Building More Decentralized Blockchains Using Secure Virtual Coordinates

Marco Silva
Instituto Superior Técnico, Lisboa, Portugal
marcofsilva@tecnico.ulisboa.pt

ABSTRACT

Blockchain provides an immutable and unforgeable ledger that can be decentralized. However, existing blockchain systems lack decentralization, e.g., due to clustered nodes most likely confined to datacenters. This centralization may raise some security concerns, e.g., allows the selection of a majority of nodes under the control of some malicious attacker. One crucial property that would help to avoid this security issue and to increase decentralization is to ensure diversity of participants in the blockchain. A way to enforce that diversity can be by choosing geographically diverse nodes, and we hypothesize that it is possible to achieve this by embedding virtual coordinates in the overlay of blockchain systems, as that would allow for topology-awareness. However, such a Virtual coordinate system (VCS) needs to be robust to malicious participants, including the ones performing attacks most effective in the context of blockchains. To address this, as a starting point for this thesis, we select Newton, a secure decentralized VCS. We evaluate it in an adversarial environment, where we simulate attack strategies trying to overcome Newton's security mechanisms, with a particular focus on attack strategies and scenarios relevant in a blockchain context, namely where the attackers form a cluster in the network, as a consequence of being operated by the same entity. We confirm that Newton can withstand the known attacks on VCS even when performed by the cluster. We then design and test a new attack strategy, Split Cluster Attack, which we found capable of disrupting Newton's defense mechanisms, degrading significantly Newton's accuracy.

Keywords

Distributed Systems; Blockchain; Decentralization; Virtual Coordinates; Cybersecurity

1. INTRODUCTION

Blockchain provides an immutable and unforgeable ledger that can be implemented in a fully decentralized manner. The importance of blockchain systems comes mainly from the ability to bring trust to a decentralized network, allowing for recording transactions between peers who do not trust each other, and removing the need for a trusted third party for validating those transactions. Given the recent success and increased visibility of this technology, blockchain-based applications are increasingly applied to various fields. This success creates a pressing need to find and overcome existing challenges and limitations of today's blockchains.

In a "permissionless" (open membership) blockchain setting, anyone can create an address and begin interacting with the network. This is derived from the nature of blockchain systems, namely the fact that they are decentralized, anonymous, and equally accessible from any computer. Beyond the advantages that result from this nature, such as increased privacy and low barrier to entry, some problems may also arise in the presence of malicious users. In particular, if an attacker creates multiple nodes, it becomes challenging to ensure that the system still works correctly [6, 5].

One crucial property to prevent such attacks is to ensure that there is a diversity of participants contributing to the blockchain, in order to minimize the odds of selecting a majority of nodes under the control of a malicious attacker, who can use that majority to subvert the system. However, in practice, existing blockchain systems lack diversity. For example, Bitcoin and Ethereum, two of the most popular blockchain-based networks, have been shown to have a significant number of clustered nodes [6], most likely confined to datacenters controlled by a single person or entity. Additionally, when these studies focus on the processing power instead of the number of nodes, they show that a low number of entities in these networks possess the majority of the processing power that maintains the blockchain [6]. As such, this lack of diversity has the potential to create security problems, thus reducing the benefit of a truly decentralized system.

A central observation of this thesis is that, enforcing geographic diversity between the nodes, would substantially raise the bar of an attacker trying to conduct this sort of attack, since he/she could no longer operate all the nodes from a single location, such as a datacenter. Furthermore, since blockchains form an overlay network between their members, it is possible to embed virtual coordinates in those overlays [3], thus allowing for choosing geographically diverse nodes, to enforce – or at least increase – diversity in the blockchain.

However, since there can be a malicious attacker that can try to subvert the protocol that determines these coordinates in order to control the blockchain, we need to deploy a virtual coordinate system that is secure in the presence of malicious participants. Thus, we choose Newton [10] as an appropriate starting point to achieve secure virtual coordinates for the nodes in the overlay.

Newton is a decentralized Virtual Coordinate System which extends Vivaldi [3] with security mechanisms that prevent

an attacker from spoofing its own coordinates or otherwise manipulate the coordinates of other nodes. The idea of these security mechanisms is to enforce Newton's laws of physics to recognize and reject tampered or malicious reports from one node to another. However, despite their potential to improve the security of blockchains, their use in this context require that they are not only robust against a single malicious node, but also against a cluster of nodes trying to disguise their real location.

In this thesis, we test Newton against novel attack scenarios that we devised, with the goal of understanding how robust, and consequently how suitable it is for incorporating in blockchains. To this end, we started by implementing Vivaldi and Newton in the context of a novel distributed systems simulator based on the Rust programming language [11]. This then allowed us to test Newton under a wide variety of adversarial scenarios: from executing the known attacks to virtual coordinate systems, performed both by randomly distributed attackers in the network and attackers forming a cluster, to designing and testing a novel Split Cluster Attack, which is a new attack strategy capable of disrupting Newton's ability to provide accurate coordinates, while the attackers deceive honest nodes into thinking that the nodes in a cluster are split across different groups. Under the known attacks, Newton is able to match the baseline accuracy, even for the cluster scenarios, however, the Split Cluster Attack, manages to degrade Newton's prediction performance.

The remaining of this document is organized as follows. Section 2 presents background on both *blockchain* and *virtual coordinate systems*. We describe our approach in this work, followed by the implementation details of Newton and the novel Split Cluster Attack in Section 3. Section 4 describes the metrics used in the measurements and shows the results obtained from our experiments using Newton. Finally, Section 5 presents the conclusions and future work.

2. BACKGROUND

Blockchain

Blockchain [9] is a peer to peer distributed ledger where transactions or digital events are recorded. This ledger is immutable, unforgeable, and simultaneously maintained by the nodes (computing devices) in the network. The transactions are grouped into blocks, and the blocks are then stored sequentially in the ledger record, thus forming a chain of blocks. For transactions to be included in a blockchain, they first need to be included in a candidate block (a possible new block to the chain). This entails a preliminary check where the nodes of the network have to confirm that it is a valid transaction. Then, the nodes run some consensus algorithm to reach an agreement on the next valid block to add the chain. In particular, the consensus algorithms that are used nowadays in permissionless blockchains are based on the notion of "proof of work". In these schemes, every node in the system acts as a miner trying to solve a cryptographic puzzle, in addition to validating transactions and creating the blocks for the ledger, usually in exchange for some reward. In this case, the consensus algorithm attempts to select the miner who appends the next block, among the nodes that succeeded in solving the puzzle.

