ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Know-Thy-Neighbour: Approximate Proof-of-Location

Sabrina Kall

School of Computer and Communication Sciences

Decentralized and Distributed Systems lab

Semester Project

June 2019

Responsible
Prof. Bryan Ford
EPFL / DEDIS

Supervisor
Cristina Basescu,
Kelong Cong
EPFL / DEDIS

Contents

1 Introduction

Applications such as the sharing economy would become more efficient with a ‘proof-of-location’, which can provide a weaker, temporary validation to speed up small transactions while stronger proofs are still being computed. Relying on GPS coordinates, however, is not secure, because nodes could provide fake information. What if we simply instruct each participant to specify its location with respect to other participants? For example, Alice says she is 30 ms round-trip time from Bob and 50 ms from Carol, and publishes this information on the blockchain. Alice could still lie, but Bob and Carol can verify Alice’s claim and thus reinforce the information, or, on the contrary, infirm it. Although latencies do not exactly map to geographical distances, very small latencies are, with high probability, between nearby nodes.

Therefore, assuming an accurate and up-to-date exchange of latencies between nodes, nodes with small latencies between them would make an adequate consensus group for two nodes seeking a low-latency validation to perform a fast, low-value transaction.

The main issue for a proof-of-location is the potential for a malicious node to craft inaccurate latencies to give to other nodes, or for a malicious node to tamper with another node’s latencies. Below, we describe a scheme that allows nodes to securely exchange latencies with each other and then, based on these latencies published in the blockchain, identify the malicious nodes whose behavior is most damaging.

2 Background

In the context of the Nyle project, proof-of-location allows us to organize nodes by geographic region. If a client wishes to validate a low-value transaction, they can choose to wait only for the region closest to it geographically, which will have the lowest latency precisely because of its proximity. For more valuable transactions, they can wait for a larger region’s validation, the largest possible region being the entire network across the globe.

This type of validation, ”trust-but-verify” has been shown to work well with other systems, such as Omniledger[1], reducing latency for low-value transactions.

There has already been some work done on location-based proofs, such as Newton [2], which relies on the Vivaldi Virtual Coordinate System and exploits physical laws to allow nodes to determine their confidence level in another potentially malicious node. However, this scheme, while strong and reliable, requires a lot of computational power as each node calculates its level of trust in the coordinates of other nodes, making it inefficient for our purposes. In contrast, know-thy-neighbor can make use of the published chain of latencies to speed things up.

3 Models, Methods and Requirements

The model we use to implement know-thy-neighbor proof-of-location is composed of two parts. The first allows the nodes to exchange latencies in a manner safe from outside tampering, and also lets new nodes join the network. With this part we periodically write latencies to the public chain, giving all the nodes access to each other's latencies. In order to keep the latencies up-to-date, nodes would be required to recompute their latencies every so often to account for changes. We refer to the time frame between two latency updates as an epoch.

In the second part, nodes can then use these computed latencies to calculate which nodes are likely malicious, and consequently blacklist them before using the trusted latencies to validate transactions. In this final part, we assume that no more than a third of the nodes in a network are malicious.

3.1 Computing latencies

In order to join (or rejoin) a network, a node must submit to the chain a block with latencies to other nodes. To do so, it must first collect a given number of latencies from a random subset of nodes assigned to it by the service. Once it has these latencies, it will submit the block to the service, which will attempt to have the block signed using the BLS Collective Signature Protocol. As part of the BLS Signature, there will be an additional check preventing nodes that are too far away from joining the network, which only works accurately on small to mid-range distances (because as explained below, the further the geographic distance, the less a round-trip time correlates to it). If the block is successfully signed, it is added to the chain.

In order to collect the latencies, a simple ping is not sufficient, as this is easily faked either by a malicious new node, or one already present in the network. To make all but the most sophisticated coordinated attacks impossible, we have created a five-message protocol in the following manner.

In this scenario, node A seeks to gain a new latency from node B, who is already in the network.

1. A sends B a message with its public key as an identifier, a nonce and the timestamp A1. The nonce and timestamp are signed with the private key matching the public key to prevent the values from being modified by a Man-In-The-Middle. If A is new to the network, the keys have to be freshly created.

2. When B gets the message, it performs a series of checks: whether A has already started creating a new latency with B (preventing a replay attack by A or a MitM), whether the public key matches the signature (preventing tampering by a MitM), and whether the timestamp is old (preventing a replay attack from an older epoch). If the message passes these checks, B computes a clock skew for A by subtracting the timestamp in the message from the time of reception. B then sends back a message to A containing A's nonce, a new nonce, and B's time of reception timestamp. All these values are also signed.

