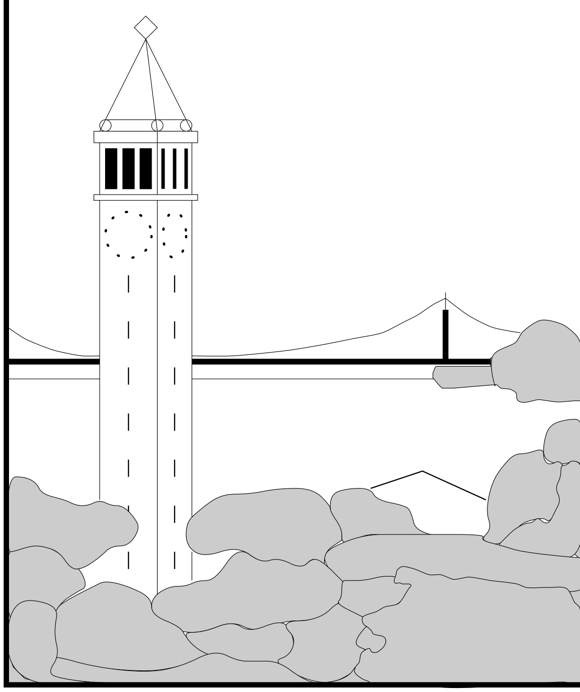
DoS Defense in Structured Peer-to-Peer Networks

Pete Perlegos
University of California, Berkeley
`perlegos@eecs.berkeley.edu`



Report No. UCB/CSD-04-1309

March 2004

Computer Science Division (EECS)
University of California
Berkeley, California 94720

DoS Defense in Structured Peer-to-Peer Networks

Pete Perlegos
University of California, Berkeley
perlegos@eecs.berkeley.edu

March 2004

Abstract

Denial of service (DoS) attacks are a large and increasing threat to the Internet community. In this paper, we propose using a distributed approach to DoS defense. Our architecture leverages the properties of a wide-area overlay network to isolate clusters of attackers while denying access to a minimal amount of legitimate users. This is done by collaborating with other members of a structured peer-to-peer network, which is inherently collaborative. Our results show that our approach is effective at both detection and suppression of a DoS attack.

1 Introduction

Denial of service (DoS) attacks are a large and increasing threat to the Internet community. A recent study [23] detected 12,805 attacks on over 5,000 distinct Internet hosts over a three week period. We propose that a distributed approach to detecting and suppressing a DoS attack is more effective than a local, centralized approach. The purpose of this paper is to study some distributed mechanisms and to compare these mechanisms to more traditional, local, centralized techniques.

There are two general classes of DoS attacks: logic attacks and flooding attacks. Logic attacks exploit software flaws to crash remote servers or degrade performance. Many of these attacks can be prevented by upgrading faulty software. Flooding attacks send large numbers of illegitimate requests which prevent a server from serving the legitimate requests. In this work we are mainly concerned with DoS flooding attacks.

Attackers can mount powerful attacks by leveraging the resources of multiple hosts; these attacks are known as distributed denial of service (DDoS) attacks. An attacker compromises a set of Internet hosts and installs a small attack daemon on each, producing a group of zombie hosts. A DDoS attack follows a hierarchical model, with one or more layers of indirection and able to control hordes of agents [12]. As a result of readily available tools, DoS attacks are easy to mount and hard to trace. An effective DoS defense must be able to respond quickly.

DDoS attacks have become an increasing problem in the Internet [23]. They are very hard to defend against because they do not target specific vulnerabilities of systems, but rather exploit the fact that the target is connected to the network [15]. DDoS attacks take advantage of the hosts on the Internet with poor security. The perpetrators break into such hosts, install slave programs, and at the right time instruct thousands of these slave programs to attack a particular target. Since this attack does not exploit a security problem at the target, no effective mechanism currently exists to defend against such an attack.

Under normal conditions TCP-like congestion control ensures fair use of the available resources. Under a DDoS attack, the arriving packets do not obey end-to-end congestion control algorithms, and instead bombard the victim, using the available resources which cause the good flows to back off and eventually starve. A large scale DDoS attack not only causes trouble to its intended victim, but also interferes with bystander traffic that may happen to use a portion of the network that is being heavily congested [15]. This happens because as it nears the target, the attacking communication hits a pinch point and overwhelms part of the network and the target.

Many previous approaches have focused on IP level methods to attempt to identify and stop attackers. This has many drawbacks which stem from a primary cause; IP was designed as a best effort service to send a message to its destination even if many network connections have failed. IP was never designed to handle such a malicious attack. Even if IP could be successfully modified, there is still the issue of deployment and cooperation among ISPs.

In recent years, there has been a trend toward deploying overlay networks on top of IP. Overlay networks bring increased resilience, distribution of load, and ease of locating data and services. Also, deployment is not an issue since communication between overlay nodes is done by standard IP traffic, making cooperation among ISPs unnecessary.

B. Zhao et al. [39] proposed a route-through overlay that tunnels traffic between existing legacy applications through a structured peer-to-peer overlay network. Standard IP traffic can be tunneled through a wide-area overlay efficiently with Brocade [40]. Once the traffic has been routed inside the overlay, we can protect the traffic.

In this paper we present a novel, distributed approach for isolating DoS attacks. A study by Y Chen et al. found that a distributed object location service (OLS) is more resilient to DoS attacks than a centralized OLS [6]. We propose using a distributed approach to DoS defense in the general case. Our proposed architecture leverages the properties of a wide-area overlay network to isolate clusters of attackers while denying access to a minimal amount of legitimate users. This is done by collaborating with other members of a structured peer-to-peer network, which is inherently collaborative. This solution is also helpful with flash crowds, which are sudden increases of legitimate traffic at a particular site. Flash crowds result in localized congestive flows and cause periods of high delay or loss. They carry many of the same signatures of a DDoS, but are legitimate traffic and are not handled well by other solutions [15].

This paper makes the following contributions. First, we propose an enhancement to Brocade [40] (Section 3.1) to improve the infrastructure for wide-area overlay routing. We propose that supernodes publish the identifications of nodes for which using the secondary overlay would be useful. This would have the effect of treating the node to be routed to as an object with the same locality awareness of the overlay network. Second, we propose some techniques to detect and suppress a DoS attack. The detection techniques range from local detection based at the target, to a global analysis, to a distributed approach. The suppression techniques involve isolating the attack as close to the source as possible. Third, we present a simulation environment to model the network and attackers. Most importantly, we present four DoS attack scenarios based on real attacks that have occurred in the Internet. Then we seperately analyze the effectiveness of each technique for detecting and suppressing the attack and analyze trade-offs such as the overhead of each technique and the amount of false positives. Finally, we analyze the overall effectiveness of our full system.

The rest of this paper is organized as follows. In Section 2 we discuss related work. In Section 3 we explain the motivation for using a structured peer-to-peer overlay network and how it fits into the world. In Section 4 we provide a detailed discussion of our DoS defense architecture. We describe our simulation environment in Section 5. In Section 6 we analyze the components of our architecture and then evaluate the performance of our full design. In Sections 7 and 8 we analyze the trade-offs and discuss future work. Finally, we conclude in Section 9.

2 Related Work

The need to protect against and mitigate the effects of DoS attacks has been recognized by both the commercial and research world for some years. There has been much work done on detecting attackers and isolating attack streams.

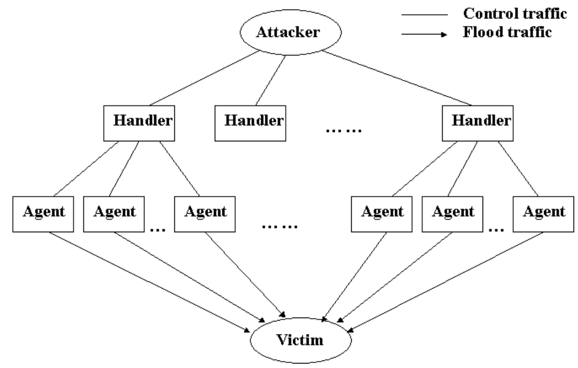


Figure 1: *Structure of a DDoS attack.* One or more attackers control handlers and each handler controls multiple agents. Handlers and agents are extra layers introduced to increase the rate of packet traffic as well as to hide the attackers from view. Each agent can choose the size and type of packets as well as the duration of flooding. While the victim may be able to identify some agents and have them taken off-line, the attacker can monitor the effects of the attack and create new agents accordingly.

2.1 DoS Attack Analysis

There has been much work done on analyzing the patterns and methodology of DoS attacks.

A recent study [23] observed 12,805 attacks on more than 5,000 distinct Internet hosts in more than 2,000 distinct DNS domains over a three week period. Most attacks are short with 90% lasting less than an hour. A DoS attack response must be quick; much quicker than picking up the phone and calling system administrators in autonomous systems.

Distributed DoS (DDoS) attacks are a flooding attack of many attacking hosts (agents) with distributed and coordinated control. Figure 1 shows the structure of a DDoS attack; one or more attackers control handlers and each handler controls multiple agents. Handlers and agents are extra layers introduced to increase the rate of packet traffic as well as to hide the attackers from view. Each agent can choose the size and type of packets as well as the duration of flooding. While the victim may be able to identify some agents and have them taken off-line, the attacker can monitor the effects of the attack and create new agents accordingly [12].

J. Jung, B. Krishnamurthy, and M. Rabinovich analyzed some signatures that differentiate flash events (FE) (also called flash crowds) and DoS [16]. During FE, the average number of requests per client is fairly constant. On the contrary, during a DoS attack, attacking clients have a much higher request rate than normal clients. The main difference between FE and DoS is that during FE, new clients arrive disproportionately from existing clusters of nodes. A cluster is an aggregate of nodes belonging to the same administrative domain, or more loosely defined as group of nodes that are local to each other in the Internet. This is not true in the case of DoS; the surge in traffic occurs because of new clusters joining the attack. FEs do not exhibit any surge in new cluster arrivals.

On October 21, 2002, all 13 of the DNS root servers were the target of a DDoS attack. Roughly two-thirds of the servers were disabled or severely hampered by the attack [27, 22]. Defenders count on having time to respond to an assault. In this case the attack abruptly ended after about an hour, so any defense that administrators could have attempted was meaningless. This emphasizes the need for a DoS attack response to be swift.

2.2 IP Level Solutions

In the past, proposals to defend against DoS attacks focused on modifying the IP level infrastructure. It is worth reviewing some of these proposals to see what insights we can be draw from them.

There are some techniques for tracing anonymous packet flooding attacks in the Internet back towards their source addresses. Savage et al. describe a general purpose traceback mechanism based on probabilistic packet marking in the network. This approach allows a victim to identify the network path(s) traversed by attack traffic without requiring interactive operational support from Internet Service Providers (ISPs) [31]. Partridge et al. present a hash-based technique for IP traceback that generates audit trails for traffic within the network, and can trace the origin of a single IP packet delivered by the network in the recent past [21]. Bellovin, Leech, and Taylor propose a new ICMP message, emitted randomly by routers along the path and sent to the destination. When forwarding packets, routers can, with a low probability, generate a Traceback message that is sent along to the destination. With enough Traceback messages from enough routers along the path, the traffic source and path can be determined [2]. Song and Perrig present the Advanced Marking Scheme and the Authenticated Marking Scheme, which allow the victim to trace back the approximate origin of the spoofed IP packets. In contrast to previous work, these techniques have significantly higher precision (lower false positive rate) and lower computation overhead for the victim to reconstruct the attack paths under large scale DDoS attacks. Furthermore the Authenticaed Marking Scheme provides efficient authentication of routers' markings such that even a compromised router cannot forge or tamper markings from other uncompromised routers [33].

