

Defending P2Ps from Overlay Flooding-based DDoS

Yunhao Liu¹, Xiaomei Liu², Chen Wang², and Li Xiao²

¹Department of Computer Science, Hong Kong University of Science and Technology, Kowloon, Hong Kong

²Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA

Abstract

A flooding-based search mechanism is often used in unstructured P2P systems. Although a flooding-based search mechanism is simple and easy to implement, it is vulnerable to overlay distributed denial-of-service (DDoS) attacks. Most previous security techniques protect networks from network-layer DDoS attacks, but cannot be applied to overlay DDoS attacks. Overlay flooding-based DDoS attacks can be more damaging in that a small number of messages are inherently propagated to consume a large amount of bandwidth and computation resources. We propose a distributed and scalable method, DD-POLICE, to detect malicious nodes in order to defend P2P systems from overlay flooding-based DDoS attacks. We show the effectiveness of DD-POLICE by comprehensive simulation studies. We believe that deploying DD-POLICE will make P2P systems more scalable and robust.

1 Introduction

There are mainly three different architectures for P2P systems: centralized, decentralized structured, and decentralized unstructured [1]. Unstructured P2P systems are most commonly used in today's Internet [2-9]. The most popular search mechanism in unstructured P2P systems is to "flood" a query to the network among peers or among super-peers. A query is broadcast and rebroadcast until certain criteria are satisfied. The simplicity of the flooding based search mechanism makes it vulnerable to overlay DDoS attacks. DDoS attacks have already become a major threat to the stability of the Internet [10-14]. The basic goal of denial of service (DoS) is to overwhelm the processing or link capacity of the target by saturating it with bogus packets. Flooding based overlay DDoS attacks are DoS attacks performed from multiple compromised peers (agents), who generate as many bogus queries as they can toward the victims. One character of P2P overlay DDoS is that the attack target is not a single site in the Internet, but the whole system.

Most previous research on DDoS focused on Network layer attacks. Since a P2P system can have millions of insecure users online simultaneously, and the IP address of a query source peer is not included in the query or query hit messages, network layer defense approaches often find it difficult to effectively protect against overlay DDoS attacks. It is therefore a worthwhile endeavor to design an overlay level defending mechanism inside P2P applications.

In this paper, we propose a detection based approach, DD-POLICE (**D**efending **P**2Ps from **O**verlay **D**istributed-Denial-of-Service), to protect P2P systems against overlay

DDoS. In DD-POLICE, peers monitor the traffic from and to each of their neighbors. If a peer receives a large number of queries from one of its neighbors, it will mark this neighbor as a suspicious DDoS peer. This peer will exchange query volume information with the suspicious peer's neighbors so that it can figure out the number of queries originally issued by the suspicious peer. Based on this number, this peer makes a final decision on whether the suspicious neighbor is a DDoS peer and should be disconnected.

The remainder of this paper proceeds as follows. Section 2 analyzes the characteristics of overlay DDoS. Section 3 presents the design of DD-POLICE. Performance evaluation of DD-POLICE is presented in Section 5. Section 4 reviews the related work. We conclude the work in Section 5.

2 Overlay DDoS in unstructured P2Ps

2.1 Overlay DDoS in P2Ps

Our belief that unstructured P2P systems are vulnerable to overlay DDoS attacks, is based on following observations.

First, the flooding based search mechanism makes overlay DDoS in P2Ps simple in design and operation without requiring any special resources. Malicious nodes may attack the system by walking in and sending out a huge number of useless search queries. The message volume will be quickly propagated and the resources in the P2P system can be exhausted by a small number of DDoS attack machines.

Second, the anonymity design of P2P helps the malicious nodes easily hide behind other peers. The forwarding peers do not know the original sender of each query and the query response is only delivered to the neighbor along the inverse path of the search path. Disconnecting all the peers who send out a large number of queries is dangerous in that a large number of "good" peers could be forwarding queries for "bad" peers. We show a simple example in Figure 1. Instead of flooding the same queries to all its neighbors, a "bad" peer issues different queries to its neighboring peers in order to make DDoS attacks more damaging to the system. We can see in Figure 1 that the query traffic in some connections between "good" peers is much higher than that in the connections between a "bad" peer and a "good" peer.

Finally, the nature of free downloading in P2P systems makes peers easily recruited as DDoS agent (slave) peers. It is challenging to handle overlay DDoS in P2P systems because of the difficulty in separating attack query traffic from legitimate traffic, while it is relatively easy to perform DDoS in P2P without being detected. Note that the victim of P2P overlay DDoS is not a single peer but the P2P system itself.

This work is supported in part by the U.S. National Science Foundation CCF-0514078, CNS-0549006, CNS-0551464, and Hong Kong RGC grants HKUST6152/06E.

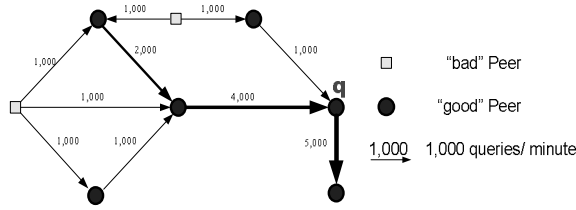


Figure 1. q sends 5000 queries/min, but it is a 'good' peer

2.2 Definition of a "good" peer and a "bad" peer

In our model, the definition of a "good" peer is twofold. First, we assume they have the same processing capacity, and they will forward as many incoming queries as they are capable of. Second, a "good" peer will not issue more than 100 queries per minute. Actually, some observations in [16] show that one peer issues less than 1 query per minute on average. We also have done some experiments in our lab, and no peer ever created more than 40 queries per minute. The assumption that a "good" peer does not issue more than 100 queries per minute makes sense as it is hard for a human user to generate more than one query every second. We further model a "bad" peer's behavior pattern as it will do everything else as a "good" peer except that it generates and issues a large number of queries during every time unit.