Blockchain [9] systems can be organized according to various different categories. The primary division is between *public* and *private* blockchains. *Private* (or *permissioned*) blockchains have restrictions on the individuals that may belong to the blockchain, usually just members of some organization. On the contrary, *public* (or *permissionless*) blockchains have an open membership, and anyone can create an address from where to send or receive transactions and begin interacting with the network anonymously. The two primary permissionless blockchain-based cryptocurrencies are Bitcoin and Ethereum. We next survey a series of measurement studies on both Bitcoin and Ethereum, regarding their level of decentralization.

Decentralization in blockchain-based Cryptocurrencies Blockchain-based systems have decentralization as one of their key underlying properties. However, a lack of diversity is currently observed in overlays such as Bitcoin or Ethereum, where a significant fraction of the participating nodes is concentrated in a few data centers, reducing decentralization.

In [6], Gencer *et al.* present a measurement study on decentralization metrics in Bitcoin [8] and Ethereum [13, 4]. When measuring *network structure*, the aim was to understand if the networks were geographically clustered, through the use of measured and estimated latencies between nodes. These latencies were used to infer estimates of geographic distances, and the results show that Ethereum nodes are more distributed around the globe than Bitcoin nodes, but that both have nodes likely to be running in data centers, due to the geographic proximity between these nodes. The authors also measured the *distribution of mining power* in Bitcoin and Ethereum. Mining in these two networks is a complex and computationally expensive process, especially due to the *proof of work* consensus algorithm. The authors try to evaluate if the participants of these networks use more powerful hardware to succeed more often in solving the cryptopgraphic puzzle, resulting in a mining process more centralized. The authors' mining power estimations for each entity are based on the ratio of main chain blocks generated by each entity. Then through the examination of the weekly distribution of mining power in Bitcoin and Ethereum for ten months from 2016 to 2017, the authors present results showing that over 50% of the mining power has been shared by only eight miners in Bitcoin and five in Ethereum. These results give us a sense of the centralization of these blockchain networks. Furthermore, powerful miners who attract more and more members might try to appear less powerful, creating multiple entities, as not to seem that they are contributing to the centralization of the network.

Virtual Coordinate Systems

Virtual Coordinate Systems (VCS) were proposed to address an issue in large-scale distributed systems, related with the fact that the cost of direct measurements of network latency or bandwidth between nodes can outweigh the benefits of exploiting topology information, to be able to select the best nodes to contact (e.g., its neighbors in an overlay network) [3]. For this purpose, VCS allows hosts to predict network

performance metrics between pairs of nodes without the need for explicit measurements, which reduces significantly the network measurement overhead. The key idea of VCS systems is to characterize the network location of a node in the overlay by modeling the real network as a geometric space. In this geometric space, all the hosts have coordinates, and the distance between them represents latency, for instance, which is usually considered as the round trip time (RTT) between hosts.

VCS systems can be divided into two major groups, Landmark-based Systems, and Decentralized Systems. Comparing both, the main difference is in the fact that the former relies on *a priori* trusted set of nodes, a fixed infrastructure of landmarks, to serve as reference nodes. The latter, in turn, does not require any fixed network infrastructure/set of reference nodes and makes no distinctions between nodes. In other words, any node in the system may be used as a reference to any other node.

Next we describe Vivaldi [3], the decentralized approach corresponding to Newton [10], with no defense mechanisms. We then present the attacks on VCS, and lastly, we describe the security aspects of Newton.

Vivaldi

Vivaldi [3] is a decentralized, low-overhead, adaptive system, with a simple algorithm that assigns synthetic coordinates to hosts, with the distance between their coordinates predicting the communication latency, specifically RTT, between them, with low error.

Vivaldi was inspired by analogy to a real-world mass-spring system, therefore on a Euclidean coordinate space. The Euclidean space has to satisfy the triangle inequality, $d_{AC} \leq d_{AB} + d_{BC}$, where d_{xy} is the distance between the nodes x and y. However, Internet routing policies often violate the triangle inequality [15, 7, 12], so Vivaldi cannot predict the exact RTT between hosts. Instead, the algorithm attempts to find the coordinates that minimize the error of predictions.

In the design of Vivaldi, all nodes update their coordinates based on interaction with a subset of other nodes (neighbor set). A node chooses half of these nodes randomly from all possible nodes and the other half from a set of low-latency (nearby) nodes. In addition to the coordinate value, each node also maintains a local error value, representing the confidence in the coordinate value. Algorithm 1 describes how each node i updates its coordinates, after it sends a request to node j for its coordinate and local error value, and measures the actual RTT when the node j replies.

First, (line 1) a sample confidence (weight) w is calculated, balancing local and remote error. Then, (line 2) the relative error of the sample is computed. (line 3) Then node **i** updates its local error with a fraction of the sample error, weighted by the confidence in the remote coordinates, where c_e is a system parameter. (line 4) An adaptive time-step δ that depends on the confidence on the remote node coordinates and a system parameter c_c is now computed. Finally, (line 5) node **i** updates its local coordinates by finding the force applied by the remote node, using a fraction of that force determined by the adaptive time-step δ and multiplying that by a unit vector with the direction it should move.

Algorithm 1: Vivaldi's Node i Coordinate Update

Input : Remote node tuple $< x_j, e_j, RTT_{ij} >$
Output : Updated local node coordinate and error x_i, e_i

1 $w = e_i/(e_i + e_j)$;
2 $e_s = |||x_i - x_j|| - RTT_{ij}|/RTT_{ij}$;
3 $e_i = (e_s \times c_e \times w) + (e_i \times (1 - (c_e \times w)))$;
4 $\delta = c_c \times w$;
5 $x_i = x_i + \delta \times (RTT_{ij} - ||x_i - x_j||) \times u(x_i - x_j)$;

One key challenge in the design of Vivaldi is that sampling only low-latency (nearby) nodes can lead to coordinates that preserve local relationships but are far from correct on a global scale. This issue is avoided by adding long-distance communications. In the proposed design, each node is assigned eight neighbors: the four immediately adjacent to it and four chosen at random (on average, the random neighbors will be far away). At each step, each node decides to communicate either with an adjacent neighbor or a far away neighbor. In the evaluation of the system, it is shown that when half of the communication is with distant nodes, coordinates converge quickly.

