Speedy: a Sybil-resistant DHT implementation*

Quanquan Liu, Qian Long, Pratiksha Thaker, Wenting Zheng

May 14, 2014

1 Introduction

Distributed hash tables (DHTs) are common services that leverage peer-to-peer (P2P) communication to provide a distributed key value service across nodes in a network. The tasks of nodes in a distributed hash table include storing key/value pairs and providing values for lookup requests or rerouting requests to other nodes. A public DHT should be able to handle nodes joining and leaving arbitrarily, and provide efficient lookups, key inserts, and value modifications.

A classic vulnerability in such a network is a *Sybil attack*, in which an adversary is able to introduce malicious nodes, which do not adhere to specified protocols, into a network. Specifically, by introducing a large number of these malicious nodes with fake identities, the adversary is able to disrupt system operation by responding to queries with incorrect information; maintaining an incorrect routing table, delaying queries; strategically placing nodes in an attempt to construct a snapshot of the entire network's data, possibly subverting security properties; or a number of other potential attacks.

In this project, we designed and implemented a DHT to mitigate some Sybil attacks and provide data consistency, while minimizing the number of hops for a lookup, given certain assumptions about the underlying P2P network. Our implementation is based on the *Whānau* routing protocol [LLK10], but includes a layer of indirection to allow key operations to be coordinated using Paxos, as well as basic key signing to provide data integrity.

2 Related work

Past projects including SybilGuard [Yu+06], SybilLimit [Yu+08], and Whānau [LLK10] have examined this problem in the context of fast-mixing social networks, with specific restrictions on the structure of the resulting graph. This work primarily addresses attacks in which malicious nodes delay or re-route queries; data integrity problems are assumed to be handled by the client via

*https://www.github.com/wzheng/speedy

signatures or another verification measure.

As in previous work, we assume that the underlying graph is a social network. As shown in [LLK10], this allows us to assume the honest region is fast-mixing and, assuming that creation of a link between honest and Sybil nodes is difficult, there exists a sparse cut between the honest region and the Sybil region.

3 Design

We separate our design into two ”layers”: the routing layer, based on [LLK10], and the data consistency layer, composed of multiple clusters of servers which coordinate key operations using Paxos.

3.1 Routing protocol

For the routing part, we use the original Whānau protocol as presented in [LLK10]. In this ”layer”, each key is matched with list of nodes that comprises of the Paxos cluster responsible for that key. In our design, we call these key/value pairs, where value is a list of server names. The true value of the key is replicated across the nodes in the paxos cluster, which is explained in the next section. The routing protocol is responsible for retrieving the Paxos cluster assigned to a key, which we will refer to as the ”value” for the remainder of this subsection.

The Whānau protocol relies heavily on random walks and random sampling to build up its routing tables. Because of the sparse cut assumption in our network model and the fast mixing properties of the honest node region, random walks from any node are likely to stay within the honest region unless the node is part of the sybil edges.

Each node maintains a local key/value store for the keys that are inserted from this node. Each key in the local kv store will get assigned a Paxos cluster value as explained in secton 3.2. After this assignment, each node creates an intermediate key/value store, called db, that contains r_d samples of key/value pairs collected from random walks on the network. Furthermore, the Whānau protocol also creates layers of routing tables, which provably mitigates Sybil attacks [LLK10] such as clustering attacks.

In each layer, there are three routing tables created: id, fingers, successors. The id for a layer is used to identify the node; it comes from the keyspace. This is chosen randomly from the db for the first layer and chosen randomly from the previous layer's fingers in subsequent layers. The fingers contains r_f (id, server) pairs that act as pointers to other nodes in the network for routing. The fingers are chosen from taking random walks on network. The successors contains records that follow the id in the keyspace. The successors are built up by taking r_s random walks and collecting the records close to the current layer's id from each of the random walks.

The routing table setup outlined above builds a static routing table on each node in the network with sybil attack resistant properties explained in [LLK10].

In order to account for new inserts into the DHT, this routing table setup must happen periodically so that the DHT can be up-to-date.