Some techniques have also been developed to suppress attack traffic. Ioannidis and Bellovin present pushback which adds functionality to each router to detect and preferentially drop packets that probably belong to an attack. Upstream routers are also notified to drop such packets in order that the router's resources be used to route legitimate traffic [15]. Mahajan et al. discuss mechanisms for detecting and controlling high bandwidth aggregates such as DoS attacks and FEs. Their approach involves both a local mechanism for detecting and controlling an aggregate at a single router, and a cooperative pushback mechanism in which a router can ask adjacent routers to control an aggregate upstream [20].

2.3 Overlay Solutions

In the last few years, there has been some initial work using overlay networks to defend against DoS attacks. This work has, so far, not exploited the true collaborative and distributed power of peer-to-peer overlay networks.

Paul Mochapetris suggests a more replicated DNS as a solution to an attack on the directory service [22]. Unfortunately, such an approach is still vulnerable and has other drawbacks. Y. Chen et al. investigated the resilience of object location services (OLS) to DoS attacks [6]. They found that a distributed OLS was more resilient than a centralized or even replicated OLS. Also, the locality properties of the distributed OLS, Tapestry, helped maintain performance under severe attacks.

N. Daswani and H. Garcia-Molina were some of the first to begin investigating the effects of DoS on P2P networks [11]. The P2P network they chose to look at is Gnutella, which is very vulnerable to attack. In their paper, they proposed some load balancing to help mitigate the effects on Gnutella. We chose to look at Tapestry, whose inherent locality properties already mitigate the effects.

Some solutions are geared toward protecting specific nodes. A. Keromytis, V. Misra, and D. Rubenstein proposed a Secure Overlay Service (SOS) to protect critical emergency services from DoS attack [17]. The goal of the SOS is to allow secure communication between a small number of pre-approved sources and a particular destination. The SOS protects the destination by allowing only secret servlets to contact the destination.

B.G. Chun, P. Mehra, and R. Fonseca propose DAM: a DoS attack mitigation infrastructure [8]. They simply use a redirection scheme with only one hop on an overlay network. If a server is attacked, the redirection overlay balances the load by using multiple server replicas. The basic premise of their 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. Our approach differs in that we leverage the structure, locality, and collaborative properties of a peer-to-peer network to isolate the DoS attack. These properties will be explained in later sections.

B.Chun, J. Lee, and H. Weatherspoon propose Netbait: a Distributed Worm Detection Service [7]. Netbait uses a collective view of a geographically distributed set of machines to detect Internet worms. Queries in Netbait are processed in parallel by distributing them over dynamically constructed query processing trees built over Tapestry. Such a distributed, global view of the network, where each node in the tree is the root of its own tree, would also be useful in detecting and suppressing DoS attacks.

3 Using an Overlay Network to Prevent DoS

In this section, we explain the motivation for using a structured peer-to-peer (P2P) overlay network for defense against DoS attacks. It is difficult to detect and suppress DDoS effectively with traditional IP networks. First, IP was designed as a best effort service and not designed to handle such a malicious attack. Second, deployment across the wide-area will be stifled by lack of cooperation among ISPs. Third, intra-ISP recovery with BGP takes too long to be useful in stopping a DoS attack near the source in order to alleviate most legitimate requesters of a service. A wide-area response is essential.

Any service that desires protection from our DoS defense techniques must use an overlay network. An overlay network must be deployed over the wide-area. Wide-area deployment is eased because participation from ISPs is not needed. Nodes can simply be set up troughout the Internet and used transparently to the ISP because inter-node communication on the overlay is done via standard IP traffic. An overlay can make the network more resilient because the overlay can adapt within seconds to an event across the network, as we will demonstrate later.

We assume the majority of nodes in the overlay network can be trusted and that a compromised node cannot forge or tamper with messages from other uncompromised nodes [33].

We follow with an overview of how a route-through overlay can be used to allow nodes outside of the overlay to leverage its properties. This will give a big picture overview of how our work fits into the world. Then we focus on the overlay network with a brief discussion of structured P2P overlay networks and their basic properties. This is followed by a description of the design of Tapestry, the specific P2P overlay network that we use in our design. Finally, we focus in on the properties of structured P2P overlays we use to defend against DoS attacks.

3.1 Wide-Area Route-Through Overlay

A route-through overlay will allow nodes outside of the overlay to leverage its properties. This will allow a service to be on any node in the Internet and still access the overlay. ISPs can offer the overlay as a value-added service to their customers who want to deliver a reliable service. The average end user does not even need to know that an overlay is being used. The overlay can even be deployed on an isolated IP network, eliminating vulnerability to an IP attack. Also, traffic can be tunneled through a wide-area overlay efficiently with Brocade [40].

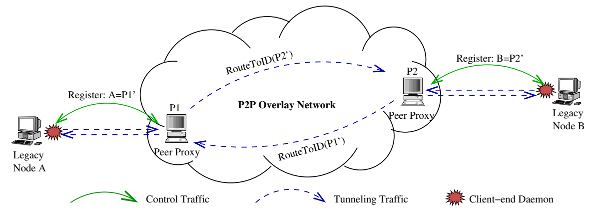


Figure 2: *Route-through overlay.* Node A registers with P1 and node B registers with P2. Messages from A to B are sent from A to P1 which encapsulates the messages and sends them to SHA-1(IPB) on the overlay. These messages arrive at P2 and are forwarded to node B. Messages are sent from B to A in a similar manner.

3.1.1 Route-Through Overlay

For nodes outside the overlay to leverage the overlay's properties, a route-through overlay can be used to tunnel traffic through the overlay with the use of nearby overlay proxies (Figure 2). First, a node (A) chooses and registers with a nearby overlay node as its proxy. Actually, node (A) does not even have to know about P1. When (A) tries to access a service at (B), (B) can install a peice of software on (A) that directs it towards P1, or P1 can be at the edge of an ISP that provides the overlay service and simply picks up the messages from (A). The proxy node (P1) assigns node (A) a proxy identifier in the identifier space of P1, such that P1(A) is the closest unassigned identifier to P1 inside its range. For example, a Tapestry proxy would assign identifiers beginning with P1-1. Under Tapestry routing rules, if no exact match can be found, the message is routed to the node with the next higher nodeId. In this case, messages to P1-1, P1-2, etc. would be routed to P1 [39].

This approach treats a node (A) as an object to be located. Once a node obtains its proxy identifier, the proxy makes a mapping between a hash of the legacy node's IP address and its new proxy identifier (<SHA-1(IP_A), P1(A)>) available to the entire network. The proxy will either use the put call on a protocol supporting the DHT interface, or store the local mapping and use the publish (Section 3.3.2) call on a protocol supporting the DOLR [10]. Node (A) can then begin to send messages to a destination (B) with IP_B by sending standard IP messages to its overlay proxy P1. Node (B) has similarly registered with a proxy P2 and has been published onto the network. Node (B) may even have a dedicated connection to proxy P2 if node (B) is in the business of delivering a reliable service. When P1 receives the messages from (A) to (B), P1 encapsulates messages to IP_B and sends them to SHA-1(IP_B) on the overlay. These messages arrive at P2 and are forwarded to node (B). Messages are sent from (B) to (A) in a similar manner (Figure 2).

Such a route-through overlay can be used to isolate all traffic for a server to the overlay. The client sends the name resolution request to the proxy which resolves the name internally to the overlay. Packet filters can also be used to only allow the server to communicate with its proxies. A node (B) using a route-through overlay via a proxy node (P2) has the added benefit of being abstracted as an object on the overlay (Section 3.3.2). This allows (B) to be protected from a direct attack to a node. If a node is under attack, it can fork off the services it is delivering to another proxy node. In the example from Section 3.4.1, directly attacking nodes 4228 and AA93 would make object 4378 unavailable.

3.1.2 Wide-Area Overlay Network

To improve point to point routing performance on an overlay across the wide-area Brocade [40] proposes a secondary overlay on top of the existing infrastructure. The premise is to find nodes which have high bandwidth and fast access to the wide-area network, and tunnel messages through an overlay composed of these supernodes. By using this secondary overlay, messages would emerge near the local network of the destination node.

The work in Brocade lacked an adequate method for locating a supernode to tunnel the messages on the secondary overlay. We propose a solution that would approximate the effectiveness of the directed method used in Brocade, but without the drawbacks of state maintenance and supernode failure susceptibility. Our approach has the simplicity of the naive approach in Brocade, but has much better performance. We propose that supernodes publish the identifications of nodes for which using the secondary overlay would be useful. This would have the effect of treating the node to be routed to as an object with the same locality awareness (Section 3.3.2 Figure 5). The closer a client is to an object, the sooner its message will likely cross paths with the object's publish path. The client's message is therefore likely to go to the local supernode, which is publishing the destination node as an object. Even if the message does not arrive at the local supernode, it will likely cross multiple autonomous systems and is likely to arrive at a supernode that is helpful.

If an isolated IP network is not feasible for the entire overlay network, it can be used for the secondary overlay only. This would be more practical since the secondary overlay has less redundancy and a greater need for high bandwidth and fast access to the wide-area.

3.2 Structured P2P Overlays

A structured peer-to-peer (P2P) network is necessary for our P2P overlay network (Figure 2). Structured overlays conform to a specific graph structure that allows them to locate objects by exchanging O(log N) messages, where N is the number of nodes on the overlay [38].

A node represents an instance of a participant in the overlay (one or more nodes may be hosted by a single physical IP host). Participating nodes are assigned uniform random *nodeIds* from a large identifier space. Application specific objects are assigned unique identifiers called keys, selected from the same ID space. For example, Tapestry [41], Pastry [30], and Chord [34] use a circular identifier space of n-bit integers modulo 2^n (e.g. n=160 for Chord and Tapestry, n=128 for Pastry).

Overlays dynamically map each object key to a unique live node, called its root. These overlays support routing of messages with a given key to its root node [10], called Key-Based Routing (KBR). To deliver messages efficiently, each node maintains a routing table consisting of the nodeIds and IP addresses of the nodes to which the local node maintains overlay links. Messages are forwarded across overlay links to nodes whose nodeIds are progressively closer to the key in the identifier space. Each system defines a function that maps keys to nodes. For example, Tapestry maps a key to the live node whose nodeId has the longest prefix match. If a digit cannot be matched exactly, the node with the next higher nodeId is chosen.

