Making Eclipse Attacks Computationally Infeasible in Large-Scale DHTs

Ren Zhang^{1,2}, Jianyu Zhang^{1,2}, Yu Chen^{1,2}*, Nanhao Qin^{1,2}, Bingshuang Liu^{1,2} and Yuan Zhang^{1,2}
^{1}Beijing Key Laboratory of Internet Security Technology, Peking University, Beijing 100871, China
^{2}Institute of Computer Science and Technology, Peking University, Beijing 100871, China
{zhangren, zhangjianyu, chenyu, qinnanhao, liubingshuang, zhangyuan}@icst.pku.edu.cn

Abstract

The security aspect of Distributed Hash Tables (DHTs), the principal model for structured P2P networks, has received considerable attention from research community, and the eclipse attack is one of the most severe threats targeting DHTs. Most of currently effective defense mechanisms suffer from significant communication cost. In this paper we present a novel approach to address eclipse attacks—making such attacks computationally infeasible. The backbone of our approach is a scheme for generating node IDs, which requires a user to solve a computational puzzle generated by her network parameters together with time-related information, in order for him to obtain a valid ID. Such procedure normally should be completed within a couple seconds of CPU time, and an ID can be easily verified for its validity. However, carrying out an eclipse attack on a specific key demands massive computing resources. We have evaluated our method by analyzing the cost of an attacker, using real-world data from BitTorrent, and the result is that it takes thousands of processors running day and night to find sufficient number of IDs. We also have simulated the computing cost of both benign users and attackers, and the outcome also supports the above claim. Unlike most existing defense mechanisms, for our method the induced communication cost and churn is negligible, and no centralized service is required.

1 Introduction

As the principal model for structured P2P networks, DHTs provide certain attractive features such as no centralized server, fast searching, and low network overhead. However, their performance is still not satisfying in many aspects: load-balancing, the availability of data, and so on. More severely, security attacks could even paralyze the entire DHTs. The most studied attacks are: (1) sybil attack—attackers generate enormous amount of bogus nodes that could paralyze the whole P2P network; (2) eclipse attack—attackers try to corrupt the routing tables of honest nodes by filling them with malicious nodes; (3) routing/storage attack—malicious nodes do not follow the routing/storage protocols correctly [24].

This paper concentrates on the eclipse attack. What makes eclipse attacks, and also sybil attacks, so easy to initiate is the loosely controlled, or even open membership policy on overlay networks. Because of such policies, an attacker can create malicious nodes (sybil attack) and isolate IDs from normal nodes (eclipse attack) effortlessly, and then manipulate lookup requests by forwarding requests to malicious nodes or returning bogus ones. Both attacks focus on the DHT protocol itself rather than the concrete client programs. Clearly sybil attacks can be used for inducing eclipse attacks, however, the presence of an effective defense against sybil attacks is not sufficient for preventing eclipse attacks. For example, in DHTs a small number of malicious nodes with legitimate identities is sufficient for initiating an Eclipse attack [19]. Current defenses against eclipse attacks focus on two approaches:

Putting constraints on neighbor set selection: Hildrum and Kubiatowicz recommended that each node selects as its neighbors the nodes with minimal network delay among all the nodes that satisfy the structural constraints for a given neighbor set member [14]. However, a large number of nodes might appear within a narrow band of delay [12], and the effectiveness of such PNS-based (PNS stands for proximity neighbor selection) defense diminishes with increasing overlay size [19]. Singh et al. presented another defense scheme in which nodes anonymously audit each other's connectivity and enforce a limit on their in-degree [19]. Admittedly, when carrying out an eclipse attack, an attacker node should try to achieve a high in-degree so as to save computing resource, however, when the attacker's only purpose is to isolate a specific ID, high in-degree becomes unnecessary and medium amount of IP addresses

*Corresponding author

plus modest computing resources are just enough. Moreover, such defense scheme induces additional communication costs, which might lower the speed of legal requests when no attack is occurring.

Circumscribing ID selection: Eclipse attacks are almost inevitable as long as a node can arbitrarily choose an ID, thus many defense mechanisms focus on circumscribing ID generation [3, 5, 7, 8, 10, 21, 23] and/or frequently redistributing IDs [4, 7, 10]. However, these mechanisms suffer from some of the following drawbacks: (1) Many of them require a certification authority [3, 7, 10, 21, 23]. The feasibility of a centralized trusted authority relies on two assumptions: all participating nodes trust on its security and incorruptibility, and the authority is able to accurately detect the attackers. These assumptions are difficult to be realized on large-scale non-commercial DHTs; (2) Frequently redistributing IDs would induce churn (participating nodes dynamically joining and leaving the underlying P2P network), which could affect the availability of the DHT protocol, and incur extra communication cost; (3) For methods neither require a certification authority nor redistribute IDs frequently, they might be vulnerable against a resourceful attacker [5, 8].

