Quantifying Network Denial of Service: A Location Service Case Study

Yan Chen, Adam Bargteil, Randy Katz and John Kubiatowicz
387 Soda Hall, #1776,
Univ. of California at Berkeley,
Berkeley, CA 94720-1776,
USA.
{yanchen, adamb, randy, kubitron}@cs.berkeley.edu

Abstract— **Network Denial of Service (DoS) attacks are increasing in frequency, severity and sophistication. Most previous work has focused on network DoS attacks that take advantage of a protocol to launch the attack. We take the broader view that DoS attack is any malicious action which reduces the availability of some resource to some users. Meanwhile, it is highly desirable to be able to measure quantitatively and verify claims pertaining to the security of IT systems and services. As the first attempt to quantify the resilience of a system to broad classes of network DoS attacks, we propose a novel benchmarking methodology and apply it to study the effect of a variety of attacks on directory services in a network setting. Preliminary simulations show the rough ranking of network DoS resilience among centralized directory services, replicated directory services and the newly-emerged distributed directory services, such as Tapestry. Finally, we discuss some potential approaches towards DoS resilience based on our experiments.**

I. Introduction

Denial of Service (DoS) attacks are increasing in frequency and severity. From 1989-1995 the number of DoS attacks increased 50% per year [22]. In addition, a 1999 CSI/FBI survey reported that 32% of respondents detected DoS attacks directed against them [27]. More recently, Yankee Group, an Internet research firm, reported that DoS attacks cost an estimated $1.2 billion in lost revenues in 2000 [15]. Yahoo, Amazon, eBay and Microsoft's name server infrastructure have suffered attacks. To make things worse, automatic attack tools (such as Tribal Flood Network (TFN), TFN2K, Trinoo and stacheldraht) allow teenagers to launch widely distributed denial-of-service (DDoS) attacks with a few keystrokes [11]. Besides these common flooding attacks, we consider a DoS attack to be any malicious action which reduces the availability of some resource to some users. For instance, an attacker might modify routing packets to fool nodes into believing that the network is partitioned, when in fact it is not. These are DoS attacks as well, and with current trends in DoS sophistication, may threaten systems to come.

Given the proliferation of network DoS attacks, many mission-critical applications are built on products claiming various and suspect DoS resilient properties and services. It has led to the widespread desire for a single number/graph by which to rate/purchase/commit to the operation/improvement/retirement of an IT system. However, although the field of "security metrics" has at least a 20-year history involving product evaluation criteria identification, Information Assurance (IA) quantification, risk assessment/analysis methodology development, and other related activities; computer science has steadily frustrated these activities: it has provided neither generally accepted nor reliable measures for rating IT security or requisite security assurance. Also, inconsistent terminology usage has complicated the development of IT metrics, such as *rating, ranking, quantifying*, or *scoring* measurements. We recommend "quantifying" and use it throughout the paper.

Facing the increasing need to develop quantifiable measures of assurance in both industry and military, what is needed is a general methodology for quantifying the resilience of a system/service to broad classes of attacks. As the first steps towards this ambitious goal, we propose a benchmarking methodology for network DoS attack and apply it to study various directory/location services, including centralized/replicated directory services and the newly-emerged distributed directory services. Our simulations also reveal some interesting findings, e.g., why distributed DoS attack is fundamentally more severe than single DoS attack and how to build a more DoS resilient service infrastrcture.

The rest of the paper is organized as follows: We describe our benchmarking methodology in the next section. Section III explains the directory services studied in this paper. Queuing analysis is applied in Section IV to predict how the QoS metrics are affected under various flooding DoS attacks. We discuss our simulation setup in Section V and give the results in Section VI. Finally, some countering attack solutions are proposed in Section VII, related work in Section VIII and conclusion in Section IX.

II. Benchmarking Methodology for Network DoS Attack

In this section, we describe a primitive methodology that can be used to measure and quantify the resiliency of arbitrary computer systems/services to network Denial of Service (DoS).

Our network DoS benchmarking can be used for multiple purposes:

- To prove a system/service has DoS vulnerablities
- To assess and compare the DoS resilience of systems/services
- To support a certification process
- To identify DoS flaws for repair and improvement

There are many challenges remaining:

1) How to define a set of Quality of Service (QoS) metrics for measurement?
2) How to develop a standard consistent network DoS simulation environment?
3) How to define a comprehensive set of network DoS attacks?
4) How to determine a security ranking from a set of discovered attacks?
5) How to differentiate between many systems/services that can be DoS attacked?

We propose some general QoS metrics in subsection II-A and address (2) in the context of Internet services (II-B). For (3), we give a network DoS attack taxonomy and discuss some typical examples of each category because we can not exhaustively cover all known DoS attacks in this paper (II-C). (4) and (5) are solved with "multi-dimensional quantifying" at subsection II-D.

A. Metrics

Essentially DoS attacks resource *availability*. Availability refers to a spectrum of service quality, not to a choice of "available eventually" versus "completely down". The particular choice of QoS metrics depend on the type of system/service being studied. Possible metrics may include *performance, completeness, accuracy* and *capacity*, as we adopted from [5].

The most popular general performance metrics for network services are *request response latency* and *request throughput*. As most services are end-to-end services, the metrics should also be end-to-end. For example, web page request latency should include both lookup latency and retrieval latency (assume no caching). *Time to recover* is another interesting metric, which measures how long the system takes to restore to its original performance after an attack.

Network DoS attack effects on these metrics sometimes can be studied through analytical models, such as queuing theory. With certain assumptions, the theoretical analysis may provide some insight on how the system behave under various degrees of network DoS attacks.

B. Simulation Setup

The challenge for benchmarking DoS attack is to have *accuracy* and *reproducibility* without a real-world testbed environment. Thus we need to set up realistic networks and workloads for simulation.

B.1 Network Configuration

Both synthetic and real network topologies can be used for benchmarking. Synthetic topology can be generated by GT-ITM [48], Tiers [29] or BRITE [32].

GT-ITM generates a transit-stub graph in stages, first a number of random backbones (transit domains), then the random structure of each back-bone, then random "stub" graphs are attached to each node in the backbones. This method ensures that the resulting sub-graph is taken at random from all possible (connected) graphs. Tiers was designed to generate networks whose topology resembles that of typical internetworks; in particular, Tiers was designed to capture the presence of locality and hierarchy in internetworks. More recently, BRITE was proposed as a parametrized topology generator that can be used to study the relevance of possible causes for power laws and other metrics recently observed in Internet topologies, such as preferential connectivity and incremental growth.

For real network topology, we can use the Internet Autonomous Systems (AS) topology graph from National Laboratory for Applied Network Research (NLANR) [18] or the Internet maps from the SCAN project of USC [12]. The AS topology graph models the connectivity between Internet AS, where each node in the topology represents an AS. It was generated based on BGP tables. The Internet maps from SCAN was collected for MBone and other ISPs in 1999. Each node represents a MBone router and these maps are anonymized–they do not contain router IP addresses.

Neither the synthetic nor the real topology above has bandwidth information. We can extend it with commonly-configured network bandwidth. Take the transit-stub model as an example, based on recommendations in [28], [26], we can model links internal to the transit domains as T3 (45Mb/s), edges connecting stub networks to the transit domains as T1 (1.5Mb/s) and edges within a stub domain as "Fast Ethernet" (100Mb/s).

