Exploiting P2P Systems for DDoS Attacks

Naoum Naoumov and Keith Ross
Department of Computer and Information Science
Polytechnic University, Brooklyn, NY 11201

***Abstract*— When a P2P system has millions of concurrently active peers, there is the risk that it could serve as a DDoS engine for attacks against a targeted host. In this paper we describe two approaches to creating a DDoS engine out of a P2P system: the first involves poisoning the distributed index in the peers; the second involves poisoning the routing tables in the peers. For both approaches, the targeted host does not have to be a participant in the P2P system, and could be a web server, a mail server, or a user's desktop. We then examine these two poisoning attacks in Overnet, a popular DHT-based P2P file-sharing system. By using limited poisoning attacks of short duration on Overnet's indexing and routing tables, we create DDoS attacks against a targeted host. We find that with modest effort, both DDoS attacks can direct significant traffic from diverse peers to the target.**

I. Introduction

In a flooding Distributed Denial of Service (DDoS) attack, the attacker exploits a large number of hosts, often referred to as "zombies," to concurrently send seemingly legitimate packets to an intended victim host. The goal of such an attack is to exhaust key resources at the target, diminishing the target's capacity to either provide or receive service. Resources that can be exhausted include the target's downstream bandwidth, upstream bandwidth, CPU processing, or TCP connection resources. From the attacker's perspective, a successful DDoS attack will not only exhaust key resources at the target host but will also involve a large number of zombies from different ISPs. This last characteristic makes it difficult for upstream devices to detect and filter the attack packets based on their source IP addresses.

We are concerned with two major classes of flooding DDoS attacks in this paper. The first type, which we call **TCP-connection DDoS attack**, is to overwhelm the victim's connection resources with *fully-open TCP connections*, thereby hampering legitimate users from making connections to the victim host. The second type, which we call the **bandwidth DDoS attack** is to generate enough traffic to tie up the bandwidth of the victim's access link (either downstream or upstream). UDP, TCP SYN or ICMP packets can be used as the raw material in a bandwidth DDoS attack.

In this paper, we explore how the indexing and routing substrates in P2P systems can potentially be manipulated to create bandwidth and TCP-connection DDoS attacks. More specifically, in this paper we explore two types of poisoning, which we call **index poisoning** and the **routing table poisoning**. With index poisoning, the attacker inserts bogus records into the P2P index system. These bogus records indicate that one or more popular files are located at the targeted IP address and port number. Importantly, the target host does not have to be a participant in the P2P system, and could be a mail server, a web server, or a user's desktop. When peers later search for the popular files, the index will inform them that the popular files are available at the targeted port of the targeted host. The peers then connect with the target and attempt to download the files, potentially overwhelming it with fully-open TCP connections or filling up the number of allowed connections and preventing legit users from obtaining services.

With routing poisoning, the attacker attempts to poison the routing tables in the P2P nodes. Specifically, the attacker attempts to make the targeted host an overlay neighbor of many of the peers in the P2P system. When a poisoned peer forwards a query, publish or overlay maintenance message, it may select the targeted host from its neighbor set, and send the message directly to the target. Given that there are millions of concurrently active peers in many P2P systems, if a significant fraction of the peers have their routing tables poisoned, the target host can potentially receive a flood of query, publish and maintenance traffic, and hence be the victim of a bandwidth DDoS attack.

The position of this paper is that, unless carefully designed, a P2P system may be vulnerable to index poisoning or to routing poisoning, and can thus be exploited as a massive DDoS flooding engine. Our principle vehicle for arguing this position is a protocol analysis and measurement study of Overnet, a DHT-based file-sharing system. With more than one million concurrently active peers, Overnet is probably the largest DHT deployed to date. It is also a core component of eDonkey, which generates today more traffic than any other content-distribution system, including BitTorrent [17]. Our experiments show that Overnet is indeed vulnerable to both index poisoning and routing poisoning. Using software developed in house, we carry out (limited and short-lived) attacks on Overnet, and measure the amount of traffic and TCP connections that are directed to our victim host (running on the Polytechnic University campus). Our measurements show that Overnet can indeed be exploited as a DDoS engine with a massive number of zombies. We conjecture that other P2P systems, besides Overnet, can also be leveraged as DDoS engines against arbitrary targets.

