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 rharrison@cs.gsu.edu anu@cs.gsu.edu

***Abstract*—This paper examines the computational effort required to perform an Eclipse attack on a P2P network. The vulnerability of Distributed Hash Tables to Sybil attacks and Eclipse attacks is well known. Most analyses of these attacks assume that the attacker is all-knowing and globally powerful. We experimentally demonstrate that these are not necessary characteristics for the attacker. A less powerful adversary can perform a successful Eclipse attack, while still obeying the protocols of the Distributed Hash Table. We show that an adversary with a small number of machines can compromise a P2P system and occlude the majority of links. The attacker only requires the use of ephemeral ports. The attacker does not poison routing tables via false advertisement. Rather, they pose many legitimate "Sybil" nodes.**

We analyze the amount of effort it takes for an attacker with a given number of IP addresses to choose a port to generate a desired hash key. We call this *mashing*. Our results show that an adversary with just three IP addresses can compromise 90% of all connections in a Chord network with 5000 nodes. We suggest methods and applications for this technique that are beneficial for performing load balancing. This provides the flexibility for any node to shoulder the extra work from bottleneck locations.

***Index Terms*—Sybil Attack; Eclipse Attack; P2P Security; Security; Distributed Hash Tables; P2P networks**

I. Introduction

One of the key properties of structured peer-to-peer (P2P) systems is the lack of a centralized coordinator or authority. P2P systems remove the vulnerability of a single point of failure 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 occludes healthy nodes from one another. This prevents them from communicating without being intercepted by the adversary. Once an Eclipse attack has taken place, the adversary can launch a variety of crippling attacks, such as incorrectly routing messages or returning malicious data [2].

One way to accomplish an Eclipse attack is to perform a *Sybil attack* [1]. In a Sybil attack, the attacker masquerades as multiple nodes, effectively over-representing the attacker's presence in the network to maximize the number of links that can be established with healthy nodes. 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 [3]. 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 easily the Sybil attack can be accomplished.

A simple method to generate node locations is to assign them at random. A Sybil attack against this method is fairly straightforward; the attacker just assigns their own nodes at "random" locations. A more sophisticated method to generate node locations would use a hash function of some unique characteristic of the node. For example, a Distributed Hash Table could use a hash of the node's IP address and port number. In this paper, we show that using a hash function provides no protection against a Sybil attack.

Our goal is to look at the Sybil attack from the perspective of an adversary executing the Sybil attack. We did a formal analysis on the breadth and depth of the presence an adversary could establish on a structured P2P network. We also constructed simulations demonstrating the steps an attacker could follow to perform a Sybil attack. Once the attacker has joined the network, they can proceed to incrementally inject malicious nodes directly in between 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 on Tor [4] emphasize the vulnerability of Tor to the Sybil attack [5]. This threatens the anonymity of Tor users.

Sybil attacks are also a threat to P2P systems such as BitTorrent, which is 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. Many implementations rely on Mainline DHT (MLDHT) [6] to connect users to other peers. 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 [7]. Current research demonstrates BitTorrent is vulnerable to Sybil attacks and a persistent attack disabling BitTorrent would be highly detrimental to many users, especially developers and system administrators [8]. BitTorrent is currently under an active Sybil attack [9], although the attackers are apparently not trying to destroy the network.

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 [10] [3]. Urdaneta *et al* find that in DHT security, a centralized, trusted authority which certifies or binds identities is the most effective strategy [3]. This solution potentially removes the Sybil attack, but reintroduces vulnerabilities to denial of service attacks and is contrary to the objective of creating fully decentralized systems. Other techniques such as proof-of-work [11] may provide a defense against Sybil attacks, but proof-of-work systems must set their computational puzzles to be simple enough for the weakest member of the network to participate. However, the attacker can reasonably obtain computational power that is orders of magnitude greater than those weakest nodes and overcome the puzzle with brute force [3].

Our work presents the following contributions:

- We first discuss the assumptions behind performing a Sybil attack on a structured P2P network and analyze how effective a Sybil attack is based on the resources available to the attacker. We show that in a size n network being attacked by an adversary with s distinct identities that the probability that *any* given link leads to a Sybil is $P_{bad\_neighbor} = \frac{s}{s+n-1}$ (Section II).
- We present our simulations that show how quickly even a naive Sybil attack can compromise a system. We validate our experimental results by comparing them with the equations we specify in our analysis (Section III).
- We discuss the broader implications of our work, including how mashing can be used for automatic load balancing and disrupting P2P botnets (Section IV).