We quantitatively define a "good" peer and a "bad" peer as follows. We use $Q_{ih}(t)$ to denote the number of queries sent out (issued plus forwarded) from peer i to peer h during the time period from $(t-1)$ th to t th time unit. We assume that a peer j has k neighbors, m_1, m_2, \dots, m_k .

Definition 2.1 Peer j 's *General Indicator* at time t is defined as: $g(j, t) = \frac{1}{qk} (\sum_{m=m_1}^{m_k} Q_{jm}(t) - (k-1) \sum_{m=m_1}^{m_k} Q_{mj}(t))$, where q is the threshold in distinguishing a "good" peer and a "bad" peer. We set $q=100$ based on the above discussion that a "good" peer does not issue more than 100 queries per minute.

Definition 2.2 Peer j 's *Single Indicator* measured by its neighboring peer i at time t is defined as $s(j, t, i)$, where

$$s(j, t, i) = \frac{1}{q} (Q_{ji}(t) - \sum_{\substack{m=m_1 \\ m \neq i}}^{m_k} Q_{mj}(t))$$

Definition 2.3 For any peer j at any time t , for any peer i other than j , if $g(j, t) > 1$ or $s(j, t, i) > 1$, peer j is a "bad" peer; otherwise, peer j is a "good" peer.

Figure 2 gives an example, in which peer j has three neighbors, i.e. $k=3$. At time t , peer j issues q_0 queries and receives q_1, q_2, q_3 queries from its three neighbors, respectively. From [15] we know that a query message will be dropped if the query message has visited the peer before. Thus, when peer j receives q_2 and q_3 queries from the other two neighbors, it may or may not send all $q_0 + q_2 + q_3$ queries to its neighboring peer i , depending on the number of duplicated query messages from the other two neighbors. For simplicity, here we assume there are no query message duplications in peer j , and all the incoming queries are sent out. This assumption is acceptable because $q_0 + q_2 + q_3$ is an upper bound of the number of queries sent from peer j to i . Both $g(j, t)$ and $s(j, t, i)$ are less than or

equal to q_0/q . If a peer's general indicator or single indicator is much larger than 1, we will have confidence to disconnect it as a "bad" peer. The two indicators at peer i for peer j in Figure 2 are show here.

$$g(j, t) = \frac{1}{100 \times 3} ((q_0 + q_2 + q_3) + (q_0 + q_1 + q_2) + (q_0 + q_1 + q_3) - (3-1)(q_1 + q_2 + q_3)) = q_0/100$$

$$s(j, t, i) = \frac{1}{100} ((q_0 + q_2 + q_3) - (q_2 + q_3)) = q_0/100.$$

In this example $g(j, t) = s(j, t, i) = q_0/100$, where q_0 is the number of queries issued by peer j per minute. Note that the assumption that all of peer j 's incoming queries should be forwarded is not always true. But at least we are sure that $g(j, t)$ and $s(j, t, i)$ could reflect q_0/q .

2.3 Implementation of an overlay DDoS agent

We build a traffic-monitoring node to collect queries flooding through the Gnutella network. Using a modified LimeWire [17] client with logging functionality, all queries passing by the monitoring node are recorded to a log file. The monitoring node, as shown in Figure 3, is configured as a super node connecting to ten peers in the Gnutella network. Our experiment to collect query trace lasted 24 hours. We collected 13, 705, 339 queries with the size of 112 MB.

It is easy to design and implement an overlay DDoS agent. To better study the characteristics of a DDoS agent, a LimeWire client is modified and an experiment is conducted as illustrated in Figure 4. We modified the command line version of LimeWire 2.0.2 by adding a new querying thread to the original source code in peer A. The querying thread reads queries from the log file collected by the monitoring node and issues these queries, and forwards queries coming from the Gnutella network to its neighbors based on the pre-configured time interval. Thus, peer A may work like an overlay query flooding based DDoS agent, a "bad" peer.

Peer B is a "good" peer who never issues any queries and only attempts to forward the queries coming from A. Peer C is a query traffic observer who counts the number of queries forwarded by peer B. Peer C never issues or forwards any queries. The three peers' network bandwidths are 10M. The machines are all PCs of Dell Optiplex GX300 with P3 733 MHZ CPU and 256M memory.

In the experiment, peer A keeps sending queries to peer B at a speed ranging from 1,000 queries per minute to as fast as it could. Indeed, eventually we find Peer A is capable of reading the log file and sending out queries to peer B at a rate of around 29,000 per minute. According to Gnutella protocol, for each received query, peer B will first look up its local sharing storage index, and then forward the query to peer C. Peer C counts the number of queries forwarded by peer B to C so as to measure the capability of a peer in processing search queries and the impact of query flood based DoS on a single Gnutella peer.