3.2 Replication and consistency

The Whānau protocol naturally results in a large amount of replication, but does not specify consistency guarantees or a protocol for key inserts and value updates. *Speedy* trades this large amount of replication for stronger consistency guarantees by creating Paxos clusters within the DHT which store the values for each key, rather than storing them in the Whānau routing tables. This gives us two advantages: first, very large data values will not be excessively replicated, which would otherwise use up large amounts of storage space; second, concurrent updates and node failures are resolved through the master clusters, which will be explained in section 3.6.

The Paxos cluster for each node is chosen by using random walks. Given user defined $PaxosSize$, *Speedy* uses $PaxosSize - 1$ random walks to construct the Paxos cluster. Each server's paxos cluster is constructed during the $SETUP$ stage. Each server, once it enters the set up stage, will perform random walks in order to find $PaxosSize - 1$ servers. The server then starts a two phase commit with those nodes in order to form the Paxos cluster.

3.3 Setup

Because the routing tables created in the Whanau Protocol are static, the setup steps must happen periodically to account for key churn and node churn in the network.

Setup works as follows for one node

1. For every key in a node's queue, ask the Master cluster to assign a Paxos cluster for it. This populates the local kvstore.

2. Build up routing tables using the Whanau Protocol

3.4 Lookup

The following outlines a lookup for key k.

1. Look in local kv store to find the Paxos cluster for k, if not found, follow the steps below to look for it.

2. Randomly choose a layer, and randomly choose a finger, n_f in that layer close to the key.

3. Look in the successors of n_f.

4. Repeat steps 2 and 3 until found or time out. The high level of replication in the protocol ensures that it will usually take 1 hop to find the Paxos cluster of a key.

5. Now that we have the Paxos cluster of k, we use the Paxos protocol within the nodes in the cluster to agree on the true value to return to the client.

3.5 Dynamic updates

Storing true values in Paxos clusters naturally allows support for dynamic value updates: a Put operation is simply routed to the appropriate cluster, and the servers involved come to a consensus on the ordering of the operation in their logs. This also addresses the potential concern of two different clients attempting to update the value of the same key concurrently. In this way, clients need not wait for a Setup round to see the effect of their value operations.

3.6 Dynamic inserts

Inserting new keys is slightly more involved than value updates, as the construction of routing tables in Whānau depends on knowledge of the existing keys in the network. To address this, we assign a certain set of servers to be ”master” servers, which also coordinate operations using Paxos. These master servers are pre-determined before the very first setup, and therefore do not change in the later setup stages. All of the servers in the DHT know of these master servers. When an insert happens, it becomes a pending insert. This has to happen because Whānau cannot handle new inserts until a new setup is run. Therefore, each server processes these new inserts by sending that information to a random master node. The master node is in charge of initiating a paxos call to the other master servers and decide on which server to send the pending insert.

After the master servers agree on a key-server mapping, all subsequent pending inserts will always be sent to that server. This Paxos operation takes care of concurrent inserts to the same key by multiple servers. Note that one master server could solve the problem, but we have multiple master servers in order to make *Speedy* more tolerant. Also note that masters are only used for insert operations, and only have limited powers.

3.7 Fault tolerance

Whānau provides fault tolerance through large amounts of replication. Our implementation maintains this at the routing layer, but the fault tolerance properties at the data layer are somewhat less obvious; we informally discuss them here.

We use $O(\log n)$ replicas in each Paxos cluster, each selected by a random walk of length $O(\log n)$, in order to ensure that each cluster has a constant number of honest nodes, where n is the total number of honest nodes in the DHT [LL10]. Each Paxos cluster is responsible for one key; the clusters are constructed and the keys assigned at Setup time. If a node fails, it can no longer participate in consensus in any Paxos cluster it is a part of. However, even correlated node failures will not result in correlated Paxos failures; since clusters are created using random walks, the effects of node failures will be ”distributed” over many Paxos clusters.