II. Analysis

We make a few assumptions for our analysis; some apply to the P2P network we are analyzing and some create rules that restrict the adversary. Without 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 called 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 [12] and SHA1 [13]. Keys generated by the hash function in our analysis are assumed to be uniformly distributed and random [14]. These hash functions are designed to make the intentional discovery of collisions computationally difficult.

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. If the choice of ports are restricted, the attacker would have to obtain more IP addresses. This is a form of very weak security that binds a particular hash key to a specific IP/port combination [15] [16]. 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 is to let nodes choose their own m-bit ID at random.[1] This is notably done in Mainline DHT, and makes the system extremely vulnerable to a Sybil attack. Since there is 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.

[1]Another commonly mentioned method is hash public keys [3] [16] [17] [18]. If the keys are certified or signed 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.

We also assume that nodes verify that a peer's advertised IP/port combination exists and that the hash function of these IP/port generates the correct node ID. This verification is not explicitly used or implemented in any DHT. Failure to validate advertised values results in a trivial security flaw.

Consider an attacker operating under these assumptions who wants to inject a malicious node in between two victims which have no other nodes in between them. The attacker must search for a valid IP and port combination under their control that generates a hash key that lies between the two victims' ID, a process we call *mashing*. The attacker's ability to compromise the network depends on what IP addresses and ports are 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 corresponding m-bit hash that falls between two given m-bit IDs. We assume the IDs are evenly distributed [14], so in a size n network there would be $\approx \frac{2^m}{n}$ unused IDs between each pair of nodes. This means the attacker is actually searching for a collision with one of $\approx \frac{2^m}{n}$ m-bit hashes, which is a much simpler problem.

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, with no others nodes with an ID in the range (a, b).

The probability that a single mash key lies in the range of (a, b) is $\frac{|b-a|}{2^m}$, which is the fraction of the network that lies between a and b. Thus the probability that the mash key does not lie in that range is $1 - \frac{|b-a|}{2^m}$. Assuming that the individual mash keys are statistically independent yields the expression:

$$(1 - \frac{|b-a|}{2^m})^{num\_ips \cdot num\_ports}$$

for the probability that no mash key lands in the range.

Thus the probability P that an attacker can mash a hash key that lands in the range (a, b) is:

$$P \approx 1 - (1 - \frac{|b-a|}{2^m})^{num\_ips \cdot num\_ports}$$

where num_ip is the number of IP addresses the attacker has under thier control and num_ports is the number of ports the attacker can try for each IP address.

If the ports the attacker can utilize are limited to the ephemeral ports,[2] 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. This means $|b-a| \approx \frac{2^m}{n}$. Therefore, the earlier probability is equivalent to:

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

This indicates that the statistical ease of mashing is independent of m.

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

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

This is effectively the number of malicous identities in the network ($num\_ips \cdot num\_ports$) over the total number of identities in the network. 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 $P_{bad\_neighbor}$:

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

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:

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

We verify these equations with our simulations.

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. These experiments were designed to analyze increasing levels of difficulty. The first experiment demonstrates a hash attack is possible and later experiments verify the feasibility on larger and more complicated networks. Our simulations were written in Python 2.7 and performed on an AMD Phenom II 965 processor.

We used SHA1 as our cryptographic function, which yields 160-bit hash keys. Again, we use the constraint

[2] 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 [19].

that victim nodes do an absolute minimum verification which forces an attacker to only present Sybils with hash keys that they can generate with valid IP and port combinations.

We assume that the attacker has no resource limitations. Previous research [9] 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 any needed routing. Nor does the adversary necessarily need to create a new running copy for every Sybil created, but doing so is quite feasible.

A. *Experiment 0: Mashing 2 random nodes*

Our initial experiment was designed to establish the feasibility of joining the network in between two random nodes. Each trial, we generated two victims with random IP addresses and ports, and an attacker with a random IP address. The experiment was for the attacker to find a key in between the two victims' keys, going clockwise. The average amount of time over 10,000 trials to mash two random keys was 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, in which case the mashing can be done quickly. Our next experiment studies a more realistic condition.

