A Peersim implementation of Whānau

A Sybil-proof Distributed Hash Table

Giovanni De Toni
Dept. of Information Engineering and Computer Science
University of Trento
Trento, Italy
giovanni.detoni@studenti.unitn.it

Andrea Zampieri
Dept. of Information Engineering and Computer Science
University of Trento
Trento, Italy
andrea.zampieri@studenti.unitn.it

***Abstract*—Whanau is a novel routing protocol for distributed hash tables (DHTs) that is efficient and robust against Sybil attacks. We replicated the original paper providing one of the first Java-based implementations of the protocol. Several experiments were conducted in order to assess the real efficiency of the protocol on large-scale systems by using the Peersim framework. We tested its resilience under the presence of clustering attacks.**

***Index Terms*—Distributed Systems, Distributed Hash Tables, Sybil Attack, Peersim**

I. Introduction

Decentralized systems are vulnerable to **Sybil attacks** in which an external attacker creates multiple fake identities in order to disrupt the normal behaviour of the network. This is an important issue especially in a system which needs to route messages between hosts, like the Distributed Hash Tables (DHT), since an attacker could prevent honest nodes to communicate. Obviously, nodes cannot distinguish between an honest and a Sybil node, otherwise, the would simply reject the malicious entities.

Several solutions are available in order to strengthen the current protocols against this type of attacks: a central authority could provide certificates for discerning between honest and malicious nodes or unstructured protocols could be used which works by flooding or gossip. However, the cost of keeping universal identities could be too high. Moreover, protocols which work by flooding would require a linear time in order to find a specific key.

Whanau [1] is a novel protocol devised by Chris Lesniewski-Laas and M. Frans Kaashoek which provides an efficient solution to implement a DHT which is resistant to Sybil attacks. It has sub-linear run time and space usage. In order to reach this goal, Whanau exploits the properties of social networks such to build its network and it achieves these performances by combining the idea of **random walks** and a new way to construct the node's IDs called **layered identifiers**. Given k keys and given the size of the network be n, the procedure build routing tables with $O(\sqrt{kn}\log(n))$ per node. Using Whanau's lookup tables, it is possible to show that a lookup provably takes $O(1)$ time. Moreover, Whanau's security does not impact the performance of the system. It scales similarly to other one-hop DHTs which provides no security.

In this work, we provided one of the first Java open-source implementations of Whanau by employing the Peersim [2] framework. We also conducted several experiments on synthetic networks such to replicate and validate the results obtained in the original paper.

The following report is structured as follow. Section II provides the scenarios to which Whanau was applied and tested. Section III shows the basic assumption on which Whanau's relies on in order to work correctly. Section IV and Section V explains the details of the Whanau protocol such as how it builds its routing tables and how the lookup procedure work. Section VI details how Whanau was implemented into the Peersim framework. Section VII shows the results of the experiment we did in order to assess Whanau's function. Finally, in Section VIII we summarize our findings and we reason about eventual future works.

II. Scenario

We applied Whanau to a simple Instant Messaging (IM) P2P application. Whanau provides the routing service for the clients. Each user is identified by a pair $(public\_key, ip\_address)$ which is saved inside the DHT. To send an IM to a specific user, we first need to look up its IP address by searching for his public key inside the DHT. Once the key is found, the client can connect directly to its target since it will know the target's IP address. Moreover, it would be possible to further complicate this application by including a validation procedure on the IP found, by checking if it was signed by the real user. In our example, each node stores just one pair $(public\_key, ip\_address)$ but, in general, there may be $k > 1$ per node.

III. Whanau's Assumptions

Whanau relies on certain features of social networks in order to work properly. In this section, we will explain the two main assumptions that Whanau's make and why they are satisfied in a real-world scenario.