A high time-step leads to high oscillation, and a low time-step leads to slow convergence. Consequently, Vivaldi uses an adaptive time step parameter, δ, to provide low oscillation, resilience against high-error nodes, and to make the system quickly converge to accurate solutions, maintaining accuracy even as a large number of new hosts joins the network.

Attacks on VCS

Now we will look at known types of attacks to VCS, aiming at lowering the performance of VCS systems. These are internal attacks, executed by *insider attackers*. Zage *et al.* [14] identified basic attacks, in a specific way considering how the malicious node can lie to perform different manipulations. Concretely, an attacker can influence the round-trip time (RTT) by delaying measurement probes, and can also lie about its coordinates and local error value, to conduct the following attacks:

Inflation Attack has attackers that lie about their coordinates, resulting in a victim node having incorrect coordinates, far from the correct ones. To accomplish this, an attacker can, for instance, report a low error and close coordinates to the victim, and delay the measured RTT.

Deflation Attack aims at maintaining a victim node immobile. For that purpose, an attacker may report coordinates that result in an estimated RTT ($|x_v - x_a|$, where x_v and x_a are the coordinates from the victim node and the attacker, respectively) similar to the measured RTT, making the victim node think that it should not move. An alternative is for the attacker to report coordinates close to the origin.

Oscillation Attack, like the disorder attack, introduces chaos in the coordinate system. Attackers choose random coordinates to report and randomly delay measurement probes. Con-

sequently, the remaining nodes have to keep updating their coordinates, not converging to their correct positions.

Additionally, Chan-Tin *et al.* [1, 2] identified more advanced attacks, *frog-boiling* and *network partition*, that are slow attacks, more subtle and difficult to identify. The objective of both is to disrupt the accuracy and stability of victim nodes' coordinates.

Frog-boiling Attack has a node lying in small amounts at a time, moving their coordinates in one direction. Over time, the coordinates of the lying node end up being far from its real position, but each step is small enough not to trigger the anomaly detection in the security mechanisms based on outlier detection.

Network partition Attack is an extension of the frog-boiling attack, with multiple nodes colluding together and moving in opposite directions.

Newton

Newton [10] is a decentralized VCS that extends Vivaldi [3] with security mechanisms. The objective is to withstand a large number of attacks on VCS systems, by using safety invariants derived from Newton's three laws of motion, and if a node lies, the system detects that it violates some safety invariant, and the update from that node can be ignored.

Newton's three laws of motion are the following: *First Law*: a body stays at rest unless acted upon by an external, unbalanced force; *Second Law*: a force f on a body of mass m, undergoes an acceleration a, such that a is proportional to f and inversely proportional to m; *Third Law*: when a first body exerts a force on a second body, the second body exerts an equal but opposite force on the first body.

The rationale underlying the detection of the attacks previously described is that, when an attacker node lies about its coordinates, it is implying that some forces have previously acted upon it, thus introducing extraneous indirect forces into the system. Introducing these forces breaks the first and third laws. Furthermore, when an attacker delays a probe or lies about its local error, it is introducing extraneous direct forces between itself and another node. This case breaks the second law. Therefore, Newton introduces the following mechanisms to detect these violations:

- Detecting extraneous indirect forces: Newton incorporates in its design two detection methods, one for randomly chosen neighbor nodes, and the other for physically close neighbor nodes:
  - Random nodes: By Newton's third law, there can be no unbalanced forces in a mass-spring system. By the definition of the first law, an extraneous indirect force introduced by an attacker will be an unbalanced force. Then, the third law implies that an unbalanced force can be detected by finding the centroid of the nodes' coordinates. Consequently, the following invariant is used to detect such violations.

    First invariant, **IN1**: If the centroid of a node i and the randomly selected nodes from its neighbor set is at the origin of the geometric space, then no unbalanced force has been introduced. However, if the centroid is not at the origin, then an attacker (or collection of attackers), has introduced an unbalanced force that has the same direction as a force vector from the origin to the centroid c. Therefore, a node i can find an attack by observing that the centroid c is non-zero, or over some threshold.
  - Physically close nodes: Because all nodes are connected via springs and are physically close, they will experience similar forces from the same nodes, thus leading to the following.

    Second invariant, **IN2**: if nodes i and k are physically close, and node i experiences a force f_{ij} from node j, then node i would expect node k to experience a force f_{kj} from j similar to the vector projection of f_{ij} onto the vector $u(x_j - x_k)$, where x_j and x_k are the coordinates of the nodes j and k.

    Node i knows where node k is expected to have its updated coordinates, and it calculates the difference, in distance, between this expected location and the location reported by k. Then, if the distance is over some threshold, i rejects the update from k.
- Detecting extraneous direct forces: The second law states how much a node should accelerate, given the force and mass of a node. Additionally, in a mass-spring system, the amount of force applied to a node is dominated by Hooke's law, $F = -kx$, which states that the amount of force on a node is proportional to the spring's current displacement x from its rest position, and where k is the spring constant [1]. These laws can be checked by the following.

  Third invariant, **IN3**: As the springs in the physical system stabilize and come closer to their rest position, nodes should decelerate, and as a consequence, the forces applied to them should decrease over time.