Existing protocols all support KBR, but differ significantly in performance. These protocols carry an overhead associated with the fact that one logical hop on the overlay network can have multiple IP hops. The ratio of IP hops to logical hops is known as stretch. These multiple IP hops can add significant delay, especially if they transit the wide-area. Such protocols must have some awareness of the structure of the underlying network to minimize the overhead. Protocols such as Pastry and Tapestry minimize latency between nodes when constructing routing tables [5]. With each hop, a message is routed to the node with the lowest network latency from all nodes that satisfy the routing constraint. Such proximity neighbor selection results in low stretch (about 2 to 5 depending on the topology) and good local route convergence. The effect is reduced latency and bandwidth consumption in the wide-area.

3.3 Tapestry Overview

We chose to use Tapestry [14, 41] as our structured P2P overlay because we have an implementation available and are familiar with it. Tapestry is one of several recent projects exploring the value of wide-area Decentralized Object Location and Routing (DOLR) services [30, 34, 25]. It enables messages to locate objects and route to them across an arbitrarily-sized network, while using a routing table with size logarithmic to the network size at each node. As a location service, Tapestry provides network applications with efficient routing of messages to locations of named objects. Such functionality in Tapestry and related projects has given rise to a new class of wide-area applications [19, 29, 3, 42].

The key distinction between Tapestry and other DOLR infrastructures is its support for point-to-point routing between named nodes. Tapestry uses similar mechanisms to the hashed-suffix mesh introduced by Plaxton, Rajaraman and Richa in [24]. Tapestry routes messages between named

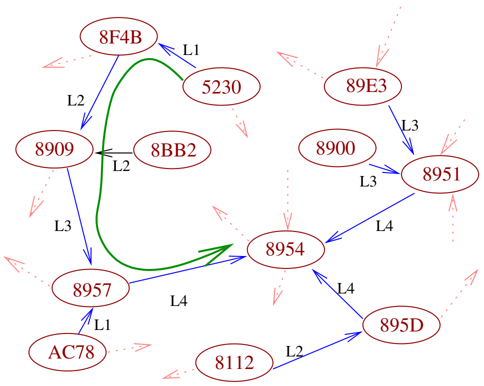


Figure 3: *Tapestry Routing Example.* Here we see the path taken by a message originating from node 5230 destined for node 8954 in a Tapestry network using 4 hexadecimal digit names (65536 nodes in namespace).

nodes across an arbitrarily-sized network using a routing map with size logarithmic to the network size. In practice, Tapestry provides a delivery time within a small factor of the optimal delivery time [41]. Previous work has leveraged Tapestry routing for application-level multicast [42] and suggested performance enhancements for wide-area operation [40].

Each Tapestry node or machine can take on the roles of server (where objects are stored), router (which forward messages), and client (origins of requests). We assume that Tapestry nodes, especially routers and servers, are well-connected over high bandwidth links. Nodes in Tapestry have names, Globally Unique IDentifiers (GUIDs), independent of their location and semantic properties, in the form of random fixed-length bit-sequences represented by a common base (e.g., 40 Hex digits representing 160 bits). The system assumes entries are roughly evenly distributed in the node ID namespace, which can be achieved by using the output of secure one-way hashing algorithms, such as SHA-1 [28].

3.3.1 Prefix-based Routing

Tapestry uses local routing maps at each node, called neighbor maps, to incrementally route overlay messages to the destination ID digit by digit (e.g., 8***=>89**=>895*=>8954 where *'s represent wildcards, as shown in Figure 3). This approach is similar to longest prefix routing in the CIDR IP address allocation architecture [26]. A node has a neighbor map with multiple levels, where each level represents a matching prefix up to a digit position in the ID. Level-1 edges from a given node connect to the 15 nodes closest (in network latency) with different values in the lowest digit of their addresses. Level-2 edges connect to the 15 closest nodes that match in the lowest digit and have different second digits, etc.

When routing, the n^{th} hop shares a prefix of at least length n with the destination ID. To find the next router, we look at its $(n+1)^{th}$ level map, and look up the entry matching the value of the next digit in the destination ID. Assuming consistent neighbor maps, this routing method guarantees that any existing unique node in the system will be found within at most $Log_b N$ logical hops, in a system with N nodes using IDs of base b. Because every single neighbor map at a node assumes that the preceding digits all match the current node's prefix, it only needs to keep a small constant size, b, entries at each route level, yielding a neighbor map of fixed constant size $bLog_b N$.

3.3.2 Object Location

A server (with GUID, S) storing an object periodically advertises (publishes) this object by routing a publish message to the objects root (Figure 4) [41]. The object's root (with GUID, O_R) is the node that would be routed to when routing to the objects GUID (O_G). Each node along the publication path stores a pointer mapping, $<O_G$, S$>$, instead of a copy of the object itself. When there are replicas (copies) of an object on separate servers, each server publishes its copy. Tapestry nodes store location mappings for object replicas sorted in order of network latency from themselves. A client locates an object (O) by routing a message to O_R (Figure 5). Each node on the path checks whether it has a location mapping for O and, if so, it redirects the messages to S. Otherwise, it forwards the message onwards to O_R which is guaranteed to have a location mapping if the object exists.

Locality Awareness The closer in network distance a client is to an object, the sooner its queries will likely cross paths with the objects publish path, and the faster they will reach the object. Since nodes sort object pointers by distance to themselves, queries are routed to nearby object replicas.

3.4 How the Overlay is Used

We now discuss how we use the properties of a structured peer-to-peer overlay network to defend against DoS attacks. We leverage the locality, structure, and collaborative properties to isolate DoS attacks.

3.4.1 Locality

If the attack can be traced back to a region of the network, a replica (copy) can be placed nearby to isolate the attack and protect the rest of the network. Therefore, it is important to know if an attack will stay local to a particular replica (Section 3.3.2). Figure 5 shows that the attack to object 4378 coming from node 4664 would go to node 4228's replica. Nodes 4B4F and 57EC would not be affected since their requests never use the affected region; however, nodes would still be vulnerable to direct attacks. The solution to this problem is presented in Section 3.1.1.

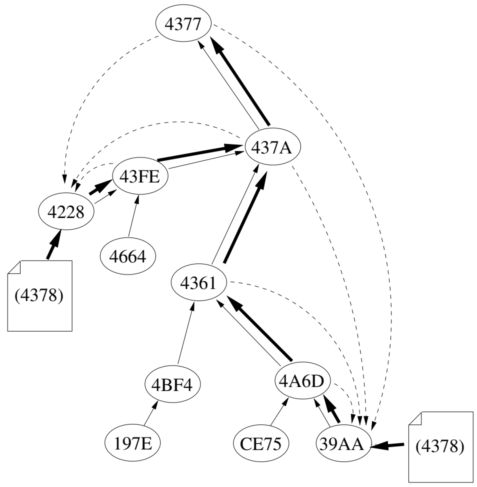


Figure 4: *Publication in Tapestry.* To publish object `4378`, server `39AA` sends publication request towards root, leaving a pointer at each hop. Server `4228` publishes its replica similarly. Since no `4378` node exists, object `4378` is rooted at node `4377`.

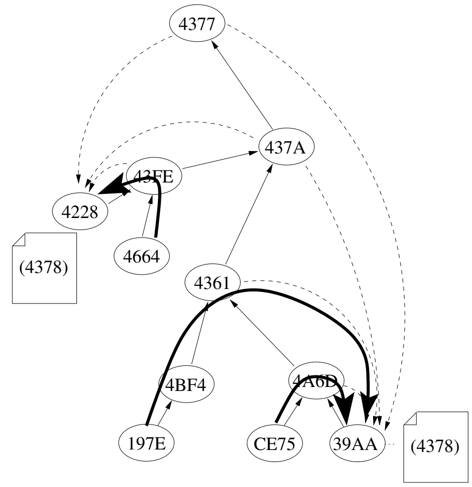


Figure 5: *Routing in Tapestry:* Three different location requests. For instance, to locate GUID `4378`, query source `197E` routes towards the root, checking for a pointer at each step. At node `4361`, it encounters a pointer to server `39AA`.

Replicas in the Network In Tapestry, replicas are dispersed throughout the network. Locality properties keep DoS attack isolated to a certain region, far away from unaffected nodes. There are pointers to replicas on every node along the path from the replicas to the root. It would be very difficult to knock out all of those nodes to prevent the location of the replica [6]. In Pastry [30], Chord [34], and CAN [25] replicas are stored on neighboring nodes and are still vulnerable. The attack will just overwhelm the backup node(s) once the root node is overwhelmed. Also, replication of all data on nodes near the root is very expensive. In contrast, Tapestry arrives at data in a distributed way, as mentioned previously. Future research should develop replication schemes to prevent correlated failures.

3.4.2 Structure

The protocols used have an awareness of the structure of the underlying network to minimize the overhead. Pastry and Tapestry minimize latency between nodes when constructing routing tables [5]. With each hop, a message is routed to the node with the lowest network latency from all nodes that satisfy the routing constraint. Such proximity neighbor selection results in low stretch (about 2 to 5 depending on the topology) and good local route convergence. The effect is reduced latency and bandwidth consumption in the wide-area. Another effect is that the overlay structure will mimic the structure of the underlying network.

Path Convergence The convergence properties of the algorithms are important in regard to their effects on a DoS attack. In Pastry [4] and Tapestry [41], when the distance between source nodes is small, the paths are likely to converge quickly. The closer two nodes are, they share a greater percentage of the distance to the destination. The proximity of the nodes is important because it can be exploited by an attacker. An attack will affect any legitimate traffic from nearby nodes to the target (or otherwise sharing the path). However, since nearby source nodes will converge paths quickly, DoS attacks are easier to trace and shut down in a more aggregate method.

3.4.3 Collaborative

P2P networks are inherently collaborative for the purpose of locating other users and locating data. This collaboration can be extended for other goals, such as suppressing DoS attacks. This lifts the impossible burden from a centralized location and distributes the duty to all members of the network. Participants are responsible for their local area. P2P networks use local discovery algorithms to discover their local nodes [14]. P2P networks can collaborate to defend against DoS attacks in a similar manner.

4 DoS Defense Architectures

In this section, we describe our techniques to detect and suppress a DoS attack. These techniques range from traditional techniques proposed as IP level solutions to some new proposals that leverage the properties of structured peer-to-peer (P2P) networks. We explain some trade-offs such as the overhead of each technique.

4.1 Detecting the Attack

The first step in suppressing a DoS attack is detecting who the attackers are and/or detecting the region(s) the attack is originating from. These detection techniques range from local detection based at the target, to a global analysis, to a distributed approach.

4.1.1 Local Detection

We have implemented a mechanism that has successfully identified the attackers by keeping running average of the request rate. During flash events (FE), the average number of requests per client is fairly constant. But during a DoS attack, attacking clients have a much higher request rate than the average [16]. Our local detection mechanism keeps a record for each requester. Each requester record keeps the time of the last request and the request rate, stored as delay with an initial value of 1000ms. When a new request from a requester arrives, the delay from the last request is calculated from the request record and the request rate is updated via a low pass filter (D_0=0.9D_{-1}+0.1$Dnew$). The average request rate is periodically updated from the requester records. If a requester has a major deviation from the average request rate, we mark that requester as an attacker.

