makes a series of requests for embedded files, using the pipelined connection. After all of the responses have been received, the client starts an OFF period, which also has the duration governed by a pareto distribution. The combined effect of the different ON-OFF client processes creates a bursty and self-similar traffic pattern [23]. The parameters used for the distributions are shown in Table 1. The Pareto distribution, with the probability density function given by (2), is a heavy-tailed distribution, which presents high variability and a non-negligible probability for high values.

$$p(x) = \alpha k^{\alpha} x^{-(\alpha+1)} \tag{2}$$

4.1.2 Server Model

We use a simple server model involving round robin scheduling of client requests. The server maintains a single queue that receives HTTP requests from the network. It processes each request for a 1ms time slice. After processing for a time slice, if the processed amount is greater than or equal to a *chunk size*, the server sends the processed amount of bytes to the client. The chunk size used in our simulations is 8KB. If the entire request has not yet been serviced, then the server places the request at the end of the queue noting the amount of service that the request has received. This scheduling attempts to model the queuing delay at a real web server. In addition, the server is able to process a fixed number of requests per second. In our simulations this processing rate was set to 100KB/s.

4.1.3 Attacker Model

We used two different types of attackers – UDP attacker and HTTP attacker. UDP attackers send 1KB UDP packets with a constant sending rate. HTTP attackers request an object of 10KB with a constant request rate. The request rate is varied to control the overall attack volume. The attacks are coordinated with a master control unit, similar to real attacks such as Trinoo [24]. Given the number of target servers to attack, attackers are partitioned into groups randomly, and each group attacks one target server.

4.1.4 Infrastructure Configuration

DAM components reside on top of Chord. We use 8 landmarks for determining client and server locations as previously outlined in Section 3.4. The resolution TTL used in our simluations is 30 seconds. Each server sends updated load information to their respective authoritative load every 30 seconds, and the gate load update interval is also 30 seconds.

4.2 Performance Metrics

We use the following metrics to evaluate the performance of DAM in comparison to existing CDN architectures.

Aggregate throughput: This is the total bytes, per second, of HTTP server responses received by clients in the network.

Inverse power: This is defined by average response time divided by aggregate throughput. Inverse power is a metric that captures two important aspects of HTTP responses: the throughput of the response and the delay in the response.

Average resolution time: This is the average time taken for the response to a resolution request issued by a client. For the CDN, this value is simply the round trip time to the authoritiative DNS server. In DAM, this value includes the time needed for chord routing to the load-balancer, the time from the server to the gate, and finally the time taken for the response from the gate to the client.

Cumulative count: This is the cumulative count of packets received at all clients based on their average response time. We do not use the cumulative distribution function (CDF) due to the large disparity in the number of responses handled by DAM in comparison to CDNs under high attack volume.

4.3 Simulation Results

We now present the results for simulations involving a 400 node transit stub topology with 3 content servers, 40 HTTP clients, and 20 attackers. Figure 2 shows the aggregate throughput, inverse power, and average resolution time for DAM in comparison to the current CDN with errorbars denoting the 90% confidence interval. Results for naive clients, and for other topolgies sizes, though not shown, are similar.

As shown in Figure 2 (a), DAM offers a slightly lower aggregate throughput than the CDN for low attack volumes, due to the overhead of communication using gate nodes. However, as attack volume is increased, DAM is able to maintain a high throughput, while the clients' throughput sharply falls off when using the CDN. As shown in the graph, for high attack volumes, DAM offers greater than an order of magnitude improvement over the competing CDN architecture. When there is no attack (i.e., attack rate is 0), there is little difference between the aggregate throughput for DAM and the CDN. In low attack volume, DAM's aggregate throughput is slightly less than the CDN, since servers can choose gates which are under attack. The relatively large confidence interval is also due to the choice of gates. This performance can be improved by utilizing bandwidth usage information from gates during the selection process, and we plan on investigating this enhancement in future work. After 500 packets/s attack rate per attacker, the CDN throughput drops sharply because the links to servers are saturated. Servers are idle most of time under high attack volume, since client requests do not reach servers.