In this work, we present a novel method to make eclipse attacks in large-scale DHTs computationally infeasible by circumscribing ID generation. Observing that a benign user can perform her searching and publishing requests at an arbitrary ID, whereas an attacker has to get a number of (typically 8) IDs closer to the target than any other benign nodes, so as to initiate an attack, we design a procedure for generating an ID and verifying an ID for its validity. In our scheme, a valid node ID is generated by solving a computational puzzle constructed by the user's network parameters together with time-related public information, which cannot be pre-computed or manipulated. Furthermore, an ID has only two weeks of period of validity, and its validity can be easily verified by a few hash operations. For the proposed scheme, generating a valid ID normally takes several seconds of CPU time at most, depending on the difficulty of the puzzle. Such time length is acceptable for a legitimate user. However, in order to initiate an eclipse attack, an attacker must utilize sufficient IP resources (hundreds of different IP addresses) for computing numerous IDs, so as to get desired ones. In BTDHT (the DHT network of BitTorrent [9]) our simulations indicate that even if the attacker can generate 10^9 IDs (45 times of current BTDHT size) in a week (it takes 4000 processors), he only has a 75% chance of getting enough desired IDs. In this paper we might use settings and parameters of BTDHT to illustrate our ideas, but our method can be applied to all DHTs.

We enumerate major advantages of our defense scheme as follows:

- It requires no centralized servers, yet still significantly increases the overall cost of a successful attack.
- It can be easily implemented in current client programs, and unlike most existing defense schemes, the induced communication cost and churn are negligible.
- Periodically renewing IDs using some public information makes pre-computation of valid IDs impossible, therefore an attacker has to generate desired IDs within a period of validity.
- Using network parameters to generate computational puzzles makes ID reusing impossible, and a valid ID can be immediately authenticated without any challenge-response message or time synchronization.

The rest of the paper is structured as follows. In Section II we provide some background on DHTs and eclipse attacks and discuss existing defenses. Section III describes our proposed defense. In Section IV, we validate our method and analyze its disadvantages using real-world data from BitTorrent and compare our method with current defenses. Section V simulates the cost for a benign user and an attacker. Section VI concludes the paper.

2 Background

2.1 Overview of DHTs

The core idea of DHTs is a decentralized lookup service denoted as $lookup(k)$, which normally returns the network addresses of the nodes who keep a record of nodes storing files associate with key k. Many popular applications of DHTs focus on file-sharing: A user (called publisher) who possesses certain values v of a file identified by key k (typically IP addresses of nodes storing the file) executes $lookup(k)$ and then stores (k, v) on the returned nodes. Afterwards, those who search for the file (searchers) can retrieve (k, v) pairs by executing $lookup(k)$ and then sending requests to the nodes in v. Such DHT architecture has an elegant feature: Although each node is just required to know the identities of only a small subset of the entire network, the lookup operation can still be executed with good enough accuracy and within acceptable time duration.

The principle structure of a DHT is a shared identifier space for both nodes and keys, with data associated with a key k stored in several nodes closest to k according to some distance function. Take Kademlia, the most popular DHT protocol, as an example: an ID is a 160-bit vector generated using the SHA-1 hash function on a random value, and the distance between two nodes is determined by XOR operation on their identifiers [16].

To locate the nodes corresponding to k, a node forwards the lookup request to another peer whose identifier is closer to k than itself, or just replies the information on "closer peers" to the requester. A lookup request is finished when no node is found with an identifier closer to the key k, or the time limit has been exceeded. The lookup time is logarithmic of the size of entire ID space. Every node maintains a *routing table*—links to a set of nodes. The size of a routing table must be small relative to the total number of participating nodes, for the sake of scalability, and it is typically of $O(m)$, where m is the length of an identifier. Take Kademlia as an example. Each node in it keeps a *k-bucket* of links (typically 10) for nodes within a distance between 2^i and 2^{i+1}, $0 \leq i < m$, from itself. Links in a bucket are selected from active nodes with preference to nodes with longer history.