This paper is organized as follows. In Section II we describe in more detail index and routing table poisoning. In Section III we describe the Overnet file-sharing system.In Section IV we describe our experiment for creating a DDoS attack against a target host and present measurement results. In Section V we discuss how the index and routing substrates can be designed to make P2P systems less vulnerable.

A. Related Work

There have been numerous books and papers written on DoS and DDoS attacks; for some recent research see [1], [2],

[10], [11], [12], [13]. None of this research has addressed how P2P systems can be exploited for DDoS attacks.

Daswani and Garcia-Molina study the query-flood DDoS attack in Gnutella, and different policies that nodes might take to mitigate the attack [7]. However, it only examines the case when the target of the attack is itself a Gnutella peer. In this paper our focus is using to P2P systems for DDoS attacks against arbitrary hosts.

We use Overnet as a vehicle in exploring P2P-driven DDoS attacks. In doing so, we develop a crawler and measurement apparatus for Overnet. Bhagwan [3] et al and Kutzner and Fuhrmann [4] have also developed Overnet crawlers for measuring Overnet characteristics, such as peer availability. Overnet is a proprietary protocol. However, the creation of Overnet crawlers is facilitated by the open-source project kadc [16], which implements the majority of the Overnet messages. Gosling describes a buffer overflow attack on the eDonkey client [5].

II. Index and Routing Poisoning

A. Index Poisoning

Many P2P systems include an index. The index contains records which map keys to values. For example, for P2P file sharing systems, the index maps file identifiers (e.g., hashes of files) to locations (that is, IP address and port number). An index may also provide other types of mappings; for example, in Skype, the index maps user names to locations.

An index may be centralized (as it was in Napster) or distributed over a large subset of peers (as it is in FastTrack, Overnet, Skype and many other "commercial" P2P systems). We refer to the peers that participate in the distributed index as the **indexing peers**. Each indexing peer includes a portion of the index, and these portions may overlap across the indexing peers.

With index poisoning, the attacker's goal is to trick indexing peers into adding bogus records into their local indexes, where the location in the bogus records is the IP address and port number of victim host and service. For example, if the attacker wants to DDoS attack the mail service at host 222.222.222.222, the bogus record would contain 222.222.222.222 for IP address and 25 for port number. Depending on the P2P system and its implementation, it may be possible to trick an indexing peer by simply sending it a message which includes a bogus record; upon receiving the bogus record, the indexing peer may include it in its local index.

After an indexing peer has been poisoned, when another peer searches for the location of a particular file, it may receive a bogus record from the poisoned peer, and then attempt to download the file from the victim host. During this download attempt, it will first establish a TCP connection with the victim host at the port number specified in the bogus record. After establishing the connection, the downloading peer will send an application protocol-specific message, indicating the file it wishes to download. Not understanding this message, the victim host may ignore the message and let the TCP connection hang, may close the TCP connection, or may even crash. If many peers attempt to download from the victim host, the victim host becomes subject to TCP-connection DDoS attack. The implications may even be worse if the client automatically retries every few minutes.

In the classic TCP SYN flood DDoS attack, zombies flood the intended victim with TCP SYN packets but do not complete the TCP handshake with TCP ACK packets (typically because the zombies are using spoofed source IP addresses). This creates a multitude of half-open connections and can exhaust the victim's connection resources. However, operating systems can (and often do) eliminate this vulnerability by using TCP SYN cookies [18], [1]. The TCP-connection DDoS attack described here is nastier than the TCP SYN flood attack. It creates multitudes of fully-open TCP connections, which cannot be countered with TCP SYN cookies.

B. Routing Table Poisoning

Recall that in a DHT-based P2P system, the peers have IDs in some ID space. These IDs are used to organize the peers into an overlay, with each peer having a set of neighbors. In many DHT systems, peers have a relatively small number of neighbors, typically, $O(\log N)$, where N is the total number of peers. A peer's list of neighbors constitutes its routing table. Each entry in the list contains the neighbor's ID, IP address, and port number.

Recall that a query message in a DHT contains a key. When a peer receives (or generates) a message, it uses to the key to select a neighbor from the routing table to forward the message to. (The forwarding can be iterative with the node originating the query being the root, as is the case in Overnet.) The neighbor selection process is DHT-dependent. For example, in the Kademlia DHT, the peer uses the XOR metric to select the closest neighbor to the key [6].