3. IMPLEMENTATION

A way to enforce diversity of participants in blockchains and thus minimizing the odds of selecting a majority of nodes under the control of a malicious attacker can be by choosing geographically diverse nodes. In this thesis, we hypothesize that it is possible to achieve this by embedding virtual coordinates in the overlay of blockchain systems, as that would allow for topology-awareness. However, since we may have to handle a malicious attacker that can try to subvert the protocol that determines these coordinates in order to control the blockchain, we need to deploy a virtual coordinate system that is secure in the presence of malicious participants. For that purpose we will use Newton [10]. Given this context, we need to make sure that the defense strategies that Newton puts in place are robust against the attacks that might be most effective in the context of blockchains. Therefore, we evaluate Newton in an adversarial environment, where we simulate attack strategies trying to overcome Newton's security mechanisms, with a particular focus on attack strategies and scenarios relevant in a blockchain context, namely where the attackers form a

[1] How stiff the spring is.

cluster in the network, as a consequence of being operated by the same entity. Consequently, our work consists of the following two main stages:

- Perform a set of known attacks to VCS to try to degrade the system performance, specifically its accuracy and in some cases stability, but in a novel context, where the attackers form a cluster in the network. The cluster varies in *size*, being formed by a different number of attackers, and *distance* to the rest of the network, where we aim at comparing attackers randomly positioned in the network, attackers forming a cluster close to the network, and forming a cluster far isolated from the rest of the network.
- **Split Cluster Attack** devises a new attack, aimed at concealing the clustering of a set of nodes by making it appear as multiple separate sets.

The measurements and results of the experiments for both are shown in Section 4. We next go through the implementation of Newton and the design of the Split Cluster Attack.

Newton

The Newton protocol [10] was explained in Section 2, in two different sections, since it consists of two components, namely Vivaldi as the base protocol, and Newton's *security invariants* applied on top of it. Additionally, this implementation is conducted in a simulation environment, which allows us to remove a lot of the complexity of a real execution. Regarding the simulator to use, we picked the novel *Corten Simulator* [11], since it is a discrete event-based distributed algorithms simulator that, among other things, models network asynchrony, namely latency, but at the same time is simple enough to allow for fast prototyping and a scalable evaluation. Specifically, *latency* is given by a matrix of inter-host internet latencies. Additionally, the Corten simulator, is coded in the Rust programming language, and requires its applications to also be written in Rust. Consequently, we had to implement all our code in Rust as well.

Our explanation of the implementation of Newton is split into two parts: first, the implementation of the base code of Vivaldi, and then the *security invariants* based on Newton's laws of physics, which add the security aspect.

Vivaldi

A node running Vivaldi needs to store the following main attributes: Local Virtual Coordinates, Local Error and Neighbor Set. Additionally, the main method, which contains the core algorithm of the system is the one that takes care of updating the coordinates, and additionally also updates the local error.

Virtual Coordinates: The coordinates that each node keeps are intended to allow the estimation of the RTT between nodes, which is accomplished by computing the distance between coordinates. As mentioned in [3], the simplest solution is to use n-dimensional coordinates and the standard Euclidean distance function, as per Equation 1, where x and y are the coordinates of the two nodes, and n is the number of dimensions.

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2} \tag{1}$$

Additionally, Seibert *et al.* [10] have shown that Newton operates well using simple two-dimensional coordinates. Consequently, and considering we run all experiments in a simulation environment and not in a real Internet deployment, we use two-dimensional coordinates in the Euclidean space.

Error: The local error value each node maintains represents the confidence the node has in its coordinate value. This error is a *float* with values within [0.0, 1.0], and starts with the value 1.0 at the beginning of an execution. In Algorithm 1, we can see how it is updated.

Neighbor Set: As previously explained, half of the neighbors are low-latency nodes and the other half are random nodes from the network. Each node creates its neighbor set at the beginning of execution and following the next two steps: (1) choosing the closest neighbors is done using the simulator's global knowledge of the network, where the $\#neighbor\_set/2$ nodes with the lowest latency to the current node are chosen; (2) choosing the random neighbors is done in a straightforward way given this global knowledge. A node uses this knowledge to pick the $\#neighbor\_set/2$ nodes randomly with uniform probability.

Coordinate Update: Regarding the updating of coordinates in the system, our implementation directly reflects the specification given in Section 2, when describing Vivaldi, with its core rationale in Algorithm 1.

Security

Regarding the security in Newton, the central idea falls in the usage of *security invariants* based on Newton's laws of motion. Therefore, in addition to the logic that is already present in the version with no security, every node after receiving an update/reply from another node checks the invariants, and, if at least one is violated, the update is discarded. Specifically, when receiving an update from a *random neighbor*, IN1 and IN3 are checked; and, if the update is coming from a *close neighbor*, then the invariants that are checked are IN2 and IN3.

IN1: A node i calculates the centroid of its local coordinates and its random neighbors' coordinates, considering also the force being applied in the current update, as we show in Equation 2, where x_p represents the coordinates of node p, f_{ij} the force that node j is trying to apply on node i in the current update, and n the number of nodes in the network.

$$c = \frac{\sum_{p=1}^{n} x_p + f_{ij}}{n} \tag{2}$$

If the distance from the origin to the centroid c is larger than some IN1 threshold, then node i detected an attack. Subsequently, it can find which node introduced the unbalanced force into the system and is therefore the attacker. For every neighbor, node i sums up all the forces that the neighbor has applied to it, and computes the vector projection of the summed forces onto the centroid vector. The neighbor node whose projection has the greatest magnitude is, therefore, the greatest contributor to the moved centroid. Specifically, if the current updating node j is the neighbor with the greatest magnitude, then the current update will be ignored.

IN2: We implemented the IN2 verification exactly as described in Section 2, when discussing the detection of extraneous indirect forces introduced in the system by physically close nodes, in Newton subsection, taking into consideration the last forces of all the neighbors (close and random) that the local node has experienced.

IN3: As the springs in the physical system stabilize and come closer to their rest position, nodes should decelerate, and as a consequence, the forces applied to them should decrease over time. To verify if nodes reporting updates are indeed slowing down over time, the local node calculates the median $\tilde{f}$ and median absolute deviation D of the magnitude of the force that each node is applying to it. If the magnitude of any force m_j is some deviations larger than the median, the node will ignore it. This is shown in Equation 3, where t is the threshold for the number of deviations.

$$m_j > \tilde{f} + t * D \tag{3}$$

Additionally, there are different thresholds defined for randomly chosen neighbors t_r and physically close ones t_c, where the median is calculated separately for both types of neighbors. The rationale for this is that close nodes exert smaller force values but deviate more from the median, while randomly chosen nodes are the opposite [10]. Hence, the threshold for the close neighbors will be higher than the threshold for randomly chosen nodes.