We present the experimental results in Figures 5 and 6. Figure 5 shows that when the number of queries sent out from peer A to B is approaching 15,000 per minute, peer B started discarding queries. Figure 6 shows the growth of the query drop

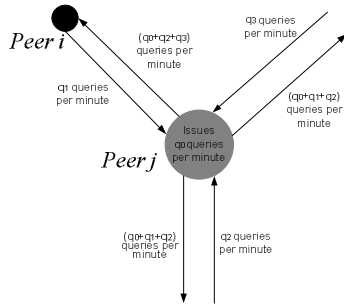


Figure 2. Query traffic analysis

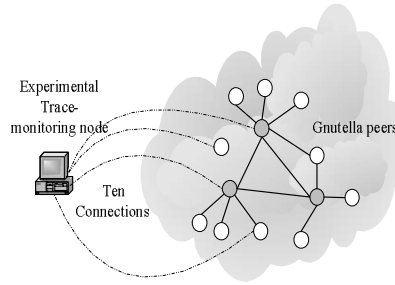


Figure 3. A query trace collection experiment

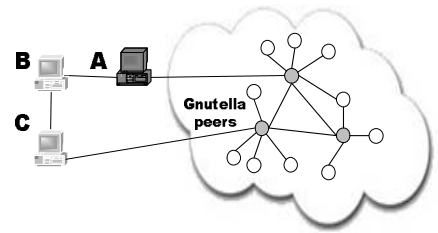


Figure 4. Implementation a DDoS agent

rate in peer B when the query density increases. When peer A sends queries to B as fast as it is capable of, 47% of the queries are dropped by peer B. Note that in our experiment both A and B are dedicated to this experiment, while in a real system a normal peer may have other conventional tasks. Normally a peer's local index includes many contents; while in our experiment the local index is almost empty, which reduces time for local look up. Based on these observations, we assume on average a "bad" peer is capable of sending 20,000 queries per minute, and a "good" peer is capable of processing 10,000 queries per minute in the rest of the paper.

3 DD-POLICE

The basic idea of DD-POLICE is that all peers are involved in policing their direct neighbors' query behavior by cooperating with each neighbor's r -hop away neighbors, and identify the possible "bad" peers for disconnection. For simplicity of the discussion, we first introduce DD-POLICE- r with $r=1$, in which a peer will collaborate with all its direct neighbors to identify "bad" peers. In Section 3.5 we will discuss why it is necessary to employ DD-POLICE- r with $r>1$. Our discussion will focus on DD-POLICE-1. DD-POLICE consists of three steps: *neighbor list exchanging*, *neighbor query traffic monitoring*, and *bad peer recognizing*.

3.1 Neighbor list exchanging

In DD-POLICE, each peer maintains a neighbor list including all its logical neighboring peers. Two neighbors exchange their neighbor lists periodically. One issue here is the

frequency at which peers exchange their neighbor lists. P2P networks are highly dynamic with peers' joining and leaving randomly. The study in [18] has shown that over 20% of the logical connections in a P2P network last one minute or less, and around 60% of the IP addresses keep active in FastTrack for no more than 10 minutes each time after they join the system. The measurement in [19] indicated that the median up-time for a peer in Gnutella and Napster is 60 minutes. If we assume that a peer's average lifetime is 60 minutes and the exchange frequency is once every 2 minutes, the probability we miss one or more neighboring peers on detecting query heavy peers in the third step (*bad node recognizing*) is around 3% ($2/60$). However, simply increase the frequency of exchanging neighbor lists to increase accuracy will cause more overhead to the system. One alternative solution is to make the information exchange event-driven, in which a peer informs all its neighbors whenever its neighboring peer is leaving or a new peer is joining as its neighbor. This solution is favorable to relatively stable networks, but will cause some peers to be super busy during some period of time if the network is highly dynamic. Clearly there is a tradeoff between overhead and accuracy in the frequency selection. Since the P2P network is highly dynamic, DD-POLICE employs a fixed frequency policy, where a peer sends its neighbor lists to all its neighbors periodically.

Another issue is what if a host lies to its neighbor. In our design, when peers exchange their neighbor lists, they will confirm the correctness of the lists with the corresponding peers. A malicious peer could lie about who are its neighbors. If a peer finds out that the claim of a pair of neighboring peers

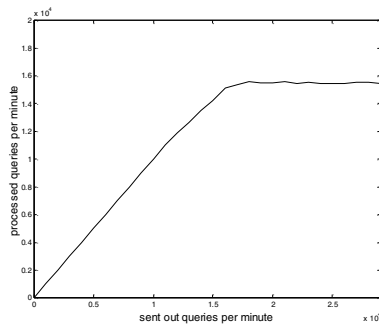


Figure 5. Queries sent out VS processed

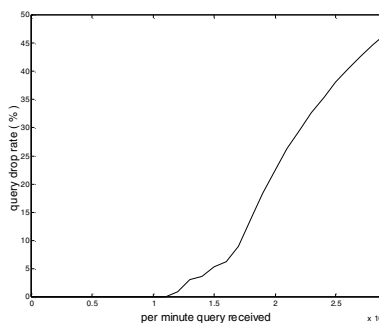


Figure 6. Query drop rate vs. query density

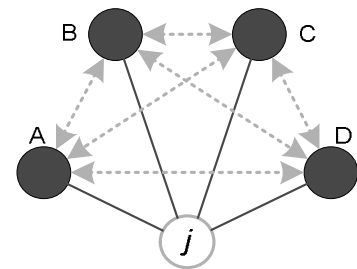


Figure 7. An example of a Buddy Group $BG1-j=\{A,B,C,D\}$

are not consistent, it will disconnect with the one which is its neighbor, and send out a message to both peers indicating the reason of disconnection. In such a way, the good peer in this pair could start to pay more attention to the other peer in this pair. If it gets too many such messages, the good peer will disconnect with the neighbor.

We define peer j 's r hop *Buddy Group* (BGr- j) as the set of peer j 's neighbors. BG1- j is illustrated in Figure 7. Depending on how many logical neighbors each peer has, a peer could belong to multiple different BGs. At the same time, each BG has multiple members. A joining peer creates its BG membership after its first neighbor list exchanging operation. A peer *ping* members within the same BG periodically to make sure that other members are online.