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 nodes in a network of size n also takes an inconsequential amount of time. We simulate this by creating a sorted 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 that results in a hash key falling in between the given pair. Our results averaged 100 trials for each network size are shown in Figure 1 and Table I.

These figures show that the larger the network size, the longer it takes for the adversary to mash a given pair of adjacent nodes. Eventually, the time it takes to mash a region asymptotes 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. While this is a short time to mash a single region, an adversary following the attack specified in our next experiment will

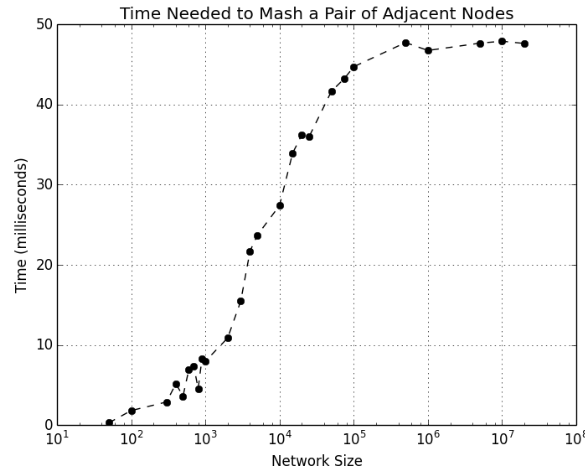

Fig. 1. This figure shows the amount of time needed for an adversary with a single IP address to mash a pair of adjacent nodes. The time it takes to mash a pair of nodes begins to markedly increase once the network size gets above 1000 until it asymptotes at 48 ms. For larger networks, more IP addresses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

TABLE I
TIMES AND SUCCESS RATE FOR MASHING ADJACENT NODES FOR A SINGLE IP.

Network Size	Success Rate	Avg Time to Mash (ms)
50	1.0	0.29
100	1.0	1.82
300	0.99	2.89
400	0.98	5.16999
500	1.0	3.62999
600	0.98	6.97
700	0.94	7.32
800	0.99	4.54999
900	0.92	8.28
1000	0.92	7.96999
2000	0.95	10.88
3000	0.88	15.47999
4000	0.74	21.7
5000	0.71	23.68
10000	0.67	27.41
15000	0.5	33.93
20000	0.37	36.24999
25000	0.39	35.98
50000	0.23	41.64999
75000	0.2	43.25999
100000	0.13	44.71999
500000	0.04	47.73999
1000000	0.03	46.75
5000000	0.0	47.65999
10000000	0.0	47.90999
20000000	0.0	47.62

want to mash $n-1$ regions. If the network size is 1 million, this process can take upwards of 13 hours in computational time.

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 160 bits or if the network uses a more expensive function such as scrypt [20].

The attacker can instead perform all the work needed to mash a network upfront, precomputing all possible valid hash keys. We have shown this 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 $176 \cdot 16383 = 2,883,408$ bits, or roughly 352 kilobytes for each IP address the attacker has.[3]

C. Experiment 2: Nearest Neighbor Eclipse via Sybil

The objective of this experiment is to completely eclipse a network using a Sybil attack, starting with a single malicious node. We 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 adjacent nodes as possible. We call this the *Nearest Neighbor Eclipse* since the attacker seeks to become the nearest neighbors of each node.

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 their 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 to it, which are provided to ensure fault tolerance between immediate neighbors. The adversary uses this information to inject Sybils in between successive healthy nodes. For example, in Pastry, a joining node typically learns about the 16 nodes closest to it for fault tolerance, in addition to all the other nodes it learns about [17]. In Chord, this number is a system configuration value r [21].

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 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 hash key k. 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 adversary moves on to the next successive pair that the adversary has learned about. In the extremely unlikely event this information is not already known, the adversary can look it up using the DHT's lookup function.

We simulated this attack on networks of up to 20 million nodes. We chose 20 million since it falls neatly into the 15-27 million user range seen on Mainline DHT [7]. 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 occluded.

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.

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.

[3] Storing the IP address is unnecessary since it is always the same.