Queuing policy plays a critical role in system resource allocation, e.g., the network congestion management. However, as the attackers often spoof their IP addresses, it does not make much difference for using smarter queuing policies without authentication of the IP addresses. We can just use the simplest drop-tail (FIFO) queueing for simulation.

B.2 Workload Generation

To build the network DoS attack benchmark, we need to generate a continuous realistic workload and to measure and analyze the quality of service. Similar to the topology generation, we can use the synthetic workload or real trace workload. Synthetic workload characterizes the properties of the Internet service traffic. For example, there are two classical World Wide Web (WWW) traffic models as following and one can expect that a real-life workload would be some mix of workloads similar to the one considered.

- **Zipf's law** Previous studies [1], [13] observes that the popularity of pages requested by clients, as well as popularity of pages on a given web site, follow Zipf's law, which basically says that if pages are ranked according to their access frequency, then the popularity of the page with rank i is proportional to $1/i$. Thus, in this workload, clients choose objects according to Zipf's law, where the object number corresponds to its popularity rank.
- **Hot-cold** All objects are divided randomly into hot and cold, with the majority of objects (e.g. 90% [38]) going to the cold bucket and the rest to the hot bucket. The hot objects takes the majority of requests (e.g. 90% [38]). This

workload models the situation when entire web sites vary in popularity.

Real Internet traffic can be found at [30], but the trace contains many more hosts than we can simulate. Thus certain preprocessing and filtering are necessary before simulation.

In order to perform DoS benchmarks, it is also necessary to provide a means of generating attacks and apply them to the system under test, *attack injection*. We can first measure the system without attacks to set up a baseline, then to test it under various *attack workloads*.

C. *Threat Model*

C.1 Taxonomy

According to the classification of DoS attacks by the CERT Coordination Center [8], we consider the following DoS attacks in our methodology:

- Consumption of network connectivity and/or bandwidth;
- Consumption of other resources, such as crucial kernel data structures, CPU and storage space;
- Destruction or alteration of configuration information.

We do not consider:

- Physical destruction or alteration of network components;

There are many dimensions for simulating the network DoS attacks. Attack workload may contain *general* attacks and *application-specific* attacks. The most popular network DoS attack is the flooding attack, in which the attacker(s) keeps injecting bogus requests to the victim(s). Flooding attacks not only overload victim's resources (such as queues and CPU), but also swamp the local router, gateway and links. Usually, the flood traffic is sent at constant rate [16].

C.2 DDoS Attack Network Model

DoS attack can also be classified as *single* or *point-to-point* attacks and *multiple distributed* attacks. There are four major point-to-point DoS techniques: TCP SYN flooding, UDP flooding, ICMP flooding and Smurf attacks [16]. Distributed Dos (DDoS) attack combines the point-to-point DoS attack above with a distributed and coordinated approach to create a powerful program capable of slowing network communications to a grinding halt.

We can model the DDoS attack as a hierarchical structure (Figure 1), with one or more *attackers* controlling *handlers*. A handler is a compromised host with a special program running on it. To eliminate a single point of failure, more than one handler is found in practice. In turn, each handler controls multiple *agents*, compromised hosts responsible for generating packet streams directed toward the victim. Handler and agents are extra layers introduced to hide the attackers from view. In most cases, each handler has equal power over its agents. Communication between the attacker and the handler,and between the handler and the agents are called *control traffic*, while the communication between agents and victims is *flood traffic*. Control traffic can be TCP, UDP, ICMP or a combination of the three. Each agent may use individual point-to-point denial of service technique, or sometimes a combination thereof to generate the flood traffic.

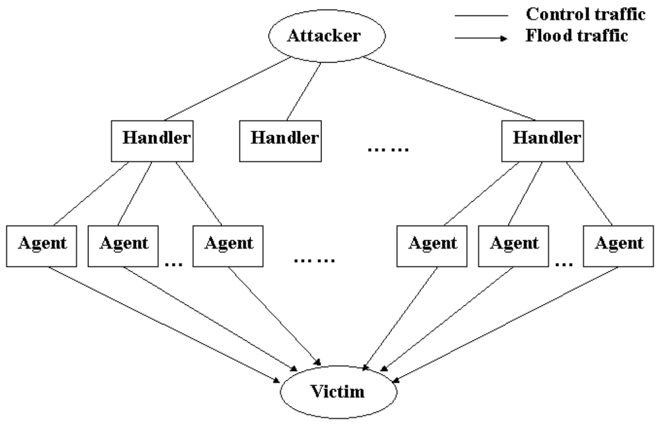

Fig. 1. *Basic hierarchical structure of DDoS attacks*

Each agent can choose the duration ("time"), size of packets and type of packet flooding directed at the victims. The duration for each host may be evenly divided among all hosts, as in "shaft" DDoS tool [16] or randomly chosen as in "TFN" DDoS tool [6]. Also in "shaft" [16], the statistics on request generation rates are possibly used to determine the "yield" of the DDoS network as a whole. Agents can be dynamically added to the simulation. Attackers stop adding when they find the number of agents is sufficient to overwhelm the victim network through the statistics collected. On the other hand, agents are dynamically removed from the network to simulate the agent systems are identified and taken off-line. And the statistics may help the attackers to know when it is necessary to add more agents to compensate for the loss. We can model the changes of the agents in our benchmarks by turning off the attack requests of an agent, waiting for exponentially distributed time intervals, then turning on the attack again at another agent. The agents may be widely distributed or co-located at certain sub-regions.

In general, the attack simulation parameters should be chosen to cover a sufficient spectrum of attack traffic vs. legitimate traffic ratio to show interesting results. This is also illustrated by theoretical analysis in Section IV.

C.3 Destruction or Alteration of Information Attacks

There are numerous ways to launch this type of attack. For example, an intruder may be able to alter or destroy configuration information that prevents you from using your computer or network. Another example is to corrupt the distance metrics, where a malicious node proclaims to a large set of distant nodes that it is very close. Target nodes that believe the attacker then include the attacker in their application-level routing tables, redirecting more traffic to the attacker. The attacker then drops or denies those requests, denying access to existing resources. There is no way we can exhaustively test all of the possible attacks. We simulate and measure some typical examples for this category in our benchmarking.

D. *Multi-dimensional Quantifying*

Security is *multidimensional* in that system A may be stronger than system B on one dimension but weaker on another. Thus, any measures of the effectiveness of the security of a system must also be multidimensional. Usually, the attack taxonomy considered defines a set of dimensions, one for each class of threat. Since there are many taxonomies, systems developed according to different threat models may be hard to compare. Our solution is to have a specific and well-defined taxonomy (e.g., consider only the network DoS attacks) and only quantify the security assurance in that category.

Dimensions should be ranked according to their relative importance. For example, while no system may be satisfactory in all dimensions, a system that is satisfactory in the important dimensions might be rated as acceptable. The ranking will also depend on frequency, severity and sophistication of attacks in that dimension. For instance, we may assign 50%, 30% and 20% as weights to the three classes of attacks in our taxonomy. The *resilience value* is the ratio of attacked performance vs. normal performance. The performance here is a combination of the interested metrics. Or the resilience value may be assigned as a binary number: "pass" (1) or "fail" (0).

