Exploiting the Structure in DHT Overlays for DoS Protection

Angelos Stavrou* Angelos D. Keromytis* Dan Rubenstein†
*Department of Computer Science †Department of Electrical Engineering
Columbia University
New York, NY
{*angel,angelos,danr*}@cs.columbia.edu

Peer to Peer (P2P) systems that utilize Distributed Hash Tables (DHTs) provide a scalable means to distribute the handling of lookups. However, this scalability comes at the expense of increased vulnerability to specific types of attacks. In this paper, we focus on insider denial of service (DoS) attacks on such systems. In these attacks, nodes that are part of the DHT system are compromised and used to flood other nodes in the DHT with excessive request traffic.

We devise a distributed lightweight protocol that detects such attacks, implemented solely within nodes that participate in the DHT. Our approach exploits inherent structural invariants of DHTs to ferret out attacking nodes whose request patterns deviate from "normal" behavior. We evaluate our protocol's ability to detect attackers via simulation within a Chord network. The results show that our system can detect a simple attacker whose attack traffic deviates by as little as 5% from a normal request traffic. We also demonstrate the resiliency of our protocol to coordinated attacks by up to as many as 25% of nodes. Our work shows that DHTs can protect themselves from insider flooding attacks, eliminating an important roadblock to their deployment and use in untrusted environments.

1 Introduction

Peer to Peer (P2P) systems are a novel and powerful way to create decentralized services for various applications. Due to their flexibility they are used for content distribution and multimedia streaming [9, 14, 4, 23] ,network storage [8, 19, 13], , resilience [3] and DoS protection [12, 11, 15]. Current P2P systems can be categorized into two distinctive groups unstructured and structured peer to peer systems.

Unstructured Peer to peer systems with randomized searches such as Gnutella [1], Kazaa [2] Freenet[6] became increasingly popular due to their content distribution abilities. These systems manage to operate fairly well and maintain robustness even under extreme uncooperative environments and flash crowds [20, 21]. This is one of the main reasons that these systems are currently used for content distribution.

On the other hand, advances in the Distributed Hash Table(DHT) construction techniques [10] led to the formulation of structured peer to peer overlay networks like CHORD [7], CAN [17], PASTRY [18] and TAPESTRY [24]. Structured P2P systems do not flood the network with search requests since they provide a small upper bound to the number of hops per search request. In addition they provide load balancing [16] and reliability. All of the above come at the cost of maintaining a network structure by using a well defined routing table at each node.

One of the advantages of structured Peer to Peer (P2P) systems is the homogeneity of the task that is assigned to each node. By design, every node plays an identical role: it is responsible for knowing the location, or storing its fair share of content, maintaining its fair share of neighbor nodes to which it can forward requests, and handling its 'air share of routing traffic. Here, 'fair share" means that, statistically speaking, the utilization of a node's memory, processing, and network bandwidth in a well balanced system will be the same for all participants.

There is a stark contrast between the homogeneity of the players in a structured P2P system and the Internet environment upon which these systems operate. For example, artifacts of the network topology can cause two otherwise identical routers to experience very different traffic loads. Routers closer to the center of the network act as a transits for numerous flows heading in all directions (interfaces), while routers closer to the network edges may experience traffic flows heavily weighted in particular directions.

The homogeneity experienced in structured P2P systems allows for a certain predictability of the system behavior. Previous work [22, 17] reveals that, with high probability, there is a natural load-balancing phenomenon: in a large system, every node can expect to play essentially the same role. We posit that it is the homogeneity and the predictability of these systems that are what makes the abstraction so powerful.

However, this homogeneity and predictability is also the P2P systems' Achilles Heel: it is *expected* to behave in this fair and equitable manner, but this need not be the case. In particular, there are three phenomena that P2P designers must concern themselves with:

1. The *content* stored within the system can have varying popularities, creating imbalanced demand.

2. With low probability, there may occur an "unlucky" configuration that violates the desired homogeneity.

3. A deliberate attack on the system geared toward inhibiting normal operations will violate the desired homogeneity

While the first item above has been addressed by [16], to our knowledge no solution has been proposed for the other two problems. In this paper, we investigate methods to handle the imbalance that can arise due to statistical misfortune or deliberate attack. Our methods are applied on aggregate flows [5] but taking into consideration the structure of P2P systems.

We posit that in P2P systems, there are distributed techniques which, by exploiting the assumption that these homogeneities **should** *exist in the system, can infer when the P2P system is behaving erratically.* We demonstrate our position on the Chord system, where we identify certain *invariants* that traffic in a well balanced, homogeneous Chord system should exhibit, and derive distributed protocols that nodes in the Chord system can use to infer whether in fact the system satisfies these invariants.

We evaluate the effectiveness of these distributed protocols in detecting a distributed denial of service (DDoS) attack. The attacks we consider are mounted from within the P2P network: a set of compromised nodes target a particular node with excessive traffic. Other work [12, 15] has examined the use of structured (DHT) systems to prevent DoS attacks from outsiders; here, we consider attacks from insiders. These attackers can be identified not because of the increase in the load with which they inject traffic, but because their distribution of requests is unnaturally biased (with respect to uncompromised nodes) toward the target of the attack. We show that, in a well balanced system, there are certain invariants among the rates of traffic arriving to a node from different neighbors, heading toward the same destination. A node can identify an attack (or a significant imbalance) when it compares a neighbor's forwarded traffic that contains an attack to another neighbor's forwarded traffic that does not contain an attack.

The detection mechanism is based on distributed statistical analysis that compares rates of traffic arriving from different neighbors. It provides a variable margin of false-positive error in detection of excessive rates. In addition, we can produce a detection threshold above which our method will detect the attacker(s) with high probability. This detection threshold depends on the percentage of compromised nodes and their injected traffic relative to normal traffic. In our attack scenarios we vary the both the fraction of P2P participants compromised and the intensity of the attack against an object of the P2P system. Our simulation results show that we can efficiently detect and mark excessive flows even when 25% of the total participants of the P2P network had been compromised for all attack intensities exceeding a specified threshold. Moreover we can detect small fraction of attackers (even one) even when the attack intensity is relatively low. The false positive error is very low: 1%, and can be modified to lower values by changing one of the parameters in the detection algorithm. The detection method uses $O(log^2(N))$ amount of memory per node where N is the number of nodes participating in the overlay network. This result shows that our approach can fit easily within the memory constraints of participating Chord nodes.

The novelty of our approach lies in the exploitation of the structure inherent in these P2P systems with inference-based techniques. In the underlying Internet, inference-based techniques are used to identify, map, and *learn* about certain properties of the environment. In a structured P2P system, the properties that *should* hold are known *a priori*, and the inference techniques are used to detect anomalous, undesirable conditions.

Paper Organization Section 2.2 gives a description of the different attack models we consider. In section 2.3 we present the various invariants of DHT systems. Distributed algorithms for the statistical estimation of aggregate flow rates are analyzed in section 3. These methods in conjunction with the invariants presented in section 2.3 to detect excessive flows in a chord system in section 4. Our experimental results for one and multiple attackers where we vary the attack intensity and the fraction of nodes compromised are discussed in section 5. Section 6 concludes the paper and gives some pointer for future work.

2 Structured P2P Model Description

2.1 Introduction

All structured (DHT) P2P systems consist of a set of keys K_{ids} , a set of nodes N and a distributed routing algorithm that all nodes use to select the neighbor to route their requests to, inside the P2P system. In most systems, there exist algorithms that maintain the routing integrity of the P2P system when nodes join or leave. For the purposes of our analysis, we assume that the P2P system always maintains its routing integrity.

The main function of the P2P system is to store and retrieve objects that are hashed in participating nodes, typically using a fixed-size hash function common to all nodes. Each node is assigned a set of keys, meaning that the node either stores in its local database all the objects that hash to these keys or knows their location inside the P2P system. If a node s wants to locate and retrieve an object from the system, it first has to create the objects hash value k. For the remainder of this paper, we will call this hash value k the object's key. Node s uses the object's key k and the routing algorithm to reach the node responsible for this key.