Figure 2 (b) demonstrates that DAM offers a low inverse power, regardless of attack volume. Meanwhile, for any attack load larger than 500 packets per second, the inverse power of the network for the CDN increases with attack volume due to the substantial decrease in throughput coupled with an the increased latency in HTTP responses. The time needed for service name resolution in DAM is slightly more than twice that needed for clients using the CDN, as shown in Figure 2(c). However, this resolution time can be decreased by utilizing an adaptive TTL scheme, which assigns larger TTLs when the system is not under attack, and smaller TTLs during times of high load. In addition, the IP addresses of load balancers can be cached with large TTL to avoid Chord routing latency in the resolution step.

Finally, Figure 3 shows the cumulative count for HTTP responses for different attack volume (0, 500, 600, and 1500 packets per second for an attacker). This figure reveals that for attack volumes larger than 500 packets per second, fewer client HTTP requests are serviced by the content servers. Investigating simulation trace files revealed that TCP backoffs due to lost packets in the network contributed to clients issuing a significantly fewer number of HTTP requests to servers, resulting in fewer server responses and degredation in client throughput.

5. CONCLUSIONS AND FUTURE WORK

DoS attacks represent a major threat to the Internet today. In this paper we propose DAM, a novel architecture to mitigate DoS attacks. DAM is a high-capacity infrastructure, shared by multiple service providers, which uses its vast resources to absorb DoS attacks with minimal performance degradation. In particular, DAM uses an indirection point to disallow direct IP access to servers. This level of indirection, tightly coupled with load balancing, offers servers significant protection from DoS attacks.

To demonstrate the viability of this approach we have implemented DAM in NS-2. Our simulations showed that a representative example of current CDN architectures collapses under heavy attack load, while DAM continues to provide good client throughput. Furthermore, our simulations also

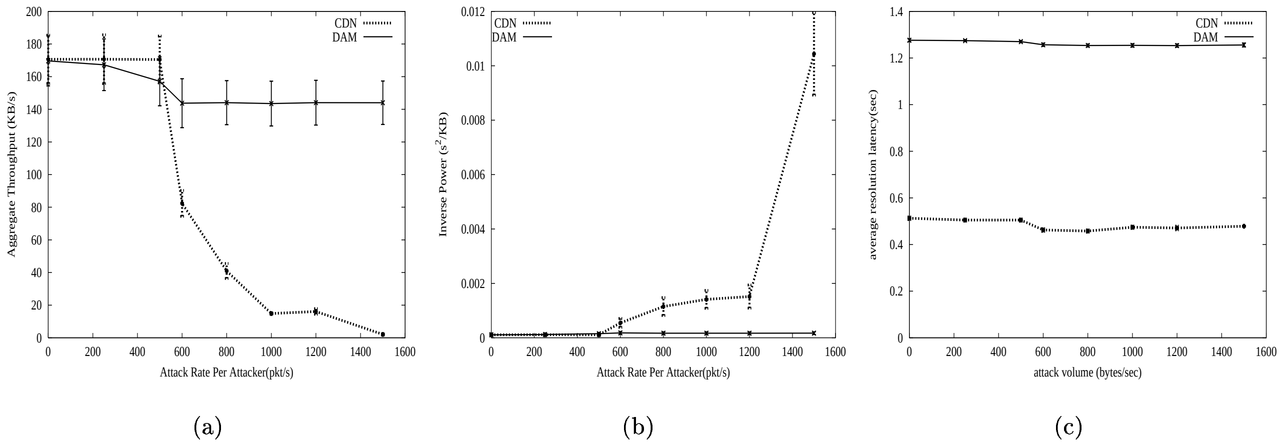

Figure 2: Simulation Results for 400 node topology. (a) Average throughput as a function of attack rate. (b) Inverse power as a function of attack rate. (c) Average resolution time as a function of attack rate.

reveal that DAM is quite efficient in normal operating conditions, i.e. with no attack load. Overall, these simulation results revealed that DAM is a very effective infrastructure for DoS attack mitigation.

