# The Sybil Attack on Peer-to-Peer Networks From the Attacker's Perspective

Andrew Rosen Brendan Benshoof Robert W. Harrison Anu G. Bourgeois
Department of Computer Science
Georgia State University
Atlanta, Georgia
rosen@cs.gsu.edu bbenshoof@cs.gsu.edu rwh@cs.gsu.edu anu@cs.gsu.edu

***Abstract*—This paper explores the feasibility of performing naive Sybil attacks on a P2P network that completely occlude healthy nodes from each other. The vulnerability of Distributed Hash Tables to Sybil attacks and Eclipse attacks has been well known for some time. However, these vulnerabilities have often been explored in a theoretical sense, assuming the attacker is an all-knowing and all-powerful adversary from the beginning. This paper proves that these are not necessary characteristics and assumptions for an adversary to have to perform a Sybil attack.**

**We examine the amount of computational effort required to perform an Eclipse attack using only Sybil identities and without poisoning routing tables via false advertisement. We do this by analyzing the amount of effort it takes for an attacker with a given IP to choose a port to obtain a desired hash key, a process we call *mashing*. Mashing requires no additional resources on the adversary's end. Our results showed that an adversary with just 3 IP addresses could compromise 90% of all connections in a Chord network with 5000 nodes. While mashing can be used for malicious behavior, we also describe methods and applications for the technique that are beneficial for performing load balancing. This provides the flexibility for any node to effortless shoulder additional work from bottleneck locations, if acting for the good of the network.**

***Index Terms*—Sybil Attack; Eclipse Attack; P2P Security; Security; Man-in-the-middle**

## I. Introduction

One of the key properties of structured peer-to-peer (P2P) systems is the lack of a centralized coordinator. P2P systems remove the vulnerability of a single point of failure and and the susceptibility to a denial of service attack [1], but in doing so, open themselves up to new attacks.

Completely decentralized P2P systems are vulnerable to *Eclipse attacks*, whereby an attacker completely occlude healthy nodes from one another. This prevents them from communicating without being intercepted by the adversary. Once a Eclipse attack has taken place the adversary can launch a variety of crippling attacks, such as incorrectly routing messages or returning bad data [2]

One way to accomplish this attack is to perform a *Sybil Attack* [1]. In a Sybil attack, the attacker creates multiple malicious virtual nodes in order to disrupt the network. If enough malicious nodes are injected into the system, the majority of the nodes will be occluded from one another, successfully performing an Eclipse attack.

This vulnerability is well known and has been for some time and extensive research has been done assessing the damage an attacker can do after establishing themselves [2]. Little focus has been done on examining how the attacker can establish himself in the first place and *precisely* how easy the Sybil attack can be accomplished.

Our goal is to look at the Sybil attack from the perspective of the adversary executing the Sybil attack. We did a formal analysis on the breadth and depth of damage an adversary could have on a generic structured P2P network. We also constructed simulations showing how an attacker could insert themselves into the network and proceed to incrementally inject malicious nodes in between normal members of the network, a process we call *mashing*.

Sybil attacks represent a significant threat to the security of any distributed system. Many of the analyses [3] on Tor [4] emphasize the vulnerability of Tor to the Sybil attack. This threatens the anonymity of Tor users, particularly those living in countries with peculiar notions about personal privacy.

Sybil attacks are also a threat to P2P systems such as BitTorrent, which are essential to a wide variety of users. BitTorrent is currently the *de facto* platform for distributing large files scalably among tens of thousands of clients. It relies on Mainline DHT (MLDHT) [5] as a

backend. The number of users on Mainline DHT ranges from 15 million to 27 million users daily, with a turnover of 10 million users a day [6]. Current research demonstrates BitTorrent is vulnerable to the Sybil attack and a persistent attack disabling BitTorrent would be highly detrimental to many users, but especially developers and system administrators.[1]

There have been many suggestions on how to defend against Sybil attack, but there is no agreed upon "silver bullet" among researchers that should be implemented for every distributed application [8] [9]. Part of this is surely influenced by the only surefire way to defend against a Sybil attack is to introduce a trusted authority to certify and/or bind identities. This solution potentially removes the Sybil attack, but reintroduces vulnerabilities to denial of service attacks, bringing us full circle. Another reason is that there is no single solution for all platforms is that not every solution is compatible with each distributed platform.