When a peer joins the system, it builds its routing table, and this table is continuously updated as other peers join and leave the system. When a peer detects that a neighbor has left the system, it removes that neighbor from the routing table. When a peer discovers a new peer in the system (for example, upon receiving a query message), it may add that peer to its routing table. The details of how a routing table is updated is highly DHT and protocol implementation specific.

With routing table poisoning, the attacker's goal is to "trick" peers into adding bogus neighbors into their routing tables, where the IP address of these bogus neighbors is the IP address of the victim. Depending on the DHT and its implementation, it may be possible to poison a peer by simply sending it a message which "announces" the existence of a bogus peer. Upon receiving the announcement, the peer may choose to include the bogus peer in its routing table. Later, when the poisoned peer needs to forward one or messages through the DHT, it may select the bogus neighbor in its routing table and forward the messages to that neighbor. If many peers are poisoned, so that they add one or more bogus entries to their routing table, with each bogus entry having the IP address of the victim host, then the victim host could receive a flood of messages from the DHT, with the messages coming from millions of different sources. Furthermore, because the victim node is not a participant in the P2P system, it will typically

reply with an error message for each message received, additionally clogging up the victim's upstream pipe.

Many P2P systems are designed so that when a peer learns that one of its neighbors has left the system (because the neighbor didn't respond with a valid message), the peer removes the neighbor from its routing table. In this case, each successful poisoning announcement generates a burst of messages directed at the target, after which the bogus neighbor is removed from the peer's routing table. Thus, routing table poisoning is similar to reflection attacks [2], with a poisoned peer serving as a reflector. Each successful announcement causes the poisoned peer to generate a single burst of one or more packets directed at the victim. A measure of efficiency of a reflector attack is its *amplification*, which can be defined as the average number of packets in a burst. We shall see in Section IV that the amplification can potentially be huge with routing table poisoning. We remark in passing that reflector attacks that send DNS requests with spoofed IP addresses to DNS servers have successfully crippled targets [1].

III. Overnet and DDoS

To gain greater insight into the index and routing table attacks, we examine in detail Overnet, which is a part of the eDonkey client. Kad, deployed in eMule, is an open-source "cousin" of Overnet. Both Overnet and Kad are based on the Kademlia DHT [6]. Kademlia is similar in many respects to Pastry [19] and Tapestry [20].

To understand how Overnet can be exploited as a DDoS engine, we review some of its relevant features. When a client joins Overnet, it joins with a 128-bit ID. Presented with any 128-bit key, the Kademlia DHT finds the peers that have the closest IDs, where closeness is defined in terms of the XOR metric. To locate the closest peers, Overnet uses UDP messages and iterative searches. In particular, the querying client sends a series of UDP messages to a sequence of peers, with each peer in the sequence having an ID that is closer to the key.

A. Constructing and Maintaining the Routing Table

When a peer joins Overnet, it attempts to make contact with at least one peer from a locally cached list of peers (created during previous sessions). After finding a peer that is alive, the joining peer finds overlay neighbors by sending search messages for its own ID. The joining peer receives routing table entries from the peers in the path between itself and the closest peer to the ID. It constructs its own routing table by aggregating the received entries [6]. Once the joining peer has constructed a local routing table, it announces its presence to all the nodes in its routing table. When a peer receives such an announcement, it can update its own routing table by including the joining peer in the table.

B. Advertising Files

After a peer constructs its routing table and announces its presence, it starts publishing information about the files it is sharing. The publishing process consists of two phases:

- During the first phase, after hashing the file to obtain the file identifier, the peer sends into the DHT a **location publish message** which contains the file identifier and the file location (IP and port). This message is iteratively routed through the DHT to peers that are close to the identifier in the ID space. When these peers receive the message, they update their local indexes.
- During the second phase, the joining peer extracts keywords from the file's name and hashes each keyword into a 128-bit key. For each such keyword, the peer sends into the DHT a **metadata publish message** which contains the hash of the keyword, the hash of the shared file, and metadata information for the file, such as artist, title, album, file size, file type, etc. This message is also iteratively routed through the overlay to peers that are close to the hash of the keyword in the node ID space. When these peers receive the message, they update their local indexes.