3.2 Neighbor query traffic monitoring

In this step, two lists are designed in a peer for each of its logical neighbors, *Out_query(i)* and *In_query(i)*, to record the number of queries per minute from and to the neighboring i .

3.3 Bad peer recognition

Based on Gnutella 0.6 protocol, we design a new message type called *Neighbor_Traffic*. In addition to the Gnutella's unified 23-byte header for all message types, a *Neighbor_Traffic* message has a message body as shown in Table 1. When a peer identifies one of its neighbors as a suspicious peer because the neighbor sends out a large amount of queries, this peer will start working with other members in the neighbor's Buddy Group by sending *Neighbor_Traffic* messages to them. A *Neighbor_Traffic* message includes five fields: Source IP Address, Suspect IP Address, Source timestamp, # of Outgoing queries, # of Incoming queries. The first three fields contain the source IP address of the current peer, the IP address of the suspicious neighbor, and the time the source sends out the message. The last two fields are the number of queries sent out from the source peer to the suspicious peer, and the number of queries that came from the suspicious peer to the source in the past one minute, i.e. *Out_query(suspicious peer)* and *In_query(suspicious peer)*. The payload type of this message can be defined as 0x83. On receiving a *Neighbor_Traffic* message, a peer in the BG will check whether it has sent a *Neighbor_Traffic* message to other members in this BG in past 5 seconds. If not, it will send such a message to other members.

Let us look at the example in Figure 7. Suppose we define the warning threshold as 500 queries per minute, meaning if peer j sends more than 500 queries to peer A in the past minute, A will mark peer j as a suspicious peer. For example, at time t , peer A marks peer j as a 'suspect' and then sends a *Neighbor_Traffic* message to each of other members in BG1- j (B, C and D). Since some of members in BG1- j (A, B, C and D) may find peer j suspicious at the same time period, and start to send *Neighbor_Traffic* messages to other members, peer A needs to check whether peer B, C or D has sent a *Neighbor_Traffic* message in past 5 seconds. On receiving all the *Neighbor_Traffic* messages from B, C and D, or waiting for another 5 seconds, peer A starts to calculate the General Indicator $g(j, t)$ and the Single Indicator $s(j, t, A)$. If any of the two indicators is greater than a predefined value (this value will be

discussed in Section 5), peer A will affirm peer j as a DDoS compromised peer and disconnect with it.

One issue is that of what criteria can convince peer A that peer j is a DDoS compromised peer with very high probability. One choice is to simply make use of definition 2.3, which implies that if one of $g(j, t)$ and $s(j, t, i)$ is greater than a threshold, s (where $s=1$), peer A can classify the suspicious peer as a "bad" peer and disconnect with it. However, in real systems, we must carefully choose the threshold. The tradeoff is that the damage of the overlay DDoS on the system will be controlled by DD-POLICE for low threshold, but some "good" peers may be miscounted as "bad" peers, while a high threshold will lead to "good" peers spending a long time before disconnecting with "bad" peers.

3.4 An example of DD-POLICE

Let us look at the example shown in Figure 8 to see how DD-POLICE works. In Figure 8, peer j has three neighbors, h , r , and m , who form a Buddy Group, BG1- j . Meanwhile, peer j is also involved in three Buddy Groups, which are BG1- h , BG1- r and BG1- m . Suppose peer j is a DDoS compromised peer, and starts issuing a large amount of queries at time t . When any of peer j 's neighbors, such as h , realizes that peer j has sent out too many queries, it will exchange query volume information (*Neighbor_Traffic* messages) with the members in BG1- j . If peer r and m are "good" peers, they will inform peer h that they did not send a large amount of queries to peer j . Thus h can figure out that a large amount of queries have been issued by peer j , and may disconnect with peer j immediately.

The question is if suspicious peer j reports a correct number of queries that it sent out to others. Since peer j issues a large amount of queries, one or all of its neighbors may forward a large amount of queries out too. Thus, they could also be questioned by their neighbors. For example, peer m could be suspected by members of BG1- m because peer m is forwarding a large amount of queries for peer j . Peer j belongs to BG1- m . There are three choices for peer j in delivering *Neighbor_Traffic* messages to other members in BG1- m : (1) to cheat, (2) not to cheat, or (3) refuse to report. If peer j does not cheat, members in BG1- m will figure out that m is a "good" peer since the majority of outgoing queries from m are the forwarded queries from j instead of initial queries issued by m . We know m will work within BG1- j and disconnect from peer j . There are two cases if peer j chooses to cheat.

Case 1, peer j reports a larger number than the number of queries it really sent to peer m . In this case, members in BG1- m will have higher intention to treat peer m as a "good" peer. Peer m will disconnect from peer j without any confusion, and so do the other two members in BG1- j . Thus, this is not a meaningful cheating for peer j .

Case 2, peer j reports a smaller number than that the number of queries it really sent to peer m . For example, peer j sent 5,000 queries to peer m in the past minute, but it reports in BG1- m that it has only sent to m 100 queries. As a result, peer m could be treated as a "bad" peer and be disconnected by the other neighbors in BG1- m . However, this choice has no benefit to peer j either. On the one hand, making peer m be wrongly disconnected as a "bad" peer will lead to peer j 's attack queries

Table 1. Neighbor Traffic message body

Source IP Address	Suspect IP Address	Source timestamp	# of Outgoing queries	# of Incoming queries
0	3	4	6	7
9	10	11		

Byte offset

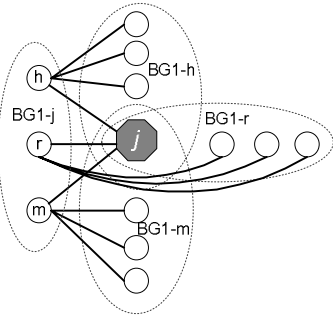


Figure 8. An example of DD-POLICE

being blocked, while peer j 's goal is to attack the P2P system instead of a single peer m . If all of peer j 's neighbors (h , r and m) are disconnected by their neighbors, queries issued by peer j will be isolated among these three peers only, which eliminate the impact of peer j 's attack and is not what peer j wants to achieve. On the other hand, in DD-POLICE, this cheating does not mislead peer m 's decision making. Peer m works with members in BG1- j , and is able to identify peer j as the "bad" peer without being confused by peer j 's cheating.