Fault tolerance in the routing layer (in particular, the effects of Sybil nodes which may re-route queries) is the same as described in [LLK10].

4 Performance

4.1 Setup: systolic mixing

A bottleneck in our implementation was the overhead in sending out many random walks throughout the Setup process. To mitigate this, we implemented the ”systolic mixing” process suggested in [LL10], in which random walks are precomputed by flooding the network with node addresses and shuffling them at each time step. This reduced the time for Setup with large numbers of servers (over 100) to less than 10% of the time it took using recursive random walks.

4.2 Whanau Lookup hop count

The Whanau Protocol guarantees a one hop lookup with high probability [LLK10]. Recall in *Speedy*, the whanau lookup is the first part of the protocol where we look up the Paxos cluster responsible for a key. One hop means that we do steps 2 and 3 of the Lookup Protocol (3.4) from the node issuing the request. If that fails, we hop to another node (via random walk) and try again.

We tested this claim experimentally using a 100 node network that is well connected and 5 pre-inserted key/value(Paxos cluster list) pairs per node, giving a total of 500 keys in the network.

After one setup phase, we performed the Whanau Lookup from every node on every key and count the number of hops in each Whanau lookup. This is $100 \cdot 500$ lookups in total. Results from 5 trials show that an average of .5 percent of all the Whanau Lookups required more than 1 hop. This shows good empirical evidence for the 1 hop lookup claim.

Furthermore, we test the claim on different network structures to observe the effect. We construct network graphs where each node has a probability P of being neighbors with any other node in the network. P is referred to as the edge probability in the table below. Once again, we use a 100 node network of honest nodes with 500 total key/value pairs. We see that the one-hop lookup claim holds up to small numbers of edges, at which point the number of lookups requiring more than one hop increases slightly but is still quite low.

Edge Prob	Fraction of Successful Lookups needed > 1 Hop	Total Lookup Success
1.0	0.004	1.0
0.9	0.006	0.99
0.8	0.003	1.0
0.7	0.004	1.0
0.6	0.0019	1.0
0.5	0.005	1.0
0.4	0.007	1.0
0.3	0.0037	1.0
0.2	0.004	.998
0.1	.0094	.998

5 Security Analysis

5.1 Threat Model

The ”Sybil Proof” security of *Speedy* is based upon the assumption that there is a sparse cut between the fast mixing honest region and the Sybil region of the underlying p2p network. There are no further assumptions on the graph structure of the Sybil region. The routing tables and Paxos clusters are all built up from taking random walks on the network. The main intuition is that a random walk from an honest region is unlikely to end up in the Sybil region because of the sparse cut, unless, of course that honest node is on the edge of the graph.

Besides the sparse cut assumption, Sybil nodes are free to behave however they wish. This can include dropping routing packets, flooding the neighbors, clustering attacks, modifying data values from within the DHT, blatantly returning wrong values, etc.

The main security guarantee *Speedy* provides is that Lookups return correct values most of the time even in the presence of a majority of Sybil nodes. This is based on the security guarantees of Whanau shown in [LLK10]. Though we extended the protocol with a layer of indirection for providing data consistency, we believe the security guarantees should still be valid.

The security analysis of *Speedy* can be broken down into the security of the routing layer (looking up the Paxos cluster for a particular key) and the security of the data layer (looking up the true value from the Paxos cluster). We explain the main theoretical guarantees and show experimental results.

5.2 Whanau Routing Layer Security

The main security guarantee in the routing layer of the protocol is that Whanau Lookups are successful most of the time in the presence of a large number of Sybil nodes. We tested this on multiple graphs of different numbers of well-connected graphs (with respect to honest nodes), numbers of Sybil nodes, and number of keys. The first simulation was run on a graph with 50 honest nodes, 50 Sybil nodes, and a maximum of 5 keys per node. The simulation with varying number of attack edges randomly and approximately uniformly dispersed throughout the honest nodes. It tested lookups of all the keys that are in honest nodes from every honest node and computed the fraction of these lookups that are successful. The results of the simulation can be seen in Figure 1.