To continue, we have to define the notion of a flow of search requests. A flow is characterized by the pair (s, k), identifying all the search requests that are generated by node s for an object that hashes to a key k. We can easily extend the notion of a search request flow to an aggregate flow of search requests. We denote by (S, K) to be the aggregate flow of search requests from a set of nodes S to a set of keys K.

In addition, for each of these search request flows (s, k), we define $\lambda_{s,k}$ to be the rate at which these search requests are injected into the P2P network. The popularity of an object k is measured by the aggregate rate of search requests $\lambda_{N,k}$ that the object receives from all the nodes N of the P2P system. By dividing the popularity $\lambda_{N,k}$ with the total number of nodes N, we get the *average popularity* λ^k_{avg} that only depends on the popularity of object k:

$$\lambda^k_{avg} = \frac{\lambda_{N,k}}{N}$$

In our model, we assume that objects stored in the P2P system may have different popularities. Initially, we also assume that for a fixed a object k, the popularity of this object is the same among the participants of the P2P network:

$$\lambda_{i,k} = \lambda^k_{avg},\ \forall i \in N, k \in K_{ids}$$

Of course, it is unlikely that individual members of the P2P network will exhibit similar popularity behavior. In our analysis, we relax this assumption by requiring only *aggregates* of large, randomly selected sets of P2P nodes to have the same popularity for the same object. Our assumption is justified by the application of the law of large numbers coupled with the fact that the nodes forming the groups are randomly selected.

2.2 DoS Attackers & Attack Scenarios

We now give a precise definition of what we consider a DoS attack in a P2P network environment. We shall only consider DoS attacks against a specific object (or set of objects) stored in the P2P

network. These attacks are mounted by nodes *inside* the P2P network. We assume that a malicious insider uses the P2P infrastructure to mount this attack by injecting excessive search requests. Later we shall justify this assumption by showing that this can be achieved by allowing traffic only from incoming neighbors to be routed or served.

We define an attacker to be a node (or a set of nodes) inside the P2P network which tries to create a starvation of network or computational resources to the node servicing the attacked objects. This denial of service is achieved by injecting excessive search requests for that objects into the P2P network.

Under 'normal' P2P network operation, we expect that the rate of a search request flow $\lambda_{S,k}$ from a fairly large, randomly selected group of nodes S towards a set of keys K, will be equal to the popularity of that object λ_{avg}^{K}.

To quantify an attack, we give the definition of a misbehaving flow of search requests: we characterize a flow (s, K) as "misbehaving" if the rate of search requests originating from a node s toward a set of keys K, $\lambda_{s,K}$ exceeds the average rate λ_{avg}^{K} by a factor of $delta$. We set this rate threshold to be λ_{max}^{K}:

$$\lambda_{s,K} > (\delta + 1) \cdot \lambda_{avg}^{K} = \lambda_{max}^{K}$$

where δ represents how many times bigger must the rate of a flow be, compared to the average search request rate, before we declare it as "misbehaving". The previous notion can be extended to a set of nodes S:

$$\lambda_{S,K} > \|S\| \cdot (\delta + 1) \cdot \lambda_{avg}^{K} = \|S\| \cdot \lambda_{max}^{K}$$

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 variance in popularity 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. A small δ allows little variance on the popularity towards a set of keys K. Larger values allow more variance on the popularity, 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 consider the cases where there is either one attacker or a set of attack nodes that are participating in a denial of service attack against the object(s) stored at a specific node, k. The attack is easy enough to mount for an attacker that has infiltrated the DHT network: the attacker simply identifies an object that hashes to the particular node[1] and sends an inordinate number of queries toward that object. To simplify our analysis, we assume that in a distributed denial of service attack, the attack effort is distributed evenly among the attacking nodes. For each of the attackers, we define β to be the proportion by which the attacker increases its traffic toward k above the normal popularity of the objects. In other words, if a "normal" node transmits queries to k at a rate of λ_{avg}^{k}, the attacker transmits queries to k at a rate of $(\beta + 1) \cdot \lambda_{avg}^{k}$.

If N_a is the number of attacking nodes, the total amount of excessive search requests injected into the system by the attackers is $(\beta \cdot N_a \lambda_{avg}^{k}$. Let $f = N_a/N$ be the fraction of nodes compro-

[1]Even easier, the attacker may be interested in attacking a particular object.

mised; we define the attack intensity, $\beta \cdot f$, to be the increase in the popularity of the target object caused by the attackers' excessive search requests:

$$\beta \cdot f = \frac{\lambda^k_{(attc-avg)} - \lambda^k_{avg}}{\lambda^k_{avg}}$$

This definition of traffic intensity is based on the fact that the contribution of the attack traffic to the overall popularity of the object is $\beta \cdot f$. For example, if $\beta \cdot f = 1$, 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, $\beta \cdot f$ 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 inject significantly more traffic toward k (high β).

In the experimental section, we shall examine the following scenarios to evaluate our detection tests in different adverse environments:

1. A Single Node attacks the network launching excessive search requests toward a set of keys.
2. Multiple Nodes (Uncoordinated) attack the network using excessive search requests toward a single key. The nodes attack do not know of each other. This is the case of different unrelated nodes trying to attack the network.
3. Multiple Nodes (Coordinated) attack a single key. The nodes know of the other attackers' position on the network, but do not have a complete map of the P2P topology.

2.3 Invariants of Structured DHT systems

In a DHT P2P network, each flow (s, k) is associated with one or more vectors of nodes that define a path, or a set of paths $P(s, k)$, from node s toward key k inside the P2P network. Since our main analysis is focused on Chord [22], we will assume that each path $P(s, k)$ is a unique vector of nodes. Additionally, we define $P_c(s, k) \subseteq P(s, k)$ to be the subset of paths from node s toward key k that pass through node c. Being part of a DHT network, each node c receives all the search requests from only a small, well defined set of nodes, the incoming neighbors In_c. We define In_c to be:

$$In_c = \bigcup Prev(s, k, c), \forall\ s \in N,\ k \in K_{ids}$$

where

$$Prev(s, k, c) = \begin{cases} \text{previous node id on path } P_c(s, k) \\ \emptyset \text{ otherwise.} \end{cases}$$

Node c looks for the key in its local database. If the lookup is unsuccessful, c forwards the search request to another set of nodes, the outgoing neighbors, Out_c, defined as:

$$Out_c = \bigcup \{n \in N : Prev(s, k, n) = c,\ s \in N,\ k \in K_{ids}\}$$

Each P2P network provides the mechanisms to discover both In_c and Out_c. These sets are relatively small, *e.g.,* $O(log(N))$, compared to the total number of nodes N. Their size depends only on the structure of the P2P network and node c's position, which is usually encoded in c's ID. For some P2P systems (*e.g,* CAN), the set of incoming and outgoing neighbors are the same. To elaborate, node c receives search requests for key k from a subset of its incoming neighbors In_c and forwards it to a subset of its outgoing neighbors Out_c. Thus node c receives all the search requests from the set of its incoming neighbors In_c. Moreover c's outgoing neighbors serve or forward all the search requests that node c generates for all the keys. The number of search request flows (s, k) that node c forwards for key k to a subset of its outgoing neighbors Out_c, depends on how "close" node c is to the key k. Closeness is defined in terms of either hops or Euclidean distance, depending on the P2P system.