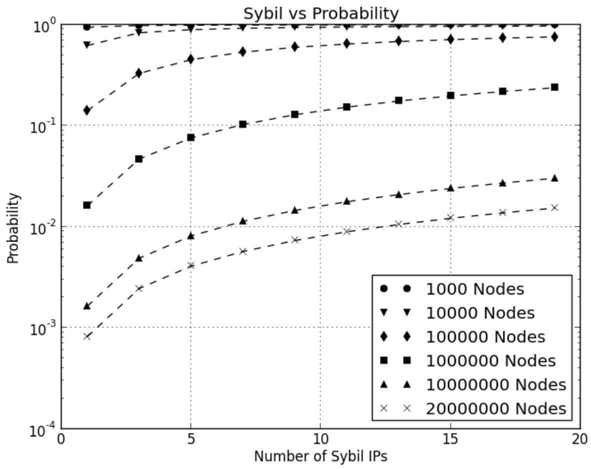

Fig. 2. Our simulation results. The x-axis corresponds to the number of IP addresses the adversary can bring to bear. The y-axis is the probability that any chosen region has been mashed. Each line maps to a different network size of n. The dashed line corresponds to values from Equation 1: $P_{bad\_neighbor} = \frac{num\_ips \cdot 16383}{num\_ips \cdot 16383 + n - 1}$

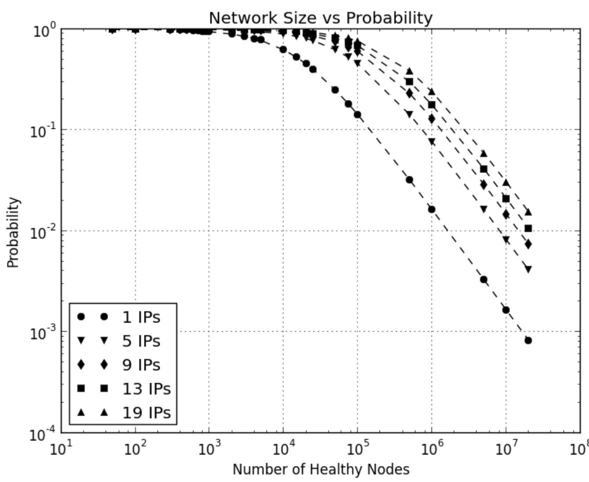

Fig. 3. These are the same as results shown in Figure 2, but our x-axis is the network size n in this case. Here, each line corresponds to a different number of unique IP addresses the adversary has at their disposal.

D. Experiment 3: Complete Eclipse via Sybil on Chord

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 attack on a P2P system built using Chord [21].

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 generates long distance links independent of information provided by other nodes, rather than directly querying neighbors. This minimizes the opportunities adversaries have to poison the node's routing table via false advertisements, which can be on other DHTs such as Pastry [17]. 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 links points to the node with the lowest key $\geq a + 2^i \mod 2^m, 0 \leq i \leq 160$.

However, many of the fingers are redundant and point to the nearest neighbor. As we have mentioned, on average, nodes are $\frac{2^m}{n}$ distance apart. The smaller the network, the further apart nodes are, and therefore each node has more redundant fingers and is easier to attack.

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 Equation 1 and the results from III-C. This means performing a Nearest Neighbor Eclipse provides the same results as actively blocking all the long range links in the network.

Our results show that an attacker needs only to focus their efforts on compromising the links between adjacent nodes to attack all the links in the network.

TABLE II

SELECTION OF RESULTS FOR NEAREST NEIGHBOR ECLIPSE.

IPs	Network Size	Success Rate	Sybils/Region
1	5000	0.7748	3.2762
1	5000000	0.0032654	0.0032768
1	10000000	0.0016364	0.0016384
1	20000000	0.00081865	0.0008192
5	5000	0.9444	16.381
5	5000000	0.0161208	0.016384
5	10000000	0.0081223	0.008192
5	20000000	0.0040801	0.004096
11	5000	0.9708	36.0428
11	5000000	0.0347646	0.0360448
11	10000000	0.0177117	0.0180224
11	20000000	0.008932	0.0090112
19	5000	0.9834	62.2452
19	5000000	0.058562	0.0622592
19	10000000	0.0301911	0.0311296
19	20000000	0.01532465	0.0155648