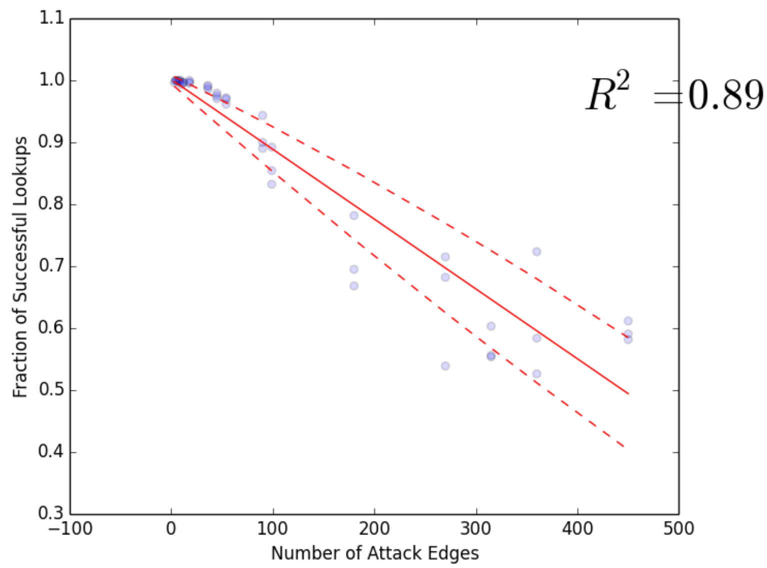

Figure 1: Number of attack edges in a graph with 50 honest nodes, 50 Sybil nodes, and 5 keys per node. Each honest node is connected to every other honest node. Each Sybil node is connected to every other Sybil node. With a certain probability an edge is determined to exist between an honest and Sybil node. The probability varies among different runs of the simulation. The fractions of successful lookups decreases as the number of attack edges increase. However, the protocol does fairly well reaching approximate 50% success for 450 attack edges (approximately 1/3 the number of edges between honest nodes).

Fig. 1 shows that the fraction of successful lookups dropped off slowly reaching 50% success with 450 attack edges, which is 1/3 of the total number of edges between honest nodes. We also varied the number of Sybil nodes present in the graph. For the second simulation, we created 70 honest nodes, 30 Sybil nodes, and 5 keys per node. The results of this simulation can be seen in Fig. 2.

We see that the *number* of Sybil nodes does affect the fraction of successful lookups. This is a surprising result because we expected to see that the fraction of successful lookups depend more on the number of honest nodes and the number of attack edges between honest nodes and Sybil nodes. In other words, a larger number of honest nodes leads to more edges between honest nodes, thus

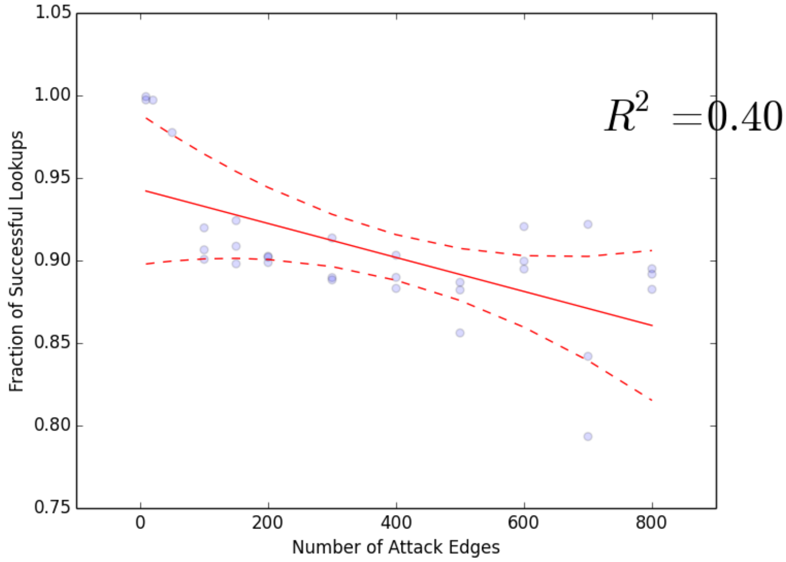

Figure 2: Number of attack edges in a graph with 70 honest nodes, 30 Sybil nodes, and 5 keys per node. The protocol has a fraction of success of approximate 88% for 800 attack edges (again approximately 1/3 the number of edges between honest nodes).

