Pushback for Overlay Networks: Protecting Against Malicious Insiders*

Angelos Stavrou[1], Michael E. Locasto[2], and Angelos D. Keromytis[3]

[1] Computer Science Department, George Mason University
[2] Institute for Security Technology Studies, Dartmount College
[3] Computer Science Department, Columbia University

Abstract. Peer-to-Peer (P2P) overlay networks are a flexible way of creating decentralized services. Although resilient to external Denial of Service attacks, overlay networks can be rendered inoperable by simple flooding attacks generated from insider nodes.

In this paper, we study detection and containment mechanisms against insider Denial of Service (DoS) attacks for overlay networks. To counter such attacks, we introduce novel mechanisms for protecting overlay networks that exhibit well defined properties due to their *structure* against non-conforming (abnormal) behavior of participating nodes. We use a lightweight distributed detection mechanism that exploits inherent structural invariants of DHTs to ferret out anomalous flow behavior.

We evaluate our mechanism's ability to detect attackers using our prototype implementation on web traces from IRCache served by a DHT network. Our results show that our system can detect a simple attacker whose attack traffic deviates by as little as 5% from average traffic. We also demonstrate the resiliency of our mechanism against coordinated distributed flooding attacks that involve up to 15% of overlay nodes. In addition, we verify that our detection algorithms work well, producing a low false positive rate ($< 2\%$) when used in a system that serves normal web traffic.

1 Introduction

Peer-to-Peer (P2P) overlay networks are a powerful and flexible way of creating decentralized routing services for various applications, including content distribution and multimedia streaming [11,13,3,20], network storage [7,18,9], resilience [2], packet delivery using rendezvous-based communications [19] and denial of service (DoS) protection [12]. A large number of overlay networks, such as CHORD [6], CAN [16], PASTRY [17] and TAPESTRY [8], are *structured*; that is, they use Distributed Hash Tables (DHTs) to perform directed routing. The use of DHTs imposes an inherent structure which dictates a well-defined and bounded set of neighbors in each P2P node. These neighbors are used by the P2P node to communicate all of its requests and replies. In

* This work was supported by the National Science Foundation under NSF grant CNS-07-14277. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

S.M. Bellovin et al. (Eds.): ACNS 2008, LNCS 5037, pp. 39–54, 2008.

addition, requests arriving from neighbors are appropriately forwarded to other neighbors until they are routed to the right overlay node. In a healthy overlay network, we expect that the load generated or forwarded by a node is, on the average, statistically similar (but not identical) to the traffic generated by any other overlay node; that is, we do not expect that the traffic will start be deviating too much from the average traffic generated by other nodes in the system. This is even more true for overlay networks that provide a service where clients do not actively participate in the overlay, such as I3 [19], Oceanstore [9], SOS [12] and others.

In this paper, we investigate methods to identify the traffic anomalies that can arise from deliberate attacks. Our detection algorithms *do not depend on object popularity*. Instead, we base them on measures of aggregate packet flows. Doing so enables us to avoid exorbitant storage and processing costs and maintain scalability in terms of the number of overlay participants. Although other researchers have used statistical methods to detect and examine aggregate flows [5,14], we are the first to consider the use of such methods in an overlay network setting that takes into consideration the neighbor-structure of P2P systems.

In general, we can protect an overlay network that has the following properties:

- The neighbors of each overlay participant are known for a window of time
- We can determine (within bounds) the fraction of requests we expect to receive from each neighbor

The above properties hold for almost all DHT-based Overlays (CAN is an exception) and even some randomized ones where the set of neighbors is of fixed size and the search requests have a predefined maximum length. Although we do not address misrouting directly as in [4], we do not allow mis-forwarding or neighbor spoofing: every node has a fixed list of known authenticated neighbors using pair-wise symmetric keys. Only these neighbors are allowed to route packets through the node and only to valid destinations according to the structure of the overlay. All other traffic is dropped, making objects reachable only by nodes that follow proper routing. By exposing traffic anomalies, our work prevents nodes from dropping or injecting requests, thereby encouraging nodes to conform to "normal" forwarding behavior with respect to the rest of the flow aggregate.

1.1 Pushback-Like Protocol

Pushback [10] is a router-based mechanism for defending against DDoS attacks. In a Pushback-enabled routing system, a router cognizant of the bandwidth limitations of *downstream* nodes may adopt a more proactive forwarding strategy. Instead of sending packets down a congested link (where they would be lost) or dropping such packets itself (which does not address the root of the problem: too much inbound traffic), the router would instruct (some of) its upstream routers not to forward certain packets. Heuristics are employed to identify the flows, or "aggregates" (packets having some common property, such as the same destination IP address and TCP port), responsible for the downstream congestion. The router examines its incoming (with respect to those flows) links, and the fraction of upstream routers responsible for most of the incoming

packets belonging to the aberrant flow are asked to rate limit that flow. These routers, in turn, recursively apply this mechanism.

Pushback, as proposed [10], works best when malicious traffic is anisotropically distributed around the Internet, so that some routers might rate limit traffic more severely than others. It also requires some level of trust between routers, an expectation that turns out to be somewhat unrealistic when crossing administrative boundaries (*e.g.,* an ISP's border or peering routers). The Pushback system also assumes that the participating routers would not misbehave in their execution of the Pushback mechanism and protocol.

There are two fundamental differences between that original setup and an overlay network: in the latter, the nodes themselves are both the originators of traffic and overlay "routers", and we have to assume that some of them will be compromised. Furthermore, not only can a compromised overlay node flood the network with traffic but it can also drop traffic meant to be routed to another destination via one of its neighbors.