Split Cluster Attack

This is a new type of attack, inspired by the actions that an entity trying to operate a cluster in a permissionless Blockchain might take, in order to deceive a secure virtual coordinate system trying to detect and deactivate such a cluster. As such, the main goal of the Split Cluster Attack is to split a cluster into multiple separate groups in the eyes of the rest of the network. It is performed by the nodes in the referred cluster, which will cooperate and lie to the remaining nodes in the network, i.e., the benign nodes. We set the goal of trying to separate the previously mentioned cluster into multiple groups *to the extent possible*, and a second goal of trying to degrade Newton's estimation performance.

Our approach was devised while taking into account the *security invariants* of Newton, and therefore we try to surpass these one by one. In particular, there are three *security invariants* that Newton tries to validate, and that consequently need to be circumvented by our strategy. With this in mind, we next explain the design of this attack strategy, and what it does to deal with each of the invariants:

IN1: First, we want to take into account IN1, which detects erroneous behavior by checking if the centroid moves away from the origin of the geometric space. Consequently, to dodge detection, the attackers need to avoid moving the centroid, and for that, our design for this attack will start by using the same base ideas as the network partition attack: splitting the cluster, with its nodes slowly getting their coordinates far away from each other, in opposite directions.

IN2: To allow the attackers from the cluster to get around this invariant, the cluster have to be isolated from the rest of the network or at least positioned far enough to avoid its nodes to be selected as *close neighbors* of any of the benign nodes in the network, since IN2 is only tested for low RTT neighbors.

IN3: Lastly, the third security invariant, IN3, checks if the forces in the system decrease over time. To overcome this invariant, the idea is for the attackers to keep the forces they are applying to benign nodes under the deviation threshold of IN3. Since a strategy along the lines of the *network partition* attack is being used, the attacker will deviate its coordinates from their real position, lying by small increments at a time. Additionally, we will have the attacker decrease the fake increment proportionally to the median magnitude of the force (which is explained in Section 3, in Newton's security implementation, under **IN3**) applied by the neighbor set of some node. Mainly, this allows the force introduced in the system and applied to another node by the wrong reports (the small increments) over time to be proportional to the IN3 threshold, which determines when to ignore an update that tries to apply a force higher than the maximum defined in each time instant. Specifically, each attacker will simply compute its median force and use it as a reference.

In summary, the attackers will be forming a cluster positioned far enough from the rest of the network, to avoid that any of its nodes are selected as close neighbors of the nodes outside the cluster, excluding this way the IN2 checks against these nodes updates. The attack consists essentially in the Network Partition behavior, where the colluding attackers avoid moving the centroid of the network (IN1 weakening), but with a dynamic value for the small fake increments, which will decrease over time to avoid detection from IN3. With all these provisions in place, the attack should be able to disguise the physical clustering and/or reduce the virtual coordinate system's performance, mainly accuracy.

4. MEASUREMENTS AND RESULTS

Newton's code is executed in each node individually, without the need of any central nodes for coordination of the algorithm, which complies with the decentralization characteristics we want for our VCS system. Regarding our choices for the system parameters, as in [3] and [10], each node picks 64 neighbors, with half being low RTT nodes and the other half random nodes. In addition, Seibert *et al.* have shown in [10] that the following values can be used in any Internet-wide deployment, and so we used them while configuring Newton: IN1 threshold 20 ms, IN2 threshold 35 ms, five deviations for random neighbors and eight deviations for close neighbors regarding IN3 thresholds.

The Corten Simulator [11] will need a matrix of inter-host internet latencies of the network in the simulations. The data set we use for our simulations contain pairwise measurements of latencies between 1130 nodes, plus the added nodes that will attack the system, either randomly distributed around the network or forming a cluster. The number of attackers will be defined by the percentage of nodes in the network that are attackers, which has different values for different experiments.

In all simulations, we use a Euclidean Coordinate Space with two dimensions, and all nodes join in the beginning in a flash-

crowd scenario and continue until the end of the execution. Every node is constantly selecting a new node from its neighbor set to which to send a probe, and consequently receive an update in its reply. A probe is sent by each node every 2500 ms, unless otherwise stated, and every node has only one pending probe at a time.

Regarding the main metrics we use to analyze the experiments, we have the *Prediction Error* to help us visualize the accuracy of the system and the *Velocity* for the stability. Accuracy and stability of the system go up when prediction error and velocity go down, respectively.

- *Prediction Error* of the system is computed in three levels. Specifically, it is the median of all the errors of each node. In turn, the error of each node corresponds to the median of the link errors involving that node. Lastly, the error of a link, which is a virtual connection between two nodes, follows Equation 4, where RTT_{actual} is the real RTT between two nodes, and $RTT_{prediction}$ is the distance between the virtual coordinates generated by each node ($||x_j - x_i||$ for nodes i and j).

$$pred\_error = |RTT_{actual} - RTT_{predicted}| \tag{4}$$

- *Velocity* is given by Equation 5, where $\triangle x$ refers to the distance that some node travels, and t is the time taken to make that distance. The system's velocity is computed as the average velocity of all the nodes, and we calculate it for different time instants.

$$v = \frac{\triangle x}{t} \tag{5}$$

Existing Attacks

In this section, we evaluate the impact that known attacks to VCS have when performed by a cluster of attackers (representing an adversary flooding the network with nodes under its control), and varying the distance between the cluster and the rest of the network. This focus on an isolated cluster is in contrast with the scenario of randomly disperse attackers, which is the type of attack tested by Seibert *et al.* [10]. In other words, we are evaluating existing attacks, but using a novel configuration for the set of nodes controlled by the attacker. All the experiments in this section run for approximately 1.4×10^7 ms, unless stated otherwise. Additionally, we vary the percentage of nodes from the network that are attackers between 10% and 30%.