A draw back to local detection is that an attacker could simulate a flash event by having many attacking hosts attack at a normal rate. So we need a solution that can obtain information from nodes on the request paths to determine what region(s) of the network the attack is originating from. Another serious drawback is that the target may be overwhelmed and cannot initiate the suppression. We therefore need a solution that pushes out the detection to nodes before the target that are not overwhelmed.

4.1.2 Clustering Analysis

To address the problems of many attacking hosts and an overwhelmed target, one requires a detection architecture with global knowledge and a distributed ability to suppress an attack.

Due to how attacking hosts are selected, DoS attacks have certain properties that can be exploited to detect and suppress the attacks. During FE, new clients come disproportionately from existing clusters of nodes. This is not true in the case of DoS; the surge in traffic occurs because of new clusters joining the attack. FEs do not exhibit any surge in new cluster arrivals (Section 2.1) [16].

An intruder finds one or more systems on the Internet that can be compromised and exploited. This is generally accomplished using a stolen account on a system with a large number of users and/or inattentive administrators, preferably with a high-bandwidth connection to the Internet (many such systems can be found on college and university campuses) [18]. It would be much more difficult to mount a DDoS attack with many attacking hosts from diverse areas. An attacker would have to compromise multiples more systems to make up for not using multiple machines on each system.

The routing process can be viewed as a tree, with messages from leaf nodes traversing intermediate nodes en route to the root (Figures 4 and 5). By observing an attack from a global vantage point, we can identify machines and clusters of machines, branches of the tree, that would otherwise have been difficult to detect, while maintaining service to as many legitimate requesters as possible. This is illustrated in Figures 6 and 7. S is the server which requesters(R) are trying to access. The request traffic is shown by a dashed line. A is the attacker which tries to prevent access to S by flooding it with request traffic. The attack traffic is shown by a bold line. With no defense (Figures 6), the attack traffic will overwhelm the server(S) as well as any link along the path from the attacker(A). With our defense scheme, we can analyze that the attack is coming from N1 and N3 but not N4. The attack can be isolated at N3 with minimal effect on legitimate requesters and most of the network (Figures 7). In this case, R1 is isolated along with A, but R2-R6 are no longer being inhibited by A's attack.

Global Clustering Analysis (GCA)

The goal is to know everything all the time. Each intermediate node (N) along the path from the source (S) to the destination (D) will keep track of any S-D pair (Figure 8 SendHBdestList). Each node N will send periodic heart beat (HB) messages to each destination to ensure that it has not been overwhelmed by an attack (line 9). If there is a missed HB from a destination (line 2), the numMissedHB for that destination is incremented (line 3). If there are too many missed sequential HBs to a particular destination (line 4) in a node's list, then the node will see if it can detect an attack and try to suppress it (line 5). If a HB from a destination is received the node marks the HB as received (line 7) and the total number of missed HBs for that destination is reset.

When D receives a HB, D adds N to a cluster list/graph (G) with sublists of Ss that each N has (this can be assembled into a graph). D periodically goes through G and checks each N's list of Ss to see if they are attackers (Figure 8 CheckClusterGraph). We check for attackers by checking attack rate, as in local detection, and whether the nodes in a particular cluster began sending at the same time [16]. Each N in G keeps the percentage of its Ss that are attackers (at each node in the graph) (line 2). If N's percentage of attackers is too high (line 3), N is notified to suppress the attack (line 4). Although this solution maintains a global view of the network, it does not scale well because of the number of states and HB messages [35].

Distributed Clustering Analysis (DCA)

We attempt to approximate GCA using a distributed approach with local knowledge (Figure 9).

Every node (N) continually checks traffic on each incom-

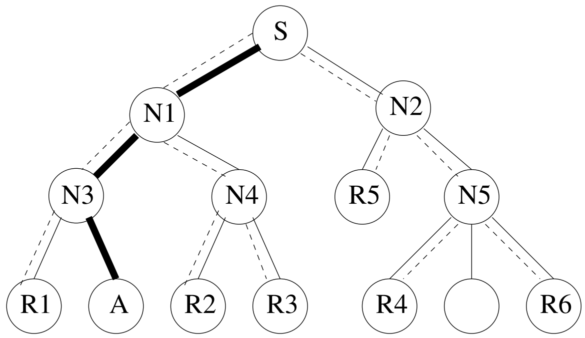


Figure 6: *DoS attack with no defense.* S is the server which requesters(R) are trying to access. The request traffic is shown by a dashed line. A is the attacker which tries to prevent access to S by flooding it with request traffic. The attack traffic is shown by a bold line. With no defense, the attack traffic will overwhelm the server(S) as well as any link along the path from the attacker(A).

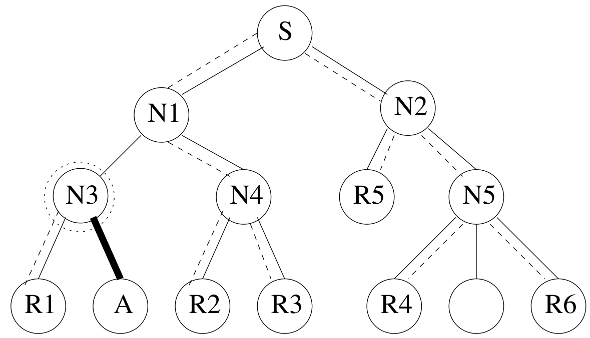


Figure 7: *DoS attack with clustering analysis defense.* With our defense scheme, we can analyze that the attack is coming from N1 and N3 but not N4. The attack can be isolated at N3 with minimal effect on legitimate requesters and most of the network. In this case, R1 is isolated along with A, but R2-R6 are no longer being inhibited by A's attack.

```
   method SENDHBDESTLIST (destlist)
1  for dest ∈ destlist
2      if ! dest.gotHBresponse
3         dest.numMissedHB++
4         if dest.gotHBresponse > 3
5            DETECTSUPPRESSATTACK(dest)
6      else
7         dest.resetGotHB
8         dest.resetNumMissedHB
9      SENDHB(dest, SourcesToDest)
   end SENDHBDESTLIST

   method CHECKCLUSTERGRAPH (clustergraph)
1  for c ∈ clustergraph
2      cAttackers ← CHECKPERCENTOFATTACKERS(N)
3      if cAttackers > AttackerThreshold
4         SUPPRESS(N)
   end CHECKCLUSTERGRAPH
```

Figure 8: *Pseudo-code for* Global Clustering Analysis(GCA).

ing link to see if it passes a certain threshold (CheckTrafficOnIncomingLink). When N notices a great deal of traffic on an incoming link (line 2), it checks if requesters are going disproportionately to a particular destination(s) (line 3). N uses the back-pointer on the congested link to contact the previous node to see if it has any congested incoming links (line 4), and also checks the congested destination(s). In case the heavy incoming link does not handle the problem, either because the node misbehaves or simply does not detect the problem, we set a timer with an appropriate delay (10000ms) (line 5).

That previous node (N') performs as N (CheckForHeavyInLinkMsg) did except with a lower threshold (c<1), since N' has been notified that it is a problem (line 3). The timer in this case is scaled down by a small factor (line 5). This concept is the same as described in trace-back as described in Section 4.2.1.

When the timer expires (CheckForHeavyInLinkTimer) and the problematic heavy incoming link is still a problem, the node will suppress the attack on its own (line 3). If a node suppresses the attack, then it must notify the node that notified it (line 4). For example, if N' suppresses the attack, N' must notify N so that N will not try to suppress again.

The notifying node (N) will watch for such a message (CheckForHeavyInLinkResponse). Using this method, N builds a map of congested links, with the percent of traffic that the congested destination(s) uses (line 3). Each node that is part of this map, checks what percent of the requesters to congested destinations are attackers. If the map on a node shows that the congested incoming links divide in previous nodes, meaning that heavy traffic is coming in from multiple incoming nodes, then that node is considered a convergence point. The convergence point then suppresses the traffic to the particular destination GUID.

This approach minimizes the number of states maintained, while it keeps false positives at a minimum. The logic behind this approach is to suppress the attack as far back as it causes a problem.

4.2 Notification of Suppression

Once an attacker or group of attackers has been detected, the attacker(s) must be suppressed. This can be accomplished by notifying a node on the attack path to suppress the attack. There are a number of methods to accomplish this notification.

```
method CheckTrafficOnIncomingLink (AllInLink)
for inlink ∈ AllInLink
    if (traffic on inlink) > TrafficTreshold
        CheckOutLink(AllOutLink)
        SendHeavyInLinkMsg(inlink)
        HeavyInLinkTimer(10000ms)
end CheckTrafficOnIncomingLink

method CheckForHeavyInLinkMsg
IncomingMessageHandler
    if msg.type == HeavyInLinkMsg
        if (traffic on inlink) > (TrafficTreshold*c)
            SendHeavyInLinkMsg(inlink)
            HeavyInLinkTimer(0.9 * msg.timer)
            SendHeavyInLink(inlink)
            inlink = msg.source
end CheckForHeavyInLinkMsg

method CheckForHeavyInLinkTimer
IncomingMessageHandler
    if msg.type == HeavyInLinkTimerMsg
        Suppress(heavyinlink)
        SendHeavyInLinkResponse(msg.inlink,
            getGraph(msg.inlink))
end CheckForHeavyInLinkTimer

method CheckForHeavyInLinkResponse
IncomingMessageHandler
    if msg.type == HeavyInLinkTimerMsg
        AddGraph(msg.inlink, msg.graph)
end CheckForHeavyInLinkResponse
```

Figure 9: *Pseudo-code for* Distributed Clustering Analysis (DCA).

```
method CheckMessagesForAttackers
1 IncomingMessageHandler
2 if msg.source == attackerGUID
3   if msg.peer == attackerGUID
4     SuppressAttack(attackerGUID)
5   else
6     SendTraceBackMsg(msg.peer, attackerGUID, traceTimer)
7     SuppressionTimer(traceTimer)
end CheckMessagesForAttackers

method CheckForTraceBackMsg
1 IncomingMessageHandler
2 if msg.type == TraceBackMsg
3   attackerGUID = msg.attackerGUID
4   traceTimer = 0.9 * msg.traceTimer
5   CheckMessagesForAttackers(IncomingMessages)
end CheckForTraceBackMsg
```

Figure 10: *Pseudo-code for* Trace-Back.

4.2.1 Trace-Back

The trace-back code is described in Figure 10.

CheckMessagesForAttackers begins by checking incoming messages (line 1) to see if the message source is the detected attacker(s) (line 2). If so, we next check to see if the attacker(s) is the next node in the trace-back (line 3). If the attacker is the next node in the trace-back, we just suppress the attack without waiting for the timer (line 4). If the attacker is not the next node, the current node SendTraceBackMsg (line 6) to the next node in the attack path. The node then sets the SuppressionTimer (line 7), with traceTimer initially at 10000ms.

Each node is constantly running CheckForTraceBackMsg. When the SendTraceBackMsg is received, the node sets the attackerGUID and traceTimer. The node will then run CheckMessagesForAttackers (line 5). This will proceed following the trace-back path until the attacker is reached or the SuppressionTimer expires. Since one does not know the number of steps the trace-back will take, decreasing by a specific number will not work. The best choice is to decrease by a multiple (0.9) at each step (line 4). This works because it allows the trace-back to go an indefinite number of steps. Also, since at each step of the trace-back there is a shorter round trip from the start of trace-back to notification of isolation, a shorter timer is appropriate. If the timer runs out, keep track of the node that is non-compliant.