Note that in each dimension, there may be multiple different attacks with different severeness. Thus there will be multiple resilience values for that dimension (e.g., the flooding attacks). We can divide the dimension weight equally among these resilience values, or better, to assign the higher weight to the resilience value with more severe attack. Take flooding attack for example, the weight can be assigned proportionally to the amount of flood traffic. The *dimensional score* is the weighted sum of all resilience values in that dimension and the *total score* is the weighted sum of all dimensional scores. Then the total score can be used for rating or ranking.

From next section, we will apply this methodology to directory services as a case study.

III. Directory Services

Current trends show that today's inter-networks are extending their reach to a wide variety of devices over the wide-area, while expanding their bandwidth capacity and reducing latency. Applications expanding to leverage these network resources find that locating objects on the wide-area network is a serious problem. Additionally, the read-mostly, write few model of the Internet and requirement for quick access has led to wide-spread object replication, compounding the object location problem. Extensive work on location services has been done in a variety of contexts [46], [14], [21], [23], [49]. These approaches can be categorized into *Centralized Directory Service (CDS)*, *Replicated Directory Services (RDS)*, and *Distributed Directory Services (DDS)*. We use examples of these categories to quantify the effects of DoS attacks on network services.

A. Centralized and Replicated Directory Services

The primary challenge to build object location services for the wide-area network is that of scalability. Until recently, existing work has focused on the use of statically configured hierarchies to provide scalability ([46], [14]), compared to less scalable alternative approaches ([21], [23]).

From the perspective of Denial of Service attacks, a centralized directory service is most vulnerable. A replicated directory

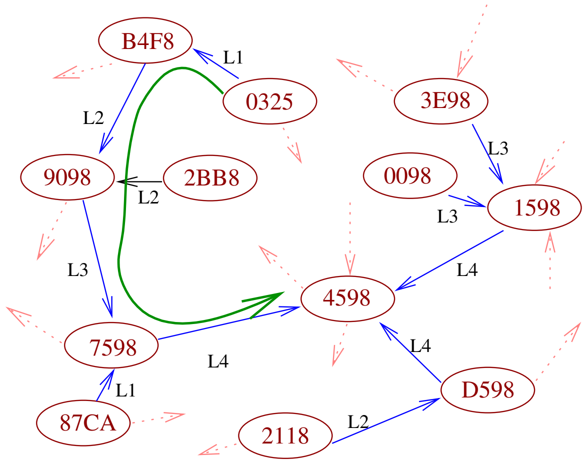

Fig. 2. *Example of Tapestry mesh routing. Node 0325 is routing to node 4598 in a mesh using hexadecimal digit representation*

service, where multiple servers handle requests using replicated location information, provides higher availability, but is still relatively easy to target and attack. Hierarchical approaches, such as DNS [35], can be seen as variations of the centralized approach, where attacking a server working at a hierarchical level can make data at that level unavailable. Meanwhile, the partial caching effects of DNS (i.e., the hosts usually cache some of the direcotry information, but not all of them) makes it, to some extent, between the CDS and RDS. Given the complextiy of simulating real DNS, our simplified models can roughly define the bounds for how it will behave under DoS attack. Our simulations will show the vulnerability of these approaches compared to the decentralized approach described below.

B. Distributed Directory/Location Services

Recent networking research has begun to explore decentralized location services and its applications [49], [39], [45]. Decentralized location services offer an evenly distributed infrastructure for locating objects quickly, with guaranteed success and locality (clients should find the replica of an object closest to them). Instead of depending on a single server for resolving the location of an object, a query in this model traverses multiple overlay nodes in the network in a "combined routing and location" model. With respect to Denial of Service attacks, the lack of a single target in decentralized location services means they provide very high availability under attack. The impact of successfully attacking and disabling a set of nodes is limited to a small set of objects.

One of these decentralized location services is the Tapestry [49] project at U. C. Berkeley. Tapestry is similar to the randomized distributed data structure first introduced by Plaxton, Rajamaran and Richa in [37]. Object location is resolved by directly routing a message to a given object using an overlay routing network. In Tapestry, every server in the system is assigned a unique random node-ID. These node-IDs are then used to construct the mesh of neighbor links forming the overlay, as shown in Figure 2. In this figure, each link is labeled with a level number that denotes the stage of routing that uses this link. In the example, the links are constructed by taking each node-ID and dividing it into chunks of four bits. The N^{th} level neighbor-links for some Node X point at the 16 *closest neighbors*[1] whose node-IDs match the lowest N-1 nibbles of Node X's ID and who have different combinations of the N^{th} nibble; one of these links is always a loopback link. If a link cannot be constructed because no such node meets the proper constraints, then the scheme chooses the node that matches the constraints as closely as possible. This process is repeated for all nodes and levels within a node.

The key observation to make from Figure 2 is that the links form a series of random embedded trees, with each node as the root of one of these trees. As a result, the neighbor links can be used to route from anywhere to a given node, simply by resolving the node's address one link at a time—first a level-one link, then a level-two link, etc. To use this structure for data location, we map each object to a single node whose node-ID matches the object's GUID in the most bits (starting from the least significant); call this node the object's *root*. If information about the GUID (such as its location) were stored at its root, then anyone could find this information simply by following neighbor links until they eached the root node for the GUID. As described, this scheme has nice load distribution properties, since GUIDs become randomly mapped throughout the infrastructure.

This random distribution would appear to reduce locality; however, it achieves locality as follows: when a replica is placed somewhere in the system, its location is "published" to the routing infrastructure. The publishing process works its way to the object's root and deposits a pointer at every hop along the way. This process requires $O(\log n)$ hops, where n is the number of servers in the world. When someone searches for information, they climb the tree until they run into a pointer, after which they route directly to the object. In [37], the authors show that the average distance traveled is proportional to the distance between the source of the query and the closest replica that satisfies this query.

IV. Analytical Model with Queuing Theory

In this section, we use queuing theory to analyze the DoS attack effects on the quality of service provided by a single server with exponentially distributed interarrival time, exponentially distributed service time and infinite buffer size (*M/M/1 queue*). We will study the average response latency and throughput of the system.

We define the following notations:

- T_{ser}: average service time;
- *req*: average number of legitimate requests/second;
- μ: server utilization;
- *r*: the ratio of the number of attack requests vs. *req*;
- T_q: average waiting time per request;
- T_{all}: average response latency ($T_{ser} + T_q$);
- $throughput$: legitimate throughput/second

Then the total request arrival rate is *(r+1)*req. Because

$$T_q = \frac{T_{ser}\mu}{1-\mu} \tag{1}$$

[1]"Closest" means with respect to the underlying IP routing infrastructure. Roughly speaking, the measurement metric is the time to route via IP.