C. Searching for Files

Searching works similarly to advertising, but the steps followed are in reverse order. The user first enters, say, n keywords into the client GUI. The client hashes each keyword and obtains $hash(keyword_1)$, $hash(keyword_2)$,..., $hash(keyword_n)$. The client then iteratively searches the DHT for each of the hashes. When a query reaches a peer that has records for the keyword, this peer returns the matching records to the querying peer. Each matching record consists of the file identifier (hash of file) and all the metadata available, such as file name, artist, file type and size. The requesting peer thus receives several sets of identifiers, one set for each keyword. The client filters the responses, keeping only those title names that match all the keywords. The GUI displays all of the filtered responses. The user then selects an entry for downloading, say, a file with identifier H. The client then performs a location search, iteratively querying the DHT for H. When a query reaches a peer that has records for the identifier H, the peer sends to the client a list of locations (IP address and port number pairs) for copies of that file. The client then tries to download the file from some of the locations with TCP connections.

D. Overnet as a DDoS Engine

We now describe how an attacker can exploit Overnet for launching DDoS attacks[1]. Let Z denote the victim host, which is not necessarily in Overnet. The attacker crawls Overnet, learning about new locations (IP address and port number) from the currently visited locations. During the crawling, the attacker either poisons the distributed index (for the index attack) or poisons the routing table (for the routing table attack).

We first describe how index poisoning can be done in Overnet. The attacker, say at Overnet node X, sends an Overnet location publish message to each of the crawled nodes while crawling. Figure 1 shows the location publish message, which gets encapsulated in a UDP packet. In these publish messages, the attacker includes the victim's IP address and

[1]We have informed the Overnet team about the vulnerabilities; however, at the time of this writing, the problems still exist.

port number. The attacker also puts any file hash it wants into the message. When node Y receives the publish message, Y adds the file hash to its index along with the location of the victim Z. Importantly, before adding this record, in Overnet, Y does not verify that Z has the file or even that Z is an Overnet peer. Later, when some node W wants the file corresponding to the file hash, W may be told by one of the poisoned indexes that victim Z has the file. Node W then establishes a fully-open TCP connection to the target service running on Z. As we shall see, this TCP connection may hang for a minute or more.

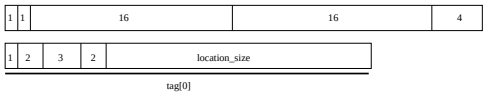

Fig. 1. Location publish message: 1 byte eDonkey, 1 byte message type, 16 bytes file hash, 16 bytes publisher's peer ID, 4 bytes number of tags, loc="bcp://ip:port"

We now describe routing table poisoning. The attacker, say at Overnet node X, sends an Overnet announcement message to each visited node during the crawling. Figure 2 shows the announcement message, which gets encapsulated in a UDP packet. In these announcement messages, the attacker inserts the victim's IP address in the peer IP field. The attacker also inserts a peer ID into the message (more about this choice later). Let Y denote the peer currently visited by the crawler. When Y receives the announcement, Y may add the victim Z to its routing table. Importantly, in Overnet, before adding Z, Y does not verify that Z actually belongs to Overnet. If many of the crawled nodes enter the victim Z into its routing table, and if the associated peer ID is such that Z is often selected as a neighbor from the routing table, then these tricked nodes will direct Overnet messages to the victim Z. We have described how routing table poisoning can be done with announcement messages. In principle, it can be done with other message types as well, including publish and query messages.

Fig. 2. Announcement message: 1 byte eDonkey, 1 byte message type, 16 bytes peer ID, 4 bytes peer IP, 2 bytes peer port, 1 byte peer type

IV. Overnet Measurement Study

In this section we show that it is indeed possible to exploit Overnet to launch a DDoS attack against an arbitrary victim host. In our experiments, the victim host resides at Polytechnic University. We show that by poisoning peers' indexes, we can create a TCP-connection DDoS attack; and by poisoning peers' routing tables, we can create a bandwidth DDoS attack. The attacks are not only DDoS attacks against our victim; they are also attacks against the users of Overnet, who are often directed to incorrect hosts during searching and downloading. For this reason, we keep the attacks brief and do not attempt to maximize the amount of traffic directed to the victim host. Instead, our goal is to merely show that with relatively little effort, Overnet can be exploited for DDoS attacks.