Thus, in an overlay network, we have two types of misbehaving flows: flooding flows and packet drops. For excessive flows we first rate limit the offending aggregate flow and notify the upstream overlay neighbor of the problem, expecting him to rate limit the offending flow. For packet drops, we contact our neighbors' neighbors asking them to give us the counts for the aggregate flows in question, until we find a conflicting count indicating the node that drops the packets or lies about its aggregate packet count. By recursive and distributed application, this pushback-like protocol allows us to isolate the attacking nodes, quenching at the same time the effects of the attack. Of course, if our system has a lot of attackers collaborating with each other to both not comply but also to falsify their aggregate rates, pushback itself can be exploited by the attackers to generate additional traffic to the overlay network. On the other hand, even if the attackers are a significant portion of the network, if they are not coordinated they can be identified and isolated by the rest of the overlay nodes. An important assumption is that neighboring nodes cannot fake their identities, that is, they cannot pretend to be another node in the overlay. There are many mechanims that we can use depending on the underlying network and operating system cryptographic facilities available to the nodes. For example, if the underlying network is the public Internet, we can set up pairwise-authenticated encrypted tunnels using IPsec, GRE, or some other encapsulation protocol used to identify the origin of the traffic.

1.2 Our Approach

We test our implementation using traffic from actual web server caches [1] to drive the detection process. To that end, we extract the source of the request and the requested object and use their hash values to map them into sources and objects in our system. This way we test the performance our system under normal traffic tuning the necessary parameters to limit false positives. Our results show that:

1. We can efficiently detect and mark excessive flows even when up to 25% of the system's nodes have been compromised.
2. Our algoritms require $O(log^2(N))$ of memory per node, where N is the overlay nodes.

3. The proposed pushback-like protocol remains effective even when up to 15% of the overlay nodes attempt a coordinated attack

Our work is the first that attempts to detect, identify and isolate DOS flooding attacks initiated from inside an Overlay Network. The novelty of our approach lies in the exploitation of the properties inherent in these P2P systems with inference-based techniques.

2 Flow Model Description

This Section presents the notation we use to describe what we consider a "structured" P2P system, formally defines our notion of an attack, and provides a description of invariants that structured P2P systems exhibit.

2.1 Structured P2P Systems

A structured P2P system maps a set of keys K_{ids} to a set of nodes of size N and provides a distributed routing algorithm among these nodes. When a node n wishes to forward a message to the node holding key k, each P2P node on the route forwards the packet to the next P2P node along the path.

A *flow*, (s, k), consists of the set of search requests sent from node s to the key $k \in K_{ids}$. Let $\lambda_{S,K}$ be the rate of packet transmissions from nodes in S to keys in K, and let $\lambda^K_{avg} = \frac{\lambda_{S,K}}{|S|}$, *i.e.,* it is the average rate at which a node transmits requests toward keys in K. Objects stored in the P2P system may have different popularity, *i.e.,* that in general $\lambda^{k_1}_{avg} \neq \lambda^{k_2}_{avg}$ for $k_1 \neq k_2$.

Initially, we also assume that for a fixed object k, the popularity of this object is similar among the participants of the P2P network: $\lambda_{S,k} = \lambda^k_{avg}$ for all S, k. We expect that as $|S|$ grows, if the nodes that comprise S are chosen at random, then $\lambda_{S,k}$ will quickly approach λ^k_{avg}.

2.2 DoS Attackers and Attack Intensity

We consider DoS attacks targeted toward a specific key or set of keys. Such an attack could be mounted to block access to data associated with a particular key. A node that is the origin point of excessive packets toward key k is said to be an *attacker* of key k.

In a healthy P2P network, the rate of a flow $\lambda_{S,k}$ from a fairly large, randomly selected group of nodes S toward a set of keys K should closely approximate the popularity of that object λ^K_{avg}. We say that a flow aggregate (S, K) is *misbehaving* if $\lambda_{s,K} > (\delta + 1) \cdot \lambda^K_{avg} = \lambda^K_{max}$ for some $\delta > 0$, where δ represents a lower bound on the proportional increase that a flow can transmit relative to the average rate before the flow is labeled as misbehaving. The previous notion can be extended to a set of nodes S as $\lambda_{S,K} > |S| \cdot (\delta + 1) \cdot \lambda^K_{avg} = |S| \cdot \lambda^K_{max}$. Note that for a set of nodes, the maximum rate allowed before we declare the aggregate flow as misbehaving depends on the size of the set. The selection of δ is a measure of the tolerance that we allow between different nodes (or groups of nodes) in the P2P system before declaring that a flow is misbehaving. If we assume a totally homogeneous system, then $\delta = 0$ — *i.e.,* no deviations from

the average rate (popularity) are allowed. Larger values allow more tolerance, but also give an attacker more freedom to deviate from the average rate and avoid detection. Typically, we select $\delta = 0.1$ which means that we detect flows that send ten percent more than the average rate of a set of keys K.