There are several directions for future work. We intend to study the effect of attacks on the infrastructure itself. Since we handle these attacks with replication of load balancers, we would like to investigate the effectiveness of this approach. We also intend to examine more accurate load models, especially the network bandwidth usage at gates and load at servers. A final area for future work is to port our implementation of DAM in NS-2 to a real-world DHT and evaluate our system's effectiveness with experiments in a wide area testbed such as PlanetLab [25].

6. ACKNOWLEDGEMENTS

The authors thank Eric Brewer, Ion Stoica, and David Wagner for valuable feedback regarding this work.

7. REFERENCES

[1] David Moore, Geoffrey M. Voelker, and Stefan Savage, "Inferring Internet Denial-of-Service Activity," in *Usenix Security Symposium*, 2001.

[2] "Ann Harrison. The denial-of-service aftermath. CNN.com. February 14, 2000. http://www.cnn.com/2000/TECH/computing/02/14/ dos.aftermath.idg/," .

[3] "Thomas R. Horton. Board level concerns for consequences of information security threats. Washington, DC Summit. April 18, 2000. http://www.ciao.gov/industry/04-18-00/ hortonprinter.html," .

[4] "Jaikumar Vijayan. Denial-of-service attacks on the rise? CNN.com. April 9, 2002. http://www.cnn.com/2002/TECH/internet/04/09/ dos.threat.idg," .

[5] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich, "Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites," in *11th International WWW Conference*, 2002.

[6] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001, pp. 149–160.

[7] "UCB/LBNL/VINT. The Network Simulator version 2, ns-2. http://www.isi.edu/nsnam/ns," .

[8] Stefan Savage, David Wetherall, Anna R. Karlin, and Tom Anderson, "Practical network support for IP traceback," in *SIGCOMM*, 2000.

[9] Ratul Mahajan *et al*, "Controlling High Bandwidth Aggregates in the Network," *ACM CCR*, vol. 32, no. 3, pp. 62–73, July 2002.

[10] Yan Chen, Adam Bargteil, David Bindel, Randy H. Katz and John Kubiatowicz, "Quantifying network denial of service: A location service case study," *Lecture Notes in Computer Science*, vol. 2229, 2001.

[11] A. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure Overlay Services," in *SIGCOMM*, 2002.

[12] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron and Dan S. Wallach, "Security for structured peer-to-peer overlay networks," in *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, 2002.

[13] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, "A scalable content addressable network," in *Proceedings of ACM SIGCOMM 2001*, 2001.

[14] Antony Rowstron and Peter Druschel, "Pastry: Scalable, decentralized object location, and routing for

large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–??, 2001.

[15] B. Y. Zhao, J. D. Kubiatowicz and A. D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," Tech. Rep. UCB/CSD-01-1141, UC Berkeley, April 2001.

[16] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proceedings of IEEE INFOCOM'02*, 2002.

[17] C. Overton Y. Chen, K. Lim and R. H. Katz, "On the stability of network distance estimation," in *Proceeding of ACM SIGMETRICS Practical Aspects of Performance Analysis Workshop (PAPA 2002)*, 2002.

[18] Byung-Gon Chun, Rodrigo Fonseca, and Puneet Mehra, "DAM: an Infrastructure for DoS Attack Mitigation," *CS 262A Project Report, U.C. Berkeley*, 2002.

[19] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee, "How to model an internetwork," in *Proceedings of IEEE INFOCOM'96*, 1996.

[20] J. Jannotti *et al*, "Overcast: Reliable Multicasting with an Overlay Network," in *Usenix OSDI*, 2000.

[21] Paul Barford and Mark Crovella, "Generating representative web workloads for network and server performance evaluation," in *Measurement and Modeling of Computer Systems*, 1998, pp. 151–160.

[22] Jeffrey C. Mogul, "The case for persistent-connection HTTP," in *Proc. SIGCOMM '95 Symposium on Communications Architectures and Protocols*, Cambridge, MA, August 1995, pp. 299–313.

[23] Vern Paxson and Sally Floyd, "Wide area traffic: The failure of posson modeling," *IEEE/ACM Transactions on Networking*, vol. 3(1), pp. 226–244, 1995.

[24] "David dittrich. the dos project's 'trinoo' distributed denial of service attack tool. http://staff.washington.edu/dittrich/misc/trinoo.analysis," .