Our work presents the following contributions:

- We first discuss the mechanics of performing a Sybil attack and analyze the theoretical effort needed to bring to bear to perform the Sybil attack.
- We present our simulations which show how quickly even a naive and inefficient Sybil attack can compromise a system. We also discuss how a more intelligent attack can be geared to each of the more popular DHT topologies.
- We discuss the implications of our work, specifically how the techniques we developed to perform the attack can be used for automatic load balancing.

## II. Formal Analysis

In our analysis we make a few assumptions. We present these assumptions to so that we can be specific in our analysis and simulations. Without many of these assumptions, the Sybil attack becomes trivial to perform.

### A. Assumptions

Our first assumption is that the systems we analyze are fully distributed and assign identities to nodes and data using a cryptographic hash function. These systems are distributed hash tables (DHTs).

Cryptographic hash functions work by mapping some input value to an $m$-bit key or identifier. Well-known hash functions for performing this task include MD5 [10] and SHA1 [11]. Keys generated by the hash function are assumed to be evenly distributed and random [12].

In distributed hash tables, $m$ is a large value, in order to avoid collisions between mapped inputs, unintentional or otherwise. An $m \geq 128$ is typical, with $m = 160$ being the most popular choice.

Our second assumption is that node IDs are generated by hashing their IP address and port. This is a form of very weak security that binds a particular hash key to a specific IP/port combination [13] [14]. This method is often used as a means of generating node IDs.

Although other methods do exist, the only other one that is mentioned as often let nodes choose their own $m$-bit ID at random.[2] This is notably done in Mainline DHT, and makes the system extremely vulnerable to a Sybil attack. Since there no way to verify that a node chose the $m$-bit ID at random, nodes in the network accept advertised keys at face value. This allows a prospective attacker to choose any specific key they want.

We also assume that nodes verify that a peer's advertised IP/port combination generates the advertised node ID. This verification is not specified or explicitly used in any DHT. Without it, our assumption about binding IDs to IP addresses and ports would be moot, since an attacker could advertise a single

Consider an attacker operating under these assumptions who wants to inject a malicious node in between two victims. The attacker must search for a valid IP and port combination under his control that generates a hash key that lies between the two victims's keys, a process we will call *mashing*. The attackers ability to compromise the network depends on what IP addresses and ports he has available.

The process for mashing is similar to searching for a hash collision, but much easier. Rather than searching for two inputs to a function that produce the same $m$-bit output, the attacker searches for a single input and $m$-bit output that falls between two given $m$-bit IDs. We assume the IDs are evenly distributed [12], so in a size $n$ network there would be $\approx \frac{2^m}{n}$ unused IDs between each pair of nodes. This mean the attacker is actually searching for a collision with one of $\approx \frac{2^m}{n}$ $m$-bit numbers, a much simpler problem.

[1] Perhaps the only reason that BitTorrent hasn't be attacked in such a way as to render it unusable is that those capable of doing so rely heavily upon it. A sobering thought, since BitTorrent is under an active Sybil attack [7].

[2] Another commonly mentioned method is generate public keys, but the specifics are often left unmentioned. If the keys are certified by an centralized source, we no longer have a completely decentralized network. If the nodes are generating the keys themselves, then we have essentially the same problem as letting nodes choose keys at random.