We define β to be the proportion by which one attacker increases its traffic toward k above λ^k_{avg} such that the attacker transmits packets toward key k at a rate of $(\beta+1)\cdot\lambda^k_{avg}$. If N_a is the number of attacking nodes, the total amount of excessive search requests injected into the system by the attackers is $\beta \cdot N_a \cdot \lambda^k_{avg}$. The rate of search requests λ_{attc}, the target node experiences under attack is:

$$\lambda_{attc} = \frac{N_a}{N} \cdot (\beta + 1) \cdot \lambda^k_{avg} + \frac{(N - N_a)}{N} \cdot \lambda^k_{avg}$$

Let $f = N_a/N$ be the fraction of nodes compromised; we define the attack intensity, or gain due to the attack, D_A, to be the increase in the popularity of the target key k caused by the attackers' excessive search requests:

$$D_A = \frac{\lambda^k_{attc}}{\lambda^k_{avg}}$$

For example, if $D_A = 2$, the node that stores the object under attack has to serve twice as many search requests for that object. For the attackers' queries to be harmful to the target node that stores the keys being attacked, D_A must be large. If an attacker only controls a limited number of nodes, their only choice is to increase β. A large β and small f means that there will be a relatively small number of attack flows, and that these attack flows will inject significantly more traffic toward k. Our methods to detect misbehaving nodes will utilize the following measure:

– Fixed-Key Variable-Source (FKVS):

$$\alpha(c, K, S_1, S_2) = \frac{\lambda^c_{S_1,K}}{\lambda^c_{S_2,K}}$$

The FKVS measure compares the rates of two sets of sources for a particular set of keys. In a healthy P2P system, for appropriately sized (large) sets S_1 and S_2, if c lies on the paths from all S_1 and S_2 to the nodes storing keys K, we should have that $\alpha(c, K, S_1, S_2) = |S_1|/|S_2|$.

3 Statistical Bounds of Flows

We now present methods that use the previously introduced metrics to identify misbehaving flow aggregates and mark packets that belong to these aggregates. The total number of possible flows in a P2P system is $N \cdot |K_{ids}|$. Thus, tracking each individual flow would require $O(N \cdot |K_{ids}|)$ memory. Even if we fully distributed the tracking load amongst all participating nodes, $O(|K_{ids}|)$ memory would be needed to collectively track all flows.

To avoid utilizing such a potentially huge amount of memory per node, we track a set of aggregate flows whose size is $O(\log(N))$ by taking advantage of the fact that the number of flows in each of the incoming and outgoing neighbors is $O(\log(N))$. We require that each node c consider as a separate aggregate all flows that arrive via the same incoming neighbor and exit via the same outgoing neighbor. Hence, there are $O(\log^2(N))$ aggregates to consider. By numbering the neighbor nodes in In_c and in Out_c, we can identify the flow aggregate that enters through the i-th neighbor and departs through the j-th neighbor as $f_{i,j}$, where $i = 0$ implies the flow originates at c and $j = 0$ implies the flow terminates at c. Aggregate $f_{i,j}$ is assigned a counter, $C_{i,j}$. When a packet arrives, we increment the counter that corresponds to the aggregate flow to which the packet belongs.

3.1 Comparison of Aggregate Flows

Our detection algorithm compares each flow's counter $C_{i,j}$ to the counter for all the aggregate of flows that exit to the same outgoing neighbor. We wish to determine the likelihood that, in a healthy P2P system, $C_{i,j}$ can have the value observed, under the assumption that C_j is made up mainly of healthy flows. More formally, let $X_{i,j}$ and X_j respectively be random variables that equal the value of these counters in healthy P2P system, and determine $\mathbb{P}(X_{i,j} \geq C_{i,j} | X_j = C_j)$. (Note that $C_{i,j}$ and C_j represent actual observed values in the real system, whereas $X_{i,j}$ and X_j are values that occur in a trial on top of a healthy P2P system.) The larger this probability, the more confidence we have that flows entering through neighbor i and exiting out of neighbor j are healthy. We define ϵ to be our *level of confidence* of the test. If $\mathbb{P}(X_{i,j} \geq C_{i,j} | X_j = C_j) < \epsilon$, then (according to this test) there is a probability $< \epsilon$ that $f_{i,j}$ is healthy. By comparing this value to the observed values of the counters, we can determine, within a certain confidence level, whether one flow's rate is higher with respect to the all the other flows' rate.

In the previous calculations, we don't need to assume anything about the flows' distribution. While any slight deviation from the normal transmission rate could be flagged as a violation, we allow a small degree of variability. Hence, we apply our "slack" factor, δ, and say that flow $f_{i,j}$ is misbehaving when the actual ratio of observed rates is larger than $(1+\delta)\alpha(c, K, S_i, S'_i)$ for the second test. Setting δ to 0 leaves no slack. If $f_{i,j}$ sends at a rate even slightly above its supposed rate in a healthy P2P system, the central limit theorem tells us that eventually, $f_{i,j}$ will be flagged as unhealthy. Setting δ to larger values allows for additional slack.

4 Application to a DHT System

A Chord peer to peer system consists of a set nodes of size N that try to serve objects that are hashed and stored in nodes using an m bit hash function. The key identifiers K_{ids} are placed in circular order, creating a ring of length $K = \|K_{ids}\| = 2^m$. To simplify the notation, all math in the remainder of this section is performed modulo K. Each node is assigned an identifier (id) from the key space, thus creating a node ring. Since we have a circular placement, we have for each node c a successor node and a predecessor node. Each node is assigned a set of keys, meaning that the node either stores or knows the location of all the objects in its local database that hash to its value.

4.1 Chord Invariants and Tests

Proposition 1 *Let $i_1 > i_2$. If the set of flows (S_{i_2}, K) that pass through c is non-empty, then:*

$$\alpha(c, K, S_{i_1}, S_{i_2}) = 2^{i_1 - i_2}.$$

Proof. In the Appendix

Proposition 2 *If the set of flows (S, K) that pass through c is non-empty, where $S = \cup_j S_j$, then:*

$$\alpha(c, K, S_{i_1}, S) = \frac{1}{2^{\log(N) - i_1} - 1}$$

Proof. The proof follows the form of Proposition 1, noting that there are $\sum_{j=0}^{\log(N)-i_1} 2^{i_1 - j}$ sources from S to K that enter c (through any finger).