2.2 Overview of Eclipse Attacks

The first piece of work on such attacks in the context of DHTs appeared in [20]. It is a form of routing poisoning, which aims to separate a set of victim nodes from the rest of the overlay network. A modest number of malicious nodes conspire to fool correct nodes into adopting the malicious nodes as their neighbors. In short, the eclipse attack is performed by an attacker that tries to intercept all the requests directed to a specific resource [15]. It is performed by initializing a limited set of nodes with identifiers numerically closer to the target ID, abbreviated as ID_e, than any real node, and then announcing these sybils to the regular peers, in order to "poison" their routing tables and to attract all lookup requests for ID_e. In practice, once receiving a lookup request to ID_e, the attacker might answer with a fake content or simply ignore it. In the extreme, an eclipse attack allows the attacker to control all overlay traffic, enabling arbitrary denial of service or censorship attacks.

There are two necessary conditions to perform an eclipse attack: (1) The attacker nodes can choose a set of desired IDs for themselves so that the attacker can control the ones closest to the target. In order to completely isolate a target ID, the node density around ID_e ought to be much higher than a randomly distributed fraction. (2) The attacker nodes must be added to others' routing tables. Unfortunately, current DHT implementations contain no authentication protocols. In Kademlia a node will insert new nodes in its buckets when old ones leave the system or the bucket is not full, nevertheless, the buckets close to the node's own ID are almost never full.

2.3 Related Work

2.3.1 Circumscribing ID selection

In this part we briefly overview some existing defenses with focus on ID redistribution or circumscribing ID generation. Since they target on the two necessary conditions of the eclipse attack, we believe they are more effective than the approaches of detecting or avoiding malicious neighbors via node behavior or network measurement.

Castro et al. imposed strong structural constraints on the neighbor sets and used redundant routing to guarantee the routing performance [7]. However this method relies on a trusted authority to distribute signed IDs. Condie et al. extended their work and suggested inducing churn periodically so as to randomly redistribute node IDs in the network [10]. While their calculation on communication overhead only includes updating routing tables, the $(key, value)$ data transmission cost in a regular churn scheme is not negligible. Furthermore, their solution still needs a certification authority.

Awerbuch and Scheideler [4] brought forward a secure DHT scheme that implemented routing process based on the concept of region. The ID of a new node is generated by a group of participating nodes using a verifiable secret sharing scheme. They introduced a new rule called cukoo rule that all nodes in the region of the new identifier of the joining node must leave the system and rejoin with new random identifiers. This rule makes the attacker difficult to locate near the victim node. However, the paper didn't carry out any experiment, so the practical effect of their method has yet to be seen.

In [8] and [5] it was suggested that IDs could be chosen as a hash result of network parameters, however, [15] indicated that such restriction is weak to a resourceful attacker: If an attacker can obtain enough IP resources, he can get the desired IDs with an adequate number of attempts. Nowadays a million-node rainbow table of network parameters and IDs can be built within a few minutes.

Aiello et al. [3] adopted a centralized certification service to generate legitimate IDs which cannot be predicted, they also introduced a handshake process to prevent man-in-the-middle attack. Centralized service (also suggested in [23] and [21]) is considered applicable at the first years of P2P, where the network scales are relatively small. However in largert-scale DHTs, it suffers from single-point failure, and even if the service is secure, as long as an attacker continuously request for more IDs, he can get the desired ones eventually (although it might take a longer time).

2.3.2 Computational Puzzles as Sybil Defenses

Ideally, an ID should be computed in a user's own computer, so that we do not have to worry about single-point failure of a certification server or collusion attacks, and no communication cost or churn should be induced for preventing a likely-to-happen attack. The defenses in [5, 8] fit in these requirements, except that they are vulnerable to resourceful attackers. In order to raise the cost of an attacker, the idea of computational puzzle is considered. In this part we discuss some usages of computational puzzles in the content of defending sybil attacks.

A pioneer work of using computational puzzles in D-HTs is [7]. The paper brought forward the key challenges of this approach: the cost of solving a crypto puzzle must be acceptable to the slowest legitimate node, yet the puzzle must be hard enough so as to sufficiently slow down an attacker with access to many fast machines. It also proposed two simple examples of using computational puzzles. However, the first example does not mention how to avoid other nodes reusing a node's work without challenge-response messages or time synchronization in the authentication procedure, whereas in the second example, the additional cost for performing an attack is very little. We believe our method perfectly solves the above challenges by incorporating the scale of the network into the cost of an attacker, and in the same time the increased cost of a normal user is very little, which can be seen from later analysis (Section IV).