We first try to understand the effect of cluster isolation on the effectiveness of the attacks. To this end, we test each attack on different topologies, varying the distance from the cluster to the rest of the network, more concretely, using 50ms and 500ms. Additionally, we run simulations where randomly distributed attackers perform the attacks. When analyzing the results for each attack, we present and compare the system's accuracy for these different scenarios. As a baseline for each attack, we present the accuracy when no attack is performed. This results in four accuracy curves over time, which are labeled as *NoAttack*, *Random*, *Cluster50* and *Cluster500*. We now present the results of the experiments for each of the different attacks, where each one starts at $4,000,000$ ms, allowing the system to stabilize first:

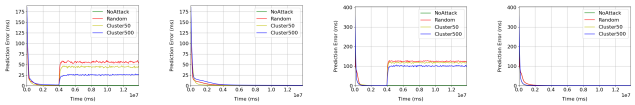
Inflation Attack: To report fake coordinates far away from the origin, an attacker randomly chooses a distance between 900 and 1000 ms, which is far enough from the origin to be beyond every correct node in the network. Observing Figures 1(a) and 1(c), we realize that the cluster scenarios have less impact on the virtual coordinates than the random one. Furthermore, Newton is able to keep the prediction error matching with the baseline, for all scenarios. This is because, regardless of the positioning of the attackers, the unbalanced forces introduced in the system during the attack, displace the centroid and allow Newton to detect the attack through IN1 (Figures 1(b) and 1(d)).

Deflation Attack: In this attack, to report coordinates close to the origin, the distance to it is randomly chosen between 0.1 and 1.0 ms. When analyzing Figures 2(a) and 2(c), we observe that when the attack starts, the prediction error of the system (using Vivaldi) rapidly peaks when under the attack of the cluster at 500 ms of distance, with higher values than with the random attackers. However, even without the security mechanisms, Vivaldi is able to match the baseline prediction error in the cluster scenarios. We can see in Figures 2(b) and 2(d) that Newton is able to successfully mitigate the attack once again.

Oscillation Attack: An attacker not only lies about its coordinates but also delays probes by up to 1 second. Regarding the coordinates reported, it randomly generates fake coordinates between 0.1 and 1000 ms of distance to the origin in any direction. We don't show the results for the oscillation attack, because they present the same conclusions and patterns as the ones from inflation attack, and just like in the inflation attack, the cluster attackers have a smaller impact than the random attackers on the system's accuracy. Once again, as in the previous attacks, Newton is able to match the baseline accuracy in all scenarios, which we attribute to the IN3 ability to detect when forces do not decrease over time.

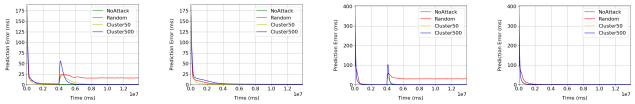
Frog-Boiling Attack: The small deviations that the attacker reports in each update have a length of 0.25 ms. Figure 3 presents the prediction error of the system under the frog-boiling attack. When there are 30% of attackers, we can see in Figure 3(c) that the cluster eventually starts degrading accuracy more than the random attackers, which had not been the case in the other attacks. However, and regardless of the scenario, the attackers are not able to prevent Newton from reaching stable and accurate coordinates.

Network Partition Attack: Like in the Frog-Boiling attack, the small inaccuracies that the attacker reports in each update that is sent, increment its fake coordinates by 0.25 ms. In Figure 4 we have the accuracy results, which allow us to see, when Vivaldi is under attack, that the network partition attack has an even slower impact than the frog-boiling attack, even though both attacks are considered to be slow attacks. Newton is still able to protect against the network partition attack, even without relying so much on IN1, since colluding attackers avoid moving the centroid.



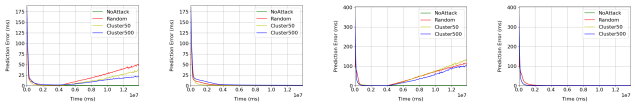
(a) Vivaldi - 10% attackers (b) Newton - 10% attackers (c) Vivaldi - 30% attackers (d) Newton - 30% attackers

Figure 1. Accuracy - *Inflation Attack*



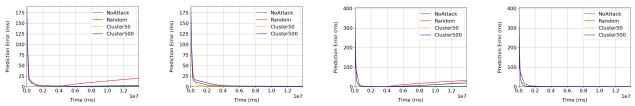
(a) Vivaldi - 10% attackers (b) Newton - 10% attackers (c) Vivaldi - 30% attackers (d) Newton - 30% attackers

Figure 2. Accuracy - *Deflation Attack*



(a) Vivaldi - 10% attackers (b) Newton - 10% attackers (c) Vivaldi - 30% attackers (d) Newton - 30% attackers

Figure 3. Accuracy - *Frog-Boiling Attack*

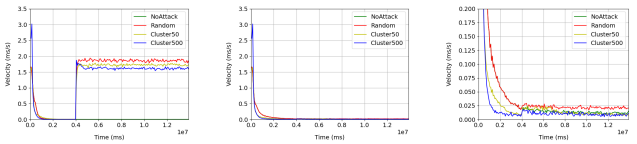


(a) Vivaldi - 10% attackers (b) Newton - 10% attackers (c) Vivaldi - 30% attackers (d) Newton - 30% attackers

Figure 4. Accuracy - *Network Partition Attack*

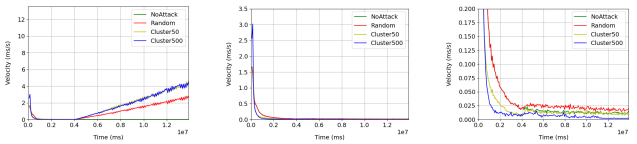
Stability: We have just seen the accuracy results obtained by Newton under all the different attacks we consider, and how it is able to match the baseline accuracy and maintain it over time. Thus, it is of no surprise that we learn from Figures 5, 6 and 7, that it obtains stable coordinates, which are represented by a small velocity value close to 0. The stability results are only presented for the oscillation, frog-boiling and network partition attacks, as these are the attacks that also intend to disrupt stability, in addition to accuracy. When comparing the impact of the frog-boiling and network partition attacks over Vivaldi, we observe that the cluster attackers have a higher impact on disrupting stability than the random attackers during the frog-boiling attack, whereas the opposite happens in the network partition attack. However, Newton can deal with both. Additionally, we present the zoomed-in velocity axis for the three attacks, which highlights that, in all scenarios, Newton reaches velocity values with minor, almost insignificant deviations from the baseline.