We can estimate the rate of search requests that node c routes or serves under normal operation for a key k, assuming that node c is on the path of this key. This is done by aggregating the rate of all the search requests that c receives for that key k from all its incoming neighbors (incoming fingers). This means that for any set of keys K, the requests for which pass through node c, we can estimate its average search request rate (popularity). We can then compare the rate of search requests node c receives from different incoming neighbors toward that set of keys K with node c's estimated popularity. More formally:

a) *The ratio of the search request rates $\lambda^c_{S,K}$, $\lambda^c_{S',K}$ that a node c routes or serves for the set of keys K from sets of sources S, S' is $\alpha(c, K, S, S')$, where :*

$$\alpha(c, K, S, S') = \frac{\lambda^c_{S,K}}{\lambda^c_{S',K}} = \frac{\lambda^c_{S,K}}{\lambda_{S',K}} = \frac{\|S\| \cdot \lambda^K_{avg}}{\|S'\| \cdot \lambda^K_{avg}} = \frac{\|S\|}{\|S'\|}$$

The above equation holds under the assumption that we have unique paths. In the general case of a DHT system with non-unique paths, we will have to add a parameter denoting the fraction of packets of flow (s, k) that are routed through c.

Thus, node c can aggregate and compare all the rates of flows of the form $(s, K),\ s \in S, k \in K$ and $(s, K)\ s \in S', k \in K$ that it routes or serves. We typically select set S' to be the total number of nodes that are allowed to be routed through node c for the set of keys K, and set S to be the nodes that arrive through a specific neighbor of c.

Since the sets S, S' contain nodes that are randomly selected from different areas, when they are large enough (in terms of the number of nodes that they contain) and of equal size, we have that $\lambda_{S,k} = \lambda_{S',k}$ (on the limit).

b) *Another property of a structured P2P system involves estimating the ratio of request distributions (popularity) of keys (or set of keys) using measurements of search request rates in a node: node c compares the popularity of sets of keys K_1, K_2 by estimating the ratio of the average search request rates that node c serves for these keys or set of keys.* More formally:

$$\frac{\lambda^{K_1}_{avg}}{\lambda^{K_2}_{avg}} = \frac{\frac{\lambda^c_{S,K_1}}{\|S\|}}{\frac{\lambda^c_{S,K_2}}{\|S\|}} = \frac{\lambda^c_{S,K_1}}{\lambda^c_{S,K_2}}$$

The above property can be used to dynamically estimate the load served by specific nodes,

and to employ load-balancing mechanisms to alleviate search request congestion due to increased object popularity.

3 Statistical Estimation and Bounds of Flow Rates

In the previous section we presented some invariant properties about the relative rates of flows that a node c should measure from different sources S, S' toward the set of keys K. We also presented a definition of an excessive flow (S, K) to a set of keys K based on its comparison to the average flow rate towards K.

In this section we present a system that detects "misbehaving" aggregate flows and marks their respective search requests. We will assume that each node in the P2P system has a unique id given by the P2P system at random. The total number of flows in a P2P system is of the order of $O(N \cdot K)$,where N is the total number of nodes in the system and K the number of the keys. Keeping track of each individual flow would require $O(N \cdot K)$ memory. In the case of a fully distributed system, it would require (with high probability) a $O(K)$ memory in each of the N nodes. *To avoid utilizing such a potentially huge amount of memory per node, we only keep track of a small number $O(log(K))$ of aggregate flows per node by exploiting the fact that, for structured P2P systems, all the flows arriving at node c are coming from nodes belonging to the set of incoming neighbors of this node (In_c).* In addition, the flows that are not served by node c are then forwarded to nodes in the set of outgoing neighbors (Out_c) of c. Node c groups the flows that arrive from the incoming neighbor i and are forwarded to the outgoing neighbor j of node c, creating the aggregate flow $F_c(i, j)$. As we mentioned in Section 2, both In_c and Out_c are of small size $O(log(K))$, thus creating a total of $O(log^2(K))$ aggregate flows per node, or $\|F_c(i, j)\| = O(log^2(K)$. To implement this scheme, each node stores a table of search request counters $C_{i,j}$ with $i \in In_c,\ \ j \in Out_c$ of size $O(log^2(K))$.

The detection of a misbehaving flow is done by checking the aggregate flow that this flow belongs to on different nodes along the path of the flow. Then, if a flow aggregate exceeds the average rate of search requests for a set of keys on a node c, this means that one or more flows of the aggregate flow are "misbehaving", exceeding the average rate for that set of keys. When a node detects that aggregate flow (S, K) is exceeding the average rate for a set of keys, that node starts marking the search requests that belong to that aggregate flow.

In general, node c groups the flows by first looking at the last node that forwarded the flow (s, k) . This node must belong to the set of $c's$ incoming neighbors In_c. Node c then looks at the search request's key and finds the outgoing neighbor that it will forward the request to. Node c checks if the flow is valid, meaning if c belongs to the path of the flow (s, k). If the flow is valid, we increase the counter $C_{i,j}$, where $i \in In_c,\ j \in Out_c$ are the IDs of incoming and outgoing neighbors of c that flow (s, k) was received from and will be forwarded to respectively. We define $I(c, i) \subset N, c \in N,\ i \in In_c$ to be the set of nodes that send search requests to node c via its incoming neighbor i, independent of the key requested. Respectively, we define $O(c, j) \subset K_{ids}, c \in N,\ j \in Out_c$ to be the set of keys that node c forwards requests to its outgoing neighbor j.

Before forwarding the search request, node c examines if the aggregate flows are "misbehaving":

Test 1. *Node c computes the rate of search requests that it receives from incoming neighbor i towards outgoing neighbor j using the counter $C_{i,j}$. It then computes the total rate of search requests C_j that it receives from all incoming neighbors for the same outgoing neighbor j:*

$$C_j = \sum_{i \in In_c} C_{i,j}$$

In addition, the results from Section 2, node c calculates the proportion of the rate of search requests that should arrive, when no attack is taking place, from incoming neighbor i towards the outgoing neighbor j (or the sets of keys $O(c, j)$ that c forwards to j):

$$\alpha(c, K, S, S_t),\ K = O(c, j),\ S = I(c, i),\ S_t = \bigcup_{m \in In_c} I(c, m)$$

Since

$$\alpha(c, K, S, S_t) = \frac{\lambda^c_{S,K}}{\lambda^c_{S_t,K}}$$

we get that, under no attack,

$$\lambda^c_{S,K} = \alpha(c, K, S, S_t) \cdot \lambda^c_{S_t,K}$$

If we take into consideration the fact that we allow the aggregate rate $\lambda^c_{S,K}$ to be at most δ times more than the mean rate, the previous equation becomes:

$$\lambda^c_{max}(S, K) = \delta \cdot \alpha(c, K, S, S_t) \cdot \lambda^c_{S_t,K}$$

Node c is now in position to use the measured aggregate search requests $C_{i,j}$ received from incoming neighbor i along with the total search requests C_j received from all the incoming fingers toward the outgoing finger j to calculate if flows comming from i are "misbehaving". To achive this c sets up a binomial distribution, where a success is considered a request from i and a failure is a request from all the other allowed fingers. The probality of success can be computed to be:

$$\mathbb{P}_{succ} = \frac{\lambda^c_{max}(S, K)}{(\lambda^c_{S_t,K} - \lambda^c_{S,K}) + \lambda^c_{max}(S, K)} =$$

$$\frac{(\delta + 1) \cdot \alpha(c, K, S, S_t) \cdot \lambda^c_{S_t,K}}{(\lambda^c_{S_t,K} - \alpha(c, K, S, S_t) \cdot \lambda^c_{S_t,K}) + (\delta + 1) \cdot \alpha(c, K, S, S_t) \cdot \lambda^c_{S_t,K}}$$

or

$$\mathbb{P}_{succ} = \frac{(\delta + 1) \cdot \alpha(c, K, S, S_t)}{1 + \delta \cdot \alpha(c, K, S, S_t)}$$

The maximum allowed number of packets X_{i,C_j} that finger i is allowed to send to finger j under normal operation can be found by computing the reverse probability from:

$$\mathbb{P}(X_{i,C_j} \geq C_{i,j}) < \epsilon$$

where ϵ is the confidence interval and X_{i,C_j} is the random variable denoting the maximum allowed search requests from incoming finger i given ϵ. If the measured C_j is bigger than the computed value of X_{i,C_j} then c declares the aggregate flow as misbehaving with a confince $1-\epsilon$. For 99.9% confidence, we choose $\epsilon = 0.001$. In practice we create a table that contains the maximum allowed values X_{i,C_j} for different values of C_j and $\mathbb{P}_{succ}$.