5 Experimental Results

5.1 Web Trace-Driven Simulations

To test our implementation, we used web traces obtained from IRCache repository [1] to drive our simulated environment. As we will soon present, our experiments show that under non-attacking conditions our system does not generate excessive false positives (less than 2% false positives).

Our first goal was to verify the effectiveness of our detection algorithm and to evaluate its performance. To that end, we use an implementation of the Chord peer to peer network in a series of simulations. In all of our experiments, some of the participating peer to peer nodes assume the role of the "attacker". The "attackers" select a key at random from the set of allowed keys and generate a disproportionate number of search requests towards that key. The node that stores the key under attack is the target. The goal of the attackers is to flood the target with search requests, crippling its ability to respond. In general, the attackers are allowed to select multiple targets simultaneously. However, if we assume that attackers have abundant but nonetheless limited resources, aiming at multiple targets will only lower their aggregate attack ability. Indeed, the attackers will have to split their attack requests between the different targets, reducing their attack intensity. We formally defined attack intensity as the increase in the search request traffic towards the target key caused by the attackers' excessive search requests. For example, an attack intensity of $D_A = 2$ means that the node that stores the key under attack has to serve twice as many search requests for that object as it would normally have.

We use the notion of attack intensity to formally quantify the attack ability of the compromised nodes. Where our test is working perfectly, marking only excessive search requests, only a fraction $\frac{\beta}{\beta+1}$ of packets in the attacking flow should be marked. We determine our performance by measuring our ability to detect the attack as close to the attacker as possible. Also, we want to mark the malicious search requests as many times as possible along the path from the attacker to the target object. All of these metrics are dependent both on the attack intensity and the relative attacker's distance from attacked key.

Another goal of our experiments was to identify the limits of our detection algorithm. Our attack detection method identifies aggregate groups containing attackers by marking packets from these groups as "excessive" whenever our estimates predict that the groups are transmitting at too high a rate. Since we operate at the granularity of groups, we introduce a false positive error for the non-attacking individual flows which happen to be grouped with "excessive" ones. As we show, this error is relatively small because the vast majority of the requests from a malicious aggregate flow belong to the attacker. Moreover, the attack requests get marked multiple times along the path from the attacker to the target. Another potential source of error can arise due to the use of statistical inference. Of course, this error can become arbitrarily small by selecting a higher confidence interval. In our simulations we used a 0.999% confidence interval.

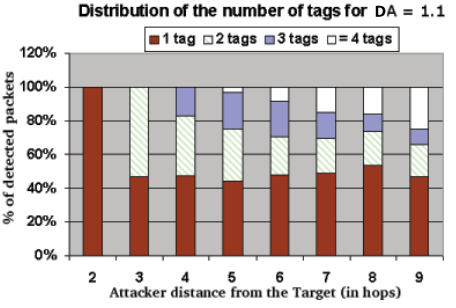

Fig. 1. Distribution of the number of tags for the attack requests for one attacker in a 1024-node Chord ring. The different plots represent the attacker's distance in hops from the target for attack intensity of 110% (D_A = 1.1) *i.e the object receives 10% more traffic.*

In the last set of experiments, we use a pushback-like protocol where a node, upon detection of a misbehaving aggregate flow, communicates with its upstream neighbor that this flow is originated. The misbehaving aggregate flow, if it is excessive, is rate limited to the average of the rest of the flows. If the upstream neighbor fails to respond with a certain amount of packets, which is a parameter for our system, this node is considered malicious and the rest of the overlay nodes are notified. The protocol is applied recursively until the source of the anomaly is identified or the flow stops being excessive. Either way the DoS attack will be prevented allowing only small, negligible spikes of packets to reach the target node.

To ensure the statistical validity of our experiments, we used approximately 4 million search requests per simulation. In addition, each simulated experiment was repeated more than 50 times. The results we present in our graphs are the average of these simulations; the variance among the different experiments was observed to be low.

5.2 Detection of Single-Attacker

We start by examining the scenario where a single node is compromised. Although simplistic for a real world attack, this scenario provides insights on the effectiveness of our approach. Furthermore, we can evaluate our ability to detect the malicious node for varying distances relative to the target and for a range of attack intensities. Initially, we placed the attacker in various distances (in hops) away from the node responsible for the target key, the target node. We then measured the percentage of the excessive search requests our algorithm detected and their detection distance from the target *i.e.,* how quickly was the traffic identified as excessive. As the distance of the attacker from the target increases, the detection distance increases accordingly. In addition, we detect

the malicious flow in multiple nodes along the path from the attacker to the target. It appears that the largest portion of the attack requests are detected close to the attacker. For example, if we place the attacker at a distance 9, the majority of its packets are detected by its next hop neighbor,with distance 8, then by the next node at distance 7 etc. Even when the attacker's proximity to the target is reduced to the minimum, *e.g.,* at a distance of 2, we detect 100% of the excessive search requests at the immediate neighbor.

Recall that each node tags all the flows of a group that appears to be "misbehaving". Figure 1 shows the relative distribution of the tags when we vary the distance of the attacker to the target. We see that, depending on the distance between the attacker and the target, a significant portion of the excessive search requests are tagged at least twice by nodes along the path from the attacker to the target. The number of tags increases as we increase the number of hops between the attacker and the target since the attack packets traverse more nodes in order to reach their target.