The third choice for peer j is not to report the number of queries it sent to m , which is the same situation as in Case 2 since, in the design of DD-POLICE, if a peer has not received a Neighbor_Traffic message from peer j within a predefined time period, it just assumes that peer j sent 0 query to peer m .

In other words, cheating or not reporting will do nothing good for peer j , and could only degrade the effects of its attacks. Therefore, we assume that peer j will not cheat in delivering the Neighbor_Traffic messages.

3.5 Experimental methodology

Using BRITE, we generate 10 logical topologies with 20,000 peers. Most peers have 3 or 4 logical neighbors, and a few peers have tens of direct neighbors. The average number of neighbors of each node is 6. Some basic settings of the workloads used in this simulation follow a 200-day trace of KaZaA P2P traffic collected at University of Washington [20]. In our first simulation, 1,000,000 search operations are simulated.

We model a malicious node such that it generates as many queries as it is capable of [21]. Measured by our implemented DDoS agent prototype described in Section 2.3, a "bad" peer could generate more than 20,000 distinct queries per minute. We assign bandwidth to each link based on the observations in [19], which show that 78% of the participating peers have downstream bottleneck bandwidths of at least 100Kbps, and 22% of the participating peers have upstream bottleneck bandwidths of 100Kbps or less. In our simulation, the number of attack queries sent by a compromised peer per minute, Q_a , is given by: $Q_a = \min\{20,000, \text{the capacity of the link}\}$

Studies in [22] show that the peer population is quite transient by arguing that measurement according to host IP addresses underestimates peer-to-peer host availability and observing that each host joins and leaves a P2P system 6.4 times a

day on average, and that over 20% of hosts arrive and depart every day. We simulate the joining and leaving behavior of peers via turning on/off logical peers. In our simulation, every node issues 0.3 queries per minute, which is calculated from the observation data shown in [16], i.e., 12,805 unique IP addresses issued 1,146,782 queries in 5 hours. When a peer joins, a lifetime in seconds will be assigned to the peer. The lifetime of a peer is defined as the time period the peer will stay in the system. The lifetime is generated according to the distribution observed in [19]. The mean of the distribution is chosen to be 10 minutes [18]. The value of the variance is chosen to be half of the value of the mean. The lifetime will be decreased by one after passing each second. A peer will leave in the next second after its lifetime reaches zero.

3.6 Consequences of overlay DDoS attack in P2Ps

We generate 1,000,000 queries and track the delivery and response of each query message under a flooding based search mechanism. In each of the simulations, k random peers, where k is ranging from 1 to 200, are selected as DDoS compromised peers and each of them keeps sending out attack queries at the maximum rate they are capable of. Meanwhile, normal peers issue queries too. We study the impact of the overlay DDoS attack on a dynamic P2P environment.

Figure 9 shows the effect on network traffic. Here traffic cost is a function of consumed network bandwidth and other related expenses. We see that ten to twenty (<0.1 %) compromised peers will double the total traffic. When there exists around 100 compromised peers, representing only 0.5% of the network, the total search traffic increases to 10 times that of the original traffic. Observations in [18, 23] have shown that P2P traffic contributes the largest portion of Internet traffic based on their measurements on some popular P2P systems, such as FastTrack (including KaZaA and Grokster) [24], Gnutella [14], and DirectConnect. Measurements in [25] have shown that even 95% of any two nodes are less than 7 hops away, the flooding-based routing algorithm generates 330 TB/month in a Gnutella network with only 50,000 nodes. Figure 10 and 11 show that the service quality of the system is also degraded by DDoS attacks in terms of response time and normalized success rate. *Response time* is defined as the time period from when the query is issued until when the source peer received a response result from the first responder.

Query success rate measures the ratio of the queries for which at least one location of the desired data is found. If we use $q_w(t)$ to denote the total number of queries issued by all the peers during the period from $t-1^{th}$ to t^{th} time unit, and use $q_s(t)$ to denote the total number of queries for which one or more locations of the desired data are found, the query success rate at any given time t , $S(t)$, is given by:

$$S(t) = \frac{q_s(t)}{q_w(t)} \times 100\%.$$

We can see that 100 comprised peers will increase the average response time of queries by 2.4 times, and up to 89.7% of queries could fail to receive query responses. These results in-

dicating that, in a real-world P2P system that usually has about 2 million peers online at any time, less than one thousand DDoS compromised peers could stress the system greatly while ten thousand DDoS agents could overwhelm the whole system.

3.7 Effectiveness of DD-POLICE

This section presents performance analysis on DD-POLICE.