A. Fast-mixing Social Networks

A **social network** is an undirected graph in which a node knows its immediate neighbours. An **attack edge** is a connection between an honest node and a Sybil one. An **honest edge** is a connection between two honest nodes. Conceptually, we can divide a social network into two parts: a **Sybil region**, which is made by all the Sybil nodes/identities, and an **honest region**, which is comprised by all the honest nodes and their honest edges.

The key assumption is that the number of attack edges g is relatively small with respect to the honest nodes n ($g \ll n$). This means that it exists a **sparse cut** between the honest region and the Sybil region (Figure 1 shows an example of sparse cut). This assumption can be justified because, in order to add a new attack edge, a Sybil node must do a social-engineering effort: an attacker must convince an honest person to create a social link to one of the Sybil identities.

Given these assumptions, the honest region forms an **expander graph**. Expander graphs are **fast mixing**, which means that the ending node of a random walk is a random node in the network, with a probability distribution proportional to the node's degree. We can then define the **mixing time**, w, which is the number of steps required in order to reach this kind of distribution. For a fast mixing graph, $w = O(\log(n))$.

B. Sampling by Random Walks

The **random walk** is the main building block of Whanau and it is the only way the protocol has to use the social network. Given the previous assumptions, an honest node can send around several w-random walks in order to sample nodes from the network (w-random walks means that the length of the path will be equal to the mixing time w). If the social network is fast-mixing and if it has a sparse cut, the resulting set of sampled nodes will contain several honest nodes and few Sybil nodes, since we will have a much higher probability of "avoiding" the Sybil region. Some nodes will be close to many attack edges, those are called **looser nodes** since they were too relaxed when making social connections. They will need to use more lookup messages in order to escape the Sybil region, thus wasting more bandwidth. Fortunately, the number of looser nodes in a network will be small (thanks to the sparse cut assumption) and most nodes will be **winner nodes**.

IV. Whanau's Structure

A. General Working Principle

In the bootstrapping phase, each node performs $r = O(\sqrt{km})$ independent w-random walks on the social network. It then collects a random $(key, value)$ record for each final nodes and stores these nodes and record in a local table.

In order to perform a lookup operation, a node u checks its local record table. If the key is not in the table, u broadcast the target key to the nodes in its local table (which were sampled before). If the table size is sufficiently large, at least one node v_i will have the needed key in its local table.

This procedure will mitigate the presence of Sybil nodes. If the number of attack edges is small, the nodes will mostly sample from the honest region. Thus, the local tables will contain mostly honest nodes/records. Moreover, nodes do not look to each other tables during the setup process. Therefore the attacker influence does not increase over time.

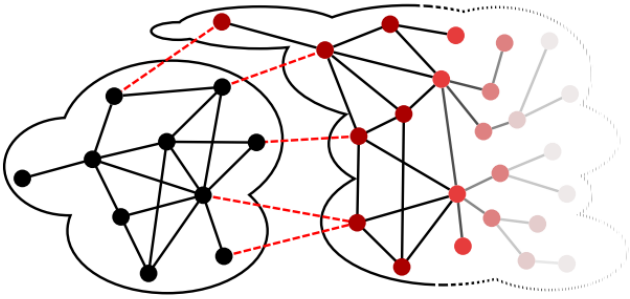

Fig. 1. An example of a sparse cut in a social network. The dashed connections are the attack edges. Black nodes are honest nodes. Red nodes are the Sybil ones.

However, this first draft of the protocol does not take into account the nodes IDs, therefore limiting its efficiency (it is basically broadcasting many messages wasting CPU and bandwidth) and it is still vulnerable to clustering attacks. Therefore, the next sections will be devoted to providing an explanation about the structures used by Whanau in order to be efficient and clustering-resistant.

B. No metric space

Whanau does not embed the keys into a metric space by using a hash function (as some other DHT like Chord [3] or Kademlia [4]) since the adversary could guess-and-check and craft many keys that fall between any two neighbouring honest node keys. Therefore, Whanau does not have any notion of "distance" between keys. However, it does assume a **global circular ordering** on keys, such to be able to determine if one key falls between two other keys (this however still requires defences to clustering attacks). This way nodes will able to build a successor's table.