Halderman and Waters [13] recommend that puzzles be derived from multiple online sources rather than from an individual source. However a valid ID in their scheme can be easily reused by an attacker. As a result, malicious nodes could just stay online listening to valid IDs and then use them.

The idea of computational puzzles is also considered by [18] and [6] in their defenses against sybil attacks, but a node can still choose its own ID in their schemes, therefore their method have no effect in defending eclipse attacks, because the attacks can be performed with a medium number of malicious nodes. More importantly, they do not solve the challenge: determination of the appropriate puzzle's difficulty in a heterogeneous environment is difficult, and continuously solving puzzles is a burden to normal users.

3 Puzzle-based ID Generation

In our opinion, an applicable ID generation procedure should meet the following requirements:

- An ID can be computed with a user's own computer within acceptable CPU time.
- An ID can be immediately verified, and no challenge-response messages or time synchronization is required.
- An ID cannot be used by another node.
- The result of ID computing cannot be predicted, so the only way for an attacker to get his desired IDs is to generate numerous IDs.
- A period of validity is carefully chosen so that ID renewal would not induce significant churn, and in the same time it is difficult for an attacker to compute desired IDs within this period.

In this section we apply the idea of *message specific puzzle* [17] and present an original method of generating IDs which satisfies all these requirements.

3.1 Basic Construction

Now let us describe the details of puzzle-based ID generation. In order to generate a valid ID, a user u enumerates all possible values of P_u until she has found one solution of the following puzzle:

$$H(IP_u|port_u|T|P_u) = \underbrace{00\ldots0}_{l \text{ bits}}xx\ldots x \tag{1}$$

where H is a cryptographic hash function, e.g. SHA-1. For simplicity we use SHA-1 as H in the rest of our paper. IP_u and $port_u$ are network parameters of current DHT node, T is a piece of public information which renews periodically (every t_{cycle}) with the length of l_1 bits, P_u has a fixed length of l_2 bits, and each x is an arbitrary value of 0 or 1.

Given IP_u, $port_u$ and T, it would take certain CPU time to find a valid P_u so that the first l bits of the hash value are all zeros. There is a possibility that the user enumerates all possible l-bit strings and still does not find a valid P_u, the possibility is very small though as long as $l_2 \geq l + 4$ (about 10^{-7}), and even if the user fails to do so, she could simply change the $port_u$.

The time-related public information T should have several characters as follows: (1) It can be automatically retrieved from many different sources, but the value is unanimous; (2) It changes periodically; (3) It cannot be manipulated; (4) It cannot be predicted. Character (4) does not affect the computing cost, but it can stop the attacker from computing valid IDs ahead of time. Such public information abounds in our world, for example, "Dow Jones Industrial Average at the end of last Friday and last Friday's date". The values can be stored as two 32-bit integers (ignoring the decimal point of DJI). They can be easily fetched from dozens of web sites, and are always unanimous. Predicting or manipulating such values is impossible.

Now the user can compute her ID:

$$ID_u = H(IP_u|port_u|T|\overline{P_u}) \tag{2}$$

in which

$$\overline{P_u} = P_u \oplus \underbrace{11\ldots1}_{l_2\text{bits}}$$

Verification of an ID is very easy. Let T_{cur} and T_{last} represent T values in current and the last period. Each ID has a validity of two periods. An ID is always transferred with its associated P_u. Whenever a node is considering adding another node (who is not behind a NAT) with certain ID_u into its routing table, it computes $H(IP_u|port_u|T_{cur}|P_u)$

and $H(IP_u|port_u|T_{last}|P_u)$, and checks if one of them satisfies the pattern. If so, it computes $H(IP_u|port_u|T|\overline{P_u})$ with corresponding T and checks if ID_u is legal.

When a new period is coming, a user u should randomly decide a time t_u in current period to renew its ID. It could continue to use the old one, and fetches new T and computes new ID after t_u. As long as t_{cycle} is chosen carefully, e.g. a week, and t_u distributed uniformly, the induced churn is negligible.

IP changing would also lead to ID renewal. Since a normal network access session lasts for hours, the workload of computing a new ID at the beginning of a new session is also acceptable. It is obvious that our method meets all five requirements we proposed at the beginning of this section.

3.2 Handling NATs

Steiner et al. [22] point out that a large amount of peers located behind NATs cannot be directly contacted in Kad. A NAT would block the lookup queries, and peers behind NATs do not participate in storing published information, and therefore make no contribution to the DHT protocol.