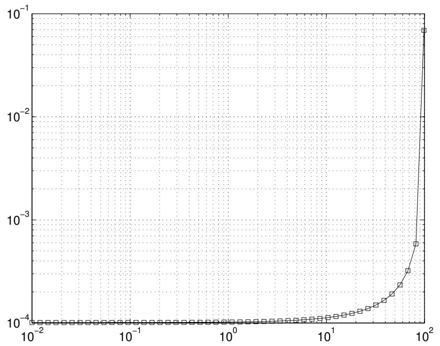

Fig. 3. *Average response latency of a single server on flooding DoS attack. X-axis is the ratio of the number of attack requests vs. legitimate requests; Y-axis is the average response latency(sec)*

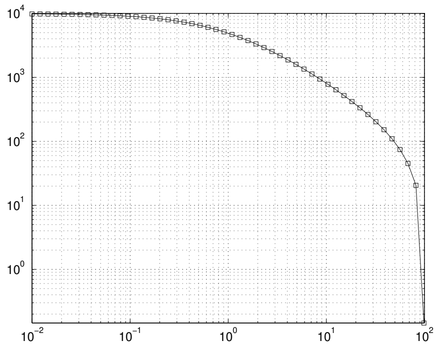

Fig. 4. *Throughput of a single server on flooding DoS attack. X-axis is the ratio of the number of attack requests vs. legitimate requests; Y-axis is the legitimate thoughput (number of requests satisfied/sec)*

$$\mu = (r+1)reqT_{ser} \tag{2}$$

Thus

$$T_q = \frac{T_{ser}^2 req(r+1)}{1-(r+1)reqT_{ser}} \tag{3}$$

$$T_{all} = T_{ser} + T_q = T_{ser} + \frac{T_{ser}^2 req(r+1)}{1-(r+1)reqT_{ser}} \tag{4}$$

$$throughput = \frac{1}{(r+1)T_{all}} = \frac{1}{(r+1)(T_{ser} + \frac{T_{ser}^2 req(r+1)}{1-(r+1)reqT_{ser}})} \tag{5}$$

If we assume T_{ser} to be 0.1ms, req to be 100/sec (these numbers are chosen so that the system in not saturated before attack), Fig. 3 and 4 show the trends how the average response latency and legitimate throughput change as the attack ratio r increases. These figures also tells us how to choose the simulation parameters (flood traffic vs. legitimate requests rate) to cover enough spectrum of various DoS effects.

V. Experimental Setup

Our experimental framework is built on top of NS [3]. Following the methodology in Section II, we create a synthetic well-behaved system, inject malicious attacks into the system, and measure the changes in availability of system resources.

We use a GT-ITM transit-stub model to construct five 1000 node graphs. Each graph is made up of five transit domains. These domains are guaranteed to be connected. Each transit domain consists of an average of eight stub networks. The stub networks contain edges amongst themselves with a probability of 0.5. Each stub network consists of an average of 24 nodes, in which nodes are once again connected with a probability of 0.5. Bandwidths are configured as suggested in Section II. We use drop-tail queueing for each node with the default NS queue size 50.

Our network has 500 objects, each with three replicas placed on three randomly chosen nodes. The sizes of objects are chosen randomly from the interval 5kB - 50 kB. We generated synthetic web traffic based on Zipf's law and hot-cold pattern as described in Section II. Nodes request a data object, wait for it and then request another, such as when a user is following a series of links in web pages.

A. *Directory Server Operation*

For our CDS simulations, a node sends each message to the directory server and waits for the response. Upon receiving the response, it communicates directly with the node hosting the replica chosen by the directory server. The choice of which replica server to return can either be "closest" in network distance, or random selection. There is no caching of object locations. The directory server is placed on a randomly chosen non-transit node.

RDS setup are similar. We choose four random widely-distributed non-transit nodes to be the directory servers. The client may make the lookup request from a random directory server or the closest directory server. The object replica location returned is randomly selected.

For DDS, we implement a simplified version of Tapestry data structures and algorithms as an extension to NS. All messages between nodes are passed by NS's full TCP/IP agent. Messages route directly to the closest object replica, and the replica responds by sending the data contents directly to the requesting node. Our Tapestry data structures are statically built at the start of the simulation using full knowledge of the topology, and using hop count as the network distance metric. It should also be noted that our implementation is un-optimized and is likely slower than a real implementation would be.

B. *The Attacks*

We inject a variety of attacks into our simulations and measured the effect of the attacks on lookup requests. Assuming that these attacks can actually be carried out, our results give an idea of how much damage they can do.

B.1 Flooding Attacks

The first attacks we simulate flood some important node(s) and overload their queues to reduce the number of legitimate requests that can get to them as shown in our queuing analysis.

All the flooding simulations run for 200 seconds in NS time. The flooding is accomplished by sending a constant bit rate from a set of nodes to the attacked node. We vary the number of flooding attackers; with DDoS attacks, we vary the severity of flooding (i.e. bit rate) for "flood traffic" and ignore the "control traffic" for simplification. The attack "agents" are placed on randomly-chosen non-transit nodes in different subnets from the attacked node. Given present Internet infrastructure, it is not easy to identify the spoofed agents, thus we set the life time of each agent to be randomly among 0 - 200 seconds. To have a conservative estimate of the DDoS resilience, we assume that once an agent is taken off, another agent will be launched immediately to keep the total number of agents constant.

For the centralized directory server case, we attack the directory server. The closest analogy in the Tapestry case is to flood the root of an object. Whereas flooding the centralized directory server will affect the lookup of all objects, attacking a root for a single object will pronouncedly affect only the attacked object and others with the same root. So to produce interesting results, we attack the root for a hot object. Tapestry trees are more resilient to such an attack because not all requests for the object will travel all the way to the root: some will be satisfied by other nodes on the insert path. For replicated directory servers case, we attack all the directory servers. To have a fair comparison, we attack the same number of root nodes of hot objects.

We only studied static directory services; a more realistic simulation of dynamic location services is our future work.

B.2 Destruction or Alteration of Information Attacks

As these attacks are system/service-specific, we only simulate and measure two attacks here as examples. A more comprehensive and systematic benchmarking will be part of our future work.

The first attack is to compromise an important node to give incorrect distance measurements as illustrated in Section II. We compromise the directory server of CDS, a random directory server of RDS and a root node of hot object of Tapestry for comparison. The compromise is to insert a false edge with negligible latency for the pair of nodes with maximum distance.

The second attack is a Tapestry-specific attack, which has a malicious Tapestry node spoofing as root nodes for all objects. By replying with a negative result to any request it receives, this attack can potentially convince clients the non-existence of requested objects, denying them access to an existing resource.

VI. RESULTS

A. *Flooding Attacks*

We perform several simulations of flooding attacks, for the CDS, RDS and the Tapestry directory service. We apply the simulations to both hot-cold workload and Zipf's law workload. They give similar trend and results; thus here we only show the results with hot-cold workload, averaged among the simulations on five topologies.

A.1 Performance Comparison of CDS vs. Tapestry

First, we compare the performance of CDS with Tapestry. The number of attacking nodes was either one or four. The bit rate for the single attacker is 500 bytes or 2000 bytes every 5ms, while for DDoS attacks, the rate is varied from 500 bytes

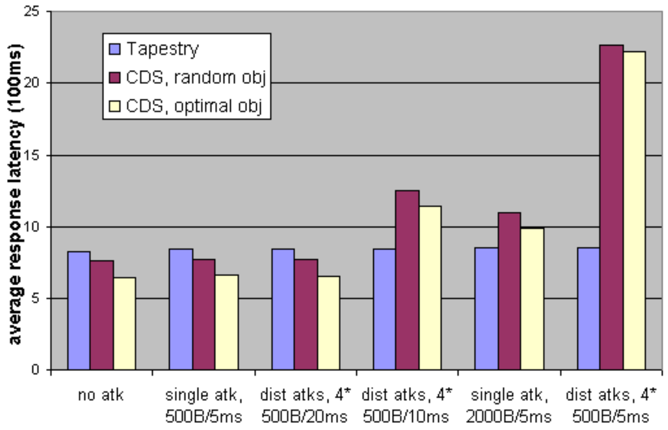

Fig. 5. *Average response latency of CDS vs. Tapestry under DoS flooding attacks*