C. Fingers and Successors

Many structured DHT have tables which contain "far pointers" (fingers) and successors. Whanau follows this pattern. All node have layered IDs which are of the same data type as the key. Moreover, each node has the following tables:

- **Finger Table**: it contains $(O(\sqrt{km}))$ pointers to other nodes which are spaced evenly over the key space;
- **Successor Table**: it contains $(O(\sqrt{km}))$ pointers to honest $(key, value)$ record which immediately follow the node ID.

This tables are built by sampling the nodes by sending out $O(\sqrt{km})$ w-random walks on the network (the successor table is actually built with a more complex sampling procedure).

The most important point is that this table architecture enables fast one-hop lookups. If a node wants to search for a

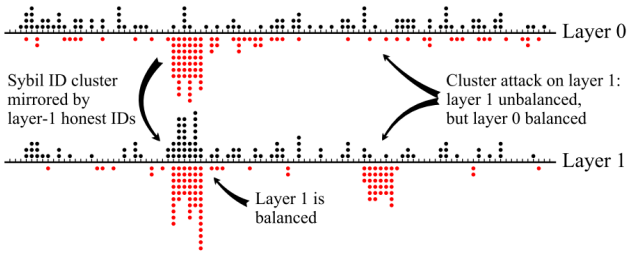

Fig. 2. Black dots represent honest nodes, while red dots represent Sybil nodes. The figures shows how Whanau deal with clustering attacks. Even if the layer-0 IDs are unbalanced, the honest nodes will "cluster" around the target key inside layer-1 IDs, thus enabling other nodes to find that key regardless of the Sybil nodes.

key, it will first look at its own successor table to see if it has already the needed value. If this does not succeed, it will send a query message to a finger node preceding the target key. It is possible to prove that this target node will have with high probability the needed value in its own successor table if it is honest.

D. Layered IDs

Whanau uses layers to defend against clustering attacks. Each node uses a w-random walk to choose a random key as its own layer-0 ID. In order to pick a layer-1 ID, each node picks a random entry from its own layer-0 finger table and uses that node's ID. To pick a layer-2 ID we sample from the layer-1 finger table and so on an so forth for all the $l = O(\log(km))$ layers.

If an attacker were to craft all its identities (IDs) to fall near a target key on layer-0, this will just cause all the other honest nodes to choose an ID in that range as their layer-1 ID (this will propagate also over all the subsequent layers). This increases the presence of honest nodes in the adversary range, thus making more likely to find the target key which is affected by the clustering attack.

V. The Whanau Protocol

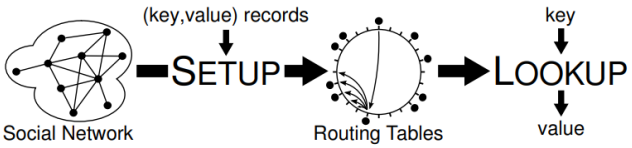

Fig. 3. The Whanau Protocol

The Whanau protocol is made by only two procedures, SETUP and LOOKUP. The former is used to initialize the DHT from the social network and it is run periodically in order to deal with network churn. The latter is used to find the target key. Figure 3 shows a visual overview of the entire process.

A. SETUP *procedure*

Algorithm V.1 Whanau's SETUP procedure

```
1: for each node u do
2:    db(u) ←SAMPLE-RECORDS(u, r_d)
3: end for
4: for i ← 0 to l do
5:    for each node u do
6:       ids(u, i) ← CHOOSE-ID(u, i)
7:       fingers(u, i) ← FINGERS(u, i, r_f)
8:       successors(u, i) ← SUCCESSORS(u, i, r_s)
9:    end for
10: end for
```

The SETUP procedure takes the social network connections, the local key value records and build for each nodes four tables:

- *ids(u,i)*: it contains, for each layer i, the node's ID;
- *fingers(u,i)*: it contains, for each layer i, pointers to the node fingers;
- *successors(u,i)*: it contains, for each layer i, key-value records contained in the successor nodes;
- *db(u)*: a sample of records which is used to construct *successors*;

Pseudocode V.1 shows the steps done by SETUP. The procedure takes also some additional parameters r_d, r_f, r_s which specify the size of the routing tables (how many samples we need to take). A detailed explanation of all the additional methods used can be found on the original paper.