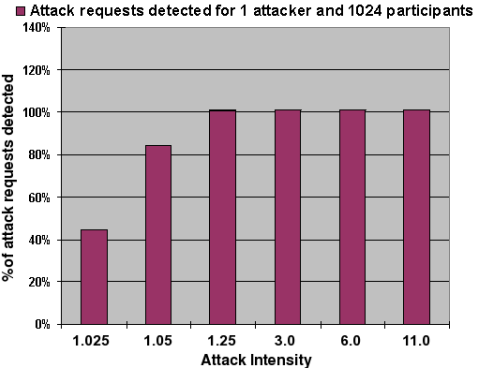

Fig. 2. Attack packets detected for one attacker randomly placed in a 1024-node Chord ring. The different bars correspond to increasing values of attack intensity D_A. The results presented are the average of multiple experiments (100 per bar).

To actually have impact on the search requests of the attacked object, the attackers intensity, D_A should be relatively high. For example, for $D_A = 2$ the single attacker has to inject search requests in the system with rate $N \cdot \lambda^k_{avg}$, or with a $\beta = N$. Although this rate may appear unnecessarily high, in practice this depends on the popularity λ^k_{avg} of the object attacked. If the object is highly popular and λ^k_{avg} is large compared to the rest of the objects in the system, the attacker will need to significantly increase the number of search requests to noticeably affect its popularity.

We have shown that our method detects and marks search requests on groups of flows. An inherent problem is that we may end up marking legitimate flows along with the attacking ones (since they are mixed in the same aggregate) but that is not the case for our method. Although we are marking flows belonging to the same "misbehaving" group, we are punishing mostly the attacker since it is the one sending the majority of the search requests through that group. The majority of the other flows are getting marked minimally, in comparison both to the total number of requests they generate and to the attacker requests marked. Naturally, blindly marking and dropping excessive requests from a misbehaving flow is a very crude method to prevent a DoS attack, although it can be effective if resources are otherwise limited.

Through our experiments we wanted to ensure that there is no attacker placement inside the Chord ring that our algorithm fails to detect. We are now in position to present more realistic results from simulations in which we have one attacker randomly placed

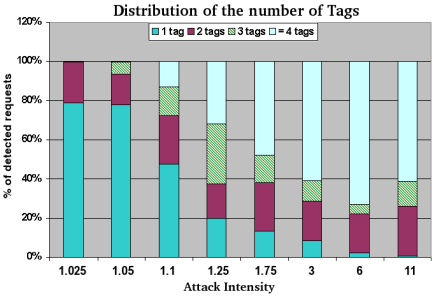

Fig. 3. Distribution of the number of tags for the detected attack packets for a 1024-node Chord ring with randomly selected attacker-target placement. Each value on the X axis corresponds to exponentially increasing attack intensity (D_A). The results averaged over 100 experiments for each attack intensity value.

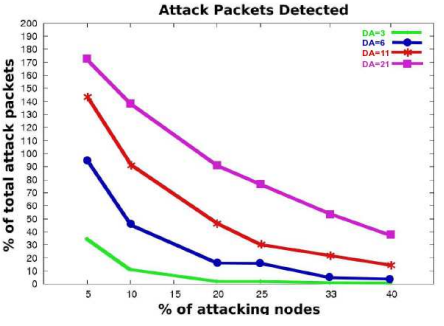

Fig. 4. Percentage of attack packets detected when we vary both the attack intensity D_A and the fraction of nodes compromised for a 4096-node Chord ring. Each line represents different attack intensity values. We can see that as we increase the attack intensity we can detect more percentage of the attackers' request even when the attack is very distributed.

in a Chord ring of size 1024 (see Figure 2). We observe that even for very low attack intensities values, $D_A = 1.025$, a mere 2.5% increase in the load of the end server, we can detect more than 40% of the attack packets. It is easy to see that for attack intensity values larger than 0.05 our method detects a significant portion of the excessive requests. As the intensity of the attack diminishes, our detection results become weaker. This is something we expected, since our algorithm detects excessive requests based on measurements done on groups of flows where small variations in the intensity of one flow does not have significant impact on the aggregate flow. For some values of attack intensity, especially for $D_A = 0.25$, we have an over-marking of the attacker search requests which fades out when the attack intensity becomes more significant. This is due to the group detection nature of our algorithm and the fact that aggregate groups of flows contain both legitimate and excessive flows originating from the attacker.

The number of tags for the attack packets is an increasing function of both the average attacker distance and of the attack intensity as shown in Figure 3. The average attacker distance seems to play a more prevalent role. This means that as the average distance between the attacker and the target increases, so does our ability to tag the attacker on multiple locations along the attacker-target path. Thus our system works better as we increase the number of participants in the DHT system, since the average distance between two nodes in the system increases. The detection behavior of our algorithm for attack intensities that are higher than $D_A = 11$ is similar to those measured for $D_A = 11$ and we omit them from our figures.

5.3 Detection of Multiple Attackers

We now study the behavior of our detection algorithm using a Chord ring where we vary both the fraction of nodes compromised and the attack intensity. Figure 4 presents the results for a 4096-node Chord ring with multiple attackers. It is clearly shown that there is a correlation between the excessive search requests detected and the attack intensity. Our results show that as the attack becomes more severe, our ability to detect excessive search requests increases; even when 40% percent of our nodes are compromised we are able to detect around 50% of the excess requests.

On the other hand, as the proportion of compromised nodes grows, there is a corresponding drop in our ability to detect excessive search requests, since the attack becomes more distributed on the Chord ring. Additionally the number of tags for the excessive requests are inversely proportional to the fraction of nodes compromised. For example, as Figure 5 shows, when the attackers constitute 5% of all the Chord participants, a large portion of their excessive requests have 2 or more tags, whereas when the attackers become 40% of the total, only 14% of the excessive requests have 2 or more tags.

In some cases we overestimate the number of attack packets sent by the attackers. This happens because we detect groups of flows where, on average, the attacker participates with multiple flows, leading to over-marking of search requests generated from the attacker. This over-marking can be used to weed out the specific flow from the group of flows detected to be misbehaving.