For large values of $C_{i,j}$, C_j, the computation of the success probability becomes time intensive, so we approximate it with a Poisson random variable X_p with rate:

$$\lambda_{i,C_j} = \mathbb{P}_{succ} \cdot C_j = \frac{(\delta+1)\cdot\alpha(c,K,S,S_t)}{1+\delta\cdot\alpha(c,K,S,S_t)}\cdot C_j$$

and $N = C_j$ with unit time period $T = 1$.

Test 2. *Another test that node c can perform is to compare the traffic that it receives from different incoming neighbors for the same outgoing neighbor. Node c computes the number of search requests routed toward outgoing neighbor j from its incoming neighbors i_1,i_2 with counters $C_{i_1,j}$ and $C_{i_2,j}$ respectively. Using the results from Section 2, node c can calculate the proportion of the rate of search requests that come from incoming neighbors i_1,i_2 toward the outgoing neighbor j.*

$$\alpha(c,K,S_{i_1},S_{i_2}), K = O(c,j),\ S_{i_1} = I(c,i_1),\ S_{i_2} = I(c,i_2)$$

Since we are trying to infer frequence of events, we use the binomial distribution with $N = C_{i_1,j} + C_{i_2,j}$ and probability of success:

$$\mathbb{P}_{succ} = \frac{(\delta+1)\cdot\lambda^c_{S_{i_1},K}}{\lambda^c_{S_{i_2},K} + (\delta+1)\cdot\lambda^c_{S_{i_1},K}} =$$

$$\frac{(\delta+1)}{1+\frac{(\delta+1)\cdot\lambda^c_{S_{i_2},K}}{\lambda^c_{S_{i_1},K}}} = \frac{(\delta+1)}{1+(\delta+1)\cdot\alpha(c,K,S_{i_2},S_{i_1})}$$

Node c then checks if $\mathbb{P}(X_{i_1,i_2,N} > C_{i_1,j}) < \epsilon$, where ϵ is the confidence interval and $X_{i_1,i_2,N}$ is a binomial random variable with probability of success $\mathbb{P}_s$, which counts the number of samples from incoming finger i_1 when N samples have arrived from i_1 and i_2. If $\mathbb{P}(X_{i_1,i_2,N} > C_{i_1,j})$ is less than ϵ, we say that with $(1-\epsilon)\%$ accuracy the flow is "misbehaving" and we split it in smaller flows. For 99% confidence we choose $\epsilon = 0.01$.

For large values of N, we approximate this distribution by a Poisson distribution with $\lambda_{i_1,i_2,N} = C_i \cdot \mathbb{P}_{succ}$, $N = C_i$ and c computes the probability $\mathbb{P}(X_p) > C_{i,j}) < \epsilon$, where X_p is a Poisson random variable.

Our second test can detect differences in the rate of search requests coming from different incoming neighbors toward the same set of keys that c forwards to a specific outgoing neighbor. The first test detects differences between the rate of search requests from an incoming neighbor for the sets of keys that c forwards to a specific outgoing neighbor, and the total rate of search requests that node c receives from all incoming neighbors for the same set of keys.

4 Chord System

In this section we will apply the tests presented in the previous sections to detect and prevent distributed denial of service attacks (DDOS) within a Chord ring. The chord model used in this section is the chord model presented in original paper[].

4.1 Chord Model Definitions

A chord peer to peer system is consisted of a set nodes 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 facilitate things all the calculus done from this point on are modulo K.Each node is assigned an id from the key space ($N \subseteq K_{ids}$) thus creating also a node ring. Since we have a circular placement we have for each node c a successor node and a predecessor node. As we mentioned each node is assigned a set of keys meaning that the node either stores in its local database all the objects that hash to these keys or knows their location. Thus when a search request reach the node responsible for the key requested the search stops and a reply to the originator of the request is issued either with the actual object or with the object's location.

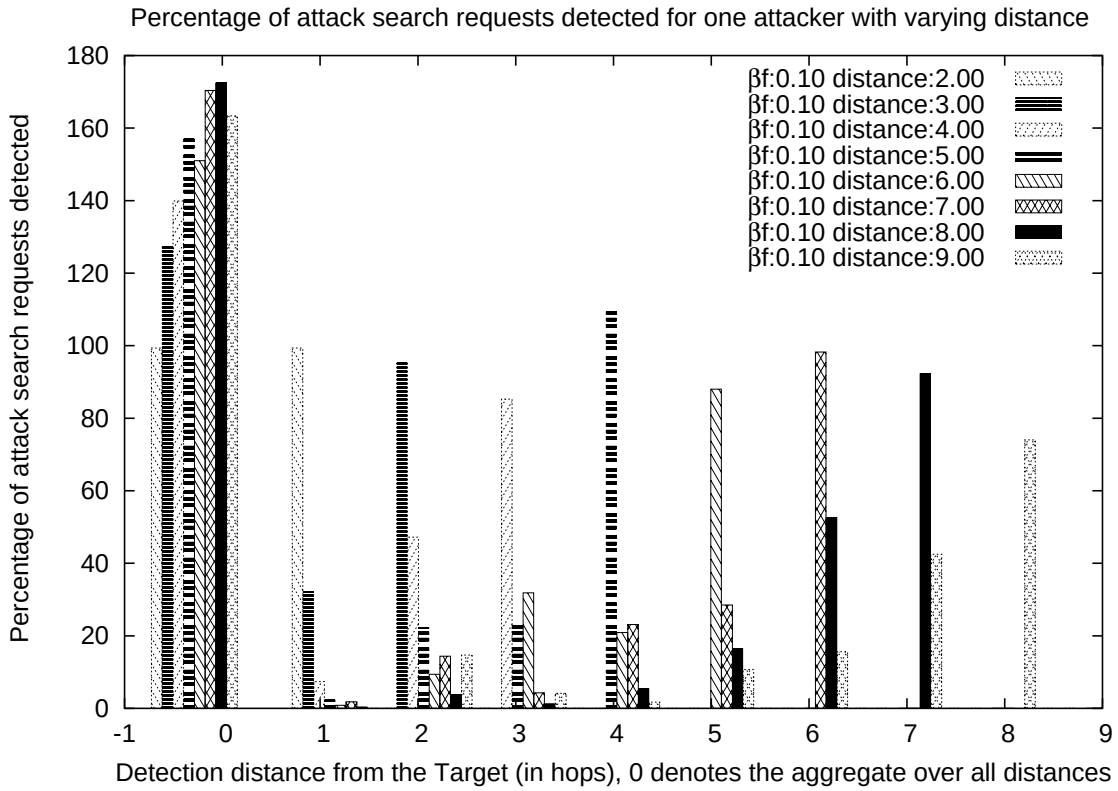

Figure 1: Percentage of excessive search requests detected and the detection distance when we vary the distance of the attacker to the target in a 1024 Chord ring. Distance 0 shows the aggregate attack requests detected for each distance.