The SETUP procedure ensure that any honest finger f which is "close enough" to the target key y will have $y \in successors(f)$.

SETUP is an iterative procedure that builds up the routing tables layer after layer starting from the first one (*layer, 0*). As previously described, the *layer-0* ID is sampled from the stored keys $db(u)$, on the node (which were sampled by random walk). Fingers (and successors) are values are then sampled by a random walk of the graph. From layer 1 to l the procedure picks:

- *id* : the CHOOSE-FINGER method picks one of the fingers of the previous layers (effectively shuffling the keys based on random walks). This "reshapes" the distribution of the keys to match the one in the previous layer, thus contrasting clustering attack on certain keys (see Fig. 4 for an example of key distribution under a clustering attack);
- *fingers* : the FINGERS methods re-samples new nodes for the current layer with random walks;
- *successors* : the SUCCESSORS method works in the same fashion as for the FINGERS one, but instead sampling keys, it samples successors from the ending nodes of the random walks.

B. LOOKUP *procedure*

The objective of the LOOKUP procedure is to find a finger node f which is honest and such that it has the target key in

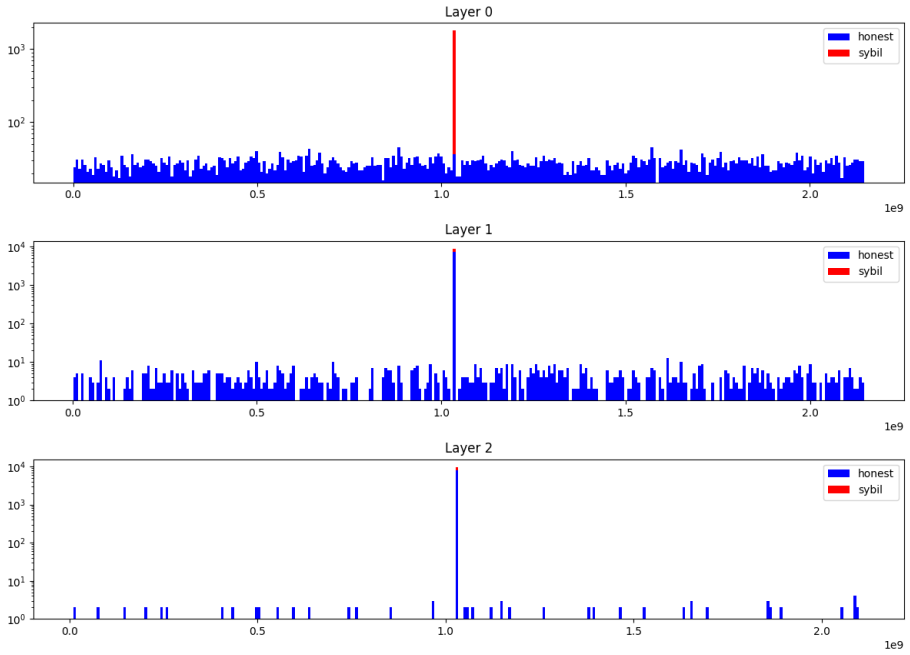

Fig. 4. Example of the distribution of the keys (integers in $[0, 2^{31} - 1]$, grouped in 300 bins) in the *ids* node's tables built on a synthetic network. The y-axis is in logarithmic scale. *Layer*-0 shows the presence of a clustering attack: ≈ 1100 nodes out of the 10^4 are Sybil and chose their $layer$-0 key near the one they want to attack. In subsequent layers ($layers$-1 and $layers$-2) it is clear that with the key sampling procedure, the honest node overcame the high concentration of malicious nodes around the key of interest, thus reducing the effect of the clustering attack.