3. When A gets this message, if checks that the signatures belong to B and that all the signed values correspond to the sent cleartext. If the nonce sent back by B is the same as the one sent in message 1, A knows that this message is a direct reply

to the original message, and that B is not simply spamming it with messages to craft a shorter latency. If the message from B passes all the checks, A can also craft a clock skew for B based on sent timestamp and time of reception. Most importantly, A now has two timestamps, one for when it sent the first message to B, and one for when it got a message back. Using these two timestamps, it can compute its latency to B. It now signs this latency and sends it back to B along with B's nonce.

4. When B gets this message, it performs the usual checks. It also tests whether the clock skew has remained similar between this message and the last. B then computes its own latency to A, and makes sure that this is similar to A's reported latency, accounting for the clock skew. Both these tests prevent node A from cooperating with another node at a different location to fake the latency while the messaging is ongoing. If B is satisfied, it signs A's signed latency with its latest timestamp, signs its own latency, and sends everything back.

5. When A gets this final message from B, it performs a similar clock skew and latency test on B's message content, and if satisfied, signs B's signed latency with its own last timestamp and returns it one last time to B.

In the end, A is holding the following combination:

$$sig\_B[timestamp\_B, sig\_A[latency\_ABA]]$$

The inner signature of the latency prevents a node other than A from claiming the combination as its own latency to B. The outer signature confirms to other nodes that B has seen and approves of this latency to A. The timestamp prevents A from reusing B's signature in later epochs. B has a symmetric combination of values:

$$sig\_A[timestamp\_A, sig\_B[latency\_BAB]].$$

Both nodes can add these combinations to their lists of latencies. Once each node has collected enough latencies, it can submit a new block to the chain. This ensures that when a new node joins the network, it does not go an entire epoch with only one-sided latencies, as nodes already part of the network can update their blocks to include it.

Note that unlike traditional round trip times, the latencies measured above include the time necessary to check the nonces and timestamps, and sign new values to send. These additional calculations add an average of 0.6ms to the round trip time.

3.2 Blacklisting nodes

By the end of this messaging protocol, we have managed to safeguard the network against all but two types of manipulations.

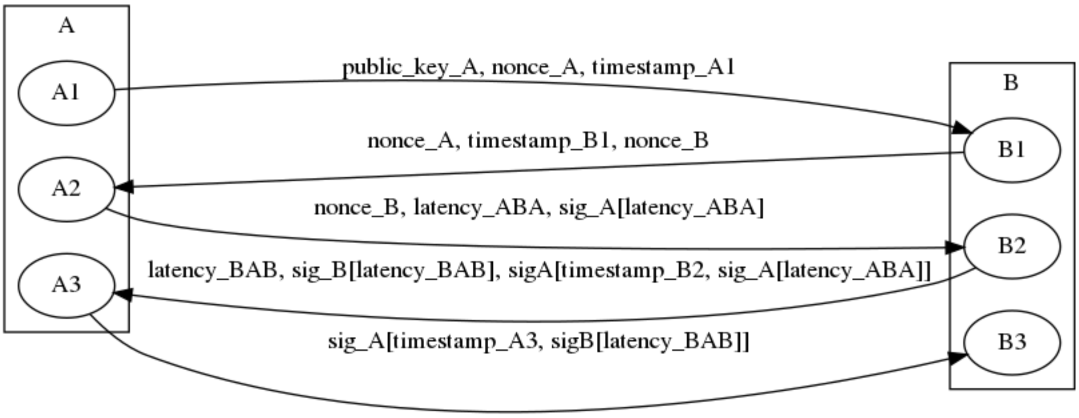

A
A1
A2
A3
B
B1
B2
B3
public_key_A, nonce_A, timestamp_A1
nonce_A, timestamp_B1, nonce_B
nonce_B, latency_ABA, sig_A[latency_ABA]
latency_BAB, sig_B[latency_BAB], sigA[timestamp_B2, sig_A[latency_ABA]]
sig_A[timestamp_A3, sigB[latency_BAB]]

Figure 1: Latency creation messaging protocol (note that all the values in this graph are also sent signed by the sender, but this is omitted for the sake of readability)

Average know-thy-neighbor latency	782.47 μs
Average Ping latency	183.748 μs
Difference	598.722 μs

Average difference between basic round trip time and measured know-thy-neighbor latency [over 100 tests]

Firstly, there remains the issue of a node merely delaying its messages in order to appear further away than it really is.