3.7.1. Frequency of neighbor list exchanging We have discussed that the frequency of neighbor list exchanging is one of the key issues to be examined. We have simulated and evaluated two policies: (1) periodic policy: to exchange neighbor lists every s minutes, where s ranges from 1 to 10; and (2) event driven policy: a peer reports its neighbor list whenever a new neighbor is joining or an existing neighbor is leaving. As we observed from our simulation results, there is no big difference on the overall performance by using the first policy as long as s is no more than 2 minutes, while the second policy incurs much higher traffic overhead compared with the first policy because the P2P network is very dynamic. But if we further increase s , such as to 4 or 5 minutes, the probability of DD-POLICE misjudging “good” peer and “bad” peers will be increased because of the inaccurate neighbor lists. Therefore, in other simulations and the DD-POLICE implementation prototype, we use periodic policy in which peers report their neighbor lists every 2 minutes.

3.7.2. Cut threshold Another issue to examine is when a peer should make the decision to disconnect a suspicious neighbor as a “bad” peer. The decision is based on two calculated indicators $g(j, t)$ and $s(j, t, i)$. In DD-POLICE, we define CT as a cut threshold. If the value of $g(j, t)$ or $s(j, t, i)$ computed by peer i is greater than CT, peer i will disconnect from peer j . CT is 1 in definition 2.3. However, we have mentioned that $g(j, t)$ and $s(j, t, i)$ are based on some assumptions which may affect the accuracy of the estimation, and the dynamic changing of the overlay topology makes the choice of CT a challenging endeavor. The dilemma is that a smaller value of CT may mislead the peers to wrongly disconnect “good” peers, while a larger value of CT could allow some “bad” peers to avoid disconnection. In order to select an optimal value of CT, we study the impact of different values of CT using three kinds of errors: false negative is the number of “good” peers that are wrongly disconnected, false positive is the number of “bad” peers that are not identified and not disconnected, and false judgment is the sum of the above two.

Another consideration in the choice of CT is the convergence speed of the algorithm. Decentralized P2Ps are fully distributed systems. When some peers recognize a “bad” peer, the only thing these peers can do is to disconnect with the “bad” peer. No mechanism can prevent the DDoS Agent from joining the system again and launching another round of attacks. We desire our approach to identify “bad” peers in a very short period of time in a dynamic P2P environment. The metric of damage recovery time is used to evaluate the performance of DD-POLICE for this consideration. We strive for short damage recovery time.

Figures 12-14 show the impact of CT on the performance of DD-POLICE when there are 100 DDoS agents in a 20,000-peer system. DD-POLICE- n means DD-POLICE scheme with $CT=n$. Figure 12 shows that DD-POLICE with $CT=3$ reduces the damage of attack faster than DD-POLICE with $CT=7$. *Damage rate*, $D(t)$, is given by:

$$D(t) = \frac{S(t) - S'(t)}{S(t)} \times 100\%$$

where $S(t)$ denotes query success rate of the P2P system when there does not exist any DDoS compromised peers, and $S'(t)$ denotes the query success rate when the system is under DDoS attack.

The damage rate of $CT=3$ scheme cannot be reduced as low as that of $CT=7$ scheme. The reason is that, compared with $CT=7$ scheme, more “good” peers are misjudged as “bad” peers and disconnected by their neighbors in $CT=3$ scheme, causing a lower success rate. However, CT cannot be too large. Figure 12 shows that with $CT=10$, DD-POLICE converges very slowly and the stabilized damage rate is much higher than that in $CT=3$ and $CT=7$.

We further show the three kinds of errors in Figure 13 and damage recovery time in Figure 14. *Damage recovery time* is defined as the time period from when the system damage rate $D(t)$ is equal or greater than 20% until when the damage is equal or less than 15%. If the damage recovery time is short, this means the DD-POLICE is effective.

We can see that in Figure 13, as CT increases, the false negative decreases, while the false positive increases. That means when we set a larger cut threshold, fewer “good” peers will be misjudged as “bad” peers and more “bad” peers will be misjudged as “good” peers. The false judgment is optimal when CT is within 5 to 7 in Figure 13. Figure 14 shows that as CT increases, DD-POLICE needs a longer time to identify a peer as a “bad” one. Thus, the damage recovery time is longer. Comprehensively considering the performance of DD-POLICE, we choose $CT = 5$ or 6.

3.7.3. DD-POLICE in dynamic P2P environments Adding DD-POLICE into the peers in the dynamic simulation environment, we repeat the experiment described in Section 5.1 with the same 1,000,000 queries. Basic setup of DD-POLICE includes exchanging neighbor lists every 2 minutes and $CT=5$. Figures 9-11 show that DD-POLICE effectively reduces the damage of overlay DDoS and is very scalable. Compared with the case of “no DDoS attack”, DD-POLICE achieves a comparable average response time and success rate with slightly higher average traffic cost.

4 Related work

Many general efforts have been made to defend against DDoS [10, 26], which are roughly divided into three categories: prevention, traceback and identification, and detection and filtering.

The first type is prevention, in which the defense tools monitor or scan the network to intercept the DDoS before the attacks start. They avoid the malicious peers’ illegal access to normal machines [27], and install security patches and virus