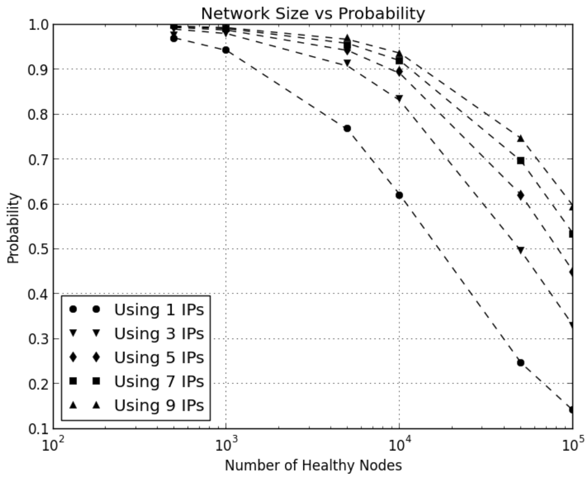

Fig. 4. This graph shows the relationship between the network size and the probability a particular link in Chord, adjacent or not, can be mashed. The dotted line traces the line corresponding to the Equation 1: $P_{bad\_neighbor} = \frac{num\_ips \cdot 16383}{num\_ips \cdot 16383 + n - 1}$

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

We can calculate that the number of IP addresses needed to compromise half the links in a 20,000,000 node network is 1221 IP addresses. 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 $43.26 USD to use 1221 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 [9].

IV. CONCLUSIONS AND FUTURE WORK

Our analysis and experiments show that an adversary with limited resources can easily compromise a P2P system and occlude the majority of the paths between nodes. All that is required is a small number of IP addresses and the unassigned ports. This implies system privileges are not required. A successful attack effectively prevents nodes from talking to one another without sending messages through Sybils. The Sybils can eavesdrop on the messages, actively reroute them, or substitute malicious data. Mashing can also be used prevent malicious behavior. Sybils inserted in a P2P botnet by this approach could effectively interfere with command and control functions to shut down the botnet [22]. Future research would involve developing an appropriate mashing algorithm for a given botnet.

Our discussion has primarily concerned Chord. We did not simulate an attack on Mainline DHT [6], the Kademlia [23] based DHT used as the backend of BitTorrent, because 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 $2^{160} - 1$. Since the choice of node ID in MLDHT is left to the client, there is no impediment to a Sybil attack. Research has examined MLDHT's vulnerability to Sybil attacks [9] and detected entities performing the attack on MLDHT.

A hash function over a unique identifier might seem to provide a level of protection against a Sybil attack. Our research demonstrates that using a SHA1 hash of the IP address and port number is no defense against a Sybil attack. We show that the adversary can easily mash Sybil into desirable locations.

However, the mashing process can be used in non-malicious purposes to benefit a DHT. The SHA1 hash has an evenly distributed output. Most sets of meaningful keys will not have a uniform distribution [24] Some nodes will be responsible for larger regions than others and therefore will be responsible for a larger portion of the data. If a node can detect when a peer is overloaded, the node can inject a virtual node into the region to shoulder some of the load. The load could be defined by the size of the region or by the volume of traffic.

A network implementing this load-balancing strategy would be self-adaptive. Nodes in this type of self-adaptive network would have a limited number of virtual nodes to mash. This limit would protect nodes from becoming overloaded themselves and ensure network stability.

We have shown that node injection via mashing is both practical and feasible for a Sybil attack on a hash-based P2P network. In addition, we have shown that mashing can be used for both malicious and beneficial purposes.

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] 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.

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

[5] 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.

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

[7] 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.

[8] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. Al Hamra, and L. Garces-Erice, "Dissecting bittorrent: Five months in a torrent's lifetime," in *Passive and Active Network Measurement*. Springer, 2004, pp. 1–11.

[9] L. Wang and J. Kangasharju, "Real-world sybil attacks in bittorrent mainline DHT," in *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012, pp. 826–832.

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

[11] H. Rowaihy, W. Enck, P. McDaniel, and T. La Porta, "Limiting sybil attacks in structured P2P networks," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, May 2007, pp. 2596–2600.

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

[13] D. Eastlake and P. Jones, "Us secure hash algorithm 1 (SHA1)," 2001.

[14] 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.

[15] 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.

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

[17] 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.

[18] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 299–314, 2002.

[19] 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.

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

[21] 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

[22] A. White, A. Tickle, and A. Clark, "Overcoming reputation and proof-of-work systems in botnets," in *Network and System Security (NSS), 2010 4th International Conference on*. IEEE, 2010, pp. 120–127.

[23] 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.

[24] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.