Secondly, if malicious node A is close to a point P, and a new malicious node B wishes to appear close to P, B can ask A to generate new keys and execute the entire messaging protocol with P on its behalf, then send the generated output to B. Thus with help, a malicious node B can appear closer to a point P (and A) than it really is.

There are multiple scenarios in which a combination of these two manipulations can prove harmful to us. A malicious node may change its latencies to make itself appear in a far-off region. This would then slow down consensus in this region.

Another, more serious attack would be if malicious nodes made themselves appear in multiple regions at once, granting themselves the power of multiple nodes and shifting the proportion of malicious nodes above one third.

Because of these two attacks, we require a way to identify nodes that are manipulating their latencies and exclude them from the transaction process to avoid their malignant influence. We do this by running a blacklisting algorithm.

Our blacklisting algorithm is composed of two parts. Note that this algorithm can be run by each node in the network independently using the public chain of latencies, and will return the same subset of blacklisted nodes regardless.

3.2.1 Triangle Inequality

In the first part, we make use of the triangle inequality theorem, which states that in any given triangle, a side cannot be longer than the sum of the two other sides. Therefore, given the latencies between three nodes as sides of a triangle, we check for each side whether the sum of the other two sides is larger or equal to it. If this is violated, then we can assume one of the nodes is being dishonest about its position, but we cannot be sure which of the three. Therefore, we award each node a strike. Once we have run through all possible triangles, we check whether any node has strictly more strikes than a given threshold. If this occurs, then the node is blacklisted.

It is worth noting that triangle inequality does not always hold over long distances in the internet. However, as we are dealing with small to mid-range distances, this should not be a problem for us.

The threshold is chosen as the maximum number of strikes an honest node can receive if, for example, it is being targeted by the malicious nodes for blacklisting. This way, we prevent malicious nodes from colluding to exclude even a single honest node.

The formula used to calculate the threshold is as follows:

Given N nodes in the network, of which at most a third are malicious, there are (N - 1) nodes (we exclude victim v) that can receive inaccurate latencies that make victim v look suspicious. Hence, we get:

$$\forall N, \theta = (\lfloor N/3 \rfloor) * (N - 1)$$

If a node has more strikes than this threshold, it is blacklisted, and we do not take it into account for transactions.

Although this part of the algorithm has efficient performance, when run alone it is very conservative, and often fails to blacklist any malicious nodes at all in an attempt to protect victims. Therefore, we make use of a more demanding algorithm on top of triangle inequality to attempt to identify more malicious nodes. For this, we exploit the fact that we know which triangles caused strikes, that is to say which nodes "blame" each other.

3.2.2 The Blame Game

At the heart of this enhancement is the fact that the behavior of the malicious nodes will influence the distribution of the strikes to the nodes in distinct ways, allowing us to differentiate between for example malicious nodes that are pretending to be in multiple places at once from malicious nodes who are trying to frame an honest node.

Specifically, the more latencies a malicious node manipulates, the more strikes it risks getting. We therefore can pick out suspected active malicious nodes by checking nodes with more strikes than average (calculated simply by summing the strikes across the network and dividing by the number of nodes).

After we have isolated the nodes whose number of strikes lies above the average, but below the threshold for triangle inequality blacklisting, we must analyze each suspect separately.

For each suspect s, we rerun all the triangles it is part of to extract the nodes that "blame" it by causing strikes to happen.

Once we have these nodes, we can exploit the following fact: if a node n is not malicious and the suspect s is not malicious, then n will not have more than N/3 strikes, one from each malicious node m the pair (suspect s, node n) interacts with. We also know that, as a converse to having at most N/3 malicious nodes, we end up with at least 2*N/3 honest nodes in the network.

Therefore, if the suspect s is not malicious, we should be able to identify at least ((2*N/3) - 1) nodes which do not have more than N/3 strikes.

As a contrapositive, if we cannot find ((2*N/3) - 1) nodes which do not have more than N/3 strikes, then the condition of (suspect s, node n) both being honest is violated, and suspect s must be malicious. We blacklist s.

This enhancement, although computationally more expensive than simple triangle inequality, works quite well, but it is not infallible. Indeed, being able to identify a sub-network of ((2*N/3) - 1) nodes which do not excessively blame the suspect does clear the suspect. A malicious node may well moderate the number and scale of its modifications so that it is not detected, or even suspected in the first place. But this implies that to a certain extend, the malicious node must restrict its manipulations in order to avoid detection, that is to say act more like an honest node.