### B. Analysis

Suppose we have a DHT with $n$ members in it, with $m$-bit node IDs between $[0, 2^m)$. Consider two victim nodes with IDs $a$ and $b$. The probability $P$ that an attacker can mash a hash key that lands in the range $(a, b)$ is:

$$P \approx \frac{|b-a|}{2^m} \cdot num\_ips \cdot num\_ports$$

where $num\_ip$ addresses is the number of IP addresses the attacker has under his control and $num\_ports$ is the number of ports the attacker can try for each IP address.

If the ports the attacker can try are limited the ephemeral ports,[3] the attacker has 16383 ports to use for each IP address. As previously noted, we assume that for a large enough $n$, node IDs will be close to evenly distributed across the keyspace, meaning there will be $\approx \frac{2^m}{n}$ unused IDs between each node ID. This makes the earlier probability equivalent to:

$$P \approx \frac{1}{n} \cdot num\_ips \cdot num\_ports$$

This indicates that the ease of mashing is independent to $m$.

The chances of an attacker mashing all $n$ nodes that partition the entire keyspace is:

$$C \approx 1 - (1 - \frac{1}{n})^{num\_ips \cdot num\_ports}$$

Given a healthy node, the probability $P_{bad\_neighbor}$ that a Sybil is his closest neighbor is:

$$P_{bad\_neighbor} = \frac{num\_ips \cdot num\_ports}{num\_ips \cdot num\_ports + n - 1} \quad (1)$$

Our experiments in Section III-D show that $P_{bad\_neighbor}$ is actually the probability that *any* of a nodes links connect to a Sybil.

From the previous equation, the adversary can compute how many unique IP/port combinations they need if they wish to obtain a desired probability $P_{bad\_neighbor}$:

$$num\_ips \cdot num\_ports = \frac{n-1}{1 - P_{bad\_neighbor}}$$

Using our previous assumption that the adversary is limited to the 16383 ephemeral ports, the attacker can computer the number of unique IP addresses needed:

$$num\_ips = \frac{n-1}{1 - P_{bad\_neighbor}} \cdot \frac{1}{16383}$$

We verify these equations with our simulations.

[3]Also known as private or dynamic ports. These ports are never assigned a specific use by the Internet Assigned Numbers Authority and are available for applications to use as needed [15].

## III. Simulations

An essential part of our analysis was demonstrating just how fast an adversary can compromise a system. We performed four experiments to accomplish this task. Our simulations we written in Python 2.7 and performed on an AMD Phenom II 965 processor.

We used SHA1 for as our cryptographic function, which yields 160-bit hash keys. Again, we use the constraint that victim nodes do an absolute minimum verification which forces an attacker to only present Sybils with hash keys that he can generate with valid IP and port combinations.

Another important aspect of our simulation is that we assume the attacker *does not* need to create or use a fully featured client to integrate with the network he's attack. This is for a couple of reasons, the primary one being that we assume the attacker is capable and competent. An adversary who wants to use a Sybil attack on a large system would likely roll his own client capable of running on the P2P protocol.

Previous research [7] shows that an adversary performing a Sybil attack does not need to maintain the state information of other nodes and can leverage the healthy nodes to perform the routing. Nor does the adversary necessarily create a new running copy for every Sybil created. If this was impossible, it is still reasonable to assume the attacker's program would be built to have a minimal memory footprint and drop any messages that require an attacker carry any of the network's load.

### A. Experiment 0: Mashing 2 random nodes

Our initial experiment was designed to establish the feasibility of injecting in between two random nodes. Each trial, we generated two victims with random IP addresses and ports, and an attacker with a random IP. The experiment was for the attacker to find a key in between the two victims's key, from the lowest to highest. The amount of time to mash two random keys was on average 29.6218 microseconds, and was achievable $99.996\%$ of the time.

This test shows that the hashing operation is extremely quick. However, two arbitrary nodes in a network can be and often are quite distant. Our next experiment seeks a bit more accuracy.