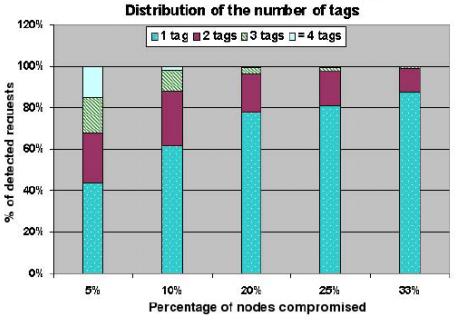

Fig. 5. Distribution of the number of tags for the excessive search requests detected when the attack intensity is $D_A = 21$ and we vary the fraction of nodes compromised for a 4096-node Chord ring. Notice that he number of tags on the attack requests are inversely proportional to the fraction of the attackers. The more "distributed" the attack is, the more difficult it is to detect and tag.

Another source for this overmarking comes from the fact that we allowed the non-attacking nodes to select the popularity of each key/object from a uniform $(0, 1]$ distribution. This generates different preference distributions for each node pushing our "normality" assumptions to their limit. However, it also confirms our initial assumption that our analysis remains the same even if we allow different preference distributions. With a mean of $1/2$ and a standard deviation of $1/6$, the uniform distribution allows for wide spreading of the possible weight values in the full $(0, 1]$ spectrum. The use of a normal or exponential distribution for the same interval would have resulted in less variance, leading to a more concentrated weight distribution. The law of large numbers dictates that any type of weight distribution would eventually lead to a normal preference distribution on the limit as the number of keys grows.

In Figure 6 we can see the percentage of packets detected for a Chord ring of size 1024. Again, each line represents different attack intensities. The detection ability dissipates as the percentage of attackers in the total node population increases. Note that the

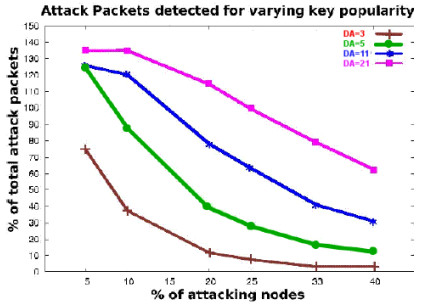

Fig. 6. Percentage of excessive packets detected when we vary both the attack intensity D_A and the fraction of nodes compromised for a 1024-node Chord ring. Each line represents different attack intensity values. For this experiment we allowed each individual non-attacking node to assign a weight to each key selected from a uniform distribution. The detection results appear to be better than using the same preference distribution function.

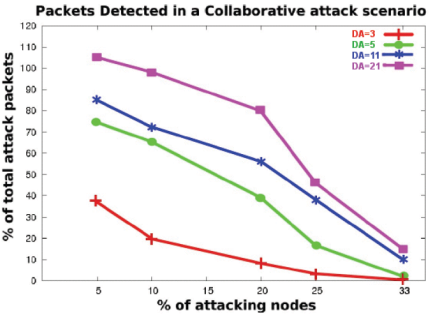

Fig. 7. In this experiment, the attackers are collaborating by not marking the excessive packets destined for the target key only (coordinated attack). Each line represents different attack intensity values and we vary the percentage of nodes compromised. Even under a collaborative attack, and with a significant portion of all nodes being compromised, we detect a large portion of the attack requests.

detection results are better than those presented in Figure 4, where we used the same preference distribution function for all the nodes. Indeed, when using different preference distribution, sometimes attackers are grouped with flows that have high preference for the attacked key and thus revealed faster. However, for the same reasons, we end up having a higher false positive detection of around 0.1% compared to the 0.07% when we used the same preference function. Overall, allowing the nodes to have a different preference distribution function leads to better detection results at the cost of an slight increase in the false positive detection percentage.

In our experiments, we used a fixed value for the threshold δ, namely $\delta = 0.1$. Recall that δ is a parameter of our detection algorithm, indicating our tolerance towards deviations from the average popularity of a key. Larger values allow more tolerance and lower our false positives, but also give an attacker more room to deviate from the average rate before our algorithm starts detecting the excessive requests. There is a trade-off between speed of detection and false positives. Figure 10 shows that, for $D_A = 20$, we get an improvement in our detection as we decrease δ from 0.1 to 0.05. At the same time we see an increase in the false positive percentage, which becomes more apparent for $\delta = 0.05$. Conversely, for $\delta = 0.07$ we get an improvement of almost 12% in the detection rate, when we have a distributed attack with $> 20\%$ of attackers. Lowering δ below 0.07 does not improve the detection rate, doubling false positives.

Our last experiment stresses our detection algorithm under the worst-case scenario: a highly distributed attack where the attackers can collaborate both by exchanging information about the location of the target and by not marking each other's excessive flows towards the target node. Even under this adverse attack scenario, our algorithm seems to detect between 70% and 100% of all attack traffic in high-volume attacks when

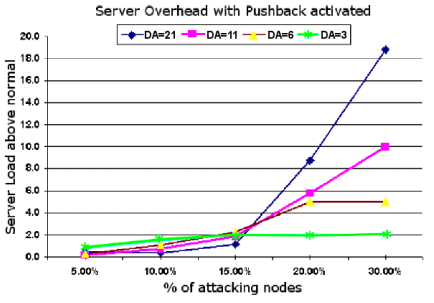

Fig. 8. Increase in server load when under attack and with pushback protocol activated. A load of value 2 means that the server is serving twice the amount of normal requests. The performance of the pushback protocol remains acceptable even when 15% of the overlay nodes are attacking the target.

