Building More Decentralized Blockchains Using Secure Virtual Coordinates

Marco Filipe Guerreiro da Silva

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Rodrigo Seromenho Miragaia Rodrigues

January 2021

Acknowledgments

I would like to thank my advisor Professor Rodrigo Miragaia Rodrigues for the guidance and support throughout this thesis.

I would like to thank my parents Eduardo e Odete Silva, my brother Bruno Silva, my girlfriend Cátia Neves, and my great friend Micael Prata for all the encouragement and support, since the day I decided to go to IST, until the end of this thesis.

Last but not least, to my great friends Hugo Guerreiro, João Sousa, Manuel Vidigueira, Maria Caixas e Nádia Fernandes, thank you for your friendship and companion, I learned a lot from you during these 5 years. Thank you guys!

Resumo

A Blockchain fornece um livro-razão imutável e infalsificável que pode ser descentralizado. Na prática, os sistemas de blockchain existentes carecem de descentralização, por exemplo, devido a nós agrupados em centros de dados. Esta centralização pode levantar algumas preocupações de segurança, como permitir a seleção de uma maioria de nós sob o controlo de um atacante malicioso. Uma propriedade crucial que ajudaria a evitar este problema de segurança e a melhorar a descentralização é a diversidade dos participantes da blockchain. Uma forma de impor essa diversidade pode ser através da escolha de nós geograficamente dispersos, e nós acreditamos que é possível consegui-lo através da incorporação de coordenadas virtuais nos sistemas de blockchain, pois isso permitiria conhecer a topologia da rede.

Tal sistema de coordenadas virtuais (VCS) precisa de ser robusto contra participantes maliciosos, incluindo os que realizam ataques que possam ser mais eficazes no contexto da blockchain. Para abordar esta questão, como ponto de partida desta tese, selecionamos Newton, um VCS descentralizado e seguro. Avaliamo-lo num ambiente adverso, onde simulamos estratégias de ataque tentando ultrapassar os mecanismos de segurança de Newton, com particular foco nos ataques relevantes no contexto da blockchain, nomeadamente onde os atacantes formam um cluster na rede, como consequência de serem operados pela mesma entidade.

Confirmamos que o Newton pode resistir aos ataques conhecidos contra VCS, mesmo quando realizados pelo cluster. Desenhámos e testámos uma nova estratégia de ataque, Split Cluster Attack, que descobrimos ser capaz de ultrapassar a defesa de Newton, degradando significativamente a sua precisão.

Palavras-chave: Sistemas Distribuídos, Blockchain, Descentralização, Coordenadas Virtuais, Cibersegurança

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

Contents

List of Tables

List of Figures

List of Algorithms

Nomenclature

RTT	Round Trip Time
VCS	Virtual Coordinate System

Chapter 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, such as their pending security and scalability problems.

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 [1, 2].

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 [1], 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 [1]. 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, if we could enforce geographic diversity between the nodes, then we would substantially raise the bar of an attacker trying to conduct this sort of attack, since he or 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. While this would not be a panacea that will completely solve this problem, we still believe that it would be a valuable contribution, with the potential to make it significantly harder for a single organization to control the network.

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, as described in this document, we started by reviewing several virtual coordinate systems and concluded that Newton [4] is an appropriate starting point to achieve secure virtual coordinates for the nodes in the overlay.

Newton's security mechanisms 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. The physical laws have to be obeyed by real-life systems, so they can represent security invariants that all nodes in the system should follow. Therefore, if a node reports incorrect values of coordinates or delays any measurements, it will introduce extraneous forces in the physics that explain the system, which breaks the expected invariants, and can therefore be detected by the system. 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 [5]. 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 the thesis is organized as follows. Chapter 2 presents background and related work on both *blockchain* and *virtual coordinate systems*. We explain the implementation details of Newton and the attacks to perform in Chapter 3. Chapter 4 describes the metrics used in the measurements and shows the validation of our implementation, as well as the results obtained from our experiments using Newton. Finally, Chapter 5 presents the conclusions and future work.