Tracking each attacker may not be necessary to stop the attack from a particular cluster. If multiple attackers are detected, the trace-backs of attackers from a common router are bundled in a single trace-back message. This will have the effect of tracing the clusters of attackers. This method will trace the attack as far as possible. Then the node at the convergence point suppresses the attack.

4.2.2 Cluster Notification

The target has identified a particular node at a convergence point of the attack and directly notifies the node at the convergence point to suppress the attack. Cluster notification is used by clustering analysis, or any other detection scheme in which the target would have some global view of the attack structure.

4.2.3 Distributed Notification

When the target has minimal information about the attacking nodes and their locations a distributed suppression notification is needed. This means that all nodes in the network are responsible for monitoring the traffic that passes through them. Any node that detects an attack must notify the appropriate node to suppress the attack or suppress the attack on its own. This would be the expected form of notification for distributed clustering analysis, which is an inherently distributed detection method.

4.3 Suppression

Once the attackers have been detected and the appropriate node to suppress the attack has been notified, the attack must be suppressed. Since a DoS attack stays local to the nearest replica (Section 3.3.2), the attack will be suppressed in that region. In a DDoS attack, this technique can be used on each of the attacking clusters. We evaluated two methods of suppression:

4.3.1 Simple Cut-off

The most simple, traditional method of suppressing a DoS attack is to drop messages to targeted destinations from identified attackers or offending incoming links. The suppressing node is notified to either drop messages to a particular destination/object or outgoing link.

4.3.2 Isolation

One can exploit the locality properties of a structured P2P network (Section 3.3.2) to isolate the attack to the source region. Placing a replica near the source of the attack will prevent nodes far away from being affected. The suppressing node can suppress messages to a particular object or destination (if destinations are treated as objects as discussed in Section 3.1).

This approach has the added benefit of being able to be used for data replication in a wide-area distributed storage system. For the purposes of the evaluation, any requests from legitimate requesters that go to a replica published by a suppressing node will be counted as false positives, even though the replica can be real instead of fake.

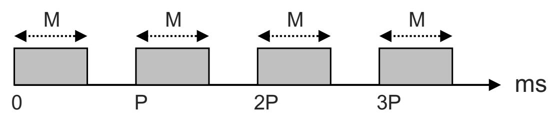


Figure 11: *Timeline of attack stream from each attacker.* Each attacker sends repeated bursts of M requests every P milliseconds. P is small enough to approximate a continuous attack. Also, M/P is a sufficient attack rate to overwhelm the target.

5 Simulation Environment

The study of algorithms to address such problems often involves simulation or analysis using a model of the actual network structure and applications [37]. It is generally more efficient to assess solutions using analysis or simulation, provided the model is a good abstraction of the real network and the participants.

5.1 Simulation Framework

We built our DoS defense mechanisms as a component for Tapestry nodes. Tapestry is implemented in Java, for portability and ease of prototyping. Internally, Tapestry nodes employ a staged event-driven architecture [36]. A stage is an independent, contained entity with its own incoming event queue. In an event-driven architecture the processing of each task is implemented as a finite state machine, where transitions between states in the FSM are triggered by external events. In the more traditional thread-driven approach the system uses a main thread which hands off tasks to individual task-handling threads, which step through all the stages of processing that task.

Each major software component (e.g. dynamic node management, router, DoSdefense) is a self-contained, event-driven stage. Stages communicate via an efficient publish/subscribe message dispatcher. Developers or administrators can add new stages to quickly add functionality. We implemented our DoS defense as one stage with modifications to the router stage. The DoSdefense and Router stages required a couple of request/response message pairs for communicating with each other. In total, the system consists of approximately 4000 lines of Java and a few additional files and scripts for testing. It was designed, built, and simulated in about 9 months.

Within our DoSdefense stage, we created 5 types of participants: node, requester, server, attacker, and master.

A node is similar to a normal Tapestry node except that it will participate in DoS defense as described in Section 4.

A requester is like a standard client. A requester sends a LocateObject message for a particular object onto the network and logs the time. When the object arrives at the requester, the requester calculates the request latency and stores it. The requester then sends another LocateObject message and so on. To keep from overwhelming the master node, which collects all information, and to smooth out the request latency, the requester calculates an average request latency every C requests. This average is sent to the master node. C can be varied to suit the simulation.

A server responds to requests for objects. When a server receives a LocateObject message, it checks if it has the object and, if so, sends the object to whoever requested it. We simulate computation time and queuing delays at servers with a delay of γ seconds for each message.

An attacker sends a large amount of requests for objects with the intention of overwhelming the server and making the object unavailable. An attacker node can individually be turned on and off and the attack rate can be varied. For simplicity of implementation, each attacker sends repeated bursts of M requests every P milliseconds. M and P can be varied. P is small enough to approximate a continuous attack. This implementation also allows for the flexibility

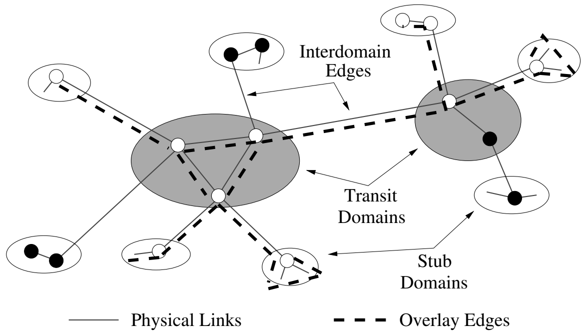


Figure 12: *A Transit-Stub Graph.* This topology mimics the structure of large networks observed in nature. Shown also is an overlay network which minimizes the number of interdomain edge crossings. Such overlays allow the topology discovery properties of the source filtering algorithm to minimize interdomain bandwidth consumption. See Section 5.2 for details.

to modify the attack rate as needed. We illustrate this in Figure 11. Also, M/P is a sufficient attack rate to overwhelm the target.

The master node controls the requesters and attackers, and collects all of the information in a centralized location.

5.2 Network Topology

Our simulator models the physical network as a graph, each edge of which has two values associated with it, α_{net} and β_{net}. To send a message along an edge takes $\alpha_{net} + s\beta_{net}$ seconds, where α_{net} is the time it takes one byte to cross an edge, β_{net} is the time to transfer one byte onto an edge, and s is the size of the message in bytes. To send a message along a path of more than one hop takes $\alpha'_{net} + s\beta'_{net}$ seconds, where α'_{net} is the sum of the α_{net} values for every edge along the path, and β'_{net} is the largest β_{net} value of any edge along the path. Our simulator does not simulate queuing delay in the network due to cross traffic on an edge. We simulate computation time and queuing delays at servers with a delay of γ seconds for each message.

Using this simulator, we constructed a physical network topology using the *transit-stub* model of GT-ITM [37]. The Internet today can be viewed as a collection of interconnected routing domains [9], which are groups of nodes that are under a common administration and share routing information. A primary characteristic of these domains is routing locality: the path between any two nodes in a domain stays entirely within the domain. Each routing domain in the Internet can be classified as either a stub domain or a transit domain [37]. An example transit-stub graph is shown in Figure 12. The purpose of transit domains is to interconnect stub domains efficiently; without them, every pair of stub domains would need to be directly connected. A transit domain comprises a set of backbone nodes, which are typically highly connected to each other. In a transit domain each backbone node also connects to nodes in a number of stub domains. A stub domain usually has one or more gateway nodes, which have links to the transit domains. Stub domains can be further classified as single or multi-homed. Multi-homed stub domains have connections to more than one other domain. Single-homed stubs connect to only one transit domain. In addition to this general layout, there are interdomain edges and several inter-stub domain edges in each graph. We augment the GT-ITM model with bandwidth numbers as follows. All stub to stub edges are 100 Mb/s, all stub to transit edges are 1.5 Mb/s, and all transit to transit edges are 45 Mb/s. These values were chosen to model Fast Ethernet, T1, and T3 connections, respectively.

Our simulations use transit-stub graphs with six transit domains of ten nodes each. Each transit node has seven stub domains of an average of twelve nodes each, yielding a total of 5,100 nodes per graph. The transit domains are fully connected to each other, and each pair of nodes internal to a domain are connected with probability 0.6. Each pair of stub nodes within a stub domain are connected with probability 0.3. We used GT-ITM to generate seven graphs given these parameters to insure that our results were not dependent on the particularities of any one graph.

On top of this physical network, we built a Tapestry overlay network as follows (Figure 12). We chose 700 of the total nodes in the graph uniformly at random without replacement and made them Tapestry servers. We then assigned node-IDs to these servers at random. While the Tapestry infrastructure includes algorithms for dynamically building a network, we assume in this work that the graph is built at the beginning of our simulation and does not change.

5.3 Attacker Model

There have been many analyses of DDoS attack methodology and the tools used in the attacks. The most prevalent model is one in which a master node controls and coordinates groups of attackers. This is demonstrated by specific tools used in DDoS attacks [12]. A master node controls handlers and each handler controls multiple attacking hosts. The handler layer exists to protect the identity of the master node. In our simulations, a master node coordinates the attacker(s), as in real attacks. For simplicity, our master node directly controls the attacker(s).

We outline 4 attack scenarios to analyze how our detection schemes respond to different situations. In each of these scenarios there is one predetermined target in the network which is under attack. Once the network starts up, 40 requesters begin requesting an object from the target and send the request latency to the master node every 5 requests. The master node will then send a message to each of the attackers telling them to begin their attack. For the simulations in Section 6.1 we are only interested in the performance of

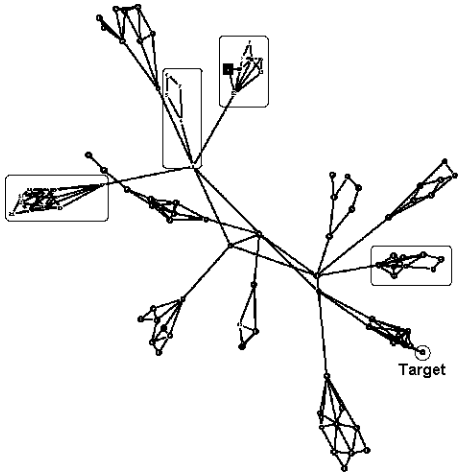


Figure 13: *Example of network layout of attack sceanio B.* We distribute groups of attackers (rectangles) throughout the network. Each group of attackers is in the same stub domain. In our simulations, we used 4 groups of 5 attackers each. The master node will send a message to each of the attackers at the same time, instructing them to start attacking the target (circle). The target is located in a stub domain different from the attackers.

the detection and suppression techniques during the attack. In Section 6.2, we are interested in comparing the latency before and after the start of the attack. So, in Section 6.2, the attack will begin 20 seconds after the requests in order to establish a baseline.

The first attack scenario (A) consists of one attacker.