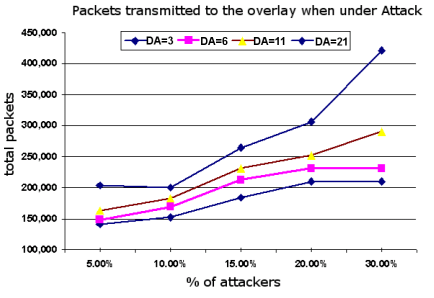

Fig. 9. Total number of packets transmitted to the overlay network, this includes packets that traversed even one hop and then where dropped. Again, the performance of the system is not affected significantly by the attack even when 15% of the overlay nodes are subverted.

even up to 15% of all overlay nodes are participating in the attack, as shown in Figure 7. When 25% of the overlay nodes are participating in the attack, our mechanism correctly identifies between 40% and 50% of all attack traffic. As the percentage of attacking nodes in the population increases, the effectiveness of our mechanism decreases. This, however, should be expected: the notion of "normalcy" in such a system is shifting towards higher-volume traffic flows. Furthermore, as we have anticipated, given more knowledge, attackers can avoid detection especially when the attack is highly distributed ($> 25\%$ of the total nodes are compromised) or the attack is of low intensity ($D_A < 6$).

5.4 Pushback Protocol Performance

In this section we quantify the performance of the simple pushback protocol we devised: we run our experiments enabling nodes to rate limit the aggregate flow that contains the malicious flow using probabilistic rate limiting. The dropping probability is determined based on the deviation of the aggregate flow from the average making the flow conform to within δ from the measured average. Moreover, the node detecting the attack notified its corresponding incoming neighbor of the problem denoting also the sets of key(s) that were affected by this excessive aggregate flow. If the node was unresponsive to the rate limiting request for more than 100 packets (i.e. the rate of the detected aggregate flow remained above the average) this node was immediately marked as malicious and was removed from the overlay by being replaced with a non-attacking node to avoid reconstruction of the overlay routing table. The selection of the 100 packet threshold is a parameter of our system and allows a little more time to the neighbor to adjust to the rate limiting request. It cannot be exploited by attackers since its too low to make a significant impact to the target server.

The most important metric for the performance of our system is the additional load imposed on the attacked node. Figure 8 shows that the load imposed on that node during an attack for various attack intensities remains low, namely below 2 and when the

percentage of the attackers is below 15%. If the fraction of the attacking nodes is increased beyond 15%, our system can only provide partial protection and the load of the server increases dramatically.

Figure 9 shows the total number of packets transmitted to the overlay network, this includes packets that traversed even one hop and then where dropped. Again, the performance of the system is not affected significantly by the attack even when 15% of the overlay nodes are subverted.

6 Conclusion and Future Work

We have examined the problem of distributed denial of service (DDoS) attacks in peer to peer (P2P) systems that exhibit certain properties due to their structured way they forward packets. This include a broad range of DHT systems including locality aware PASTRY and KADEMLIA[15]. However, our method does not apply to CAN since the traffic arriving from various neighbors cannot be well defined. To our knowledge, this is the first work that examines the problem of insider DDoS attacks in such systems. We presented a distributed and scalable mechanism for identifying anomalous traffic flows. Our main intuition was to exploit certain invariant characteristics of a well behaved DHT-based P2P system, in particular, the tendency of aggregate traffic flows through a node to exhibit roughly equal loads. Attackers that introduce (or drop) excessive traffic are identified by measuring the deviation of their flows from the behavior of other flows, as seen by any node in the system. Our simulations show that our mechanism can be extremely effective, detecting attack traffic with 100% accuracy. Using different attack scenarios applied in real Web traces from IRCache[1], we show that our detection can protect against both uncoordinated and coordinated attacks. For uncoordinated DDoS attacks, our algorithm detects most of the attack traffic even when up to 25% overlay nodes are participating in the attack. In the case of coordinated DDoS attacks, the worst type of attack possible, our mechanism works even when up to 15% of the nodes have been subverted.

Our detection and reaction mechanism is fully distributed: it does not depend on any particular node for its operation. When an attack is detected, information about it is "pushed back" to the incoming neighbors that are better positioned to differentiate misbehaving flows. Such nodes are alerted about the attack and recursively apply our approach.We believe that our investigation opens up an important new area in DHT P2P networks. Future directions include the application of our method in localized PASTRY, KADEMLIA and other systems that have structure and evaluation in a real-world (not simulated) environment.

References

1. Ircache web trace repository, `http://www.ircache.net`
2. Andersen, D., Balakrishnan, H., Kaashoek, M., Morris, R.: Resilient Overlay Networks. In: SOSP (October 2001)
3. Banerjee, S., Kommareddy, C., Kar, K., Bhattacharjee, B., Khuller, S.: Construction of an efficient overlay multicast infrastructure for real-time applications. In: INFOCOM (April 2003)