Chapter 2

Background

This chapter presents background and related work, including research on both *blockchain* and *Virtual Coordinate Systems* (VCS). We begin by addressing blockchain-based systems in Section 2.1, focusing, in particular, on the level of *decentralization* achieved in practice by blockchain-based systems/cryptocurrencies, and hence motivating the use of VCS to enforce diversity in blockchains (namely to further decentralize permissionless consensus protocols). In Section 2.2, we describe VCS and focus on their *security* aspects in Section 2.3.

2.1 Blockchain

Blockchain [6] 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 [6] 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. Some entity by the name of Satoshi Nakamoto published the original Bitcoin protocol [7], and created the Bitcoin network in 2009. Bitcoin was the first decentralized digital currency, and, ever since, blockchain technology has increased its popularity. A popular alternative to Bitcoin is Ethereum [8, 9], also based on blockchain technology. However, unlike Bitcoin, Ethereum enables the deployment not only of digital currency (*ether*) but also of decentralized apps and smart contracts.

One of the key properties that blockchain [6] has to offer is the high level of trust that can be derived from a decentralized system, which allows for transactions between peers who do not need to trust each other, while removing the need of a trusted third party to mediate those transactions. Since this trust is closely tied to the lack of a centralized control over the system, we next survey a series of measurement studies on both Bitcoin and Ethereum, regarding their level of decentralization.

2.1.1 Decentralization in blockchain-based Cryptocurrencies

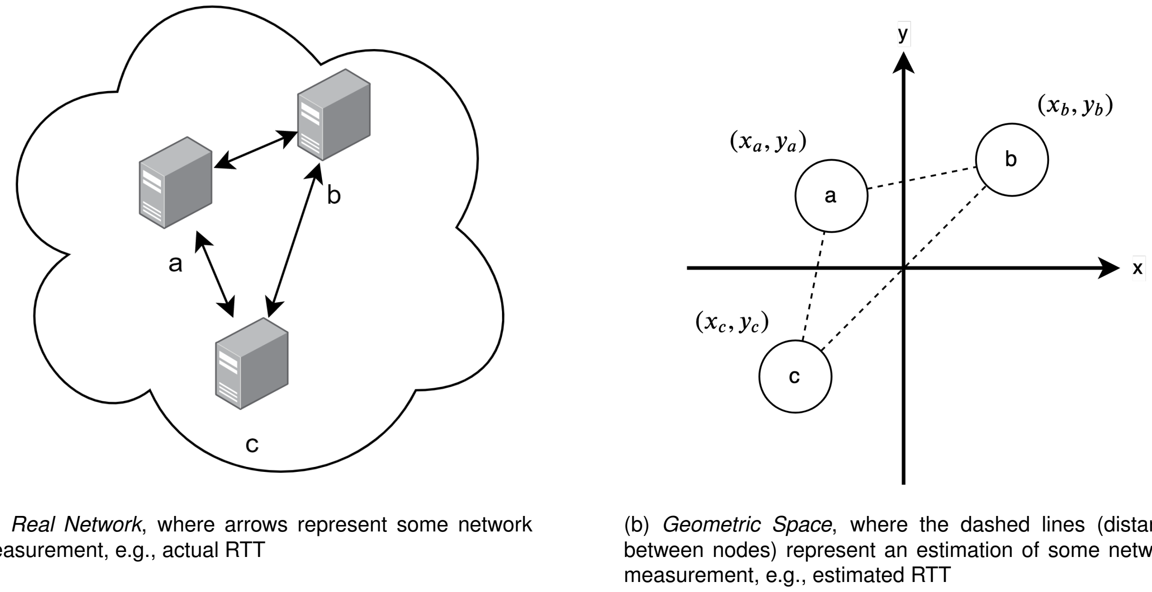
Blockchain-based systems have decentralization as one of their key underlying properties. However, a lack of diversity is currently observed in overlays such as Bitcoin or Ethereum, where a significant fraction of the participating nodes is concentrated in a few data centers, reducing decentralization.

In [1], Gencer *et al.* present a measurement study on decentralization metrics in Bitcoin [7] and Ethereum [8, 9]. 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 same 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, as explained before. The authors try to evaluate if the participants of these networks are using more powerful hardware to succeed more often in solving the cryptopgraphic puzzle, with the end result that the mining process becomes 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, during the ten months. These results give us a sense of the centralization levels 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.