In our second scenario (B), we grouped the attackers into clusters. We illustrate this in Figure 13. We distribute groups of attackers (rectangles) throughout the network. Each group of attackers is in the same stub domain. In our simulations, we used 4 groups of 5 attackers each. The master node will send a message to each of the attackers at the same time, instructing them to start attacking the target (circle). The target is located in a stub domain different from the attackers.

Our third attack scenario (C) is identical to attack B except that attack C is dynamic and clusters of attackers are turned on and off every few seconds during the attack. In our simulation, we used 4 groups of 5 attackers. The first group is turned on at 20 seconds. After 5 seconds the first group is turned off and the second group is turned on. After 5 more seconds, the second group is turned off and the third group is turned on, and so on. After the fourth group, we cycle to the first group. This is identical to real DDoS attacks (Section 2.1) [12].

An attacker may use a virus to distribute the slave program [32]. Recently there has been a number of self-propagating Internet worms [1] that have been able to infect a great number of systems before they were stopped. There is the possibility of that one of these worms could be used to set off a DDoS attack at a particular time to allow for a critical mass of nodes to be infected. To see how well we handle such an attack, our fourth attack scenario (D) will have attackers that are dispersed through the network instead of in clusters.

5.4 Experiment Descriptions

In our experiments, we place an object to locate on a server, which all requesters will try to access. We distribute 40 requesters uniformly throughout the network and each requester will request the object. A requester will request the object and wait for a response from the server. When the response arrives, the requester calculates the request latency and resends the request. Every 5 requests, the requester will take an average of the request latency and send this average to the master node which logs the data. We do not simulate cross traffic in the network because our simulator is not able to accurately model the queuing delay caused by cross traffic at links in the network.

In our experiments for Section 4, we analyze the effectiveness of each of the components of our design. Each of our analyses is performed under all 4 attack scenarios. We first do a quantitative analysis of our detection methods by evaluating the percent of attackers detected and the percent of legitimate requesters that are falsely identified as attackers (false positives). Second, we do a qualitative analysis of how fine grained the notification is for the purpose of suppressing the detected attackers. Third, we do a quantitative analysis of our suppression methods by evaluating the percent of identified attackers that are successfully suppressed and the percent of legitimate requesters that are falsely suppressed (false positives).

In our experiments for Section 6, we analyze the effectiveness of our full system at detecting and suppressing a DDoS attack. We evaluate the request latency under no defense and three defense schemes: the three detection schemes with their corresponding notification scheme (local-trace, global-cluster, distributed-distributed) each under the isolation suppression scheme. After a period of time (20 seconds), used to establish a non-attack baseline for the object request latency, the master node notifies the attackers to begin flooding object requests. Once the attack begins, the network will react appropriately under the defense scheme in use. All data for each node is logged to a local file. All data used in our results is logged by the master node.

6 Simulation

We evaluated our system as described in Section 5. In our simulations, we used seven different topologies; our graphs show the median value with error bars showing the minimum and maximum values. Our results show that a dis-

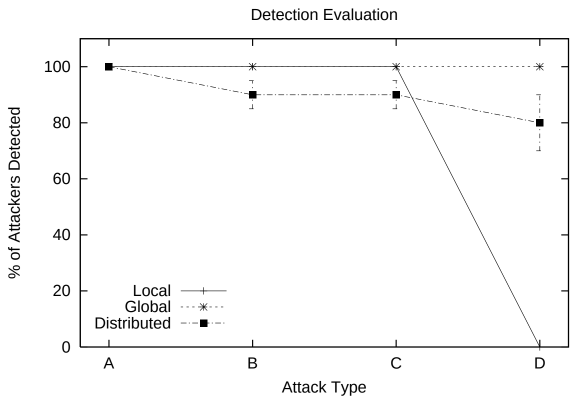


Figure 14: *Detection Evaluation.* % of Attackers Detected. Local detection fails under attack scenario D because it has no sense of the network topology. Global detection successfully detected all attackers under all 4 attack scenarios. Distributed detection is nearly as effective as global at detecting attackers.

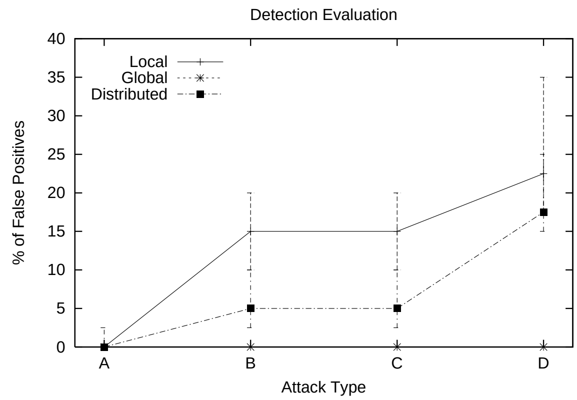


Figure 15: *Detection Evaluation.* % of False Positives. Local detection is susceptible to more dispersed attacks because it lacks awareness of network topology. Global detection has knowledge of the entire network topology and is therefore able to have no false positives. Distributed detection also has knowledge of the network topology, but on a local level. As a result, distributed detection is able to minimize false positives when the attack is concentrated from certain regions of the network. Unfortunately, distributed detection does not operate as well under a highly distributed attack.

tributed approach with local knowledge is almost as effective as a global approach.

6.1 Analysis of Components

6.1.1 Detection Evaluation

To evaluate the effectiveness of our detection methods, we analyzed two properties: percent of attackers detected (Figure 14) and percent of false positives (Figure 15).

Detecting attackers is important in order to suppress them and keep resources available for legitimate requesters. Local detection fails under attack scenario D because it has no sense of the network topology. Global clustering analysis (GCA) successfully detected all attackers under all 4 attack scenarios. Distributed clustering analysis (DCA) is nearly as effective as GCA at detecting attackers. DCA is only slightly less effective than GCA but the state that must be maintained and messages passed are minimal.

Minimizing the number of false positives is essential so that legitimate requesters are not alienated. Once again, local detection is susceptible to more dispersed attacks because it lacks awareness of network topology. The local detection false positive rate is unacceptably high. GCA has knowledge of the entire network topology and is therefore able to have no false positives. DCA also has knowledge of the network topology, but on a local level. As a result, DCA is able to minimize false positives with a manageable amount of state and messages when the attack is concentrated from certain regions of the network. Unfortunately, DCA does not operate as well under a highly distributed attack.

Notification Scheme	A	B	C	D
Trace-back	+	0	-	-
Cluster Notification	+	+	0	+
Distributed Notification	+	+	+	0

Table 1: *Notification Evaluation.* This is a quantitative analysis of the notification methods under the 4 attack scenarios. The +, 0, and - stand for good, acceptable, and poor respectively. We evaluate the notification scheme in terms of effectiveness and feasibility. A more detailed explanation of the reasoning can be found in Section 6.1.2.

6.1.2 Tracing Attackers Evaluation

This is a quantitative analysis of the notification methods under the 4 attack scenarios (Table 1).

Trace-back attempts to trace each individual attacker from the target, along the attack path, back to the attacker. Such a method would only be effective against one or at most a limited number of attackers. Because of the delay in trace-back, this method would not be effective against dynamic attackers. The overhead involved with tracing each individual attacker would make tracing widely dispersed attackers infeasible.

Cluster notification would be highly effective against most attack scenarios, except a situation where the attackers are dynamic. In the dynamic case, cluster notification must first go through a centralized detection process and no-

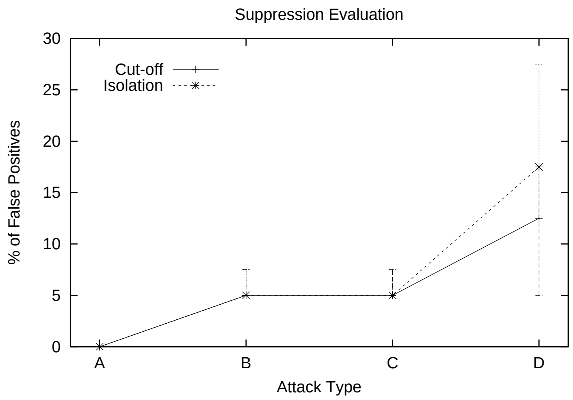


Figure 16: *Suppression Evaluation.* % of False Positives. Isolation performs only slightly worse than Cut-off under attack scenario D.

tification of the cluster each time there is a change, which could be on the order of seconds. Distributed notification, on the other hand, could detect a change locally and respond locally.

Distributed notification would be highly effective under all scenarios except for the scenario with widely distributed attackers. Distributed notification relies on an attack reaching a convergence point where it can be detected locally. In the distributed attacker scenario, an attack cannot be suppressed close to the source.

6.1.3 Suppression Evaluation

Both Cut-off and Isolation suppress all of the intended traffic under all 4 attack scenarios. The more distributed an attack is the more likely that a legitimate requester will intersect with an attacker and so the more likely that a requester's traffic will be suppressed along with the attackers' traffic. Isolation performs only slightly worse than Cut-off under attack scenario D (Figure 16). The Cut-off and Isolation publish are on the attack path. With the Isolation publish, any requesters that intersect the publish path after the publish will increase the number of false positives. These results show that Isolation is a viable alternative to Cut-off.

6.2 Full System Simulation

We did full system simulations as described in Section 5.4. The left axis is the average latency of all requesters. We chose to show the latency for the median without error bars for the other topologies because they follow the same trend and the chart would be too difficult to see. The right axis is the percentage of legitimate requesters that have not been suppressed with the attacker(s) and are still requesting. This data is collected every 5 seconds.

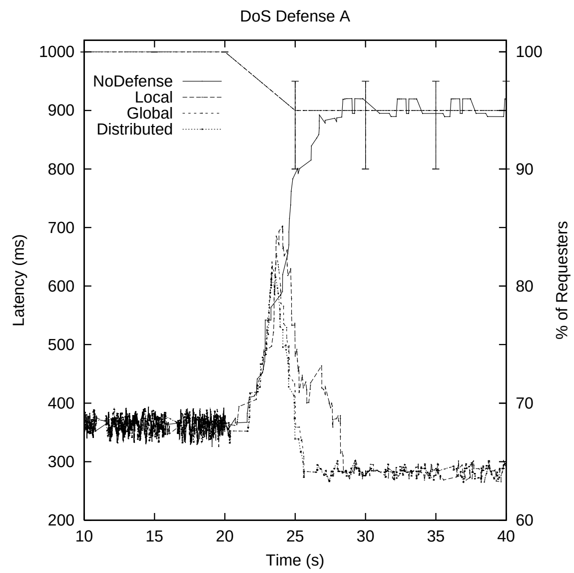


Figure 17: *Performance of defense under attack scenario A.* The attack begins at 20 seconds. As the request queue at the server fills, the legitimate requesters face an increasing delay. With no defense, a server will become overwhelmed within seconds. Isolation can be observed when the request latency reaches a peak and begins to drop. Global and Distributed defenses isolate the attack after approximately 2 seconds. Local isolates the attack after approximately 3 seconds. After the attack has been isolated, the server receives no more messages from the attacker and the request latency returns to the pre-attack level after approximately 2 more seconds.