### B. Experiment 1: Time Needed to Mash a Region

After showing that mashing two arbitrary nodes takes a minuscule amount of time, the next step is to demonstrate that mashing the region between two adjacent

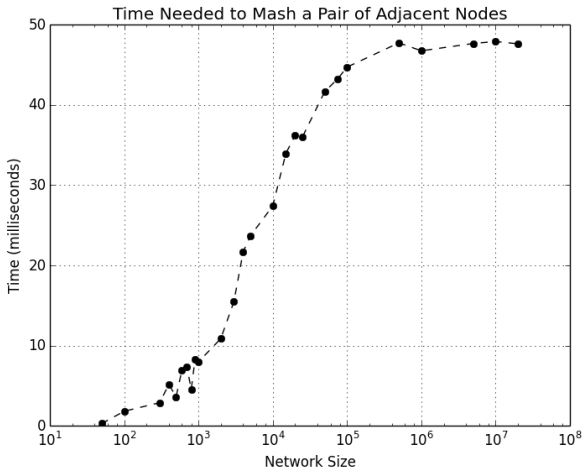


Fig. 1. This figure shows the amount of time needed for an adversary with a single IP address to find the appropriate hash key needed to mash a pair of adjacent nodes. The time it takes to mash a pair of nodes begins to markedly increase once the network size gets above 1000 until it asymptotes at 48 ms. For larger networks, more IP addresses are needed and precomputing becomes necessary

nodes in a network of size $n$ also takes an inconsequential amount of time. We simulate this by creating a list of node IDs for $n$ random IP/port combinations and pick a random adjacent pair of numbers from the list. The adversary then hashes their IP with their ports until they find an IP/port combination results in a hash key falling in between the pair. Our results are shown in the Figure 1 and Table I.

These figures show that the larger network size, the longer it takes the adversary to mash a given pair of adjacent nodes. Eventually, it the time asymptotes out to about 48 milliseconds. This is the amount of time it takes for the adversary with a single IP to generate all 16383 hash combinations using ephemeral ports. While this is a short time to mash a single region, an attacker following the attack specified in our next experiment will want to mash $n-1$ regions. If $n = 1,000,000$, this can take upwards of 13 hours.

Since the attacker's IP addresses and ports do not change over the course of the attack, the attacker would waste time by generating the same keys over and over. The mashing process could also take substantially longer if the network a hash function that produces numbers larger than the 160 bits we are assuming or if the network uses a more expensive function such as scrypt [16].

The attacker can instead perform all the work needed to mash a network upfront, precomputing all possible valid hash keys they can generate. We've shown this

TABLE I
TIMES AND SUCCESS RATE FOR MASHING ADJACENT NODES.

| Network Size | Success Rate | Avg Time to Mash (ms) |
|---|---|---|
| 50.0 | 1.0 | 0.29 |
| 100.0 | 1.0 | 1.82 |
| 300.0 | 0.99 | 2.89 |
| 400.0 | 0.98 | 5.16999 |
| 500.0 | 1.0 | 3.62999 |
| 600.0 | 0.98 | 6.97 |
| 700.0 | 0.94 | 7.32 |
| 800.0 | 0.99 | 4.54999 |
| 900.0 | 0.92 | 8.28 |
| 1000.0 | 0.92 | 7.96999 |
| 2000.0 | 0.95 | 10.88 |
| 3000.0 | 0.88 | 15.47999 |
| 4000.0 | 0.74 | 21.7 |
| 5000.0 | 0.71 | 23.68 |
| 10000.0 | 0.67 | 27.41 |
| 15000.0 | 0.5 | 33.93 |
| 20000.0 | 0.37 | 36.24999 |
| 25000.0 | 0.39 | 35.98 |
| 50000.0 | 0.23 | 41.64999 |
| 75000.0 | 0.2 | 43.25999 |
| 100000.0 | 0.13 | 44.71999 |
| 500000.0 | 0.04 | 47.73999 |
| 1000000.0 | 0.03 | 46.75 |
| 5000000.0 | 0.0 | 47.65999 |
| 10000000.0 | 0.0 | 47.90999 |
| 20000000.0 | 0.0 | 47.62 |

takes about 48 milliseconds for 16383 keys. Storing these values in a sorted list costs 160 bits for each SHA1 key and 16 bits for each port, for a total of $176 \cdot 16383 = 2883408$ bits, or roughly 352 kilobytes for each IP address the attacker has.