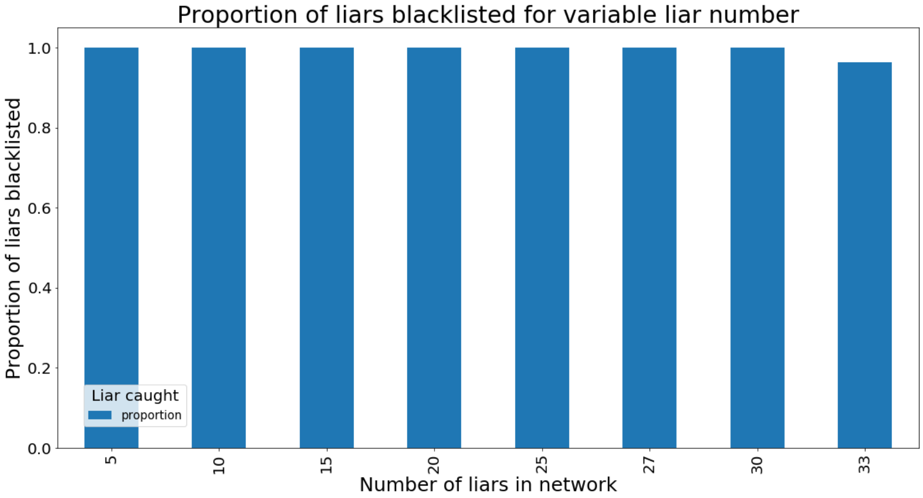


Figure 2: Proportion of malicious nodes blacklisted for increasing number of malicious nodes

4 Results

A series of simulations has allowed us to test the precision of our proof-of-location implementation when confronted with various types of behavior by a subset of malicious nodes.

We simulate a network of N nodes by building a chain with latencies varying between 0.5ms and 1ms when nodes are in the same ”region” and 1.5ms to 2ms for separate, disjoint regions. We then can modify latencies to simulate the behavior of selected malicious nodes.

4.1 Varying number of malicious nodes

We start by examining how the proportion of malicious nodes affects our ability to identify and remove them. In a network of 100 nodes, we fix a set of manipulated latencies between 1ms and 5ms, and give them to a growing subset of randomly selected malicious nodes. We then run our blacklisting algorithm 50 times over 50 different networks with crafted latencies in various locations to average the number of malicious nodes blacklisted for each proportion of malicious nodes. (Note that while this results in a large number of scenarios with rather inept malicious nodes that are easily detected, we occasionally also come across a cleverer set of attackers.)

As can be seen in Figure 2, for most networks, we manage to detect all malicious nodes, only missing a few of them as the number of malicious nodes nears one third. For the following simulations, we therefore concentrate on the worst case scenario, when the number of malicious nodes is maximized.

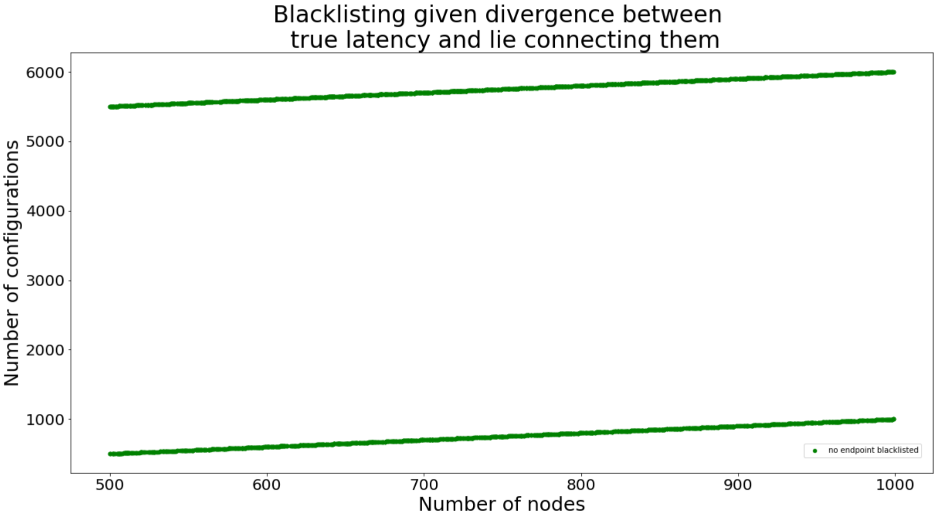


Figure 3: Liars coordinate to move further away from the non-liars and create their own subgraph

4.2 Varying divergence of crafted latencies from true latencies

We go on to analyze how large the difference between a crafted latency, and its real value can get before the malicious node manipulating it is blacklisted. This is important because it allows us to find out within what distance from its true position a node can project itself.