Our crawler works by sending connect messages to all peers that it learns about and search messages to peers that it already knows are alive. Considering the vast number of peers in Overnet, we ran our crawler from 16 machines. For simplicity, the crawlers running on the different machines do not communicate with each other and simply begin with different seed peers. The combined work of the crawlers covered the vast majority of the participating Overnet nodes. At the victim host, we ran a measurement program to record statistics about incoming data. The victim host did not run an Overnet client.

As a preparation for the attacks, we selected several popular titles from the music charts. We then searched for versions of those songs, obtaining a list of hashes of existing versions that the DHT already knows about. From the popular songs, we also selected 10 popular keywords and obtained the hash of each keyword.

A. Experimental Results

For index poisoning, we sent location publish messages to the crawled peers, with the file hash in the publish messages being the file hash of a popular version, as described above. In this manner, when a users wanted to download one of the popular files, with high probability the user attempted to download it from the victim host (as well as from other nodes, as Overnet employs parallel downloading). To keep the impact on Overnet low, we limited our advertisements for a 45 minute period. We continued recording the incoming traffic after the advertising processes stopped. We used a list of 7,564 file hashes, but we didn't send a publish messages for each file hash to each Overnet peer. Instead, for a particular file hash, we sent a publish message to a peer if the first 6 bytes of the peer ID and the file hash were the same. This allowed us to cover the majority of the Overnet nodes in less than an hour. We ran an Apache web server at the target host and measured the number of connections present every second.

Figure 3 shows the total number of connections and the number of ESTABLISHED connections as produced by netstat every second. It is important to note that for index poisoning, the poisoning persisted for hours after the publishing stopped, which occurred at 45 minutes. This is because the bogus records persisted in the indexes for hours, even after peers failed to download from the target host. The TCP connections originated from thousands of different peers (see discussion below on routing table attack).

Figure 4 shows that not only does the attack generate TCP connections at the target node, but that the connections also have significant durations. For our 9-hour trace, the average connection duration was about one minute.

For routing table poisoning, we sent announcement messages to crawled peers, with the peer IDs in the announcement messages being the hashes of the popular keywords. In this manner, a fraction of the search and publish messages for each of the popular keywords are directed to the victim host.

Figure 5 shows the amount of UDP traffic received at the target node. The traffic starts at 0 kbps and rapidly increases to 1 Mbps in less than a minute once the attack is launched. The traffic then remains in the 1.2 to 1.6 Mbps range until

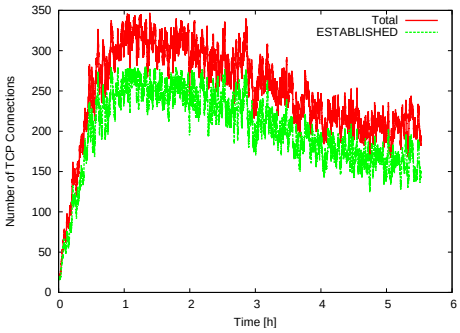

Fig. 3. Number of TCP connections present in one-second intervals

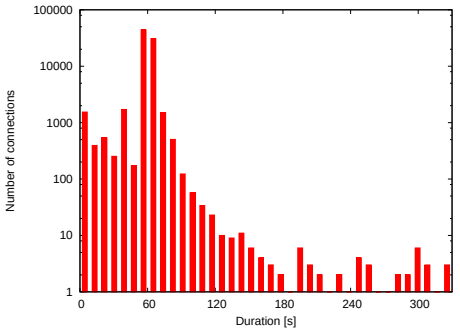

Fig. 4. Histogram of TCP connection durations

the crawler is stopped; the traffic then quickly tails off. On average, there is about 1.3 Mbps of downstream traffic (not including Ethernet headers) at the victim node during the attack period, not counting any traffic blocked by our university firewall. There is also, on average, about 1.5 Mbps upstream traffic out of the victim node, consisting of ICMP error messages generating by the victim's operating system. The reason for the rapid decrease of traffic at the victim after the attack is terminated is that in Overnet nodes frequently reannounce themselves, whereas we didn't frequently reannounce the victim. The victim host also doesn't properly reply to any of the UDP messages that it receives; so that after sending traffic to the victim, the peer removes the victim from its routing table.