Algorithm V.2 Whanau's LOOKUP procedure

1: $value \leftarrow null; v \leftarrow u$
2: **repeat**
3: $value \leftarrow \text{TRY}(v, key)$
4: $v \leftarrow \text{RANDOM-WALK}(u)$
5: **until** TRY found a valid $value$, or hit retry limit
6: **return** $value$

its *successor* table. Pseudocode V.2 shows the step executed. LOOKUP first tries using its own finger table, then, if the search has failed, it selects a random delegate and retries the search from there (until we reach a retry limit). The TRY procedure searches the finger table for the closest layer-0 ID x_0 to the target key. It then selects a random layer i and a random layer-i finger f whose ID lies between x_0 and the key. TRY then queries f for the target key.

It can be proved that there is a high probability that the TRY procedure will select an honest finger f which will have $key \in successors(f)$. In conclusion, LOOKUP should always succeed after only a few calls to TRY.

C. Network Churn

Churn refers to a change in the availability of nodes or keys in the system (i.e. new nodes join/leave the network and/or new keys are added/deleted from the DHT).

Node churn is not an issue: the original paper states that temporary unavailability of nodes (e.g. due to network failures such as latencies) is not a problem and it results in small delays in the retrieval of the wanted key(s). These results come from simulations on **PlanetLab** (1353 nodes). However, as the original authors noted, these conclusions cannot be inductively extended to bigger networks due to lack of empirical experiments.

As a matter of fact, once initialised, the structure of the network is rather **static**: new nodes that joined after the setup stage and/or nodes leaving may result in inefficiencies for the protocol. A simple re-run of the WHANAUSETUP protocol fixes this issues. This can be safely done with a small overhead in with a repetition timeout in the order of minutes/hours. The optimal solution to this problem may differ from application to application and different approaches may perform better w.r.t. other in certain contexts.

Key churn is handled in the same way as we deal with addition/deletion of nodes, by repeatedly running the SETUP procedure after certain time intervals. Key churn (and no re-assessment with SETUP) may cause unbalanced *successors* tables and create "hotspots". This can result in network ID distributions on the nodes that do not resemble the actual updated distribution of keys.

VI. IMPLEMENTATION

The protocol's implementation was done in Java 1.8 and by using the Peersim framework such to be able to perform test on large scale networks. Peersim was chosen among other possible solutions (like AKKA [5]) because of its simplicity and scalability. As a matter of fact, we were unable to replicate such large system with other frameworks

because they caused too much overhead. The source code used is available on Github (https://github.com/geektoni/whanau-sybil-proof-DHT) such to encourage peer-review and replicability of our results from other researchers.

A. Configuration Parameters

In order to replicate exactly the experiment done in the original paper, we adopted the same configuration parameters. Table I shows the social network configurations. Moreover, since we wanted to provide an highly customizable architecture, such to be able to test several possible scenarios, we also provided some additional parameters:

- **layers**: the total number of layers l used by each node;
- **database_size**: the number of entries $(key, value)$ stored by each node (the default is 1);
- degree_node: the starting degree of the node used by the synthetic graph generator;
- **target_node**: the target node we want to find using the LOOKUP procedure;

TABLE I
SOCIAL NETWORK PARAMETERS

	Magnitude	Description
n	$n \geq 1$	Number of honest nodes
m	$O(n)$	Number of honest edges
w	$O(\log(n)$	Mixing time of the honest region (length of the random walk)
g	$O(n/w)$	Number of attack edges

B. Package Structure

Following the Peersim programming model, we developed several Java classes which are used by the `peersim.CDSimulator` to run the algorithm (Figure 5 shows the package structure).