The result under attack scenario A is illustrated in Figure 17. The attack begins at 20 seconds. As the request queue at the server fills, the legitimate requesters face an increasing delay. With no defense, a server will become overwhelmed within seconds. All of the defense schemes take some initial time to detect that an attack is occurring. Then the notification must be sent to the node which must isolate. Isolation can be observed when the request latency reaches a peak and begins to drop. Global and Distributed defenses isolate the attack after approximately 2 seconds. Local defense isolates the attack after approximately 3 seconds. Global and Distributed defenses isolate slightly faster than Local defense. This is because Global detection and isolation notification of the offending cluster is done by the target, while Local defense must do the notification hop-by-hop. Distributed defense is approximately the same as Global defense. Distributed defense is very slightly faster than Global defense since the detection and isolation command is closer to the attacker. The main advantage of Distributed defense over Global defense is that much less state must be maintained and less messages are passed, and so Distributed defense scales better. After the attack has been isolated, the server receives no more messages from the attacker and the request latency returns to the pre-attack level after approximately 2 more seconds. Also, in this case, the one attackers is easily detected and the false positives are low, meaning the legitimate requesters after isolation is still high. You may notice that the latency after isolation is slightly lower than the latency before the attack began. The reason for this can be understood by observing the requesters near and far from the server separately.

To clearly illustrate the effect, we placed the attacker far away from the server. For the half of requesters nearest to the object requested on the server (Figure 18), the post-isolation latency is the same as the pre-attack level. There are no legitimate requesters isolated because no isolation occurred near the server. The far requesters (Figure 19), which in this case are nearer to the attacker, contained all of the isolated legitimate requesters. In fact, the requesters that were isolated were the ones nearest to the attacker and farthest from the server, making their latency higher. Since they were isolated with the attacker, their latency is not included in the average, resulting in a lower post-isolation latency.

The result under attack scenario B is illustrated in Figure 20. The result is almost identical to attack scenario A, except that the peak of the latency is slightly higher. Also, since there are 4 locations from which attacks are coming from, there are more legitimate requesters that are isolated along with the attackers. Even if every attacker is not detected, a beneficial side effect of isolation is that nearby attackers will also be isolated. In this attack scenario, each of the attack clusters, or at least one attacker from each, is detected and each cluster is isolated.

The result under attack scenario C is illustrated in Figure 21. The end result is identical to attack scenario B be-

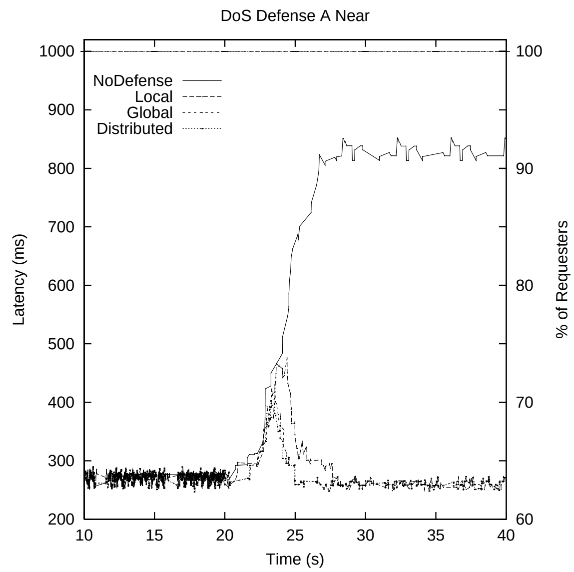


Figure 18: *Performance of the half of the requesters nearest to the object requested, under attack scenario A.* The post-isolation latency is the same as the pre-attack level. There are no legitimate requesters isolated because no isolation occurred near the server.

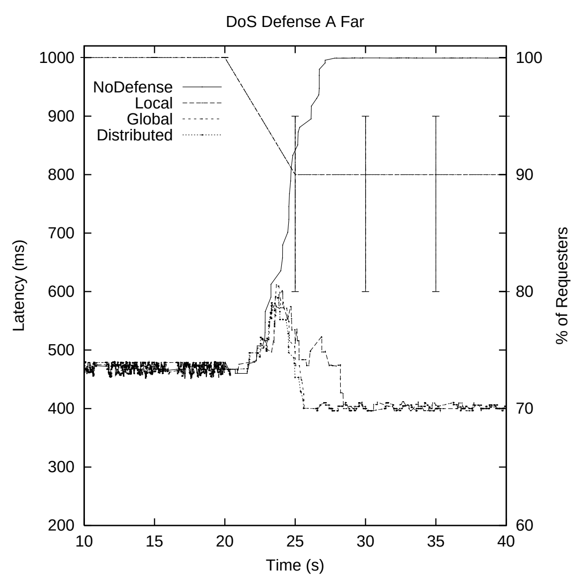


Figure 19: *Performance of the half of the requesters farthest from the object requested, under attack scenario A.* The requesters that were isolated were the ones nearest to the attacker and farthest from the server, making their latency higher. Their latency is not included in the average, resulting in a lower post-isolation latency.

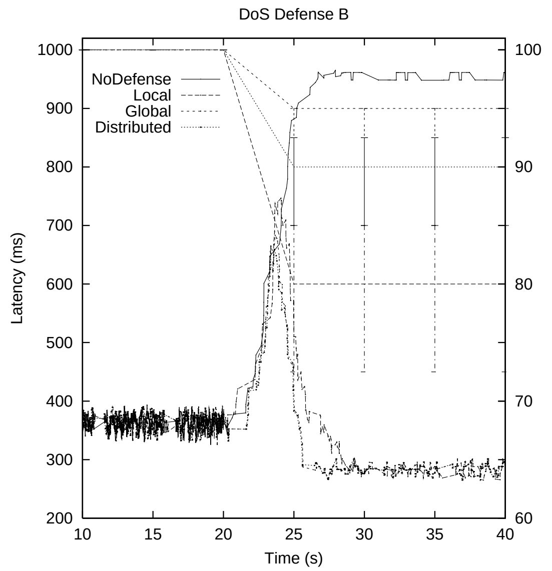


Figure 20: *Performance of defense under attack scenario B.* The result is almost identical to attack scenario A, except that the peak of the latency is slightly higher. Also, since there are 4 locations from which attacks are coming from, there are more legitimate requesters that are isolated along with the attackers.

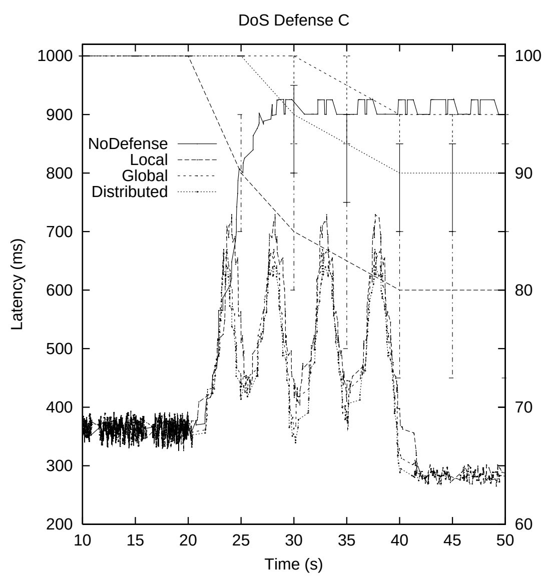


Figure 21: *Performance of defense under attack scenario C.* The only difference from attack scenario B is that we cycle through the 4 attack clusters one at a time. The defense successfully isolates each attack cluster as it appears.

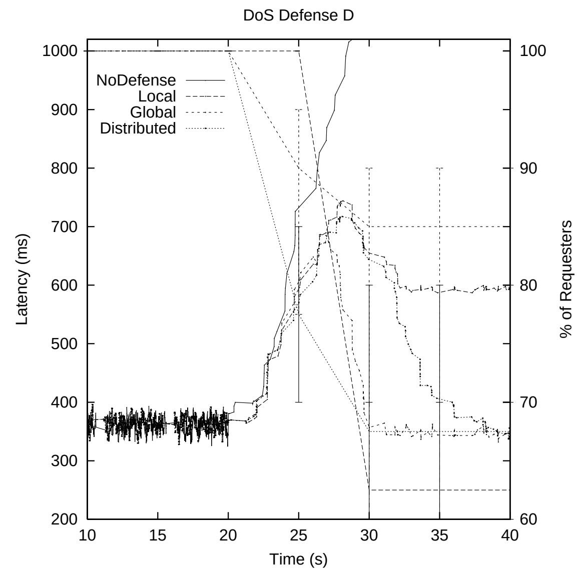


Figure 22: *Performance of defense under attack scenario D.* Local detection is not able to identify all attackers and therefore does not isolate all of the attack. Global and distributed detection manage to isolate all attackers even though they do not detect all attackers. Some attackers are not noticed until others are isolated, hence the longer time to isolate.

cause the same attack clusters were used. The only difference is that in attack scenario B, the 4 attack clusters were all activated at once, while in attack C, we cycle through the 4 attack clusters one at a time. The defense successfully isolates each attack cluster as it appears, and the attackers stay suppressed even after that first attack cluster reactivates after the fourth. Intuitively one might think that the amount of legitimate requesters isolated would increase by the same amount after each cluster of attackers is activated. In fact there is some overlap. Some of the requesters that would have been isolated while isolating a later attack cluster were isolated while isolating an earlier attack cluster.

The result under attack scenario D is illustrated in Figure 22. Under all detection methods, it takes longer to isolate the attack. This attack scenario is more difficult to detect and isolate because it does not come from a focused point. If an attacker is detected and isolated, that will not isolate many other attackers as a side effect. Each attacker must be detected and isolated individually. Some attackers are not noticed until others are isolated, hence the longer time to isolate. Local detection is not able to identify all attackers and therefore does not isolate all of the attack. Global and distributed detection manage to isolate all attackers even though they do not detect all attackers.

7 Discussion

Many previous approaches have focused on IP level methods to attempt to identify and stop attackers. IP was designed as a best effort service and was never designed to handle such a malicious attack. In recent years, there has been a trend toward deploying overlay networks on top of IP. Overlay networks bring increased resilience, distribution of load, and ease of locating data and services. A route-through overlay will allow nodes outside of the overlay to leverage its properties. Once people route through the overlay, we can automatically detect and suppress a DoS attack.

We believe that our DoS defense architecture will defend against the most prevalent DoS attack scenarios. These results suggest that using a distributed approach with local knowledge is effective in suppressing an attack. These results rely on synthetic benchmarks running on an artificial topology. While both were designed for a degree of realism, the behavior of our approach should be examined in a global-scale network with real users providing the workload.

The benchmark we use as the ideal solution is Global Clustering Analysis (GCA). GCA knows everything, everywhere, at all times in a centralized location and therefore allows for highly effective detection (perfect in our simulations). Our goal was to approach the effectiveness of GCA in a distributed manner with only local knowledge. If an attack comes from a concentrated location, the distributed approach is highly effective. The more distributed an attack is, the less effective a distributed approach is. We use very simple detection based upon a deviation from average traffic. More effective heuristics could, perhaps, improve the effectiveness.