Peers behind a NAT have no information about their public IPs on the Internet, thus their ID generation always leads to an invalid value in our scheme. Luckily, since their IDs are invalid, they will never get a chance to enter a node's routing table, and thus make no contribution to an eclipse attack as well.

Consequently, if a node can detect that it is behind a NAT, it can skip the generating procedure and use a random number as its ID.

4 Analysis

4.1 Choice of Parameters

As stated, a week's period is long enough to maintain the stability of the network, and 64 bits (namely $l_1 = 64$) of T should induce adequate inscrutability. A P_u of $l+3$ bits will definitely suffice, which becomes obvious in the following analysis. The most important parameter in our method is the length of the pre-determined pattern, i.e. l. Given a puzzle length l, the possibility of finding the first puzzle solution on the x-th trial is $P_{x,l} = 2^{-l} \times (1 - 2^{-l})^{x-1}$. Therefore, the expected number of trials of finding a puzzle solution P_u is

$$E(x) = \Sigma_{x=1}^{\infty} P_{x,l} \cdot x = 2^l$$

Thus on average it takes 2^l trials to find a solution, and one trial requires a SHA-1 computation on 55 bytes at most (the total length of IP_u, $port_u$, T, P_u, which is exactly one data block after padding). Our goal is to limit the computation for a normal user in reasonable time. According to the well-known Crypto++ 5.6.0 benchmark [11], an AMD Opteron 8354 2.2 GHz processor on Linux platform can perform 3.49×10^6 trials per second, and an Intel Pentium 4 (Prescott) CPU (2.9GHz) can perform 2.51×10^6 trials per second. When $l = 23$, it takes 2.40 seconds on average to find a solution with an AMD Opteron 8354 2.2 GHz processor, and 3.34 seconds on an Intel Pentium 4 (Prescott) CPU. The l value can be easily adjusted as the computers become faster.

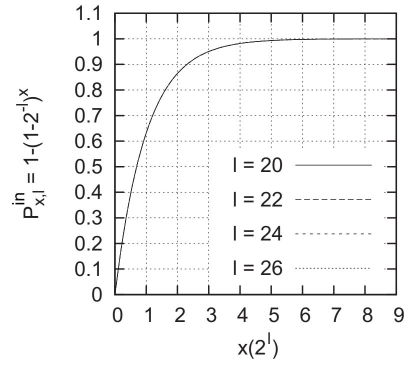

Figure 1. Probability of finding a puzzle solution within $x \cdot 2^l$ trials

Now let us consider the worst case. Figure 1 (cited from [17]) shows the possibility of finding a puzzle solution within $x \cdot 2^l$ trials, e.g.$P^{in}_{x,l} = 1 - (1 - 2^{-l})^{x \cdot 2^l}$. We can see that there is a 95% possibility of finding a solution within 3 times of average duration, and in 6×2^l trials the possibility is almost 100%. Note that these lines almost overlap with each other, which means the choosing of l has almost no effect on these possibilities:

$$P^{in}_{x,l} = 1-(1-2^{-l})^{x \cdot 2^l} = 1-(1-2^{-l})^{-2^l \cdot (-x)} \approx 1-e^{-x}$$

Hence $l_2 = l + 4$ (which means 16×2^l trials at most for a single $port_u$) is a reasonable choice. The possibility of failure is $e^{-16} \approx 1.1 \times 10^{-7}$.

4.2 ID Generation From an Attacker's Point of View

Because it is impossible for a malicious node to claim to be its desired ID, an attacker would have to generate enormous IDs to get enough ones closer to the target than any other node in the network. The desired ID space of the attacker is $N^{-1} \times \frac{1}{2}$ on average, where N is the total amount of nodes (who are not behind NATs) in the DHT. Figure 2 illustrate the estimation. In the interest of being closer to the target ID (ID_e) than the nearest benign nodes n_1 and n_2, the desired ID space has a span of

$$\min\{|ID_e - n_1|, |n_2 - ID_e|\} \times 2$$

which is $\frac{1}{2}|n_2 - n_1|$ on average. The expectation of $\frac{1}{2}|n_2 - n_1|$ is $\frac{1}{2N}$ of the entire ID space. Given a desired ID space of $\frac{1}{2N}$, the possibility of finding a desired ID on the x-th trial is $P_{x,N} = \frac{1}{2N} \cdot (1 - \frac{1}{2N})^{x-1}$. As a consequence, the attacker has to generate $2N$ IDs to find a useful one on average, the analysis is similar to the discussion on choosing l. This assumption is very loose, for the real-world ID distribution is random rather than uniform.