4.2 Chord Invariant properties and Tests

Proposition 1. *For a chord node c, $c \in N$, we can compute the rate of search requests it receives from sets of nodes S_{i_1}, S_{i_2} that pass through its fingers ($i_1, i_2 \ \in In_c$) respectively, for the set of keys $K = O(c, j)$ that node c forwards to its j outgoing finger with $j < min(i_1, i_2)$. Then we use the property 1 of section 2 to compute the expected relative rate of search requests $\alpha(c, K, S_{i_1}, S_{i_2})$ for a system with no "misbehaving" flows. More formally we have:*

$$\alpha(c, K, S_{i_1}, S_{i_2}) = \frac{\lambda^c_{S_{i_1},K}}{\lambda^c_{S_{i_2},k}} = \frac{\lambda_{S_{i_1},K}}{\lambda_{S_{i_2},k}} \tag{1}$$

$$K = O(c, j),\ S_{i_1} = I(c, i_1),\ S_{i_2} = I(c, i_2)$$

For a full chord system with no attackers we are expecting to have:

$$\alpha(c, K, S_{i_1}, S_{i_2}) = \frac{\|S_{i_1}\|}{\|S_{i_2}\|} = \frac{\|I(c, i_1)\|}{\|I(c, i_2)\|} = \frac{2^{(m-i_2-1)}}{2^{(m-i_1-1)}} = 2^{i_2-i_1}$$

Proof. For this proof we will need the following lemma:

Lemma 1. *In case of full chord the number of distinct node ids that come through the i^{th} incoming neighbor (finger) of node c, or ($\|I(c, i)\|$), $with\ 0 \leq i \leq (m-1)$ to route their search requests to the allowed key set $K = c+2^o,\ o < i$ (not to terminate in node c) are $\|I(c, i)\| = N_r(i) = 2^{(m-i-1)}$ independent of the selection of node c for all the allowed keys K.*

Proof. We have that the number of nodes that can contact node c through its i^{th} incoming finger include first of all the actual node with id $P_i = (c - 2^{(i)})mod(K)$. We also have to add all nodes that corespond to the incoming fingers j with $j > i$ of node P_i. Using this argument recursively we obtain the total number of nodes $N_r(i)$ by the following formula :

$$N_r(i) = 1 + \underbrace{N_r(\text{i + 1}) + N_r(\text{i + 2}) + \ \ldots\ + N_r(\text{m - 1})}_{m-i-1},\ \ 0 \leq i \leq (m-1)$$

In this formula $f(m-1) = 1$ and $f(j) = 0,\ \forall j > m-1$ because we now that for the maximum finger in our input finger table we have only one node to count since there is no entry in any node's input finger table for distance greater than $(m-1)$. We see that:

$$N_r(i) = 1 + \sum_{l=i+1}^{l=m-1} N_r(l),\ N_r(i+1) = 1 + \sum_{l=i+2}^{l=m-1} N_r(l)$$

$$\text{Let } a_i = 1 + \sum_{l=i+1}^{l=m-1} N_r(l) \text{ then } a_i = 1 + 2 \cdot a_{i+1} \text{ and } a_{i+1} = 1 + 2 \cdot a_{i+2} \ \Rightarrow$$

$$a_i = 2 + 2 \cdot a_{i+2} = 2 \cdot (1 + a_{i+2}) = 2 \cdot (1 + (1 + 2 \cdot a_{i+3}) \Rightarrow$$

$$a_i = 2^{j-1}(1 + a_{i+j}), 0 \leq j \leq (m-i-1)$$

$$a_i = 2^{(m-i-2)}(1 + a_{m-1}) \ \text{ for } \ j = (m-i-1).$$

Using the fact that:

$$a_{m-1} = \sum_{l=m-1}^{l=m-1} N_r(l) = N_r(m-1) = 1$$

we get that:

$$a_i = 2^{(m-i-2)}(1+1) \Rightarrow a_i = 2^{(m-i-1)}$$

and we know by definition that $N_r(i) = a_i,$ so $N_r(i) = 2^{(m-i-1)}$ □

Corollary 1. *The total number of nodes that can route through node c using node's c outgoing finger o (without including c) are:*

$$N^{(r)}_{(i>o)} = 2^{(m-o-1)} - 1.$$

independent of the selection of node c.

We now return to the proof of equation (1): From lemma (1) for two different input fingers i_1, i_2 the number of distinct chord nodes and the coresponding flows (S, k) and (S, k) that pass through them for a single key are:

$$N_r(i_1) = 2^{(m-i_1-1)} \text{ and } N_r(i_2) = 2^{(m-i_2-1)} \Rightarrow \alpha(c, K, S_{i_1}, S_{i_2}) = 2^{(i_2-i_1)}$$

for every key k . The ratio number of flows $\frac{(S_{i_1},K)}{(S_{i_2},K)}$ don't depend on the selection of set of keys K but rather on the ratio of the number nodes in the sets S_{i_1} and S_{i_2}. This happens because all flows that pass through incoming fingers i_1 and i_2 for the set of keys $K = O(c, j)$ are valid because $j < min(i_1, i_2)$ and the set of keys K is common.

Proposition 2. *Node c can compute the proportion of the rate of search requests it receives from its i_1^{th} incoming finger ($i_1 \in In_c$), with the total rate of search requests that c receives from all allowed incoming fingers, for the set of keys $K = O(c, j)$ that node c forwards to its j outgoing finger with $j < i_1$. Then we use the property 1 of section 2 to compute the expected relative rate of search requests $\alpha(c, K, S, S_t)$ for a system with no "misbehaving" flows. More formally we have:*

$$\alpha(c, K, S, S_t) = \frac{\lambda^c_{S,K}}{\lambda^c_{S_t,k}} = \frac{\lambda_{S,K}}{\lambda_{S_t,k}} \tag{2}$$

$$K = O(c, j),\ S = I(c, i_1),\ \ S_t = \bigcup_{i>j, i\in In_c} I(c, i)$$

For a full chord system with no attackers we are expecting to have:

$$\alpha(c, K, S, S_t) = \frac{\|S\|}{\|S_t\|} = \frac{\|I(c, i_1)\|}{\sum_{i>j, i\in In_c} \|I(c, i)\|} = \frac{2^{(m-i_1-1)}}{2^{(m-j-1)} - 1} = \frac{1}{2^{i_1-j} - 1}$$

Proof. The proof for this test is easily derived by the use of the corrolary (1) Now using the previous two properties and the generalized tests defined in the tests of the previous section we can define the following tests:

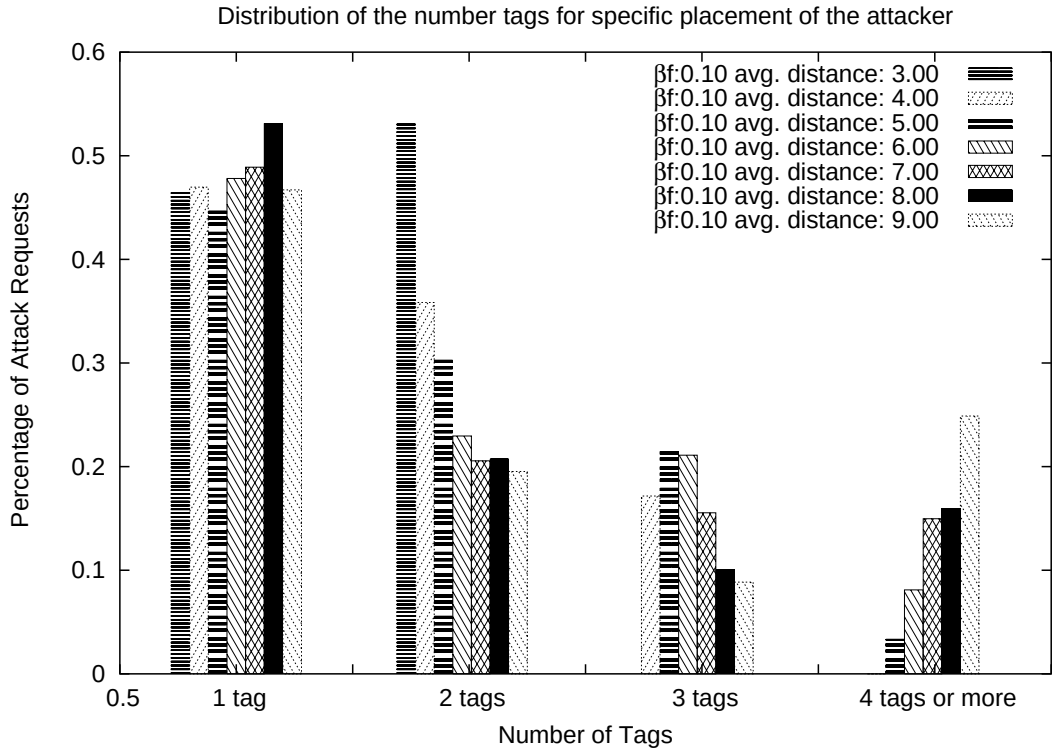

Figure 2: Distribution of the number of tags for the attack requests for one attacker in 1024 chord ring. The different plots represent the attacker's distance in hops from the target for a $\beta f = 0.1$.