### C. Experiment 2: Nearest Neighbor Eclipse via Sybil

The objective of this experiment is to completely ensnare a network using a Sybil attack, starting with a single malicious node. With simulate this by creating a network of $n$ nodes. Each node is represented by a key generated by taking the SHA1 of a random IP/port combination.

The goal of the attacker is to mash as many pairs of the nodes as possible. We call this the *Nearest Neighbor Eclipse* since the attacker seeks to block as many lines of communication between neighboring nodes as possible.

The attacker is given $num\_ips$ randomly generated IP addresses, but can use any port between 49152 and 65535. This gives the attacker $16383 \cdot num\_ips$ possible hash keys to use for Sybils. As mentioned previously

in Section III-B, the attacker can easily precompute all of theses hash keys and store them in a sorted list to be used as needed, requiring only 352 kilobytes per IP. Since this list is sorted and this attack will go in order through the network, searching for a key that mashes a pair of nodes takes constant time.

To perform the attack, an adversary choses any random hash key as a starting point to "join" the network. This is his first Sybil and the join process provides information about a number of other nodes. Most importantly, nodes provide information about other nodes that are close in the hash space, which provided to enable fault tolerance between immediate neighbors. The adversary uses this information to inject Sybils in between successive healthy nodes.

For clarity we present this example. Consider the small segment of the network made up of adjacent nodes $a$, $b$, $c$, and $d$. The Sybil joins between nodes $a$ and $b$, and the joining process informs the adversary about node $c$, possibly node $d$, and a handful of other nodes in the network. The adversary will always learn about node $c$ because a normal node between $a$ and $b$ would need to know about node $c$ for fault tolerance.

The adversary's next move would be to inject a node between nodes $b$ and $c$. This is done by selecting a precomputed hash key $k$, such that $b \leq k \leq c$. The adversary injects a Sybil node with key $k$, which joins in between $b$ and $c$, and the joining process informs the adversary about node $d$ and several other nodes, including many close nodes. The adversary then aims to inject a node in between $c$ and $d$, and continues *ad nauseam*.

Depending on the network size and the number of keys available to the adversary, it is entirely possible the adversary will not have a key to inject between a pair of successive nodes. In this case, the node moves on to the successive pair that the node has learned about.

We simulated this attack on networks of up to size 20 million. We chose 20 million since it falls neatly into the 15-27 million user range seen on Mainline DHT [6]. We gave the attacker access to up to 19 IP addresses. Our results are in Figures 2 and 3 and Table II. For Table II, we have included only the results for larger sized networks, as the smaller sized networks were completely dominated.

Our results show that an adversary, given only modest resources, can inject a Sybil in between the vast majority of successive nodes in moderately sized networks. In a large network, modest resources still can be used to compromise more that a third of the network, an important goal if the adversary wishes to launch a Byzantine attack.