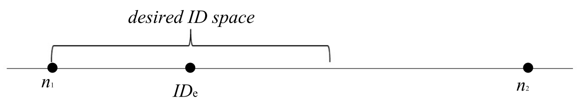

Figure 2. Estimation of the desired ID space for an attacker

His overall computing effort in an attack can be estimated as:

$$C_{attacker} = t \times m \times 2N$$

in which $C_{attacker}$ represents the computing effort of the attacker (in seconds of CPU time), t is the average time of generating a valid ID, and m is the malicious nodes he need to isolate a target ID. To complete the computation within t_{cycle} (so that he can continuously perform the attack), he needs $C_{attacker}/t_{cycle}$ processors working continuously without interruption.

We now apply our estimation using real-world data from BTDHT. There are about 22 million online nodes in BTDHT at most simultaneously [25]. Estimating m is difficult, because different client programs have different implementations. In BTDHT, a lookup request can ask for 8 nodes, so it is impossible to perform the attack with less than 8 valid IDs. Therefore we use 8 to estimate the minimal nodes to perform the attack.

To execute an eclipse attack in BTDHT, assuming that the attacker uses AMD Opteron 8354 2.2 GHz processors which can compute an ID in 2.40 seconds, the total CPU time he needs is $2.40 \times (22 \times 10^6 \times 2) \times 8 = 8.45 \times 10^8$ seconds (about 10^4 days). To complete the computation within a week (so that he can continuously perform the attack), he needs 1397 processors running day and night.

We can use Amazon EC2 pricing standard [2] to give out a simple estimation on the expense of ID generation. The most economic choice of an attacker is to use 84 Cluster GPU Instances [1], and the cost is \$4234 per day (on-demand instances) or \$$4.73 \times 10^5$ per year (one-year-term reserved instances).

Moreover, because each IP can only provide 6.5×10^4 available ports and 16 valid P_u per port on average, the attacker needs 339 IPs to perform the attack at any ID_e he wants. If the attacker's IP resources are not enough, it is very likely that he cannot reach his desired ID space no matter how much he cost on computing IDs.

4.3 ID Repetition

ID repetition is a very common problem in DHTs. About 19.5% of peers in routing tables and 4.5% of active peers (those who respond to the DHT protocol) in Kad (the DHT network of eMule) do not have unique IDs [26]. ID repetition degrades Kad's performance on publishing and searching. It can be considered as a special case of eclipse attacks.

In our scheme this form of attack is almost impossible. The difficulty of generating a specific ID is guaranteed by the mathematical property of cryptographic hash functions.

4.4 Comparison with Previous Techniques

In this part we compare our work with three existing defenses discussed in 2.3.2: the second example proposed by Castro et al. [7], the methods proposed by Halderman and Waters [13] and that by Rowaihy [18]. For simplicity we assume the workload of a normal user is 1, and the network scale is 20 million. An attacker who generates 10% of the number of existing nodes can perform a sybil attack and paralyze the network [18], and 8 IDs is enough to perform an eclipse attack.

	[7]	[13]	[18]	Our method
Eclipse	8	1.6×10^7	8	1.6×10^7
Sybil	8	2×10^6	2×10^6	2×10^6
Replay	N	Y	N	N
Pre-compute	Y	N	N	N

Table 1. Comparison with similar methods

The result is presented in Table 1. The first data row represents the attacker's workload in a typical eclipse attack, and the second row represents that in a typical sybil attack. The third row indicates whether the attacker can replay a user's authentication message. The last row is whether the attacker can pre-compute the puzzle. The table shows that our method has the best overall performance with regard to the four aspects.

4.5 Disadvantages

The main disadvantage of this method is that the computing cost of an attacker is associated with the size of the network. Generating IDs during an eclipse attack on a 500-node DHT can be accomplished by a single processor, thus our method can only be applied to large DHTs. On the other hand, a centralized certification service is applicable in small DHTs. In practice, say that a Hollywood company wants to block the distribution of its latest movies, it might still afford the computational cost of performing attacks on all large-scale file-sharing DHTs. Nevertheless, our method still greatly enhances the difficulty of an attack without complicating DHT protocol, and in the same time, the additional computing and communication cost for a normal user is negligible.