Start Attack from Beginning: In an attempt to make the attacks more effective against Newton, we ran each one of them from the beginning of the system execution, with 30% of attacker nodes in the network. We present the prediction error results of these experiments in Figures 8 and 9, which run for 2.5×10^7 ms.As we can observe, Newton mitigates all five attacks. The main observation is that during the network partition



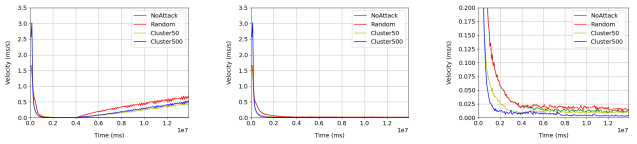
(a) Vivaldi - 30% attackers (b) Newton - 30% attackers (c) Newton - 30% attackers (zommed in)

Figure 5. Stability - *Oscillation Attack*



(a) Vivaldi - 30% attackers (b) Newton - 30% attackers (c) Newton - 30% attackers (zoomed in)

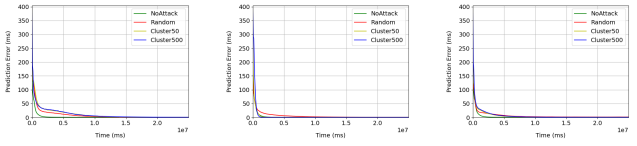
Figure 6. Stability - *Frog-Boiling Attack*



(a) Vivaldi - 30% attackers (b) Newton - 30% attackers (c) Newton - 30% attackers (zommed in)

Figure 7. Stability - *Network Partition Attack*

attack, the system takes longer to match the baseline accuracy, specifically in the random attacker scenario.



(a) *Inflation* (b) *Deflation* (c) *Oscillation*

Figure 8. Accuracy - Newton with 30% of attackers. *Simple attacks* from beginning.

(a) *Frog-Boiling* (b) *Network Partition*

Figure 9. Accuracy - Newton with 30% of attackers. *Advanced attacks* from beginning.

Summing up from the results obtained in this section, we observe that the defense mechanisms of Newton are strong enough to mitigate not only the known attacks performed by random attackers but also by attackers forming a cluster, even if isolated from the rest of the network. In fact, the cluster attackers were less effective than the randomly distributed ones. We attribute this to the fact that half of the neighbor sets are

composed of close (low RTT) neighbors. An isolated cluster, sufficiently distant to avoid its nodes from being chosen as close neighbors, will consequently reduce its influence over the network, since fewer of its attacker nodes will be neighbors (reference nodes) of the benign nodes.

Split Cluster Attack

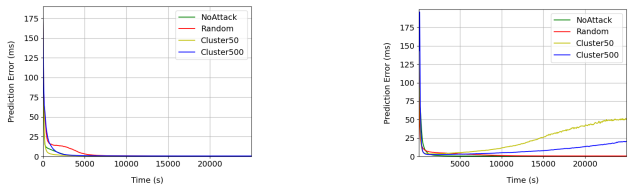
Unlike in the experiments from the previous section, in this attack, nodes send a probe every 2 seconds because this is enough to allow for one pending probe at a time. The rationale behind the *split cluster attack* and how we intend to avoid each one of the three *security invariants* is as follows:

For **IN1**, which is looking out for the displacement of the centroid of the network, the approach is to minimize the deviation of the centroid by adopting the core behavior of the *network partition attack*, with colluding nodes counter-balancing and canceling the extraneous forces introduced in the system by their lies.

Regarding **IN2**, we obtained the total number of close neighbors of benign nodes that are attackers, for the network used in our simulations, as follows. For 10% Attackers - randomly distributed attackers: 1446; cluster at centroid of network: 210; cluster at 50 ms and 500 ms: 0. For 30% Attackers - randomly distributed attackers: 3797; cluster at centroid of network: 244; cluster at 50 ms and 500 ms: 0. As we can see, comparing the random scenarios with the cluster ones, in the former, there are significantly more close neighbors of benign nodes that are attackers, than in the latter. We are interested in the scenarios where this number is as low as possible, because IN2 is only verified for updates coming from close neighbors, and, in the case of the network we are using for the simulations, there are no nodes from the cluster being chosen as low RTT (i.e., close) neighbors when it is at least at 50 ms away from the rest of the network.

To deal with **IN3**, as previously explained, we produce small lies that push the coordinates of an attacker slowly away from its correct position and decrease over time, making the force generated by these fake movements remain below the IN3 threshold in benign nodes. To analyze the effect of the attack after its completion, we present the results for accuracy of our experiments in Figure 10, where the attack starts from the beginning, and we varied the percentage of attackers between 10% and 30%. Serving as a baseline is the prediction error curve of a simulation of Newton under no attack, then a curve that resulted from the scenario where the attackers are randomly distributed around the network, and finally the prediction error curves for the scenarios with the attackers forming a cluster, whose distance to the rest of the network changes between 50 ms and 500ms.

We first observe that in order for the attack to be effective, the attackers must represent at least 30% of the nodes in the network, since we have a clear increase of the error in Figure 10(b), but not in Figure 10(a). In fact, Newton was able to maintain the accuracy of the system during the three scenarios of attack (random, cluster50, cluster500), matching the accuracy of the benign scenario, with 10% of attackers. For 30% of attackers, we can see in Figure 10(b), that randomly

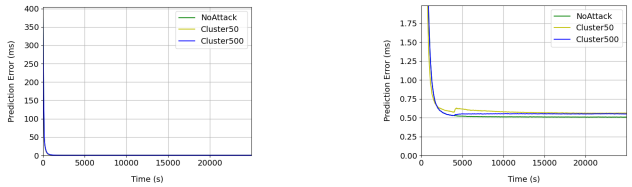


(a) 10% Attackers (b) 30% Attackers

Figure 10. Split Cluster Attack - Accuracy

distributed attackers have no significant impact on the system's accuracy, and once again Newton is able to match it with the accuracy of the benign setting. On the other hand, we observe that the curves for the cluster scenarios have an increase, with Cluster50 reaching around the double of Cluster500, at around 50ms and 25ms of error, respectively. Regarding the reason that makes the attack successful when performed in the cluster scenarios, but not in the random one, this is related to the detection of malicious updates by IN2. Specifically, the attackers are isolated and distant enough so that the remaining nodes in the network do not choose any of them as close neighbors, and consequently do not even run the IN2 check on their updates.