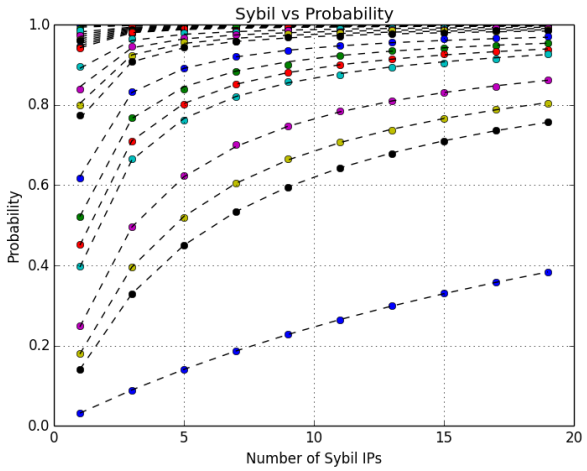


Fig. 2. Our simulation results. The $x$-axis corresponds to the number of IP addresses the adversary can bring to bear. The $y$-axis is the probability that a random region between two adjacent normal members of the network can be mashed. Each line maps to a different network size of $n$. The dotted line traces the line corresponding to the Equation 1: $P_{bad\_neighbor} = \frac{num\_ips \cdot 16383}{num\_ips \cdot 16383 + n - 1}$

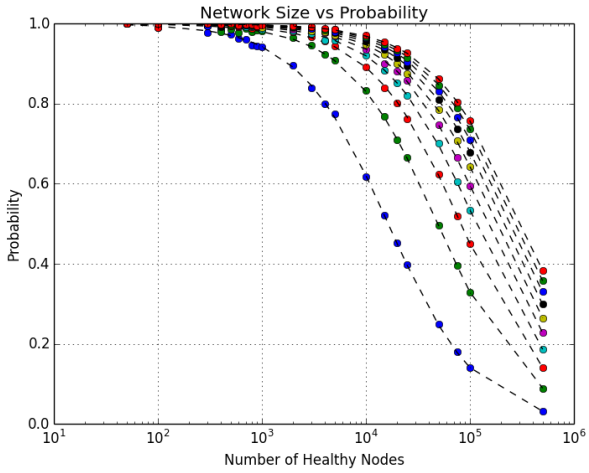


Fig. 3. These are the same as results shown in Figure 2, but our $x$-axis is the network size $n$ in this case. Here, each line corresponds to a different number of unique IP addresses the adversary has at their disposal.

Our results match values predicted by Equation 1. However, this experiment only covers the short links of the network, but not the long distance links.

### D. Experiment 3: Fully Complete Eclipse via Sybil

We extended the previous experiment by considering the long-distance hops of each node in addition to the short-range links of the DHT. We choose to model an

TABLE II
SELECTION OF RESULTS FOR NEAREST NEIGHBOR ECLIPSE.

| IPs | Network Size | Success Rate | Sybils/Region |
|---|---|---|---|
| 1 | 5000.0 | 0.7748 | 3.2762 |
| 1 | 5000000.0 | 0.0032654 | 0.0032768 |
| 1 | 10000000.0 | 0.0016364 | 0.0016384 |
| 1 | 20000000.0 | 0.00081865 | 0.0008192 |
| 5 | 5000.0 | 0.9444 | 16.381 |
| 5 | 5000000.0 | 0.0161208 | 0.016384 |
| 5 | 10000000.0 | 0.0081223 | 0.008192 |
| 5 | 20000000.0 | 0.0040801 | 0.004096 |
| 11 | 5000.0 | 0.9708 | 36.0428 |
| 11 | 5000000.0 | 0.0347646 | 0.0360448 |
| 11 | 10000000.0 | 0.0177117 | 0.0180224 |
| 11 | 20000000.0 | 0.008932 | 0.0090112 |
| 19 | 5000.0 | 0.9834 | 62.2452 |
| 19 | 5000000.0 | 0.058562 | 0.0622592 |
| 19 | 10000000.0 | 0.0301911 | 0.0311296 |
| 19 | 20000000.0 | 0.01532465 | 0.0155648 |

TABLE III
SELECTION OF RESULTS FOR A SYBIL ATTACK ON CHORD.