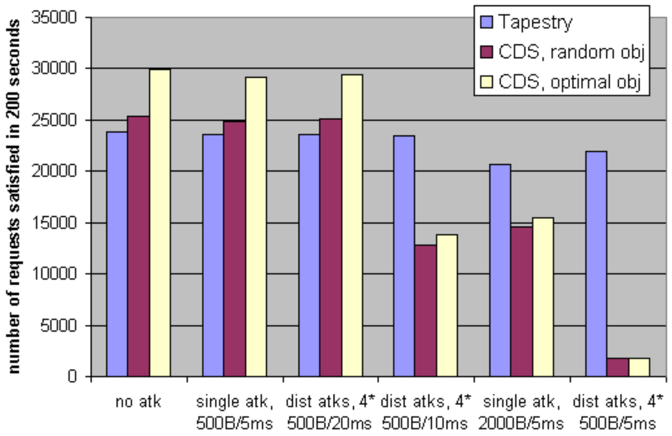

Fig. 6. *Throughput of CDS vs. Tapestry under DoS flooding attacks*

every 20ms (least severe) to 500 bytes every 5ms (most severe), with each agent set to the same rate. These numbers are chosen to have reasonable ratios of attack bandwidth vs. legitimate throughput to show some interesting results.

The results are shown in Figures 5 and 6, which reveal that a single attacker does not significantly influence performance, while the distributed attackers, each flooding at the same high rate, cause severe denial of service. Thus as the attacks get more severe, the average response latency of the CDS skyrockets and the throughput dives. Interestingly, the trends match well with the prediction from our analytical model in Section IV. On the other hand, the Tapestry-based directory service shows resistance to flooding attacks. It can be explained by the distributed nature of Tapestry – there is no single server that keeps all the lookup information. Under no attack, the Tapestry-based location service performs slightly worse than the directory server, especially in throughput. This is because an object request must not only pass through several nodes on its way to the root, but each of these nodes must do some processing of the request.

For each simulation, we normalize the the throughput under various attack vs. normal throughput and plot as Figure 7. The

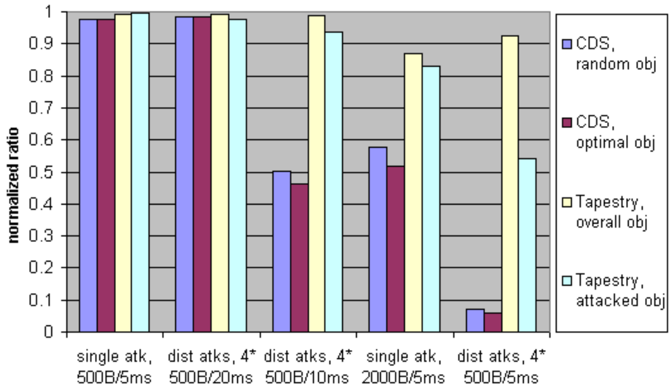

Fig. 7. *Normalized thoughput of CDS vs. Tapestry under DoS flooding attacks*

normalized graph shows that the throughput of CDS drops to 6-7%, while Tapestry remains more than 90% of the original throughput. Even the attacked object of Tapestry keeps more than 50% of throughput under the most severe attacks. This is because in Tapestry, a request travels toward the root until it encounters a node which was on the insertion path of one of the replicas (this node then forwards the request directly to the replica). Thus even the root node is swamped, those "back-pointers" can still help the clients to find a local replica.

In addition to performance comparison, we try to answer two interesting questions regarding flooding DoS attack from our experiments:

1) Why the distributed attack is fundamentally more severe than the single attack? Is that only because more hosts will have more power to inject flood traffic?
2) How do the different policies of CDS or RDS affect their resilience?

For the first question, there are three other reasons revealed from our experiments. One is that single DoS attack traffic is restricted by the bottleneck bandwidth from the attacker to victim, e.g., the T1 line in our experiment. Note that although the network bandwidth keep increasing and the bottleneck bandwidth in our simulation may be out of date soon, the machine power also double every 18 months (Moore's law). Thus the network bandwidth may still limit the attackers' impact, depending on their connectivity. The second one relies on the distribution of clients – the attacker swamps the path from itself to victim, any client that shares some link of the path (especially the bottleneck link) will be affected. As shown by the two rightmost sets of data in Figures 5, 6 and 7, the distributed attackers cause more severe DoS than a single attacker, although they both inject the same amount of flood traffic. Finally, as attackers or agents are identified and taken offline from time to time, distributed attackers are much harder to eliminate completely.

The second question is answered in both this and next subsections. For CDS, when there is no attack, the directory server which chooses the optimal replica performs better than the one that chooses a random replica for about 20% with throughput. Surprisingly, with more and more severe attacks, the gap is shrinking and finally the two perform similarly. One explana-

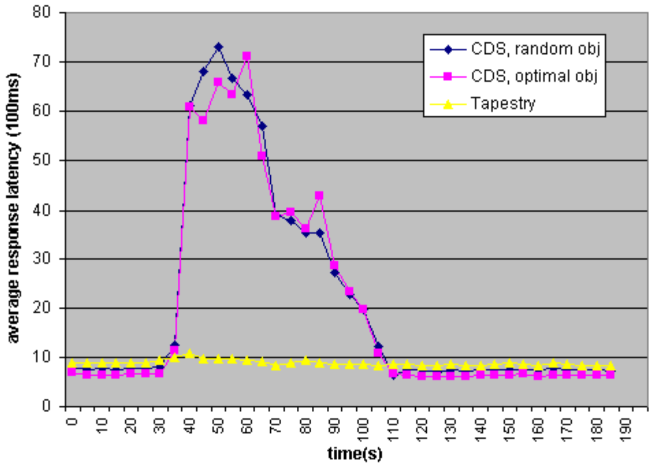

Fig. 8. *Dynamics of average response latency of CDS vs. Tapestry under DoS flooding attacks*

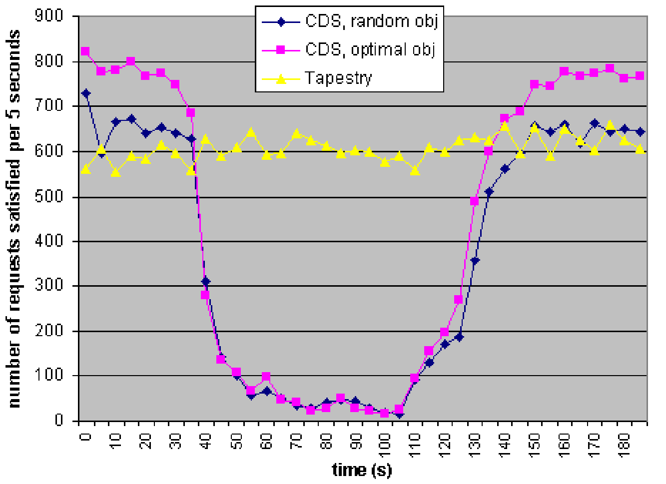

Fig. 10. *Dynamics of throughput of CDS vs. Tapestry under DoS flooding attacks*

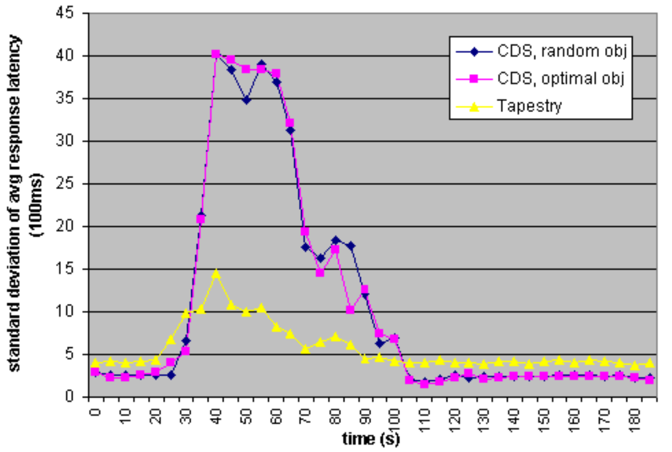

Fig. 9. *Standard deviation dynamics of response latency of CDS vs. Tapestry under DoS flooding attacks*