The attack must start right from the beginning of the system execution, because of the higher tolerance of IN3 to the forces applied between nodes. In Figure 11, we have the accuracy of the system when performing this attack after the system has stabilized, at 4000 seconds, which shows no significant impact. Looking at the Figure 11(b), where we zoom in the *prediction error* axis, we can notice where the attack starts, but the prediction error peaks after an insignificant increase of less than 0.1 ms, then it starts decreasing and stabilizes with an insignificant difference of around 0.04 ms above the accuracy of the system under no attack.



(a) 30% Attackers (b) 30% Attackers (Zoomed in)

Figure 11. Split Cluster Attack - Accuracy with attack starts at 4000 seconds

A noteworthy aspect of this evaluation is the following: in prior work [10], the authors say that Newton can avoid significant degradation of its accuracy until the system reaches 50% of attacker nodes. In this thesis, however, we just presented an attack strategy where Newton's security invariants cannot provide enough protection to allow for accurate coordinates, accomplishing that with a minimum of 30% attackers.

5. CONCLUSION

In this thesis, we highlighted the current lack of decentralization in blockchain systems, and how in order to minimize the odds of selecting a majority of nodes under the control of a

malicious attacker (who can use that majority to subvert the system), one crucial property is the diversity of participants contributing to the blockchain. With that in mind, we build on the concept of virtual coordinate systems, which model networks as geometric spaces, attributing virtual coordinates to each node in this space. These virtual coordinates allow for efficient estimation of latency between nodes in the network. A central observation of this thesis is that we could increase the diversity in blockchains by embedding virtual coordinates in the overlay of the blockchain and choosing geographically diverse nodes for contributing to the blockchain.

We showed, through various simulations, that Newton is indeed very robust even when under attack, being able to mitigate all the attacks presented in Section 4.1, including the cluster attack scenarios we devised.

Another contribution of this work is the Split Cluster Attack, which we designed with inside knowledge about Newton's protocol. This attack strategy is able to disrupt Newton's prediction accuracy. In particular, the nodes from the cluster performing the attack are able to deceive the remaining nodes from the network into thinking that they (the attackers) are split across different groups. This is, to our knowledge, the first negative result that is presented in the context of Newton, given that the original paper only mentions the ability to cope with advanced attackers that leverage insider knowledge about calibration-specific parameters used by the algorithm [10].

For the Split Cluster Attack to be effective, the cluster of attackers must represent at least 30% of the network, it must be performed right from the start of Newton's deployment, and the cluster must be at a minimum distance that prevents the honest nodes from choosing attacker nodes as close (low RTT) neighbors. However, all these requirements reduce the number of occasions when attackers could perform the attack. In particular, the need to start the attack from the beginning of the system's execution can be a substantial barrier. Consequently, despite having created an attack strategy capable of degrading Newton's performance significantly, we still believe, from the remaining experiments done in this thesis, that Newton is suitable and robust enough to be useful when deployed on a blockchain system.

A promising avenue for future work is to design a modification for Newton, in order to optimize it and make it also resilient to the Split Cluster Attack. Additionally, another possibility is to run the scenarios tested in this work in a real implementation. In particular, the main avenue for future work resides in incorporating Newton [10], as a virtual coordinates algorithm, in the blockchain source code. This would require building an overlay between nodes (or adapting the existing overlays used by the blockchain) and embedding virtual coordinates for the nodes in that overlay, which could allow for topology-awareness, and from there enforcing greater diversity of participants in the blockchain network by choosing geographically diverse nodes. With this purpose, either Bitcoin [8] or Ethereum [13] could be used as a representative blockchain system where to implement the Newton algorithm, as they are two of the most popular blockchain systems at this time and have a large amount of documentation available.

REFERENCES

[1] Eric Chan-Tin, Daniel Feldman, Nicholas Hopper, and Yongdae Kim. 2009. The Frog-Boiling Attack: Limitations of Anomaly Detection for Secure Network Coordinate Systems. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering* 19 LNICST (2009), 448–458.

[2] Eric Chan-Tin, Victor Heorhiadi, Nicholas Hopper, and Yongdae Kim. 2011. The Frog-Boiling Attack: Limitations of Secure Network Coordinate Systems. *ACM Trans. Inf. Syst. Secur.* 14, 3, Article 27 (Nov. 2011), 23 pages.

[3] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. 2004. Vivaldi: A decentralized network coordinate system. *Computer Communication Review* 34, 4 (2004), 15–26.

[4] Ethereum. Last edit Jun. 2019. A Next-Generation Smart Contract and Decentralized Application Platform. (Last edit Jun. 2019).

[5] Ittay Eyal and Emin Sirer. 2013. Majority Is Not Enough: Bitcoin Mining Is Vulnerable, Vol. 8437. DOI: http://dx.doi.org/10.1007/978-3-662-45472-5_28

[6] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert Van Renesse, and Emin Sirer. 2018. *Decentralization in Bitcoin and Ethereum Networks*. 439–457.

[7] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. 2007. Network Coordinates in the Wild.

[8] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. *Cryptography Mailing list at https://metzdowd.com* (03 2009).

[9] Marc Pilkington. 2016. *Blockchain Technology: Principles and Applications*.

[10] Jeff Seibert, Sheila Becker, Cristina Nita-Rotaru, and Radu State. 2014. Newton: Securing virtual coordinates by enforcing physical laws. *IEEE/ACM Transactions on Networking* 22, 3 (2014), 798–811.

[11] Ines Sequeira. 2019. Large Scale Distributed Algorithms Simulator. *IST Dissertation* (2019).

[12] Guohui Wang, Bo Zhang, and T. Ng. 2007. Towards network triangle inequality violation aware distributed systems. 175–188.

[13] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. (2014).

[14] David John Zage and Cristina Nita-Rotaru. 2007. On the accuracy of decentralized virtual coordinate systems in adversarial networks. *Proceedings of the ACM Conference on Computer and Communications Security* (2007), 214–224.

[15] Han Zheng, Eng Lua, Marcelo Pias, and Timothy Griffin. 2005. Internet Routing Policies and Round-Trip-Times, Vol. 3431. 236–250.