| IPs | Network Size | Success Rate | Occlusions/Node |
|---|---|---|---|
| 1 | 1000 | 0.94186875 | 150.699 |
| 1 | 10000 | 0.618480625 | 98.9569 |
| 1 | 100000 | 0.141753625 | 22.68058 |
| 3 | 1000 | 0.97915625 | 156.665 |
| 3 | 10000 | 0.83425 | 133.48 |
| 3 | 100000 | 0.3286290625 | 52.58065 |
| 5 | 1000 | 0.988725 | 158.196 |
| 5 | 10000 | 0.894415 | 143.1064 |
| 5 | 100000 | 0.4488916875 | 71.82267 |
| 7 | 1000 | 0.99091875 | 158.547 |
| 7 | 10000 | 0.919071875 | 147.0515 |
| 7 | 100000 | 0.5337635625 | 85.40217 |
| 9 | 1000 | 0.9948125 | 159.17 |
| 9 | 10000 | 0.935495625 | 149.6793 |
| 9 | 100000 | 0.5936118125 | 94.97789 |

attack on a P2P system built using Chord [17].

We chose to model Chord for a number of reasons. Chord is an extremely well understood DHT and is very simple to evaluate using simulations. Nodes in Chord generate long distance links independent of information provided by other nodes, rather than directly querying neighbors. This prevents adversaries from poisoning the node's routing table via false advertisements, which can be on other DHTs such as Pastry [18]. This makes the Sybil attack the most straightforward means of effecting an Eclipse attack on a Chord network.

Nodes in Chord have $m$ long-range links, one for each of the bits in the keys, which is 160 in our experiments. Each of node $a$'s long-range hops points to the node with the lowest key $\geq a + 2^i \mod 2^m, 0 \leq i \leq 160$.

The attack is very similar to the Nearest Neighbor attack demonstrated above. Beside injecting a node in between successive nodes, the attacker also attempts to place a Sybil in between each of the long-range links. We simulated this attack under the same parameters as above and are presented in Table III.

Surprisingly, the percentage of links from healthy nodes that connect to Sybil nodes follows III-D.

We can calculate that the number of IP addresses needed to compromise half the links in a 20,000,000 node network is 2442. While this seems like a daunting number of IP addresses, cloud computing has made solutions for an attacker much more accessible and affordable to attackers. At the time of writing, it would cost $73.26 USD to use 2442 instances for an hour on Amazon's Elastic Cloud Compute service. In fact, one of the attacker launching a Sybil attack was identified as having an IP address associated with Amazon's Elastic Cloud Compute [7].

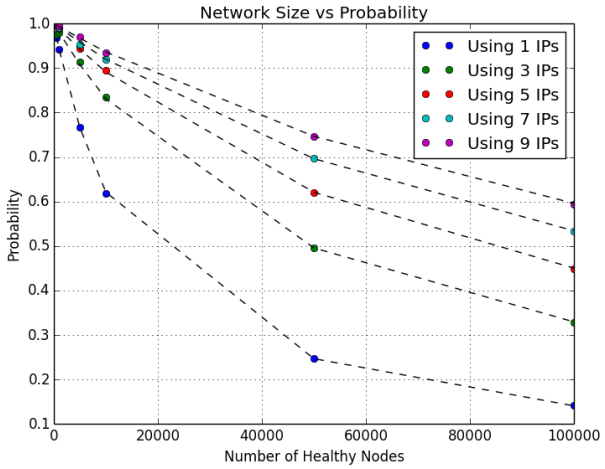


Fig. 4. This graph shows the relationship between the network size and the probability a particular link in Chord, adjacent or not, can be mashed. The dotted line traces the line corresponding to the Equation 1: $P_{bad\_neighbor} = \frac{num\_ips \cdot 16383}{num\_ips \cdot 16383 + n - 1}$

## IV. CONCLUSIONS AND FUTURE WORK

Our analysis and experiments show that an adversary with a small number of machines and limiting their port usage to just the ephemeral ports can easily compromise a P2P system and mash the majority of the regions between nodes. This effectively prevents nodes from

talking to one another without first sending messages through Sybils, who can eavesdrop on them or disregard the eaves and move straight on to the dropping.