In itself, as long as malicious node A behaves in the exact same manner towards all the nodes, A projecting itself to a single other place will not make or break any consensus, but merely slow it down. The network will stay consistent. If Alice tells both Bob and Carol that she is 20ms further away from them than she really is, there is no issue as long as she keeps up the pretense throughout the epoch. This situation can be seen in Figure 3, where all 33 malicious nodes have announced themselves to be 5ms further away from the 67 honest nodes than they really are, creating a separate subgroup. Since they are isolated and do not affect the honest nodes, there is no need to track and blacklist them.

Problems only start occurring when a malicious node gives different honest nodes contradictory information in order to gain some advantage. Alice might tell Bob she is 2ms away (putting herself in Bob's region), and tell Carol she is 3ms away (putting herself in Carol's region), but if Bob and Carol are 100ms away from each other and not in the same region, we should be able to detect and blacklist Alice. Otherwise she gains influence over too many regions, to the point where she and the other malicious nodes could control the entire network.

We can see the reaction of the blacklisting algorithm to this behavior in Figure 4. When malicious nodes craft latencies in contradictory fashion, an increasing number

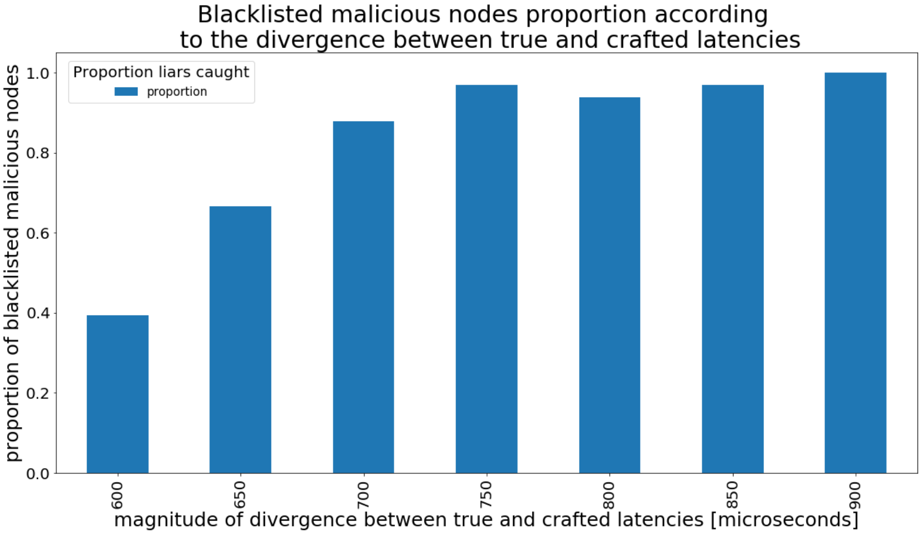


Figure 4: Uncoordinated attack by malicious nodes against honest nodes with growing discrepancy between real and crafted latencies

of them can be blacklisted as the magnitude of the discrepancy between the true and crafted latency increases, until we blacklist the vast majority of them at around 0.85ms discrepancy.

We can examine this reaction in more detail when we graph the distribution of latencies, real and crafted, for each of the nodes they affect, in Figure 5. It is immediately noticeable that the malicious nodes have an average discrepancy between real latencies and crafted ones that is much larger than the honest nodes. The size of this discrepancy implies that malicious nodes will also receive more strikes, making them detectable to the blacklisting algorithm.

4.3 Infiltrating multiple regions

An extremely important aspect which must be handled by the algorithm is the situation in which a node attempts to present itself as being in multiple regions at once. For example, as discussed above, Alice may tell Bob she is 2ms away from him, and tell Carol she is 3ms away from her, and if unchecked, this essentially gives Alice the power of two nodes. In the worst case scenario, a third of the network being malicious, these nodes could all multiply their locations until they and their clones make up more than half of all effective nodes, giving themselves power over the network in its entirety. Therefore, we must limit this kind of behavior as much as possible.

We have modelled this behavior with growing numbers of nodes as networks with various distributions of nodes across regions. Each number of regions and number of nodes implies multiple scenarios for the distribution of nodes into regions. As can be seen in Figure 6, for a single malicious node, the more regions a malicious node attempts