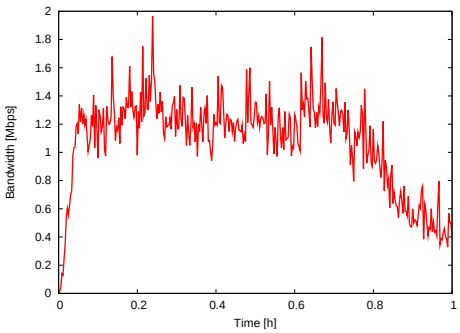

Fig. 5. Routing Table Attack: UDP Traffic at Victim Host

Figure 6 shows the number of unique IP addresses that send traffic to the target host in each one-second interval. We see that in each one-second interval the target host receives packets from about 1,600 Overnet peers. For the entire duration of the attack, the target host received traffic from 340,274 peers from 22,484 Autonomous Systems (ASes). This illustrates that the attack is highly distributed, making ingress filtering by source IP addresses difficult if not impossible [1].

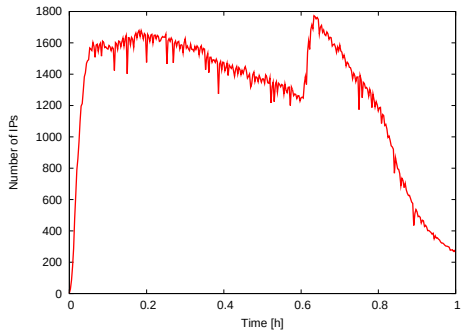

Fig. 6. Number of distinct Overnet hosts sending traffic to target in one-second intervals.

In addition to source peer diversity, amplification is an important measure (see Section II-B). Figure 7 shows the CDF of packet bursts sent per announcement message. We see that 44% of the bursts contained only 1 packet. Typically, one-packet bursts are queries and multiple-packet bursts are bursts of publish messages. Remarkably, many bursts contain hundreds or even thousands of bursts. This indicates that it may be possible to optimize the announcements and the peers to which they are sent, generating significant amplification in the DDoS attack. We note in passing that the bursts with thousands of messages most likely emanate from attackers which have been hired by music and film companies [9]

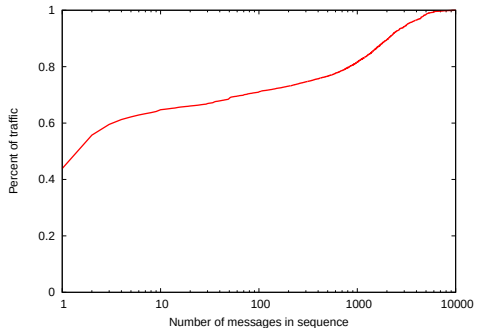

Fig. 7. CDF of Bursts

We can conclude from these experiments that with little effort in optimizing the attack, a significant amount of traffic and connections can be directed at an arbitrary victim host. For example, our routing table poisoning generated on average about 1.5 Mbps of upstream traffic at the victim, which is enough to exhaust the bandwidth of most victims with broadband residential access or institutions with T1 access; moreover, the traffic emanated from hundreds of thousands of different Overnet nodes. Our index poisoning caused over 300 TCP connections to hang at the victim, which persisted for hours after the attack. By increasing the number of attacked titles, and optimizing the attack procedure, more traffic and

connections can be generated, perhaps by a factor of 10 or more.

V. Issues with Counter Measures

In this section we briefly discuss issues surrounding counter measures to the attacks. Recall that in routing poisoning, a peer Y receives a message which announces the existence of a node Z, where Z is the victim and not a participant in the P2P system. A counter measure is to have peer Y check to see if Z is a peer in the P2P system. There are a few ways to do this:

- Y can send Z a message, eliciting a valid response. Traffic would still be reflected to Z, but now only with an amplification factor of 1.
- Encryption and closed-source software, so that nodes can only be announced by themselves. But those techniques can often be reverse engineered and circumvented [9]

Recall that in the index attack, an indexing peer Y receives a publish message which advertises the existence of a file at a location Z. One way to a counter a DDoS attack (on a non-participating host) is to have peer Y verify that Z is a participant in the P2P network. This can again be done by having Y handshake with Z. However, handshaking with each peer Z that advertises content to Y may introduce significant additional overhead traffic, both for Y and Z.

A. NATs

Complicating counter measures further is the need to accommodate NATed hosts that share content. In order to understand why it is not applicable we need to look at how Overnet handles NATed hosts.