leading to a higher threshold for the number of attack edges that can be tolerated before the number of successful lookups decreases significantly. However, Fig. 2 shows that despite having 800 attack edges (1/3 of the number of honest edges), the fraction of success still hovers around 88%. We are not quite sure why this occurs but one explanation could be that the number of potential Sybil clusters that could occur decreases as the number of Sybil nodes decreases.

Real life social networks might not be made of graphs that are well-connected. For the next set of simulations we performed, we looked at whether varying the edge connectivity probability of honest nodes with other honest nodes will affect fraction of lookup success. We tested the connectivity probability, 0.4, on a graph with 50 Sybil nodes, 50 honest nodes, and 5 keys per node. Fig. 3 shows the fraction of successful lookups when the edge probability between honest nodes is 0.4.

We see that the connectivity of the graph affects the success rate of lookups although not to the extent that we would have predicted. The lookup success rate at fewer than 500 attack edges has a lower success rate than when the edge probability is 1. However, as the attack edges increase, the success rate does decrease rapidly, indicating the robustness of the protocol against Sybil attacks even in graphs that are not well-connected.

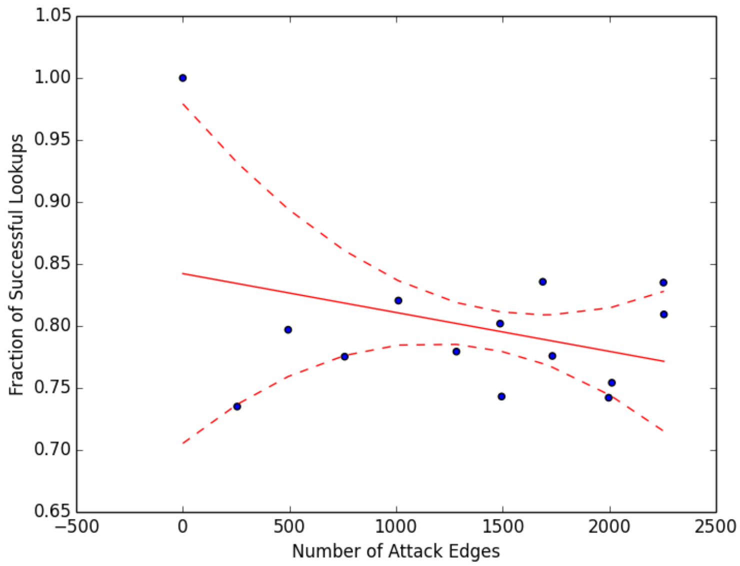

Figure 3: Number of attack edges in a graph with 50 honest nodes, 50 Sybil nodes, and 5 keys per node and honest node edge probability of 0.4. The protocol successful lookup rates remains around 80% even as attack edges increase.

5.2.1 Clustering Attack

One form of clustering attack is when Sybil nodes all choose their ids to cluster around a particular key. This clustering could leak into the finger tables during the routing protocol so that requests for that particular key are likely to be routed to a Sybil node.

One of the main contributions of the Whanau protocol is to prevent against clustering attacks. This is done by the layered construction of routing tables described in section 3.1. The intuition for preventing clustering attacks is that if Sybil nodes choose their ids to cluster around a particular key in one layer, then in the next layer, that clustering will be replicated in the honest node ids. This is because node ids for one layer is chosen randomly from the previous layer's finger tables. Sybil nodes can anticipate this layered routing table construction and choose a particular layer to cluster in, which is why many layers are used. [LLK10] shows that $O(logn)$ layers is the optimal number.