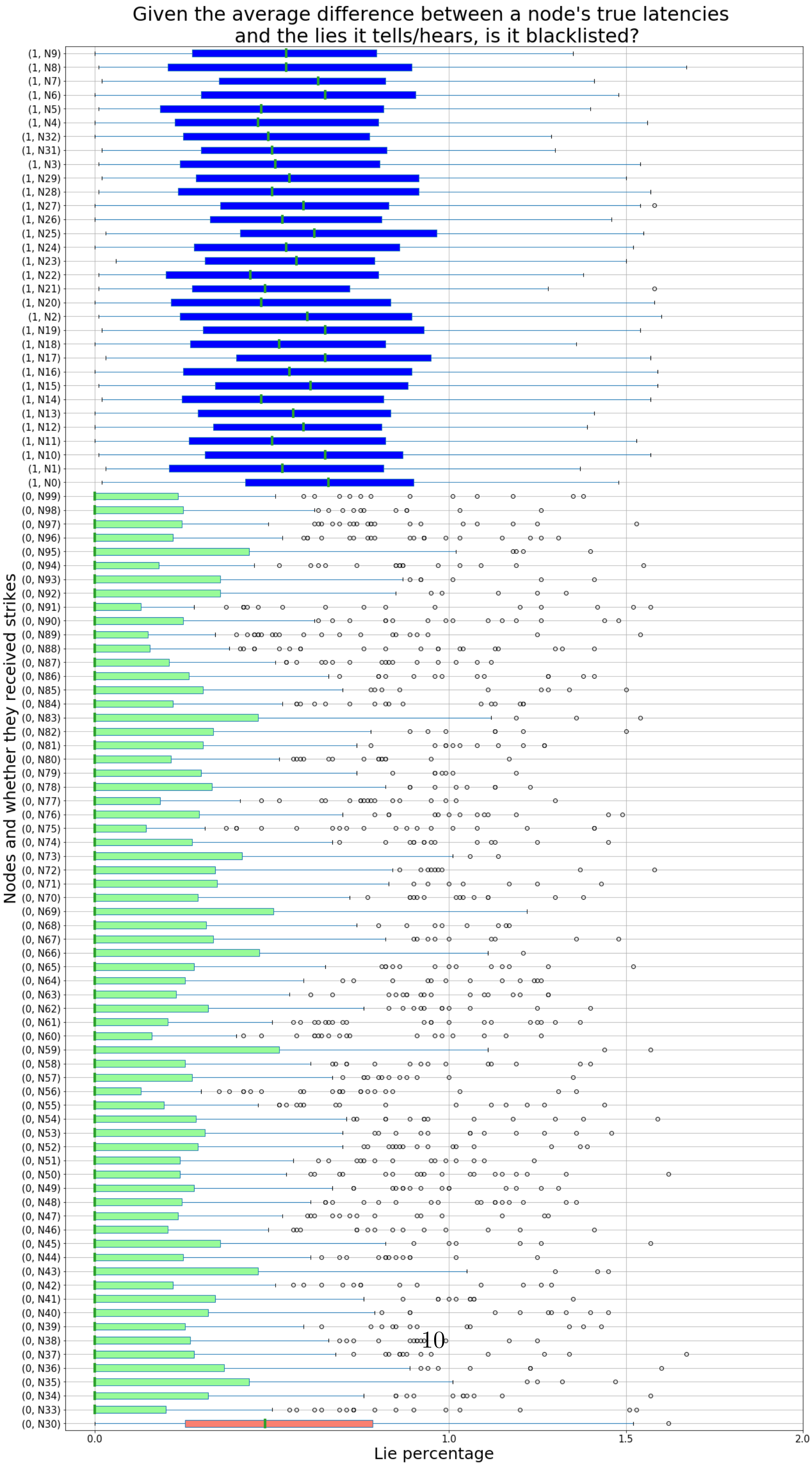