Our discussion have primarily concerned Chord, but an astute reader may wonder why did we not simulate an attack on Mainline DHT [5], the Kademlia [19] based DHT used as the backend of BitTorrent. It is ostensibly the largest P2P system in use, so it seems more suited for analyzing our mash-based attack than Chord.

We avoided simulating the mashing process on MLDHT because, as we noted earlier in Section II-A, it is completely unnecessary to perform any mashing. In MLDHT, a node ID is not chosen by hashing an IP and port combination, but by picking an address uniformly at random between 0 and $2^{160} - 1$.

An adversary who wants to a completely isolate a node on the network can choose Sybils with "random" hashkeys completely eliminate that node from the routing tables of healthy peers. Research has examined MLDHT's vulnerability to Sybil attacks [7] and detected entities performing the attack on MLDHT. Our research demonstrates that switching to using a SHA1 hash of the IP and port is no defense against a Sybil attack, since the adversary can easily mash into appropriate places.

However, the mashing process can be used in non-security related settings to benefit a DHT. As we have mention previously, the SHA1 hash has an evenly distributed output, but no set of keys will be completely evenly distributed. Some nodes will be responsible for larger regions than others and therefore be responsible for a larger portion for the work.

If a node can detect a when a peer has too much of a load, the node can inject a virtual node into the region to shoulder some of the load. Too much of a load could be defined by having a too large region or by watching traffic requests. This could be integrated into a distributed key allocation system, so that a node can bind these keys to itself, but only use a small portion of the keys alloted to it to aid with load-balancing. The only cost would be that node would have to keep track of the keys it can inject, which we mentioned was only 320 kilobytes of data.

## REFERENCES


[1] J. R. Douceur, "The sybil attack," in *Peer-to-peer Systems*. Springer, 2002, pp. 251–260.

[2] M. Srivatsa and L. Liu, "Vulnerabilities and security threats in structured overlay networks: A quantitative analysis," in *Computer Security Applications Conference, 2004. 20th Annual*. IEEE, 2004, pp. 252–261.

[3] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against tor," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. ACM, 2007, pp. 11–20.

[4] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

[5] A. Loewenstern and A. Norberg, "BEP 5: DHT Protocol," http://www.bittorrent.org/beps/bep_0005.html, March 2013.

[6] L. Wang and J. Kangasharju, "Measuring large-scale distributed systems: case of bittorrent mainline dht," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, Sept 2013, pp. 1–10.

[7] ——, "Real-world sybil attacks in bittorrent mainline dht," in *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012, pp. 826–832.

[8] B. N. Levine, C. Shields, and N. B. Margolin, "A survey of solutions to the sybil attack," *University of Massachusetts Amherst, Amherst, MA*, 2006.

[9] G. Urdaneta, G. Pierre, and M. V. Steen, "A survey of dht security techniques," *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, p. 8, 2011.

[10] R. Rivest, "The md5 message-digest algorithm," 1992.

[11] D. Eastlake and P. Jones, "Us secure hash algorithm 1 (sha1)," 2001.

[12] M. Bellare and T. Kohno, "Hash function balance and its impact on birthday attacks," in *Advances in Cryptology-Eurocrypt 2004*. Springer, 2004, pp. 401–418.

[13] J. Dinger and H. Hartenstein, "Defending the sybil attack in p2p networks: Taxonomy, challenges, and a proposal for self-registration," in *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*. IEEE, 2006, pp. 8–pp.

[14] E. Sit and R. Morris, "Security considerations for peer-to-peer distributed hash tables," in *Peer-to-Peer Systems*. Springer, 2002, pp. 261–269.

[15] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry," *work in progress*, 2011.

[16] C. Percival, "Stronger key derivation via sequential memory-hard functions," *Self-published*, pp. 1–16, 2009.

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, pp. 149–160, August 2001. [Online]. Available: http://doi.acm.org/10.1145/964723.383071

[18] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*. Springer, 2001, pp. 329–350.

[19] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.