Another disadvantage is that for a user with an unstable network connection, her IP may change every time her connection is lost, and frequent ID computation would affect the performance of her P2P application. To avoid such situation, she can simply skip the generating procedure and use a random number as her ID, just like a node behind a NAT. After all, a node whose IP changes frequently is not likely to contribute much to a DHT.

5 Simulations

In this section we validate our method by simulating the computing effort of the users and the attackers.

5.1 ID Generation Time Distribution

To measure the cost of a normal user, we generate 1000 valid IDs with our method and record the time it took for generating each ID. The parameters in Equation (1) are chosen in the following way: a random combination of IP_u, $Port_u$ and T is generated for each ID before its computation. This simulation is carried out on a Lenovo Ideapad Y450 Laptop with an Intel Core2 Duo T6600 2.20GHz processor and 2.00GB RAM, such computing power is available to a normal user. The operation system is Win 7 and the program is written in C++. Figure 3 presents the cumulative distribution of generation time. As we can see the curve is very close to the theoretic prediction of Figure 1.

The average computing time is 8.68 seconds, and over 97% of IDs is computed within 30 seconds. The longest generation time for an ID is 56.54 seconds. Since the computation is required only once in a week and can be performed in background, the cost is acceptable for most users.

5.2 The Attacker's Effort

In this part we analyze the relation between an attacker's computing effort and his attack capacity. For simplicity we use a 56-bit ID space and the distance between two nodes is computed as the Euclidean distance between their IDs. We randomly distribute 2.2×10^7 benign nodes (the current BTDHT size) and 10^5 target IDs in the space. An attacker has to generate 8 IDs closer to the target ID than any benign nodes does, in order to isolate the target ID. Our fist simulation shows that valid IDs distribute randomly in

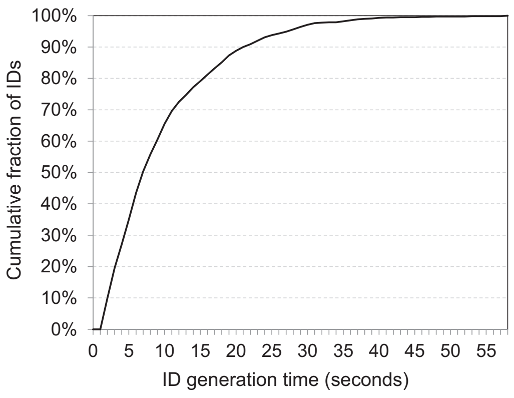

Figure 3. Cumulative distribution of ID generation time

the space, so we use SHA-1 function on sequential numbers to simulate an attacker's effort. We assume that the attacker performs the computation within one t_{cycle} (otherwise, part of his effort would be invalid at the end of the computation). According to our theoretic prediction in section IV, an attacker has to generate $m \times 2N = 3.52 \times 10^8$ IDs on average to perform an attack, which perfectly meets our simulation results in Figure 4.

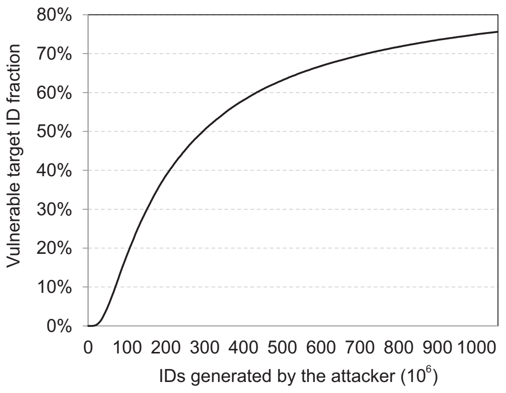

Figure 4. Number of nodes generated by an attacker vs. number of IDs he can attack

Our simulation indicates that few target IDs (less than 1%) can be eclipsed until the number of nodes generated by the attacker reaches the number of existing nodes. And when he has 3.52×10^8 IDs he can only attack 55% of IDs. If he computes as much as 3 times of the number, he can attack 75% of IDs. Our simulation on different network scales shows similar results.

6 Conclusion

An eclipse attack is very difficult to detect or prevent, since the attacker nodes can act normally in every way except when they are asked for certain target IDs. Defending against the eclipse attack involves a trade-off between performance and complexity. In this work we present a method of making eclipse attacks in large-scale DHTs computationally infeasible by a puzzle-based ID generating and verifying procedure. The induced communication and computing cost is negligible, other than an ID renewal procedure taking several seconds per week. To the best of our knowledge, it is the first ID generating method, which does not involve a centralized certification server and induce little churn.