4. Castro, M., Drushel, P., Ganesh, A., Rowstron, A., Wallach, D.: Secure routing for structured peer-to-peer overlay networks. In: OSDI (2002)
5. Chen-Nee Chuah, R.H.K., Subramanian, L.: DCAP: Detecting Misbehaving Flows via Collaborative Aggregate Policing, vol. 33(5) (October 2003)
6. Dabek, F., Kaashoek, F., Morris, R., Karger, D., Stoica, I.: Wide-area cooperative storage with cfs. In: SOSP (October 2001)
7. Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with CFS. In: SOSP (October 2001)
8. Zhao, B.Y., et al.: Tapestry: A Global-scale Overlay for Rapid Service Deployment. IEEE Journal on Selected Areas in Communications, Special Issue on Service Overlay Networks (January 2004)
9. Kubiatowicz, J., et al.: OceanStore: An Architecture for Global-scale Persistent Storage. In: ASPLOS (November 2000)
10. Ioannidis, J., Bellovin, S.M.: Implementing Pushback: Router-Based Defense Against DDoS Attacks. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (February 2002)
11. Jannotti, J., Gifford, D.K., Johnson, K.L., Kaashoek, M.F., O'Toole Jr., J.W.: Overcast: Reliable multicasting with an overlay network. In: OSDI, October 2000, pp. 197–212 (2000)
12. Keromytis, A.D., Misra, V., Rubenstein, D.: SOS: Secure Overlay Services. In: SIGCOMM, pp. 61–72 (August 2002)
13. Li, Z., Mohapatra, P.: QRON: QoS-aware routing in overlay networks. IEEE Journal on Selected Areas in Communications, Special Issue on Service Overlay Networks (January 2004)
14. Matrawy, A., Oorschot, P.C.: v., Somayaji, A.: Mitigating Network Denial-of-Service Through Diversity-Based Traffic Management. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 104–121. Springer, Heidelberg (2005)
15. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, Springer, Heidelberg (2002)
16. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content-Addressable Network. In: SIGCOMM (August 2001)
17. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
18. Rowstron, A.I.T., Druschel, P.: Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In: SOSP, pp. 188–201 (October 2001)
19. Stoica, I., Adkins, D., Zhuang, S., Shenker, S., Surana, S.: Internet indirection infrastructure. IEEE/ACM Trans. Netw. 12(2), 205–218 (2004)
20. Tran, D.A., Hua, K., Do., T.: A peer-to-peer architecture for media streaming. IEEE Journal on Selected Areas in Communications, Special Issue on Service Overlay Networks (January 2004)

A Appendix

A.1 Chord Invariants and Tests

Proposition 3 *Let $i_1 > i_2$. If the set of flows (S_{i_2}, K) that pass through c is non-empty, then:*

$$\alpha(c, K, S_{i_1}, S_{i_2}) = 2^{i_1 - i_2}.$$

Proof. Assume the set of flows (S_{i_2}, K) that pass through c is non-empty. For each key $k \in K$, let us count the number of sources whose transmissions would pass through node c, entering node c via the i_2-th finger of node $c - 2^{i_2}$ en-route to key k. Recall that the Chord forwarding protocol requires that a transmission be forwarded on the finger that takes the transmission as far as possible in the clockwise direction without passing the destination. Since the i_2-th finger travels a distance of 2^{i_2} around the ring, all transmissions prior to reaching node c must traverse a distance greater than 2^{i_2}, and the sequence of fingers taken after reaching c to then reach k is unique.

Consider a sequence of ℓ bits, $b_1 b_2 \cdots b_\ell$, where $\ell = \log(N) - i_2$, and consider the node s that is at distance $\sum_{j=1}^{\ell} b_j 2^{j+i_2}$ from c in the counter-clockwise direction. Then, for node s to reach k, it will first traverse a series of nodes where it exits through the $j + i_2$-th outgoing finger of some node along the path when and only when $b_j = 1$. After taking this series of fingers, the transmission will end up at node c by taking the i_2-th finger of node $c - 2^{i_2}$ en-route to its final destination of k.

Each of the 2^ℓ possible bit sequences produces a unique node s. Hence, there are 2^ℓ such nodes whose transmissions to k first traverse a path that proceeds through $c - 2^{i_2}$ and takes its i_2-th finger to reach c en-route to k. Furthermore, since the only way a transmission can originate at a source s and reach k by taking the i_2-th finger of $c - 2^{i_2}$ is to previously take a strictly decreasing sequence of fingers, this set of 2^ℓ nodes is unique.

It follows that there are $2^{\log(N)-i_2}$ sources that can reach key k by transiting through the i_2-th finger of $c - 2^{i_2}$, and there are $2^{\log(N)-i_1}$ sources that can reach k by transiting through the i_1-th finger of $c - 2^{i_1}$, so there are $2^{i_2-i_1}$ times as many sources entering the i_1-th finger of c to reach k as there are entering c through its i_2-th incoming finger to reach k.

Since we assume that all sources have (approximately) the same interest in key k, and since this ratio remains fixed irrespective of the final distance of key k from c, the Lemma holds.

A.2 Evaluation Results

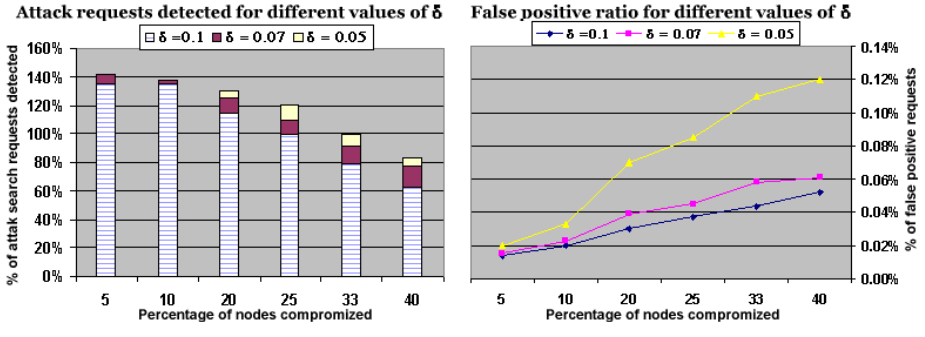

Fig. 10. The left graph shows the increase in the detection rate as we decrease our tolerance threshold from $\delta = 0.1$ to $\delta = 0.05$. The right graph shows the impact of this increase in the false positive percentage: lowering δ below 0.07 offers little benefit to the detection rate while doubling false positives.