2.2 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.



(a) *Real Network*, where arrows represent some network measurement, e.g., actual RTT

(b) *Geometric Space*, where the dashed lines (distance between nodes) represent an estimation of some network measurement, e.g., estimated RTT

Figure 2.1: Correspondence between the real hosts and their virtual coordinates in the geometric space

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, as shown in Figure 2.1. In this geometric space, all the hosts have coordinates, and the distance between them represents some network performance measurement, like latency or bandwidth. Latency is usually considered as the round trip time (RTT) between hosts, and bandwidth is the maximum rate at which information can be transferred in the network.

In contrast, in peer-to-peer file sharing, bandwidth between a peer looking for some file and all the peers that have that file is favored over latency.

In our case, location-awareness can enable the blockchain protocol to know how to choose nodes in a way that enforces geographical diversity and increases decentralization. Therefore, in order to achieve location-awareness, latency is the most relevant metric to be modeled by our implementation of the VCS system, to be embedded in the blockchain scenario. Hence, all the approaches we present next focus on latency.

VCS approaches have to guarantee sufficient accuracy for them to be useful in predicting distances through virtual coordinates. As a way to accomplish that, these VCS algorithms tend to minimize some error function when computing a prediction of these coordinates, with the meaning that a value of 0 in this error function stands for a perfectly accurate coordinate.

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.

In Section 2.2.1, we give an overview of existing VCS systems and approaches, and in Section 2.2.2 we focus on Vivaldi [3], one of the decentralized approaches. We explain Vivaldi in more detail than the other decentralized systems because it was subsequently extended to accomplish a robust decentralized VCS called Newton [4], which is an important component of our work. Afterwards, we will explain Newton, when talking about security in VCS, and discuss the reasons why we decided to focus on this approach. Finally, in Section 2.2.3, we discuss the limitations of the approaches shown.

2.2.1 Existing Approaches

In this subsection we present various existing approaches of virtual coordinates.

Landmark-Based Systems The majority of landmark-based approaches consist of systems with two types of nodes, the fixed landmarks and the regular hosts. Concretely, these landmarks are nodes that compute their own coordinates after making latency measurements between all of them. The regular hosts, in turn, are all the other nodes in the network system. The operations of regular hosts when computing their coordinates include getting the coordinates of the landmarks while measuring the latency to each one of them. With both the latency measurements and the coordinates of the landmarks, the regular hosts can then compute their own coordinates.

There are several landmark-based systems, but since our primary focus is on the decentralization aspect of a network, we do not study them in detail. An overview of some landmark-based systems may be found in [10]. The reason why we do not focus on this class of systems is that, in most blockchain systems, all nodes are indistinguishable, and therefore this is not compatible with relying on a fixed and *a priori* set of trusted nodes (landmarks). Also, an unavoidable drawback of landmark-based approaches is that landmarks represent a single point of failure and limit the scalability of these systems.

Decentralized Systems Decentralized VCS systems aim to accurately predict the latency between nodes, through the creation and maintenance of virtual coordinates, without using fixed infrastructure nodes. The essential features in the design of most decentralized approaches are a *reference/neighbor set*, which is essentially a small set of topological neighbors in the network, a *distance prediction mechanism*, such as the Euclidean distance, and an *error minimization technique* of some distance error function.

For a node x to determine its coordinate, a reference set is first selected. Then, the node x queries the nodes from the reference set, and based on the reported information, x computes its coordinate. This information may include metrics like *local coordinates*, *measured RTT*, and *local error*, which represents the confidence of a node on its local coordinates. Finally, after a node has determined its coordinate, it refines it by periodically querying the reference set.

We now present various decentralized virtual coordinate systems, more concretely, Big-Bang Simulation [11], PIC [12], and PCoord [13, 14], leaving Vivaldi [3] for the next section.