[25] "PlanetLab. http://www.planet-lab.org," .

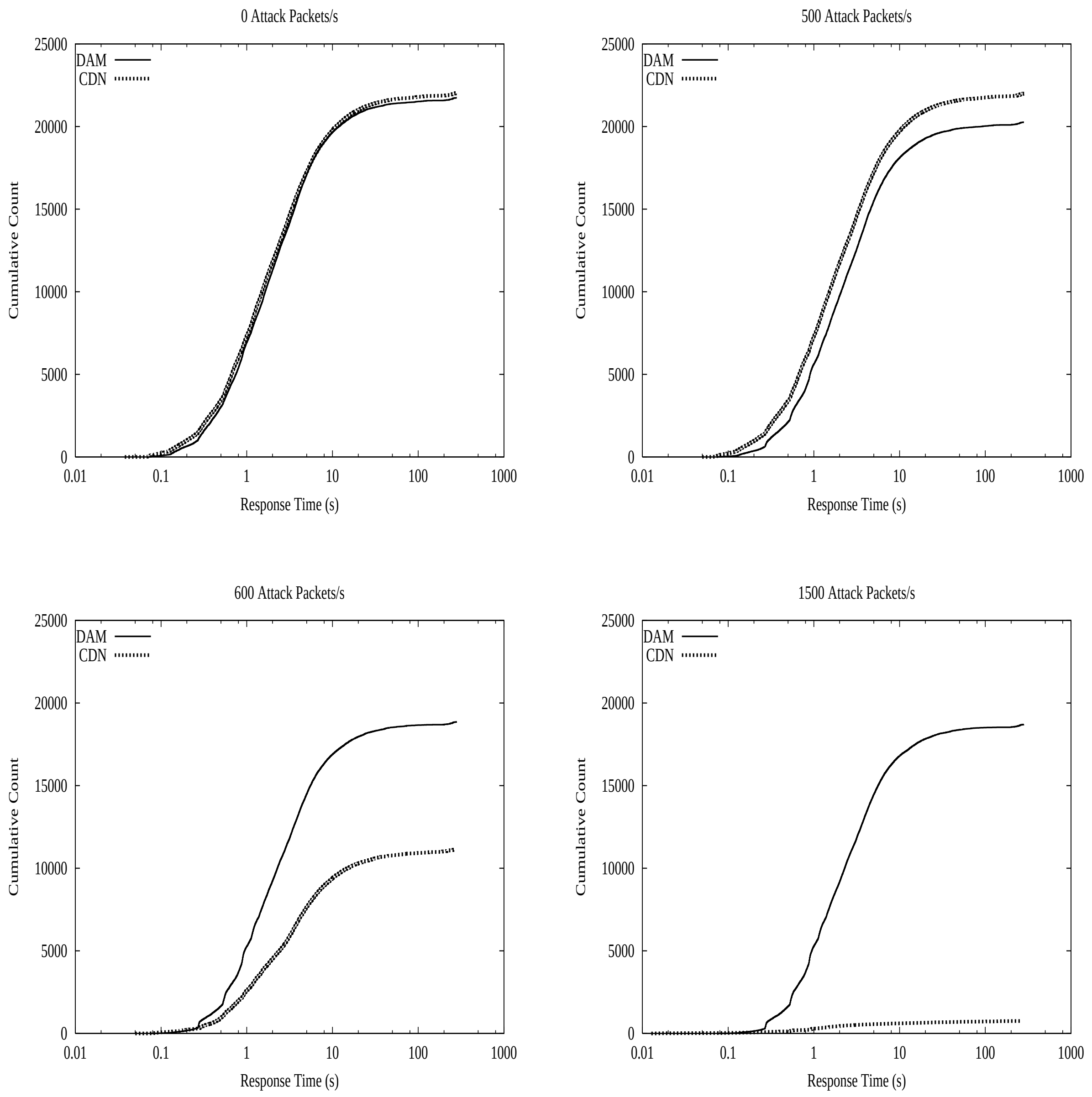

Figure 3: Cumulative count of HTTP response times for different attack loads