It is a good idea to involve a human interaction in the authentication procedure [15], since it would make automatic ID generation impossible. It would be a great improvement to our method if that can be achieved without the help of a centralized server.

Acknowledgment

We would like to thank Dr. Peng Ning of North Carolina State University, USA: His course in Dragon Star Program helps us bring about the key idea of this paper. We also thank Dr. Tielei Wang of Peking University for his insightful suggestions on an early version of this paper.

References

[1] Amazon ec2 instance types.
Available: http://aws.amazon.com/ec2/instance-types/.

[2] Amazon ec2 pricing.
Available: http://aws.amazon.com/ec2/pricing/.

[3] L. M. Aiello, M. Milanesio, G. Ruffo, and R. Schifanella. Tempering kademlia with a robust identity based system. In *Proc. IEEE International Conference on Peer-to-Peer Computing (P2P'08)*, 2008.

[4] B. Awerbush and C. Scheideler. Group spreading: A protocol for provably secure distributed name service. In *Proc. 31st International Coll. on Automata, Languages and Programming (ICALP)*, Turku, Finland, 2004.

[5] S. Balfe, A. D. Lakhani, and K. G. Paterson. Trusted computing: Providing security for peer-to-peer networks. In *Proc. IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, Kostanz, Germany, 2005.

[6] N. Borisov. Computational puzzles as sybil defenses. In *Proc. IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, 2006.

[7] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, USA, Dec. 2002.

[8] D. Cerri, A. Ghioni, S. Paraboschi, and S. Tiraboschi. Id mapping attacks in p2p networks. In *Proc. IEEE GLOBECOM'05*, St. Luis, USA, 2005.

[9] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems (P2PECON)*, 2003.

[10] T. Condie, V. Kacholia, S. Sankararaman, J. M. Hellerstein, and P. Maniatis. Induced churn as shelter from routing-table poisoning. In *Proc. Network and Distributed System Security Symposium (NDSS)*, San Diego, USA, 2006.

[11] W. Dai. Crypto++ 5.6.0 benchmarks. Available: http://www.cryptopp.com/benchmarks.html, Mar. 2009.

[12] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *Proc. ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003.

[13] J. A. Halderman and B. Waters. Harvesting verifiable challenges from oblivious online sources. In *Proc. ACM Conference on Computer and Communications Security(CCS'07)*, Alexandria, USA, 2007.

[14] K. Hildrum and J. Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *Proc. International Symposium on Distributed Computing (DISC)*, Sorrento, Italy, Oct. 2003.

[15] L. Maccari, M. Rosi, R. Fantacci, L. Chisci, L. M. Aiello, and M. Milanesio. Avoiding eclipse attacks on kad/kademlia: an identity based approach. In *Proc. IEEE ICC'09*, Dresden, Germany, 2009.

[16] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. IPTPS'02*, Cambridge, USA, 2002.

[17] P. Ning, A. Liu, and W. Du. Mitigating dos attacks against broadcast authentication in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4, Jan. 2008.

[18] H. Rowaihy. Limiting sybil attacks in structured peer-to-peer networks. In *Proc. IEEE INFOCOM'07*, St. Louis, US-A, 2007.

[19] A. Singh, T. W. Ngan, P. Druschel, and D. S. Wallach. E-clipse attacks on overlay networks: Threats and defenses. In *Proc. IEEE INFOCOM'06*, Barcelona, Spain, 2006.

[20] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proc. IPTPS'02*, Cambridge, USA, 2002.

[21] M. Steiner, E. W. Biersack, and T. En-Najjary. Exploiting kad: Possible uses and misuses. *Computer Communication Review*, 37:65–70, Oct. 2007.

[22] M. Steiner, T. En-Najjary, and E. W. Biersack. Analyzing peer behavior in kad. Technical Report EURECOM+2358, Institut Eurecom, France, Oct. 2007.

[23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *Proc. ACM SIGCOMM*, San Diego, USA, 2001.

[24] G. Urdaneta, G. Pierre, and M. van Steen. A survey of d-ht security techniques. *ACM Computing Surveys (CSUR)*, 43(2):1–53, 2011.

[25] S. Wu. Measurement methods of dht network behavior. Master's thesis, Peking University, Beijing, China, 2011.

[26] J. Yu. Id repetition in kad. In *Proc. IEEE International Conference on Peer-to-Peer Computing (P2P'09)*, 2009.