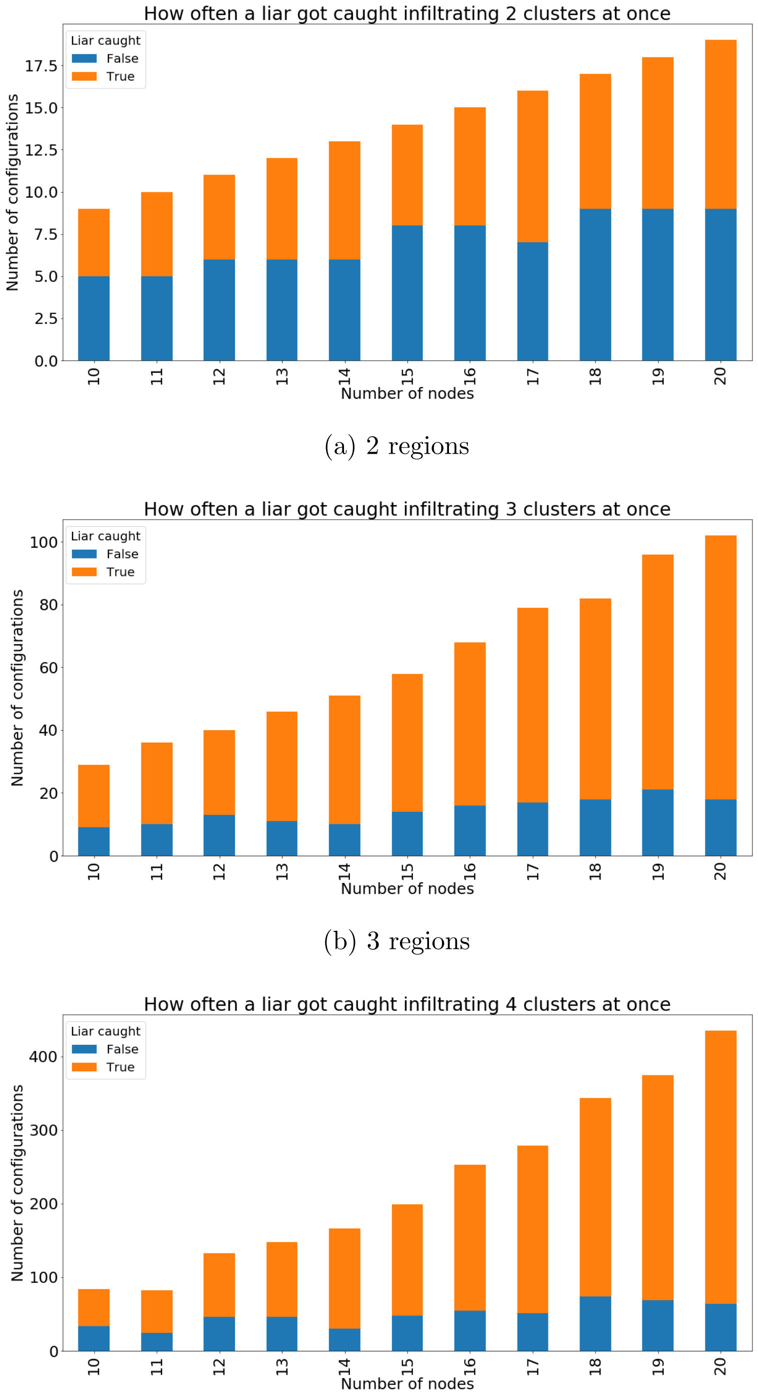