1) Protocol: The only protocol we implemented was the `WhanauProtocol`. It contains all the routing tables defined before which are used by Whanau. It also contains several helper methods which are used by the SETUP and SETUP procedures.

2) Controls: We implemented three controls which are used to build the DHT and then to perform the LOOKUP actions:

- `WhanauSetup`: this control is run only once at the beginning of the simulation and it is used to build the initial structure of Whanau. It basically contains the implementation of the SETUP method outlined above;
- `WhanauWireNetwork`: this control is used to generate the social network topology. It can easily build a random topology using the **Barabàsi-Albert method** [6] or it can process an existing social graph (e.g. DBLP, Youtube, etc.). The link protocol used was the already available `IdleProtocol`, since we did not need advanced or custom functions;
- `WhanauLookup`: this control implements the LOOKUP procedure of Whanau. It is used to test the performance of the protocol under several conditions of the network.

3) Observers: Besides using the already available observers provided by Peersim, we implemented two additional classes, `KeyDistributionObserver` and `WhanauObserver`. The former is used to display the distribution of the IDs over the several layers (in order to check visually for clustering attacks). The latter is used to simply print informations about a given node (its internal tables content).

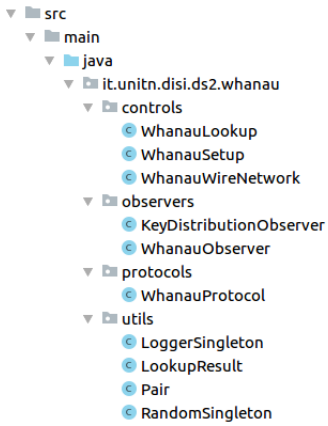

Fig. 5. Java Package Structure of Whanau.

VII. EXPERIMENTS

The experiments were done by considering two synthetic networks which consisted of 10^4 and 10^5 nodes. These two networks were generated using the same method used by the paper's authors, the Barabási-Albert model with preferential attachment, which enabled us to generate graphs which have similar properties as the social networks.

Each node contained only one random $(key, value)$ entry. In a real-world application, the key should be a hash or a public key, here we simply generated a random number $k \in [0, 2^{31} - 1]$ and a random IP address. Since $n \ll 2^{31} - 1$ we assumed that all the elemenent generated were unique in the DHT.

In order to add Sybil nodes, we repeatedly chose a random node in the network and we turned it into a Sybil one. We kept converting nodes until we reached a number of attack edges equal to g.

Moreover, in order to simulate a clustering attack, we chose a random key and we made all Sybil nodes to choose their layer-0 IDs such that: $\forall id \in ids(u, 0)$ where u is Sybil, $id < key$. Ideally, Whanau do not embed a metric space into their keys, therefore, we artificially placed all the layer-0 Sybil's ids such that to make them "close enough" to the target key.

We did several experiments in order to verify the most important assumption of Whanau: (1) **number of lookup messages decreases as the table size increases** (2) **clustering attacks only causes an increase of lookup messages** (3) **using layers makes Whanau resilient to clustering attacks** (4) **the number of lookup messages is constant if there is no attack**.

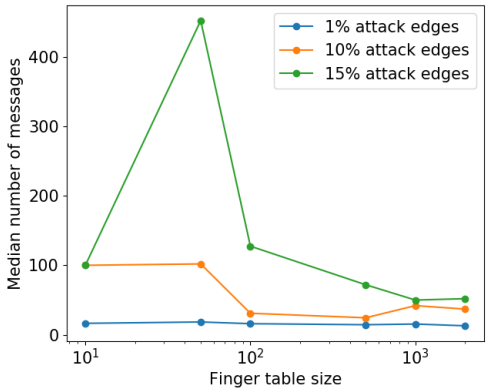

Fig. 6. Median number of messages delivered by LOOKUP calls under different fingers table sizes. The performance were recorded also for different Sybil attack conditions (% of attack edges).