5 Simulation Results

5.1 Experiments Setup

To evaluate our detection algorithm we use an implementation of the Chord system. In this system we randomly select a portion of the nodes to become the "attackers" and object from the set of allowed keys. The node that stores that object (assigned by the hash function of the DHT) becomes the "target". The goal of the attackers is to overwhelm the "target" with search requests so the target becomes unresponsive to legitimate search requests leading to a denial of service. In general the attackers are be allowed to select multiple "targets" but this will only lower their attack intensity since they will have to split their search request rate among different targets.

To quantify the ability of our algorithm to detect the attackers we introduced in section 2.2 the notion of attack intensity βN_a, to be the increase above the normal popularity of the target object ,k, caused by the attackers' excessive search requests. In other words, if a "normal" node transmits queries to k at a rate of λ^k_{avg}, the attack search requests to k from all the compromises nodes in the system have a rate of: $(\beta f + 1)\lambda^k_{avg}$ causing an increase in the popularity of the object that is measured by:

$$\beta \cdot f = \frac{\lambda^k_{(attc-avg)} - \lambda^k_{avg}}{\lambda^k_{avg}}$$

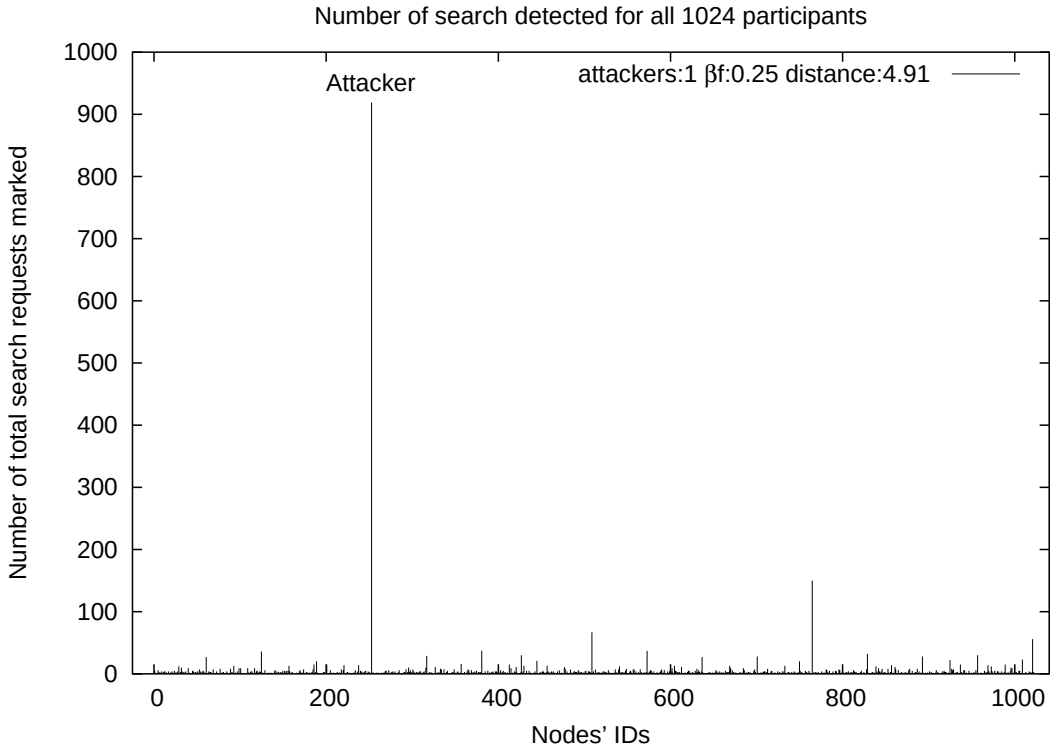

Figure 3: Total excessive packets detected when we vary both the attack intensity βf and the fraction of nodes compromised for a 4096 participants chord ring. Notice that in some cases we overestimate the number of excessive requests and that is why we have values above 100%.

Our attack detection method identifies aggregates that contain attackers by marking packets within these aggregates whenever our estimates predict that the aggregate is sending at too high of a rate. Our detection mechanism introduces a false positive source of error inherited form the aggregate detection mechanism. This error can become arbitrarily small by selecting a higher confidence interval. For our simulations we used a 0.999% confidence interval. Every nodes generated 2048 search requests on average with at total of approximately $2x10^6$ per simulation. Each simulation was repeated more that 20 times and the results we present are averages of these simulations. Were 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. Moreover our detection mechanism should be able to detect the attack as close to the attacker as possible and the malicious search requests should be marked as many times as possible along the path from the attacker to the target object. It appears that both of these factors for the one attacker scenario are dependent on the attack intensity and the attacker's distance to the attacked object.

5.2 Single attacker scenarios

We first investigated the scenario where one of the nodes participating in the overlay has been compromised and attacking a single object. Since we assume only one malicious node the attack

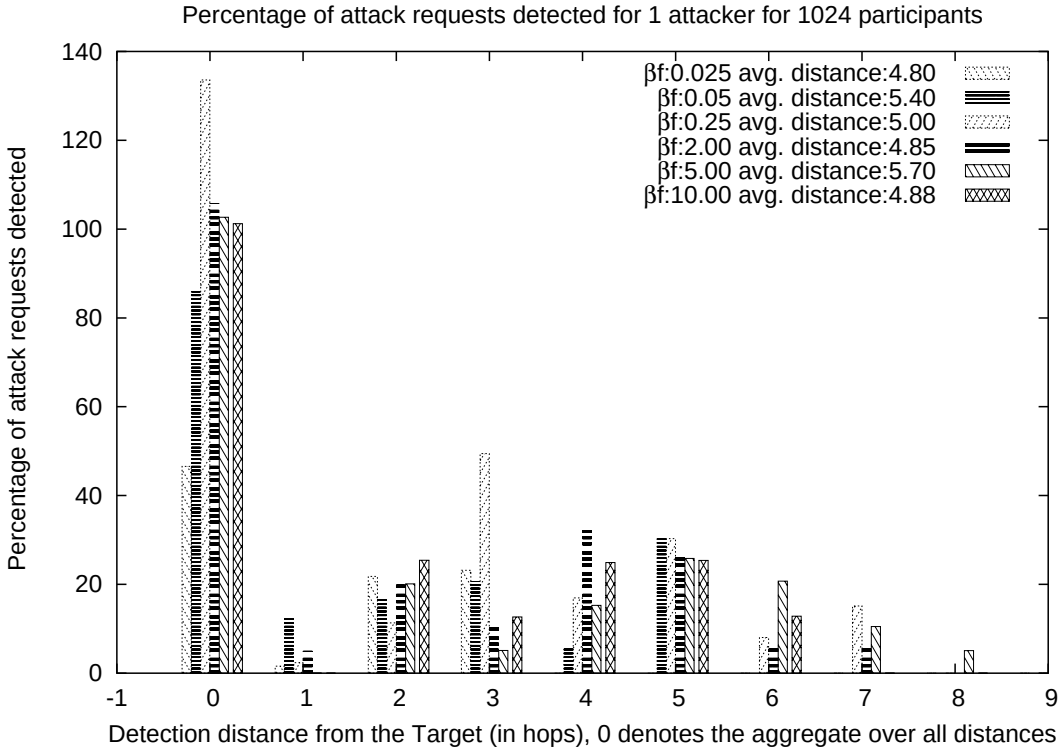

Figure 4: Percentage of attack search requests detected for one attacker randomly placed in a 1024 Chord ring and their distance in hops from the target. Distance 0 shows the aggregate attack requests detected. The different plots represent increasing values of attack intensity βf. Since these plots are the average of multiple experiments (100 for each plot), the average distance of the attacker from the target is also displayed

intensity becomes:

$$\beta \cdot f = \beta \cdot \frac{1}{N} = \frac{\beta}{f}$$

In order for the single attacker to have impact on the search requests of the attacked object $\beta \cdot f$ should be large. For example for $\beta \cdot f = 1$ the single attacker has to inject search requests in the system with rate $N \cdot \lambda_{avg}^{k}$ or with a $\beta = N$. Someone can argue that this is too much of a rate but in practice this depends on the popularity λ_{avg}^{k} of the object attacked. If the object is highly popular and λ_{avg}^{k} is large compared to the rest of the object in the system the attacker will need to inject a bigger amount of search requests to further increase its popularity.