tion for this phenomenon is that with more and more severe attack, the directory lookup service becomes the bottleneck of the whole system (web object retrieval). That is, the throughput will be solely determined by the throughput of the directory services. The time to do the actual retreival of the object is negligable.

If we watch through the attacking process, Figures 8, 9 and 10 show the dynamics of the most severe flooding attack effects on the average response latency and throughput in CDS and Tapestry simulations. The attack(s) start at 40 seconds and end at 110 seconds. We filter the data of last 10 seconds, divide the rest 190 seconds into five-second bins and calculate the mean and standard deviation for each bin. The system's performance is degraded right after the DoS attack, however, it is much slower for the system to restore its original performance after stopping DoS attack. Given our simulation setup, the time to recover for CDS with both policies are 40 seconds. As Tapestry is not really affected much, its the time to recover is 0.

A.2 Performance Comparison of RDS vs. Tapestry

We also run these simulations on replicated directory servers. We put four directory servers on four widely-distributed non-transit nodes. Two different policies are investigated: the client can request random directory server or always request the closest one. The latter assumes that the client can use some bootstrap mechanism to find its closest directory server. We do not simulate the overhead traffic to keep these directories consistent.

Again, the single flooding attack cannot affect the performance much. So we only show the results of DDoS attacks in Figure 11 and 12. When there is no attack, the multiple directory servers outperform Tapestry service by 20%-30% in terms of throughput. Then we simulate four random non-transit attack agents, each attack one directory server in different subnet from the agent. For Tapestry, we have each of these agents to attack the root node of a random hot object. The performance of all these directory services remain mostly unchanged.

Furthermore, we have sixteen random non-transit attack agents, each four from four different subnets attacking one directory server of RDS or the Tapestry root node of a random hot object. The rate is varied from 500 bytes per 10ms (least severe) to 500 bytes per 1ms (most severe), with each agent set to the same rate.

In contrast to the CDS case, the optimal RDS always performs better than the other two. The reason may be that although the attackers swamp the bottleneck link from the directory server to the backbone (in our experiment, the T1 line connecting the stub domain and the transit domain), all the clients in the same subnet as the directory server can still be serviced. Thus replication and topology-aware locality can significantly increase the DoS attack resilience for directory services. Meanwhile, Tapestry outperforms the random RDS on severe attacks.

B. Destruction or Alteration of Information Attack

For the "compromising the important node with a false edge attack", the CDS and RDS which access the random replica

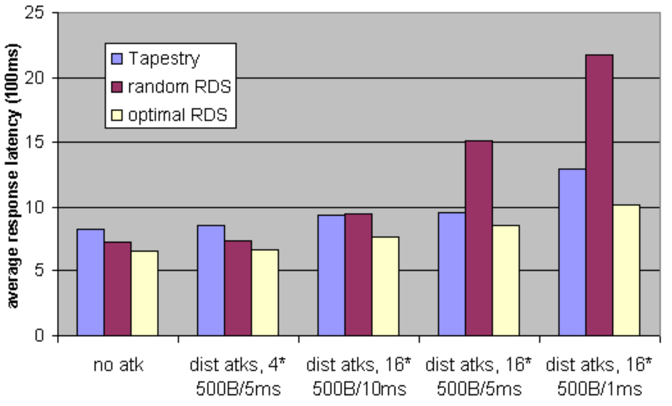

Fig. 11. *Average response latency of RDS vs. Tapestry on DDos flooding attacks*

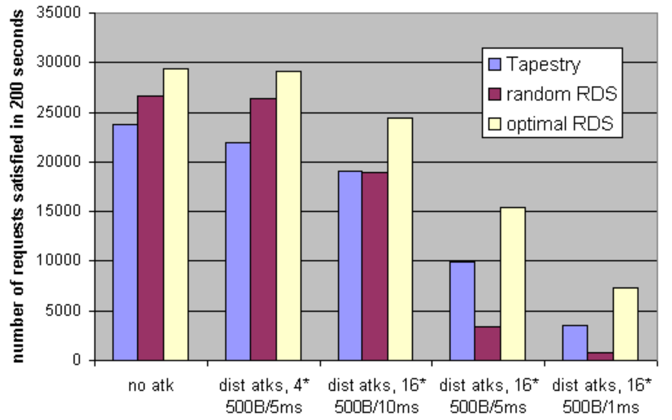

Fig. 12. *Throughput of RDS vs. Tapestry on DDos flooding attacks*

are not affected (we assume that the directory server(s) are not routers or gateways). The performance of CDS which access the optimial replica was degraded to 85%. The impact to Tapestry is neligible – overall performance reduced only by 2.2%, with resilience value 97.8.

We also simulate the Tapestry-specific node spoofing attack. The effects of the attack are displayed in Figure 13 [2]. The attack affects 24% of the network. With evenly distributed workload, the resilience value is 0.76.

C. Quantifying

Following the methodology in Section II, we quantify the network DoS resilience of CDS, RDS and Tapestry and rank them in Table I. Since flooding attacks belong to both the first and second categories of our taxonomy, they provide 80% (the sum of the two dimensional weights) of the score. We simulate all eight attacks in Figures 5, 6, 11 and 12 for all three types of directory services and use the combination of latency and throughput results. The weights are assigned in proportion to their amounts of flood traffic. The two attacks in "destruction

[2]We reduce the simulation size to 100 nodes and 60 objects (15% hot) for better visualization.

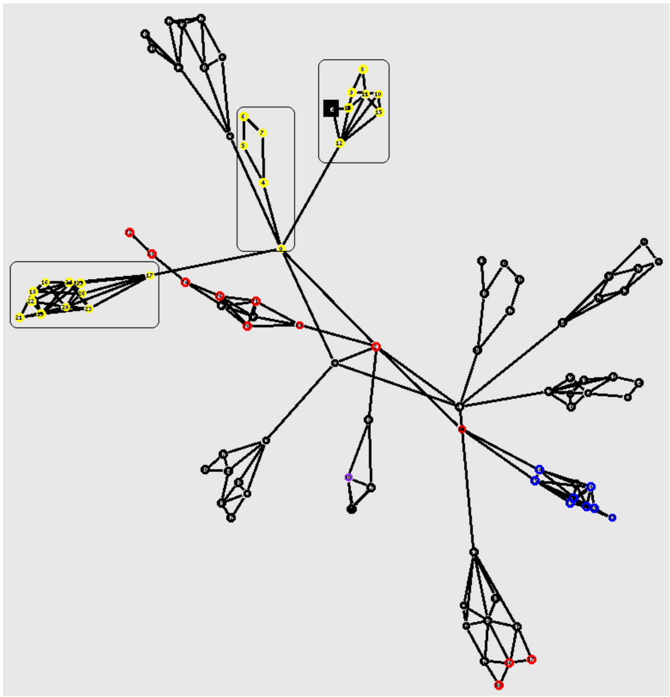

Fig. 13. *Nodes accessing each replica of an attacked object. Corruption of the neighbor tables at the black square node renders all the nodes enclosed by round-corner rectangles unable to lookup the object.*

or alteration of information" are comparable and have the same weight.