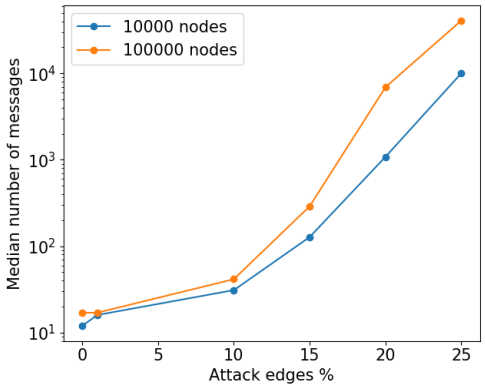

Fig. 7. Median number of messages sent by LOOKUP node, given fixed layer and fixed routing tables size, by varying the percentage of attack edges.

In order to have a statistically meaningful synthesis of the Whanau protocol behaviour, for each configuration (regarding network size, number of layers, etc.) we collected the statistics of 100 LOOKUP calls (whether it failed or not and the number of messages sent around) in order to have a significant sample of data points.

A. LOOKUP *calls vs. Routing rable size*

Figure 6 illustrates the behaviour of the protocol w.r.t. (1) finger table size (x-axis), (2) median number of messages (y-axis) and (3) percentage of attack edges.

As shown, as the finger table grows in size the number of messages tend to decrease: it's clearer in the examples with a higher percentage of attack nodes (10-15%). The behaviour with 1% of attack edges is basically identical to a network without malicious nodes, so it can be seen as a baseline performance for comparison purposes. The collected datapoints are the different combinations of:

- *finger table size*: 10, 50, 100, 500, 1000, 2000
- *attack edges %*: 1%, 10%, 15%

and *network size* has been fixed to 10^4. Moreover, we set to 3 the number of layers. Note that the x-axis has a logarithmic scale.

These results confirm the simple intuition that if a node stores more known neighbours for each layer, it is easier to find the target key, avoiding all the Sybil nodes clustering. On the other hand, even though the fingers increase in an exponential fashion, having routing tables larger than a certain threshold do not bring much improvement to the performances w.r.t the number of messages.

B. LOOKUP *calls vs. Percentage of attack edges*

The second experiment aimed to study the response of the network under different increasing percentages of attack edges by using the default configurations parameter of Whanau.

We conducted two series of tests varying in the network size (respectively 10^4 and 10^5). The percentage of attack edges used were: 0%, 10%, 15%, 20%, 25%. The number of layers is fixed to 3 and the fingers and successors tables were set to the theoretical defined size of ($O(\sqrt{kn}) \approx 100$).

The paper specifies that the protocol has one-hop LOOKUP performances when the number of attack edges is less than a predefined value $g \leq O(\frac{n}{w})$. Figure 7 shows that the median number of messages grows in the order of the thousands when the percentage of attack edges g grows past the threshold the protocol allows to tolerate (e.g. g should be in the order of the 10% of the total nodes for $n = 10^4$). It can be noticed that when $g \leq O(\frac{n}{w})$ the number of messages sent around is comparable to the case without attack edges.

C. LOOKUP *call is* $O(1)$

This experiment empirically shows how the protocol is a one-hop DHT in the absence of malicious identities. The number of layer of 3 and the fingers and successor tables are set as described above in the *Experiment 2*.

The four test networks had respectively 5000, 10^4, 10^5 and $2 \cdot 10^5$ nodes. The boxplots shown in Figure 9 shows that there are no particular differences regarding the number of messages sent for the different network sizes (despite having a different magnitude of nodes).

D. *Layers improve performances under clustering attacks.*

Figure 8 highlights the impact of having multiple layers when dealing with clustering attacks caused by Sybil nodes. The y-axis shows the percentage of the successful queries related to the percentage of edges that are compromised.

The protocol starts to suffer rapidly when there is just 1 layer and the percentage of attack edges grows out of the tolerable threshold of the system: around the 20% mark, performances drops to 73%. By using 3 or more layers, the protocol can cope with the difficulties imposed by the Sybil nodes, keeping the performance around 90% of successful queries.