During lookup, one layer is chosen at random, so an honest node is likely to be chosen even in the presence of a clustering attack.

Our results did show show a major difference in lookup success rate when a simulation was performed on a network of 50 honest nodes, 50 Sybil nodes, and 5 keys per node.

5.2.2 Put Security and Data Integrity

Speedy provides some data integrity guarantees using signing. In a public DHT, Sybil nodes could constantly update new values for existing keys. There is no good way to know whether a particular put operation inserted an "incorrect" value because we cannot tell whether a node is malicious or not. We can only provide a guarantee that a particular value will always be tracked back to the node that executed that put operation. This information could then be used in a reputation system for this public DHT. The reputation system is outside the scope of this project, so we will only provide an explanation for a method for ensuring data integrity.

In our data integrity model, each node in the network has a secret key and public key. We refer to the originator as the node who performs the put request. When a `(key, true value)` is inserted, the originator signs a concatenation of `(true value || originator ip address || originator public key)` with its secret key. The originator then concatenates the true value, originator ip address, signature, and his public together as the new true value of the key. If a node does not include all of the valid information, then its put request will be rejected.

During a lookup, after the true value is retrieved from a Paxos cluster, the receiving node can verify the integrity of the data by verifying the signature with the attached public key.

Note that this scheme does not provide any confidentiality because the original value is always stored in plain text. Therefore, a malicious node can always take someone else's value and sign with its own secret key. Information requiring confidentiality should be encrypted using public key encryption, and it should be taken care of on the client side.

Sybil nodes could potentially try to imitate an honest node by signing with its secret key, but then say that the information came from an honest node's IP. This problem can be solved by requiring the originator IP address to be included in the true value so that any node can ask the originator for its public key to check with the one that is attached.

Of course, sybil nodes can still coordinate with each other to verify each others public keys. In this case, we can build a reputation system to keep track of suspicious nodes. For example, if the data received is a virus, then that ip address/public key pair will get a low ranking in the reputation system. More research is needed in this area, but we believe that our scheme is a good start for providing some basic data integrity in *Speedy*.

5.2.3 Lookup Security

Lookup security of the data layer is partially guaranteed by the *Whānau* layer. Because each key resides in both the *Whānau* layer and the Paxos layer, we know that every single key that is found in the *Whānau* layer *must* be found in the data layer as well. If a Sybil node decides to return an error saying that no key was found, we know that is Sybil behavior because we have already found the key in the *Whānau* layer. In this case, the server can retry at another node in the cluster.

If a Sybil node decides to return an incorrect value instead of the correct value, then the client should accept that value. The client could try to do another lookup to make sure that it was indeed the correct value that was returned, but this extra work is unnecessary because a Sybil node can always legitly update an existing with that incorrect value. We believe that this problem is taken care of by the digital signature.

5.2.4 Paxos Security

While the security on the Paxos protocol level is out of the scope of this project, *Speedy*'s design does try to minimize the number of Sybil nodes within a Paxos cluster. A Paxos cluster for a particular key is formed by taking some random walks from the node which a key is inserted/updated. If that node does not lie on an attack edge with the Sybil region, then there is a high probability all of the nodes contacted will stay within the honest region due to the sparse cut assumption.

As long as a majority of the nodes in a Paxos cluster is honest, then the agreement part of the protocol will work correctly because this is analagous to a majority of nodes not failing.

We tested the robustness of the random walk by experimentally showing the relationship between the number of attack edges and the percent of Paxos clusters that end up with a Sybil node majority after the setup phase. As proven in [**Whanau**], the security of the Whanau protocol can support up to $O(n/logn)$ attack edges, where n is the number of honest nodes in the network. We relax this assumption and vary the number of edges (attack edges) between the two regions and observe the effect of the number of Sybil nodes that end up in the Paxos clusters.