For the first experiment in the one attacker scenario we placed the attacker in various distances (in hops) away from the target node. Then we observed the percentage of the excessive search requests our algorithm detected and their detection distance from the target. Distance zero denotes the total percentage of the search requests we detected independent of the distance from the target. Figure 1 shows our results. Notice that as the distance of the attacker from the target increases the detection distance also increases. For example if we place the attacker at a distance 9 we have that the bigger portion of its packers were detected its next hop neighbor on the path to the target. Moreover when the attacker is very close, lets say distance 2, we detect 100% of the excessive search requests. As the distance from the target increases we start over-marking the attacker requests. This happens because as we move away from the target more flows generated by attacker participate in the groups of flows we detect and thus marked. Moreover in this experiment we used an attack intensity of $\beta f = 0.1$ or a 10% increase in the popularity of the attacked object which is a relatively small value. For larger values of βf we have minimal over-marking of the attacker search requests since the excessive requests become a bigger portion of the total search requests.

As we mentioned in our experiments each node tags all the flows of a group that appears to be “misbehaving”. Figure 4 shows the distribution of the tags when we vary the distance of the attacker to the target. We see that depending on the distance of the attacker we have a significant portion of the excessive search requests with two or more tags from 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 our method detects and marks search requests on group of flows an inherent problem of our method comes up: a group contains both misbehaving and legitimate flows so we maybe end up marking legitimate flows along with the attacking ones. As figure 3 shows that is not the case. Although we are marking flows belonging to the same “misbehaving” group we are punishing mostly the attacker since he is the one sending the majority of the search requests through that group. All the other flows are getting marked but only minimally, at least the majority of them, in comparison both to the total number of requests they generate and in comparison with the attacker. Of course blindly marking and dropping excessive requests from a misbehaving flow is a very crude method prevent a denial of service but we can do it in cases that our resources are limited.

All the previous analysis was done to ensure that there is no attacker placement inside the chord ring that our algorithm fails to detect. Now we are in position to present more realistic results from simulations in which we have one attacker randomly placed in a chord ring of size 1024 (see figures 4,5). Figure 4 depicts the percentage of the attacker requests detected and their corresponding detection distance from the target. It is easy to see that for attack intensity values

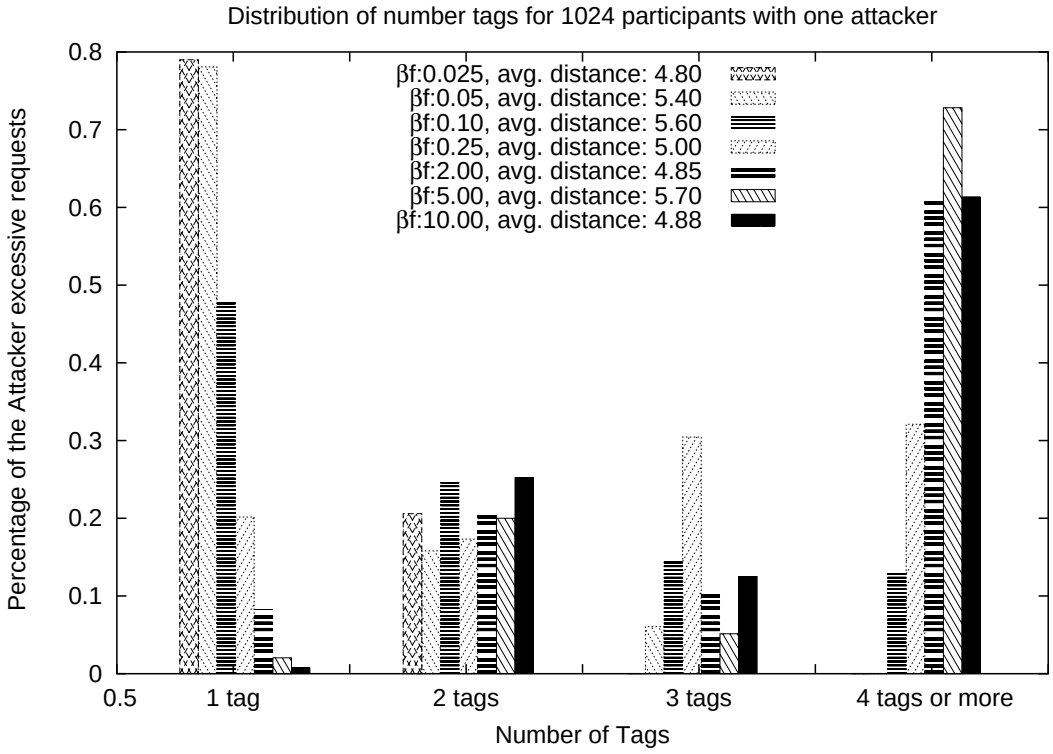

Figure 5: Distribution of the number of tags for the attack search requests. In this plot we have one attacker randomly placed in a 1024 ring. The different plots represent increasing values of attack intensity βf. Since these plots are the average of multiple experiments (100 for each plot), the average distance of the attacker from the target is also displayed

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.

On the other hand, 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 5 although 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. For attack intensities that are larger than ones shown in figure 4 we detect all of the attack search requests and thus we have similar results with the ones for $\beta f = 10$. Finally for $\beta f = 0.25$ we have an over-marking of the attacker search requests which fades out when the attack intensity becomes more significant. The reasons for this over-marking where discussed thoroughly in the previous paragraph.