Measurement studies have shown that 30%-40% of the peers in a P2P system may be behind NATs [8]. Many NATs do not allow external hosts to establish connections to hosts behind the NAT. In file sharing systems, this might prevent users from downloading files from a NATed host, and in VoIP applications the NATed host may not be able to accept incoming calls. Many P2P applications have implemented workarounds for the NAT problem. For example, in FastTrack, in order to request a file from a NATed peer, the user contacts that peer's supernode and that node tells the NATed peer about the request. Subsequently, the NATed peer will initiate the TCP connection, thus creating a "reverse" TCP connection. In Overnet the NATed peer sends location and metadata publish messages itself, but instead of providing its own IP address as the source for the shared file, it uses a delegate peer's IP. Later, if a peer wants to download the file shared by the NATed host, it will contact the provided delegate peer and have the delegate peer tell the NATed host to establish a reverse connection. This is possible due to the two different formats of location entries in the DHT. Typically, file locations are published with the following format: |**version_hash**|**peer_id**|**ip:port**| where the version hash is a hash over the content of the file, peer id is the ID of the peer that has the content, ip:port is the actual location of an Overnet client that has the file. If, however, a NATed peer is sharing files and sending publish messages, the file locations have the following format: |**version_hash**|**peer_id**|**peer_id:delegate_ip:delegate_port**| the difference here is that the location tag includes the ID of the NATed peer, and the IP and port of a delegate peer.

Because of this protocol specific NAT-accommodation technique, in Overnet the IP header source is not necessarily the same as the published source. This complicates anti-DDoS measures. A way to solve that problem would be to have the delegate nodes send the publish messages instead of the NATed host, and then verify with a 3-way handshake. Since a delegate node may be a delegate for many NATed hosts, this could substantially increase the traffic at the delegate nodes.

VI. Conclusion

We have argued that P2P index and routing substrates can potentially be exploited for DDoS attacks. As a case study, we showed how Overnet can be exploited. This issue needs to be brought to the forefront, as future P2P systems need to be designed without this DDoS vulnerability.

Acknowledgment

We would like to thank Jian Liang for his comments and help. This work was partially supported by NSF grant CNS-0412029.

References

[1] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher. Internet Denial of Service: Attack and Defense Mechanisms. *Prentice Hall 2004*.
[2] V. Paxton. An Analysis for Using Reflectors in Distributed Denial-of-Service Attacks. *ACM SIGCOMM Computer Communication Review, July 2001, pp.38-47*.
[3] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding Availability. *Proc. of IPTPS 2003*.
[4] K. Kutzner and T. Fuhrmann. Measuring Large Overlay Networks - The Overnet Example. *KiVS 2005*.
[5] I. Gosling. eDonkey/ed2k: Study of A Young File Sharing Protocol *GIAC practical repository*.
[6] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. *IPTPS 2002*.
[7] N. Daswani and H. Garcia-Molina. Query-Flood DoS Attacks in Gnutella. *CCS 2002*.
[8] J. Liang, R. Kumar, and K.W. Ross. The FastTrack Overlay: A Measurement Study. *Computer Networks (Special Issue on Overlays) 2005*.
[9] J. Liang, R. Kumar, Yongjian Xi, and K.W. Ross. Pollution in P2P File Sharing Systems. *INFOCOM 2005*.
[10] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial-of-Service Activity. *USENIX Security, 2001*.
[11] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with Capabilities. *HotNets, 2003*.
[12] C. Jin, H. Wang, and K.G. Shin. Hop-count filtering: An effective defense against spoofed DDoS traffic. *CCS 2003*.
[13] M. Sung and J. Xu. IP Traceback-Based Intelligent Packet Filtering: A Novel Technique for Defending against Internet DDoS Attacks *ICNP 2002*.
[14] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal, vol. 6, no. 1, 2002*.
[15] D. Stutzbach and R. Rejaie. Characterizating Today's Gnutella Topology. *IMC 2005*.
[16] The kadc library. *http://kadc.sourceforge.net/*
[17] Cache Logic. Peer-to-Peer in 2005. *http://www.cachelogic.com*
[18] E. Skoudis. Counter Attack. *Prentice Hall 2002*.
[19] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware) 2001*.
[20] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Tech. Rep. UCB//CSD-01-1141, 2000*.