Figure 5: Latency distribution for 100 nodes in the context of an uncoordinated attack with maximal discrepancy 0.85ms [green: honest nodes, blue: blacklisted malicious nodes, red: unblacklisted malicious nodes]

to infiltrate, and the more honest nodes it is confronted with, the more likely that it will get caught.

Concerning the cases in which malicious nodes go undetected, we have analyzed the distribution of nodes across two regions. As can be seen in Figure 7, where the size of one region is divided by the size of the other, blacklisting occurs most easily when the regions have a similar size. If one of the regions becomes too small, it is easier for a malicious node to infiltrate it, as there are not enough nodes to signal its presence to the rest of the network in a way that gets the malicious node blacklisted. However, small regions being compromised is a risk the Nyle project is willing to accept as long as it is limited to said region.

From these graphs, we can also see that by increasing the number of malicious nodes attempting to position themselves in multiple networks, we also increase the number of configurations in which these malicious nodes are blacklisted. This is excellent news, as the feared situation where a large group of malicious nodes attempts to take over the network by duplicating their positions is shown to be extremely difficult to pull off.

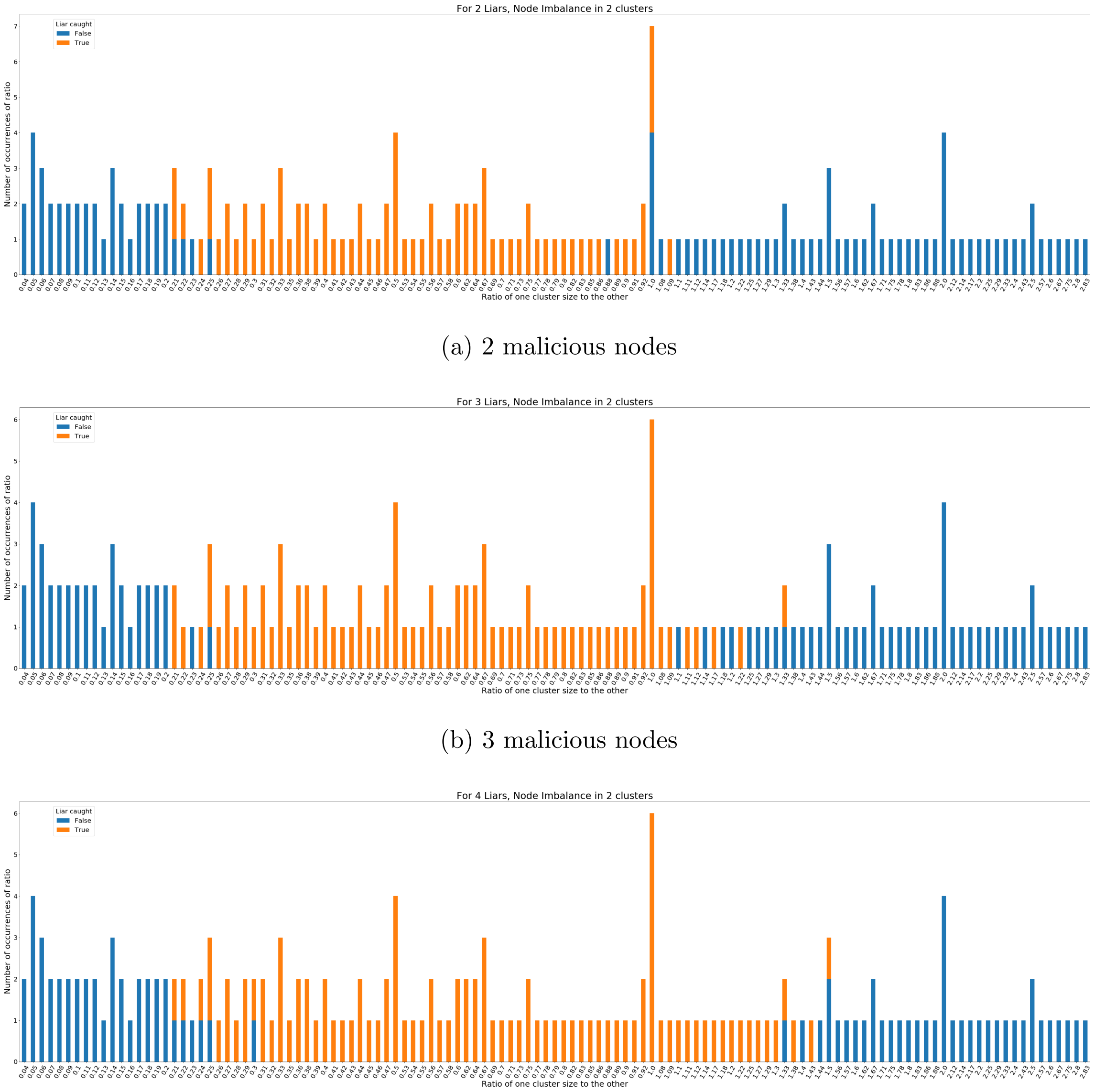


(a) 2 regions

(b) 3 regions

(c) 4 regions

Figure 6: Number of times a malicious node got caught infiltrating variable numbers of regions for growing number of total nodes in a network



(a) 2 malicious nodes

(b) 3 malicious nodes

(c) 4 malicious nodes

Figure 7: For various distributions of N nodes across 2 regions, ratio of nodes between the two regions mapped to whether malicious nodes pretending to be in both gets caught

5 Discussion and Future Work

This implementation has so far permitted us to exchange latencies between nodes in a manner secure to outside attacks, and blacklist malicious nodes with particularly harmful behavior.

The current system works on the assumption that all latencies between all nodes are computed and published in the identity chain. An expansion that could be made would be to reduce the number of known latencies to a subset of nodes for each newcomer, making the system more robust and efficient. This future work would include finding an optimal way to select a subset of nodes for a newcomer such that a malicious node cannot predict which nodes it will be paired with, but that other nodes can verify. It will also be necessary to measure the performance of the blacklisting algorithm with this smaller subset of latencies at its disposal.

6 Summary

In order to exploit proof-of-location, we have implemented a messaging protocol allowing the secure exchange of latencies between validators and blacklisting of malicious nodes with harmful behavior. Under various simulations, this protocol has proven resistant against attacks by up to one third of malicious nodes. This includes attacks in which malicious nodes target a single victim and attempts to infiltrate multiple regions of similar size.

7 Acknowledgements

The author thanks Cristina Basescu and Kelong Cong, who steered and supervised the development of this project.

References

[1] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598, Los Alamitos, CA, USA, may 2018. IEEE Computer Society.

[2] J. Seibert, S. Becker, C. Nita-Rotaru, and R. State. Newton: Securing virtual coordinates by enforcing physical laws. *IEEE/ACM Transactions on Networking*, 22(3):798–811, June 2014.