Note that we did not test all possible attacks under all possible severeness, thus the ranking is by no means very accurate. However, it does give an idea how these directory services differ in terms of network DoS resilience and how to apply the general benchmarking methodology to a specific case.

VII. Mechanisms for Resilience to Attacks

Denial of Service attacks have varying impact depending on the target system architecture. In the instance of location services, there are several different approaches to minimize the impact of attacks.

We discuss several potentially interesting approaches towards DoS resilience:

A. Decentralization and Topology-awareness

A primary reason that today's systems are extremely vulnerable to DoS attacks is their primarily centralized design. In terms of applications, most website content is delivered by a single web server. In terms of infrastructure, clients on the Internet hardcode a single DNS server into their network configuration. These design choices present to would-be attackers a small number of clear targets to focus efforts on. Attacking and disabling these targets results in wide-spread havoc.

As we learned from the experiments, replication and topology-awareness dramatically improve the DoS resilience of systems/services. Some form of Internet distance estimation service, such as IDMap [19] may suffice for topology

Directory services	Flooding attack (80%)	Distance corruption attack (10%)	Node spoofing attack (10%)	Total score	Rank
CDS, access random replica	0.027	1.0	1.0	0.2216	4
CDS, access optimal replica	0.023	0.85	1.0	0.2034	5
RDS, user request random dir server	0.17	1.0	1.0	0.336	3
RDS, user request optimal dir server	0.48	1.0	1.0	0.584	1
Tapestry	0.35	0.978	0.76	0.4538	2

TABLE I
Quantifying three types of directory services

awareness. Meanwhile, the advent of infrastructures such as Tapestry make possible a decentralized model of operation, where all nodes serve similar functions to similar sized client pools. With no clear central target, and lots of less-important potential targest, an attacker's resources are spread thin in order to make any impact on the performance of the overall system.

B. Obscure and Redundant Naming

In order to make a successful attack, an attacker needs to associate the set of physical node(s) that provide a given resource. One way to prevent meaningful attacks is to obscure the name of the given resource, so that attackers cannot readily identify which physical nodes provide it. Another is to create and advertise with multiple names for each protected resource, such that an attacker cannot prevent clients from finding the resource.

C. Separate Channels for Data and Control Transmission

Services deployed on top of the existing TCP/IP-based Internet infrastructure automatically inherit vulnerability to flooding-based DoS attacks. The existing infrastructure treats packets impartially: routers are generally oblivious to the source of traffic routed through them, and the same channels are used to transmit both data and control information ("in-band signaling"). To handle control information out-of-band, as the phone network now does, can be used to deal with many of the Internet's security vulnerabilities.

D. Detection and Reaction

While the previous three approaches focus on making attacks harder and more costly, we now discuss an approach for handling the inevitable attacks that do happen. The general approach is to quickly detect the attacks while they are in progress, and either take action to eliminate the attack or its impact, or failing that, notify authorities of its findings.

To detect DoS attacks in progress, recent research has provided several techniques for tracing DoS attacks with low overhead [41], [43]. The IOS software of Cisco router can use access lists to filter out attacks, characterize unknown attacks and trace "spoofed" packet streams back to their real sources [24]. In addition to these protocol-specific techniques, we believe that "introspection" or self-examination of a system, should occur on a regular basis on several potential layers (application, link, transport, etc). Once an attack has been identified via abnormal behavior or protocol-specific mechanisms, group consensus protocols can be used to provide a form of authority, and create a way to eliminate the offending party. In the Tapestry infrastructure, if a large enough set of nodes agree on the identity of a malicious node, they can actively sever the node's connection to the overlay network, therefore refusing to relay its traffic and rendering it harmless.

VIII. Related Work

Denial of service attacks have been studied for some time. Early work by Gligor and Yu [20], [47] built on the classic notion of a trusted computing base to define a "DoS Protection Base". Yu and Gligor also pointed out that denial of service is in fact an attack on resource availability. Millen believed that denial of service is fundamentally a problem of improper resource allocation [33], [34].

The recognition that denial of service is fundamentally a problem of improperly reducing availability of resources has inspired more recent work by Meadows [31], who has worked on characterizing susceptibility of network services to DoS attacks on resources used before a remote host can be authenticated. Such network-based attacks are increasingly problematic. Some such attacks rely on protocol weaknesses to consume resources with minimal attacker effort, as in TCP SYN flooding [42]; other attacks depend simply on the ability of an attacker to produce sufficient traffic to overwhelm a victim by brute force [10].

Several counters to network denial-of-service attacks have been proposed. In [42], the authors investigated several approaches to fighting TCP SYN attacks and developed a tool which actively monitored the network for suspicious attack behavior and terminated dangling connections left by the attacker. In [44], the authors describe the use of an end-to-end resource accounting in the Scout operating system to protect against resource-based DoS attacks. Both these works present microbenchmarks testing the effectiveness of the proposed countermeasure. Our approach differs partly in that we investigate attacks on availability of a service, rather than on a particular server.

There are more approaches proposed recently towards improving DoS attack resilience, such as network ingress/egress filtering [17], packet rate limiting and unicast reverse path forwarding [25], improving intrusion detection capabilities (e.g., use Snort [40]), and auditing hosts for DDoS tools (e.g., use NIPC find_ddos [9]). In contrast, the mechanisms that we proposed in Section VII focus on the more fundamental network and services infrastructural improvements for DoS resilience.

There has been a great deal of recent interest in quantifying service availability. Brewer [4] suggests several metrics for measuring availability of networked data services, of which the latency and throughput metrics we use are some of the simplest. Brown and Patterson [5] investigate the use of workloads including benign faults as a way of benchmarking availability, and applied their methodology to studying software RAID systems. Our work is similarly based on injecting faults into a workload and investigating the effect, but our faults are malicious in nature.

In 1999, International Organization for Standardization (ISO) published "The Common Criteria for Information Technology Security Evaluation (CC) version 2.1" to be used as the basis for evaluation of security properties of IT products and systems [7]. Jennifer Bayuk recommended using an automated approach for assigning quantitiative weights or values to the pre-defined criteria for assessment [2]. More recently, "backscatter analysis" was proposed to estimate the denial-of-service attack activity in the Internet [36].

IX. Conclusions and Future Work

We have proposed a benchmarking methodology to quantitatively characterize network DoS attacks and applied it to several directory services, such as CDS, RDS and the newly-emerged distributed directory services. Using elementary queueing theory, we analyzed the effects of DoS flooding attacks on a single server. We also created an NS network simulation with a synthetic workload motivated by web-traffic. We then injected malicious attacks into the system and measured the availability, as characterized by latency, throughput and time to recover, of the attacked services. Our simulation framework is the first attempt to quantify the network DoS resilience of arbitrary systems/services. Finally, some potential approaches to improve the network DoS resilience are proposed.