The distributed detection is able to detect most of the attackers, even in the more distributed attack. Under attack A the attack comes from one location. Under attacks B and C, the attack comes from a few concentrated locations. As a result, there are not many requesters near the attack source(s) whose requests will converge paths with the attack traffic. But under the distributed attack (D), the attackers are dispersed throughout the requester population. As a result, many requesters converge with the attack traffic and are grouped with attackers once they have converged enough to be detected.

Unfortunately, our distributed approach is not as successful against the emerging threat of highly distributed DoS attacks. As we described in Section 5.3, attack scenario D is intended to emulate a totally distributed attack. While this is not the most prevalent form of attack, it offers benefits to the attacker; namely easier takeover of hosts and many more distributed attacking hosts.

The evaluation of our two suppression techniques is comparable. Even under a highly distributed attack, isolation is almost as effective simple cut-off. Under both suppression techniques, all of the detected attackers are suppressed. Also, both suppression techniques have relatively low false positives; most likely because the detection technique can detect very close to the attack. While isolation did have slightly higher false positives, a beneficial side effect was that some of the false positives were attackers. This benefit to cost was highest in cases where attackers were near each other in the network.

In this paper we make a number of contributions. First, we explain that an overlay network is necessary to defend against DoS because standard IP is incapable. We then show how an overlay network can be used in the Internet today through the use of a route-through overlay network. We propose an enhancement to such a route-through overlay to improve the infrastructure for wide-area overlay routing. We propose some techniques to detect and suppress a DoS attack in a distributed manner by leveraging the inherent collaborative properties of peer-to-peer overlay networks. Also, we present a simulation environment to model the network and attackers. Most importantly, we present four DoS attack scenarios based on real attacks that have occurred in the Internet.

We hope that this work will refocus efforts on an important research topicand make the case for how services should be delivered in the future. This work emphasizes the great promise of peer-to-peer overlay networks in the future of service delivery.

8 Future Work

There are several different directions this work can take in the near future.

While our transit-stub topology is fairly realistic, there are some new simulation topologies, such as power-law degree networks, which better approximate the nature of the internet. Also, our current simulator does not effectively simulate cross-traffic in the network because it does not simulate queing delay on a network edge. We simulate computation time and queuing delays at servers to approximate such delays as much as possible. While it is reasonable to expect that isolating the attack as close to the source of the attack as possible would mitigate the effect on the cross-traffic from senders and requesters not associated with the attack, this should be measured in a quantitative way. Ultimately, the true behavior or our approach should be examined in a true, global-scale network with real users providing the workload.

Aside from the application to DoS defense, the most exciting application of our detection and suppression techniques is replication on demand. Analyzing which regions of the network requests for a service are coming from, will allow the service to handle flash events. These techniques can be used to determine the placement of replicas for data replication in a wide-area distributed storage system [13] to handle load balancing in general.

Our route-through overlay can be on an isolated IP network. This will protect us from any attack at the IP level.

This will also allow us to provide certain assurances to help the overlay. The IP network can provide certain information to the overlay so that the overlay can better perform its path selection. The IP network can provide a certain level of QoS to the overlay. Also, intelligent peering points between ISPs can provide a smart market which provides aggregate QoS, pricing, security, etc. information about different ISPs and paths to allow the overlay to balance its needs. The intelligent peering points are an ideal location for the secondary, wide-area, overlay nodes (Section 3.1.2). Also, if an isolated IP network is not feasible for the entire overlay network, it can be used for the secondary overlay only. This would be more practical since the secondary overlay has less redundancy and a greater need for high bandwidth and fast access to the wide-area.

In this work, we do not develop the ability to immediately track down the culprit behind the attack, but instead focus on suppressing the attack. The contribution we make to quickly track down the attacking nodes while the attack is ongoing will allow for the tracking of the attack's control traffic. Currently administrators/investigators take on the order of hours or days to track down the source of the attacking traffic [27, 22]. One should also attempt to track down the culprit behind the attack while the attack is occurring.

9 Conclusion

We have built a system for suppressing a DoS attack. This system uses a distributed approach with local knowledge to isolate an attack to its originating region.

Overlay networks bring increased resilience, distribution of load, and ease of locating data and services. A route-through overlay will allow nodes outside of the overlay to leverage its properties. We can then automatically detect and suppress a DoS attack. Previous work using overlay networks to defend against DoS attacks has taken the simplistic approach of a redirection layer. Our system leverages the collaborative and distributed power of peer-to-peer overlay networks to detect and suppress DoS attacks. It also relies upon DoS attacks having some kind of concentration to certain regions of the network.

In a larger system, for example 20,000 nodes and 200 attackers, our defense techniques would be more effective. A larger system would allow for greater dispersal of participants. If requesters are more dispersed, larger sections of the network can be isolated to stop a DoS attack with minimal effect on legitimate requesters.

We show that a distributed approach with local knowledge is effective in suppressing an attack. Even though there is a high amount of false positives under a highly distributed attack, if an attack will completely overwhelm it's target it is better to sacrifice some legitimate requesters than all legitimate requesters.

Our initial measurements show the direction that future research should take. Collaborating with other members of a structured peer-to-peer network can provide a solution to many problems in the Internet today. We hope that this work will encourage more research into peer-to-peer networks to address DoS attacks and other issues with service delivery.

10 Acknowledgments

I would like to thank my fellow project members, who built Tapestry and OceanStore and helped me understand their work: Sean Rhea, Hakim Weatherspoon, Ben Zhao, and Ling Huang. I would like to thank my friend Daniel Kan for helping with this report. Finally, I thank my research advisor, John Kubiatowicz.

References

[1] Cert advisory ca-2001-19 ”code red” worm, July 2001. `http://www.cert.org/advisories/CA-2001-19.html`.

[2] Steven Bellovin, Marcus Leech, and Tom Taylor. Icmp traceback messages. ISOC Internet Draft, Feb 2003.

[3] M. Castro, P. Druschel, A-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. In *Journal on Selected Areas in Communications (JSAC)*. IEEE, 2002.

[4] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical Report MSR-TR-2002-82, Microsoft Research, 2002.

[5] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. Technical Report MSR-TR-2002-82, Microsoft Research, 2002.

[6] Yan Chen, Adam Bargteil, Randy H. Katz, and John Kubiatowicz. Quantifying network denial of service: A location service case study. In *Proceeding of Third International Conference on Information and Communications Security*, Nov 2001.

[7] Brent N. Chun, Jason Lee, and Hakim Weatherspoon. Netbait: a distributed worm detection service. Technical Report IRB-TR-03-033, Intel Research Berkeley, September 2003.

[8] Byung-Gon Chun, Puneet Mehra, and Rodrigo Fonseca. Dam: a dos attack mitigation infrastructure. Class Paper CS261: Computer Security, UC Berkeley, 2003.

[9] D. Clark. Policy routing in internet protocols. RFC 1102, `http://www.faqs.org/ftp/rfc/rfc1102.txt`, 1989.

[10] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common api for structured peer-to-peer overlays. In *IPTPS*, 2003.

[11] N. Daswani and H. Garcia-Molina. Query-flood dos attacks in gnutella. In *Computer and Communications Security*. ACM, 2002.

[12] Sven Dietrich, Neil Long, and David Dittrich. Analyzing distributed denial of service tools: the shaft case. In *Proceedings of USENIX*, Dec 2000.

[13] Dennis Geels. Data replication in oceanstore. Technical Report UCB/CSD-02-1217, UC Berkeley Master's Report, November 2002.

[14] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of SPAA (Winnipeg, Canada)*. ACM, August 2002.

[15] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *NDSS*. ISOC, 2002.

[16] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *Proceeding of 11th Word Wide Web conference*. IEEE, May 2002.

[17] A. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services. In *SIGCOMM*. ACM, 2002.

[18] Gary C. Kessler. Defenses against distributed denial of service attacks, November 2000. `http://www.garykessler.net/library/ddos.html`.

[19] John Kubiatowicz et al. OceanStore: An architecture for global-scale persistent storage. In *Proc. of ACM ASPLOS*, pages 190–201. ACM, Nov 2000.

[20] Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network. Technical report, ACIRI, February 2001.

[21] Alex C. Snoeren (MIT), Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Stephen T. Kent, and W. Timothy Strayer (BBN Technologies). Hash-based ip traceback. In *SIGCOMM*. ACM, 2001.

[22] Paul Mockapetris. Keeping ahead of dns attacks, January 8 2003. `http://zdnet.com.com/2100-1107-979650.html`.

[23] David Moore, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial of service activity. In *In Proceedings of Usenix Security Symposium*, August 2001.

[24] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the 9th Annual Symp. on Parallel Algorithms and Architectures*, pages 311–320. ACM, June 1997.

[25] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proc. of SIGCOMM*, pages 161–172. ACM, Aug 2001.

[26] Yakov Rekhter and Tony Li. An arch. for IP address allocation with CIDR. RFC 1518, `http://www.isi.edu/in-notes/rfc1518.txt`, 1993.

[27] Paul Roberts. Fbi: Dns server attacks came from u.s., south korea, November 1 2002. `http://www.computerworld.com/printthis/2002/0,4814,75564,00.html`.

[28] Matthew J. B. Robshaw. MD2, MD4, MD5, SHA and other hash functions. Technical Report TR-101, RSA Laboratories, 1995. version 4.0.

[29] Anthony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of SOSP*, October 2001.

[30] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM Middleware 2001*, November 2001.

[31] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for ip traceback. In *Proc. of SIGCOMM*. ACM, Aug 2000.

[32] Daniel Sieberg. Msblaster: Internet worm spreading rapidly, August 12 2003. `http://www.cnn.com/2003/TECH/internet/08/12/windows.worm/index.html`.

[33] Dawn Xiaodong Song and Adrian Perrig. Advanced and authenticated marking schemes for ip traceback. In *In Proceedings of the IEEE Infocom 2001*, April 2001.

[34] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM*, pages 149–160. ACM, Aug 2001.

[35] Hakim Weatherspoon and John D. Kubiatowicz. Efficient heartbeats and repair of softstate in decentralized object location and routing systems. In *Proceedings of the SIGOPS European Workshop*, September 2002.

[36] Matt Welsh. *The Staged Event-Driven Architecture for Highly Concurrent Server Applications*. PhD thesis, UC Berkeley, November 2000.

[37] Ellen W. Zegura, Ken Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE INFOCOM*, 1996.

[38] Ben Zhao, Ling Huang, Anthony D. Joseph, and John Kubiatowicz. Exploiting routing redundancy using a wide-area overlay. Technical Report UCB/CSD-02-1215, UC Berkeley, November 2002.

[39] Ben Zhao, Ling Huang, Jeremy Stribling, Anthony D. Joseph, and John Kubiatowicz. Exploiting routing redundancy via structured peer-to-peer overlays. In *Proceedings of 11th International Conference on Network Protocols*. IEEE, November 2003.

[40] Ben Y. Zhao, Yitao Duan, Ling Huang, Anthony D. Joseph, and John D. Kubiatowicz. Brocade: Landmark routing on overlay networks. In *IPTPS*, March 2002.

[41] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. In *Journal on Selected Areas in Communications*. IEEE, 2003.

[42] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. of the 11th International Workshop on Network and Operating System Support for Digital Audio and Video*. ACM, June 2001.