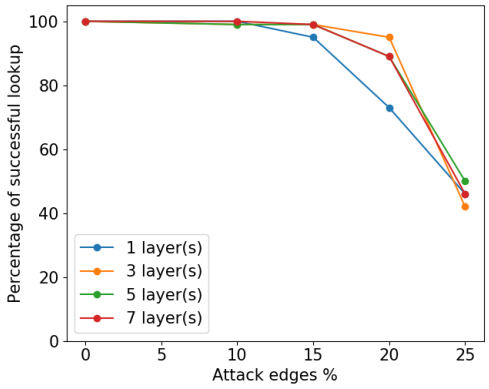

Fig. 8. Percentage of successful LOOKUP calls under different percentages of attack edges. We tested several numbers of layers.

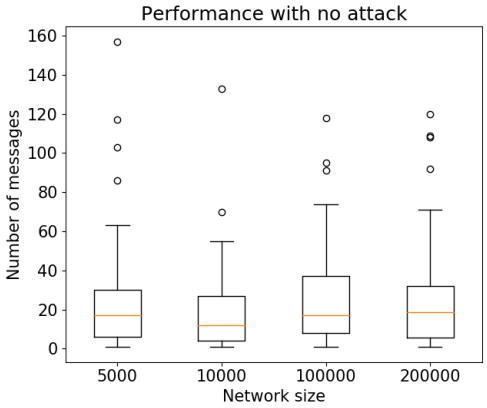

Fig. 9. Total number of messages sent by LOOKUP procedure with a different network size n.

VIII. Conclusions

The Peersim implementation allowed us to test extensively the protocol under several conditions. The subsequent empirical analysis conducted proved Whanau to be resilient against malicious entities that could try to deny/delay access to the resources made available by the nodes. More importantly, it has shown to be resistant to clustering attack. In addition, the protocol does this in a completely distributed manner and without using external entities (e.g. central/third-party authorities).

Thanks to the w-random walk sampling and the layered IDs, it manages to keep a fast one-hop LOOKUP mechanism without compromising the Sybil-resistance property. It also succeeds to achieve this quality by keeping its routing table size to $O(\sqrt{mk}\log(km))$ (sublinear w.r.t. the network size n) entries per node.

Moreover, the majority of the bandwidth used globally by the protocol is utilised by the numerous random walks needed. These random walks just have to send $w = O(\log(n))$ messages each, thus leading to a small system load w.r.t the network size. This holds under the assumption of having few keys per node ($k = O(m)$ total keys, where m is the number of honest edges), but also in the case of $k \gg m$. Whanau would still be able to work using an acceptable share of network resources, without any major drawbacks.

There are however several ways in which we could improve our project and that are left to future works. We were not able to test it on real-world social network datasets, because mainly of experimental constraints (not enough powerful computing architectures). Moreover, the code itself could be vastly improved such to reduce memory usage and speed.

However, despite these limitations, it safe to conclude that Whanau is effective in different cases and scenarios, making it affordable and reliable for systems which need to be Sybil-proof.

IX. Acknowledgements

Figures 1, 2, 3 were taken from the original Whanau paper [1] and the copyrights belongs to the images owners. The pseudocodes explaining the LOOKUP and SETUP procedure were taken from the same paper.
The sections describing the behaviour, the principles, the assumptions and the main ideas behind Whanau are inspired to the original work [1].

References

[1] Christopher Lesniewski-Laas and M Frans Kaashoek. Whanau: A sybil-proof distributed hash table. 2010.
[2] Alberto Montresor and Márk Jelasity. Peersim: A scalable p2p simulator. In *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*, pages 99–100. IEEE, 2009.
[3] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.
[4] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.
[5] Martin Thurau. Akka framework. *University of Lübeck*, 2012.
[6] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.