Future work includes simulating DoS attacks on dynamic systems/services, applying our methodology to other services, such as content distribution and web hosting and modeling a greater variety of attacks for benchmarking as well as modeling the proposed countermeasures we have suggested.

References

[1] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveria. Characterizing reference locality in the WWW. In *Proceeding of the IEEE Conf. on Parallel and Distributed Information Systems*, 1996.

[2] J. Bayuk. Measuring security. In *First workshop on information-security-system rating and ranking*, May 2001. `http://www.acsac.org/measurement/position-papers/Bayuk.pdf`.

[3] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[4] Eric Brewer. Lessons from giant-scale services. submitted for publication.

[5] Aaron Brown and David Patterson. Towards availability benchmarks: A case study of software RAID systems. In *Proceedings of the 2000 USENIX Annual Technical Conference*, San Diego, CA, June 2000.

[6] David Brumley. Remote intrusion detector, 2001. `http://theorygroup.com/Software/RID/`.

[7] Common Criteria Implementation Board (CCIB) and International Organization for Standardization (ISO). Common Criteria version 2.1 / ISO IS 15408, 1999. `http://csrc.nist.gov/cc/ccv20/ccv2list.htm#CCV21`.

[8] CERT Coordination Center. Denial of service attacks. `http://www.cert.org/tech_tips/denial_of_service.html`, 1999.

[9] National Infrastructure Protection Center. find_ddos: find distributed denial of service (ddos), 2001. `http://www.nipc.gov/warnings/alerts/1999/README`.

[10] Symantec AntiVirus Research Center. W32.dos.trinoo. `http://www.symantec.com/avcenter/venc/data/w32.dos.trinoo.html`, 2000.

[11] CERT/CC advisory ca-2000-01 Computer Emergency Response Team. Denial-of-service developments. `http://www.cert.org/advisories/CA-2000-01.html`, 2000.

[12] Self configurable active network monitoring. Internet maps, 2001. `http://www.isi.edu/scan/`.

[13] C. A. Cunha, A. Bestavros, and M. Crovella. Chracteristics of www client-based traces. Technical Report TR-95-010, Boston University, Department of Computer Science, April, 1995.

[14] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *Proceedings of ACM MOBICOM*, August 1999.

[15] Michelle Delio. New breed of attack zombies lurk, May 2001. `http://www.wired.com/news/technology/0,1282,43697,00.html`.

[16] S. Dietrich, N. Long, and D. Dittrich. Anaylzing distributed denial of service tools: the Shaft case. In *Proceedings of the 14th Systems Administration Conference (LISA)*, Aug 2000.

[17] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing, 1998. RFC2267, `http://www.landfield.com/rfcs/rfc2267.html`.

[18] National Laboratory for Applied Network Research (NLANR). Raw routing table information, 2001. `http://moat.nlanr.net/Routing/rawdata/`.

[19] Paul Francis, Sugih Jamin, Vern Paxon, Lixia Zhang, Daniel Gryniewicz, and Yixin Jin. An architecture for a global internet host distance estimation service. In *Proceedings of IEEE INFOCOM*, March 1999.

[20] V. Gligor. A note on the DoS problem. In *Proceedings of the 1983 Symposium on Security and Privacy*, 1983.

[21] Erik Guttman, Charles Perkins, John Veizades, and

Michael Day. Service Location Protocol, Version 2. IETF Internet Draft, November 1998. RFC 2165.

[22] John D. Howard. *An Analysis of Security Incidents on the Internet*. PhD thesis, Carnegie Mellon University, Aug. 1998.

[23] Timothy A. Howes. The Lightweight Directory Access Protocol: X.500 Lite. Technical Report 95-8, Center for Information Technology Integration, U. Mich., July 1995.

[24] Cisco Systems Inc. Characterizing and tracing packet floods using Cisco routers, 2001. `http://www.cisco.com/warp/public/707/22.html`.

[25] Cisco Systems Inc. Strategies to protect against distributed denial of service (DDoS) attacks, Feb. 2000. `http://www.cisco.com/warp/public/707/newsflash.html`.

[26] Frontier Network Inc. Internet topology. `http://www.frontier.net/about/topology.html`, 2000.

[27] Computer Security Institute and Federal Bureau of Investigation. 1999 CSI/FBI computer crime and security survey. In *Computer Security Institute publication*, March 2000.

[28] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole Jr. Overcast: Reliable multicasting with an overlay network. In *4th Symposium on Operating Systems Design & Implementation*, Oct. 2000.

[29] M. B. Doar K. Calvert and E. W. Zegura. Modeling internet topology. *IEEE Communication Magazine*, June 1997.

[30] Lawrence Berkeley National Laboratory. The Internet traffic archive, 2001. `http://ita.ee.lbl.gov/`.

[31] C. Meadows. A formal framework and evaluation method for network denial of service. In *Proceedings of the IEEE Computer Security Foundations Workshop*, June 1999.

[32] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in internet topologies. In *ACM Computer Communication Review*, April 2000.

[33] J. Millen. A resource allocation model for DoS. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, 1992.

[34] J. Millen. DoS: A perspective. In *Dependable Computing for Critical Applications 4*, 1995.

[35] P. Mockapetris. Domain names - implementation and specification, November 1987. STD 13, RFC 1035.

[36] David Moore, Geoffrey Voelker, and Stefan Savage. Inferring Internet denial of service activity. In *Proceedings of the 2001 USENIX Security Symposium*, Aug. 2001.

[37] Greg Plaxton, Rajmohan Rajaraman, and Andre'a W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the 9th Annual SCP Symposium on Parallel Algorithms and Architectures*, June 1997.

[38] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. A dynamic object replication and migration protocol for an internet hosting service. In *Proceedings of IEEE Int. Conf. on Distributed Computing Systems*, May 1999.

[39] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. to appear in Proceeding of ACM SIGCOMM, 2001.

[40] Marty Roesch. Snort: The lightweight network intrusion detection system, 2001. `http://www.snort.org/`.

[41] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for IP traceback. In *Proceedings of ACM SIGCOMM*, August 2000.

[42] C. Schuba, I. Krsul, M. Kuhn, and et. al. Analysis of a DoS attack on TCP. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, May 1997.

[43] Dawn Song and Adrian Perrig. Advanced and authenticated marking schemes for IP traceback. In *Proceedings of IEEE Infocom Conference*, 2001.

[44] O. Spatscheck and L. Peterson. Defending against DoS attacks in Scout. In *Proceedings of the 3rd Symposium on Operating System Design and Implementation*, February 1999.

[45] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. to appear in Proceedings of ACM SIGCOMM, 2001.

[46] Maarten van Steen, Franz J. Hauck, Philip Homburg, and Andrew S. Tanenbaum. Locating objects in wide-area systems. *IEEE Communications Magazine*, pages 104–109, January 1998.

[47] C. Yu and V. Gligor. Specification and verification method for preventing denial of service. *IEEE Transactions on Software Engineering*, 16(6), June 1990.

[48] Ellen W. Zegura, Ken Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE Infocom*, 1996.

[49] Ben Y. Zhao, John Kubiatowicz, and Anthony Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. UCB Tech. Report UCB/CSD-01-1141.