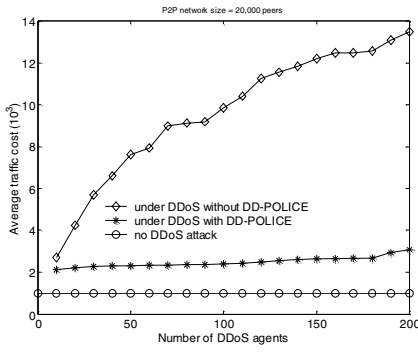


Figure 9. Average traffic cost

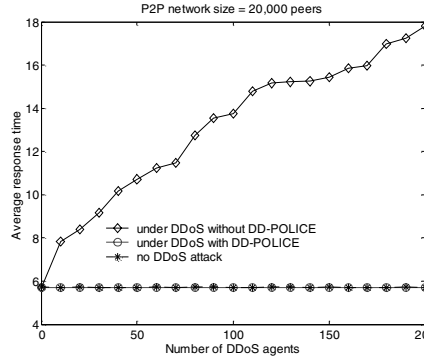


Figure 10. Query response time

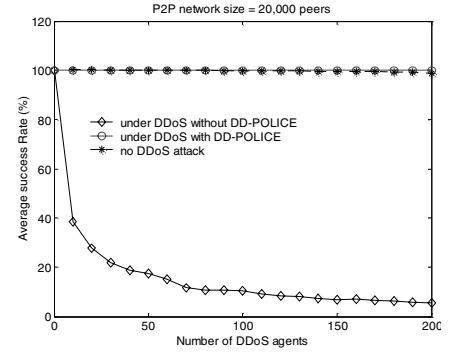


Figure 11. Success rate

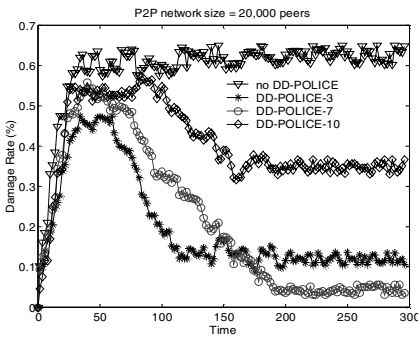


Figure 12. Effectiveness of DD-POLICE in Dynamic P2P environments

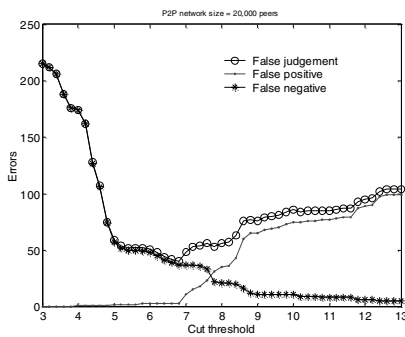


Figure 13. Errors vs. cut threshold

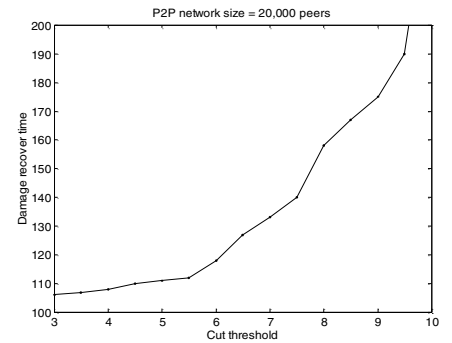


Figure 14. Damage recovery time vs. cut threshold

scanners[28, 29]. One basic goal of these approaches is to prevent DDoS attackers from recruiting a large number of agents. Most of these methods are based on knowledge of existing DDoS attacks to recognize attackers' behaviors. Although it is of great importance to improve Internet security, it is hard to believe that preventive approaches could successfully avoid DDoS attackers recruiting hundreds of agents in a P2P network with millions of peers online at any given time. We have shown that only tens of DDoS agents in a 20,000-peer system cause serious damage.

The second type is the TraceBack and Identification approach [30-34], which is usually used after experiencing attacks. Most of them are based on IP traceback. They try to track the attackers and identify them via the routers' records or by sending special traceback packets. However, these approaches are not effective for P2P overlay DDoS attacks because the query and query hit messages do not include the IP addresses of query source peers. In P2P systems, the anonymity requirement makes it hard to know who originally issued the queries.

The third type is based on Detection and Filtering. Our proposed DD-POLICE is of this type. They detect the occurrence of DDoS attacks and respond to it. For example, although IP spoofing is not a necessary component for DDoS attacks, it helps the attackers hide [35]. Ingress/egress filtering [36] prevents packets with spoofed source IP addresses from entering or leaving the network. Theoretically, ubiquitous ingress packet filtering (UIPF) [10] can stop all address-spoofed direct attack packets as well as the attack packets sent to reflectors. But it is hard for them to be employed in P2P systems with such a large scale. Route-Based defense is similar to UIPF, and employs some distributed detection systems and filters attack traffic at

some key nodes. Pushback mechanism [37] is a useful means of detecting the attack packets' flow and dropping them. In a P2P system, it is hard for a peer to trace the source of a query because the values of TTL and hops could be easily modified by DDoS compromised peers. Thus, it is hard to separate good traffic and bad traffic. A source-end defense system, such as D-WARD [38] or Reverse Firewall, attempts to observe incoming and outgoing flows and connections over time, aiming at separating the normal links from the attack links to provide good service to normal clients. However, in a P2P system, since normal traffic and attack traffic may come from the same logical link because of the flooding search, the source-end system is not effective in P2P systems.

To our knowledge, the most related work to this research to date is discussed in [21], where application-layer load balancing techniques are proposed to give clients a fair share of available resources, so as to alleviate damage of application-layer DoS attacks. It is basically a survival approach: it does not require servers to distinguish attack queries from normal queries, but maintain a fair load distribution in the P2P system. However, this approach could be less effective when the number of DDoS agents is getting large.

5 Conclusion and future work

Unstructured P2P systems are vulnerable to overlay DDoS attacks. Most previous techniques protect networks from network-layer DDoS attacks and cannot be applied to overlay DDoS attacks. Overlay flooding-based DDoS attacks can be more damaging in that a small number of messages are inher-

ently propagated to consume a large amount of bandwidth and computation resources.