In the experiment, we construct a 100 node network with 50 honest and 50 sybil nodes, where each region is well connected within the partition. We insert 5 keys into each node and run the entire setup protocol. We count the fraction of Sybil nodes in the Paxos clusters for all the keys that were inserted by honest nodes. In the experiment, n is 50, so the $O(n/logn)$ upper bound is around 12 attack edges. As seen in Figure 3, the percentage of Sybil majority Paxos clusters is near zero for 12 attack edges and increases linearly with the number of attack edges.

6 Future work

We have implemented the Whanau protocol as it is described in [LL10], including a layer of indirection to provide data consistency as well as a layer of key signing to provide data integrity. However, our implementation requires a set of master nodes, which could turn into a bottleneck and increase latency significantly; future work might involve distributing the work of the master nodes

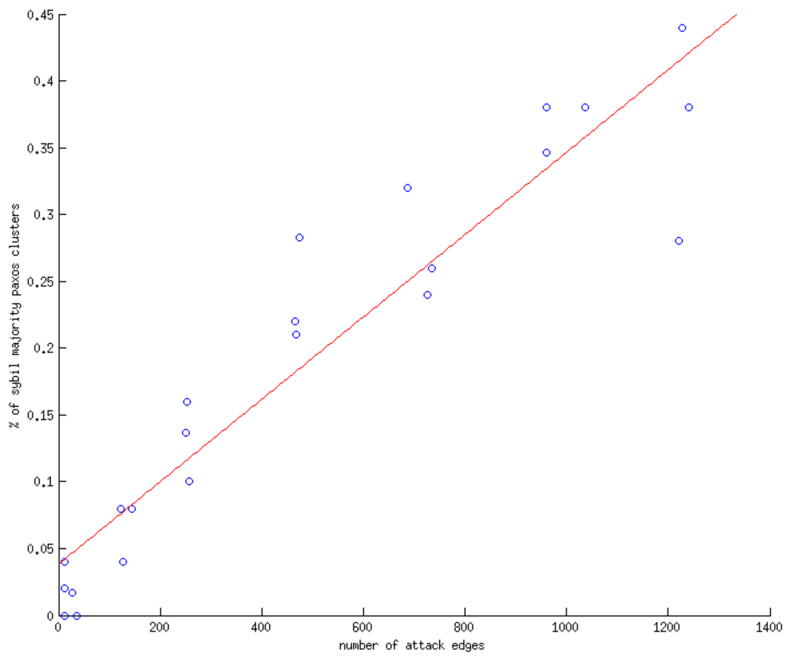

Figure 4: Number of attack edges in a graph with 50 honest nodes, 50 Sybil nodes, and 5 keys per node. The Paxos cluster size was set to 3.

further.

We would also like to do correctness and integrity testing on a much larger scale to determine how many node failures our system can tolerate. Furthermore, we have not yet tested our system with significant changes to the graph structure in between Setup phases, although we believe we have increased its robustness to such changes by adding Paxos logging so that nodes leaving the network nevertheless allow the remaining nodes to agree on operations.

References

[1] Frank Li, Prateek Mittal, Matthew Caesar, and Nikita Borisov. "SybilControl: practical sybil defense with computational puzzles". In: *Proceedings of the seventh ACM workshop on scalable trusted computing*. ACM. 2012, pp. 67–78.

[2] Chris Lesniewski-Laas and M Frans Kaashoek. "Whanau: A sybil-proof distributed hash table". In: *7th USENIX Symposium on Network Design and Implementation*. 2010, pp. 3–17.

[3] Christopher T Lesniewski-Laas. "Design and Applications of a Secure and Decentralized Distributed Hash Table". PhD thesis. Massachusetts Institute of Technology, 2010.

[4] Haifeng Yu, Phillip B Gibbons, Michael Kaminsky, and Feng Xiao. "Sybil-limit: A near-optimal social network defense against sybil attacks". In: *IEEE Symposium on Security and Privacy, 2008.* IEEE. 2008, pp. 3–17.

[5] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham Flaxman. "Sybilguard: defending against sybil attacks via social networks". In: *ACM SIGCOMM Computer Communication Review* 36.4 (2006), pp. 267–278.