5.3 Multiple attacker scenarios

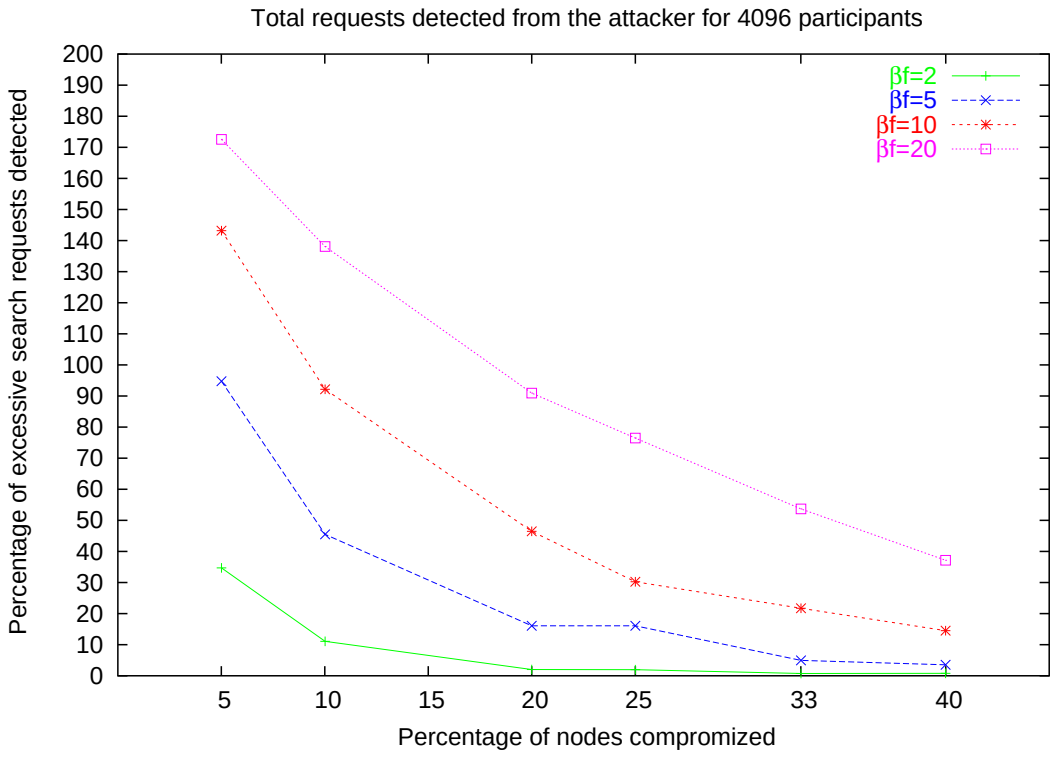

Figure 6: Percentage of excessive packets detected when we vary both the attack intensity βf and the fraction of nodes compromised for a 4096 participants chord ring.

In this section we simulate the behavior of our detection algorithm using a chord ring where we vary both the fraction of nodes compromised and the attack intensity. In figure 6 we present our

simulation results for a 4096 chord ring with multiple attackers. It is clear that there is a dependence 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. In figure 7for example, when the attackers constitute the 5% of all the participants a large portion of their excessive requests have 2 or more tags whereas when the attackers become 40% of the total nodes only 14% of the excessive requests have 2 or more tags.

Notice that in some cases we overestimate the attack packets generated from 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 misbehaving flow from the group of flows detected to be misbehaving. To achieve this we can use sampling on the misbehaving flow or Bloom filters.

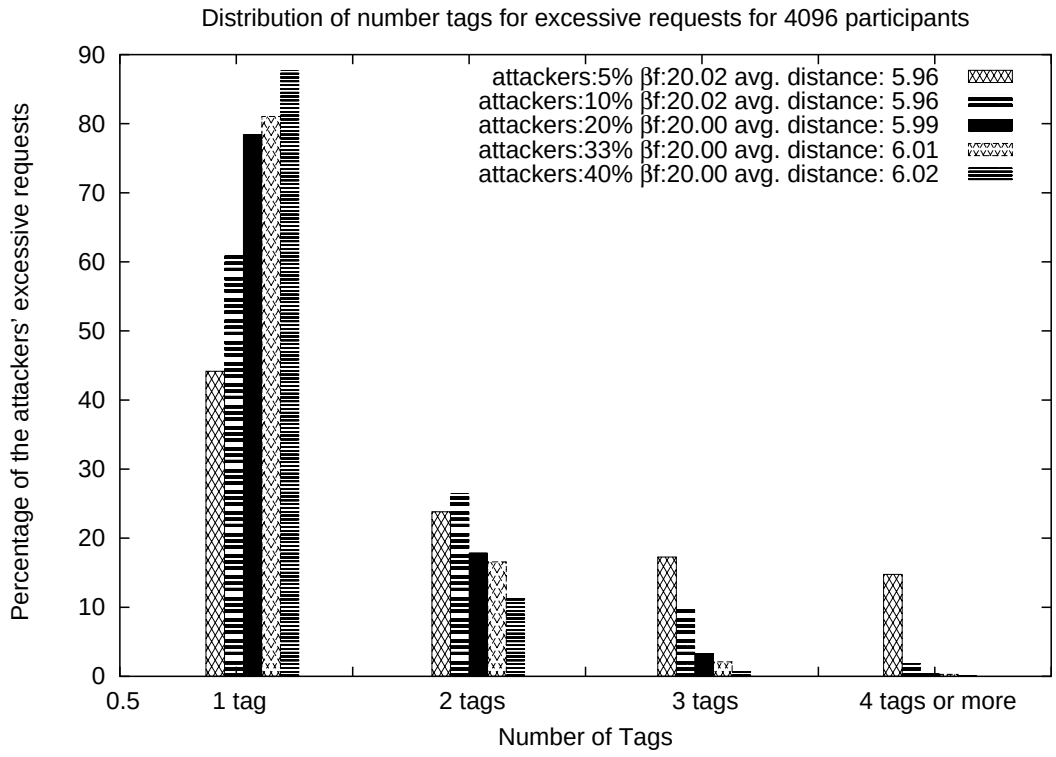

Figure 7: Distribution of the number of tags for the excessive search requests detected when the attack intensity $\beta f = 20$ and we vary the fraction of nodes compromised for a 4096 participants chord ring. The number of tags are inversely proportional to the fraction of nodes compromised.

6 Conclusion and Future Work

In this paper we identified a distributed and scalable method of detecting anomalous traffic flows in DHT P2P networks. This technique can assist in responding to DoS attacks within these types of networks.

Although our findings are preliminary, there is a wealth of challenging problems that need to be addressed. Our investigation opens up an important new area in DHT P2P networks. Future directions include application of our method in CAN/PASTRY, recursive detection of the attacker (push back mechanisms), and ways of isolating individual "misbehaving" flows out of aggregates flows.

References

[1] Gnutella, a file-sharing system, *www.gnutella.com*.

[2] Kazaa media desktop, *www.kazaa.com*.

[3] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)*, October 2001.

[4] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *INFOCOM, San Francisco, CA, April 2003*, 2003.

[5] R. H. K. Chen-Nee Chuah, Lakshminarayanan Subramanian. Dcap: Detecting misbehaving flows via collaborative aggregate policing. In *SIGCOMM Computer Communication Review*, volume 33, October 2003.

[6] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, 2009:46, 2001.

[7] F. Dabek, F. Kaashoek, R. Morris, D. Karger, and I. Stoica. Wide-area cooperative storage with cfs. In *Proceedings of ACM SOSP*, October 2001.

[8] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[9] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *In Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI), October 2000.*, pages 197–212.

[10] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[11] F. Kargl, J. Maier, and M. Weber. Protecting web servers from distributed denial of service attacks. In *World Wide Web*, pages 514–524, 2001.

[12] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of ACM SIGCOMM*, pages 61–72, August 2002.

[13] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[14] Z. Li and P. Mohapatra. QRON: QoS-aware routing in overlay networks. *IEEE Journal on Selected Areas in Communications, Special Issue on Service Overlay Networks*, January 2004.

[15] W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Misra, and D. Rubenstein. Using Graphic Turing Tests to Counter Automated DDoS Attacks Against Web Servers. In *Proceedings of the 10th ACM International Conference on Computer and Communications Security (CCS)*, pages 8–19, October 2003.

[16] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Structured P2P Systems. In *In 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003*, 2003.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM*, August 2001.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[19] A. I. T. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *Symposium on Operating Systems Principles*, pages 188–201, 2001.

[20] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *1st International Peer To Peer Systems Workshop (IPTPS 2002)*, Cambridge, MA, USA, Mar. 2002.

[21] A. Stavrou, D. Rubenstein, and S. Sahu. A lightweight, robust p2p system to handle flash crowds. *IEEE Journal on Selected Areas in Communications, Special Issue on Service Overlay Networks*, January 2004.

[22] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Application. In *Proceedings of ACM SIGCOMM*, August 2001.

[23] D. A. Tran, K. Hua, and T. Do. A peer-to-peer architecture for media streaming. *IEEE Journal on Selected Areas in Communications, Special Issue on Service Overlay Networks*, January 2004.

[24] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A Global-scale Overlay for Rapid Service Deployment. *IEEE Journal on Selected Areas in Communications, Special Issue on Service Overlay Networks*, January 2004.