In this paper, we show the serious consequences of query flooding based overlay DDoS attacks on P2P systems, and propose a distributed and scalable method, DD-POLICE, to detect malicious nodes and defend P2P systems from overlay DDoS attacks. We discuss the design of DD-POLICE and show the effectiveness of DD-POLICE by comprehensive simulation studies. Results show that DD-POLICE can help peers disconnect with DDoS agents in a very short time period after attacks are launched.

Other future work includes employing and measuring DD-POLICE in larger scale systems, such as PlanetLab[39], and studying overlay DDoS in structured P2P systems [40]. Overall, we believe that deploying DD-POLICE will make unstructured peer-to-peer systems more scalable and robust.

6 References

- [1] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-peer Networks," in Proceedings of the 16th ACM International Conference on Supercomputing, 2002.
- [2] L. Xiao, Y. Liu, and L. M. Ni, "Improving Unstructured Peer-to-Peer Systems by Adaptive Connection Establishment," *IEEE Transactions on Computers*, 2005.
- [3] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei, "Quantifying Skype User Satisfaction," in Proceedings of SIGCOMM, 2006.
- [4] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid Search Schemes for Unstructured Peer-to-Peer Networks," in Proceedings of IEEE INFOCOM, 2005.
- [5] H. Jiang and S. Jin, "Exploiting Dynamic Querying like Flooding Techniques in Unstructured Peer-to-Peer Networks," in Proceedings of ICNP, 2005.
- [6] B. Liu, W.-C. Lee, and D. L. Lee, "Supporting Complex Multi-dimensional Queries in P2P Systems," in Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS), 2005.
- [7] D. Qiu and R. Srikant, "Modeling and Performance Analysis of Bit Torrent-Like Peer-to-Peer Networks," in Proceedings of SIGCOMM, 2004.
- [8] R. Zhang and Y. C. Hu, "Assisted Peer-to-Peer Search with Partial Indexing," in Proceedings of IEEE INFOCOM, 2005.
- [9] L. Zhou, L. Zhang, F. McSherry, N. Immerlica, M. Costa, and S. Chien, "A First Look at Peer-to-Peer Worms: Threats and Defenses," in Proceedings of the 4th International Workshop on Peer-To-Peer Systems (IPTPS), 2005.
- [10] R. K. C. Chang, "Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial," *IEEE Communications Magazine*, pages: 42-51, October, 2002.
- [11] A. Stavrou and A. D. Keromytis, "Countering DoS attacks with stateless multipath overlays," in Proceedings of ACM Conference on Computer and Communications Security (CCS), 2005.
- [12] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS Defense by Offense," in Proceedings of SIGCOMM, 2006.
- [13] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten, "New client puzzle outsourcing techniques for DoS resistance," in Proceedings of ACM Conference on Computer and Communications Security (CCS), 2004.
- [14] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting network architecture," in Proceedings of SIGCOMM, 2005.
- [15] "The Gnutella protocol specification 0.6," 2002.
- [16] K. Sripanidkulchai, "The Popularity of Gnutella Queries and Its Implications on Scalability," 2001.
- [17] Limewire, <http://www.limewire.com/>
- [18] S. Sen and J. Wang, "Analyzing Peer-to-peer Traffic Across Large Networks," in Proceedings of ACM SIGCOMM Internet Measurement Workshop, 2002.
- [19] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in Proceedings of Multimedia Computing and Networking (MMCN), 2002.
- [20] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," in Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP), 2003.
- [21] N. Daswani and H. Garcia-Molina, "Query-Flood Dos Attacks in Gnutella," *ACM Computer and Communications Security*, 181-192, 2002.
- [22] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding Availability," in Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), 2003.
- [23] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An Analysis of Internet Content Delivery Systems," in Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 2002.
- [24] Fasttrack, <http://developer.berlios.de/projects/gift-fasttrack/>
- [25] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the Gnutella Network," *IEEE Internet Computing*, 2002.
- [26] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity," in Proceedings of Usenix Security Symposium, 2001.
- [27] Tripwire, <http://sourceforge.net/projects/tripwire/>
- [28] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," in Proceedings of Networks and distributed system security symposium (NDSS), 1999.
- [29] J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms," *UCLA CSD Technical Report no. 020018*,
- [30] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical Network Support for IP Traceback," in Proceedings of ACM SIGCOMM, 2000.
- [31] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network Support for IP Traceback," *ACM/IEEE Transactions on Networking*, 9(3), June, 2001.
- [32] D. X. Song and A. Perrig, "Advanced and Authenticated Marking Schemes for IP Traceback," in Proceedings of IEEE INFOCOM, 2001.
- [33] A. Yaar, A. Perrig, and D. Song, "Pi: A Path Identification Mechanism to Defend against DDoS Attacks," in Proceedings of IEEE Symposium on Security and Privacy, 2003.
- [34] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer., "Single-Packet IP Traceback," *IEEE/ACM Transactions on Networking (ToN)*, 2002.
- [35] J. Mirkovic, G. Prier, and P. Reiher, "Challenges of Source-End DDoS Defense," in Proceedings of NCA, 2003.
- [36] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing," *RFC 2827*, 2000.
- [37] J. Ioannidis and S. M. Bellovin, "Implementing Pushback: Router-Based Defense Against DDoS Attacks," in Proceedings of NDSS, 2002.
- [38] J. Mirkovic, G. Prier, and P. Reiher, "Attacking DDoS at the Source," in Proceedings of ICNP, 2002.
- [39] L. Peterson, D. Culler, T. Anderson, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," in Proceedings of HOTNETS, 2002.
- [40] N. Naoumov, and K.W. Ross, "Exploiting P2P Systems for DDoS Attacks," *International Workshop on Peer-to-Peer Information Management (keynote address)*, Hong Kong, May 2006.