Big-Bang Simulation (BBS) [11] simulates an explosion of particles that will be traveling in the Euclidean space under the effect of the potential force field. The particles represent the geometric image of the network nodes in the geometric (Euclidean) space, and they are initially placed at its origin.
This proposal is based on the simulation of Newtonian mechanics, where nodes are seen as particles influenced by the force field induced between different nodes. As a way to reduce the total system error, each pair of particles either pulls or repulses each other.
This force field is derived from the total embedding error, which is equal to the potential energy error. The embedding error of the distance between each pair of particles drives the force field to make them pull or repulse each other, to minimize their pair embedding error. The force affecting a specific particle is equal to the sum of the induced forces by the other particles. If the force is positive, it pulls the two particles together, whereas if it is negative, it repulses them apart.
The BBS method is based on several calculation phases performed sequentially. In the first phase, the induced pair force field is equal to the difference between the geometric (Euclidean) and the network pair distances. Also, BBS starts with an insensitive, less optimal potential function to minimize, the simple squared error. As it moves along to the next phases, it uses increasingly sensitive potential functions. Then, during the calculation phases, the particles remain traveling in trajectories, tending to reduce the potential energy of the system. At the end of each phase, the system approximately achieves an equilibrium point. This equilibrium point corresponds to the potential energy minimum, and it serves as the initial position of the particles in the next phase.
If the particles were only under the normal force field effect, they would move too fast and oscillate forever around their correct position. Therefore, a friction force exists alongside the normal force field. This friction force dissipates some of the energy, slowing down and constraining the movements of the particles, allowing them to stabilize after a while in positions of minimum potential energy.

Practical Internet Coordinates (PIC) [12] In this other proposal for a virtual coordinate system, nodes compute their coordinates only once when joining the system. Additionally, any node that knows its coordinates can be used as a reference for new nodes. For a new node to compute its coordinates, it first measures the latency to a set of reference nodes and obtains the coordinates of each reference. Then the new node runs the Simplex Downhill optimization algorithm [15], to minimize the errors of the predicted distances between the node and the references, as a way to compute its coordinates.
To choose the reference set, PIC uses a *hybrid strategy*. This strategy is a mix between two other strategies, where the first is a *random strategy*, which picks some of the nodes randomly; then the second one is the *closest nodes strategy*, which picks, among the rest of the nodes, the closest to the new node in the network topology.
One challenge of the hybrid strategy is to find the closest nodes in the network because the new node does not know its coordinates yet. For that, the new node uses the PIC algorithm with the random strategy to compute an estimate of its coordinates. Then uses this estimate to find the closest nodes, and, in the end, the new node uses the PIC algorithm with the hybrid strategy to refine its coordinates.

In *PCoord* [13, 14], each node measures its round trip latency to some reference set of nodes and gets their coordinates. Then, updates its coordinates in a way that minimizes the squared normalized difference between the measured and the computed distances, using the Simplex Downhill algorithm [15]. In particular, as for all the VCS systems we presented, the distance metric corresponds to the measured latency, and the computed distance corresponds to the distance between the coordinates of the nodes in the geometric space.

PCoord has three schemes, *RandPCoord, ClusterPCoord* and *ActivePCoord*. The first two schemes assume the existence of a reference set, acting as bootstrap nodes to guide joining nodes. In *RandPCoord*, the reference nodes are chosen randomly, while in *ClusterPCoord* each joining node exploit the topological information, separating the existing nodes in clusters based on their coordinates. Then, the joining node chooses a random node from each cluster, to obtain a well-distributed reference set. The third scheme, *ActivePCoord*, is an iterative process, where each node starts in the origin of the geometric space. Then, to refine its own coordinates in each iteration, the node uses triangulated distances to choose a well-distributed reference set, to whose members it measures the distance.

Lehman *et al.* incorporated the following mechanisms in PCoord [14], to allow the system to converge to low prediction errors in faster time, as well as to create robustness against faulty or misbehaving nodes and triangle inequality violations in the network paths. (1) A weighted loss function is used to give a higher weight to the reference nodes' coordinates with higher prediction accuracy. (2) The loss function also has a weighted ”resistance” that helps stabilize the coordinate convergence process, avoiding oscillation. (3) A threshold-based ”damping” mechanism helps to avoid instability derived from faulty reported information. For that purpose, the distance a node has to move towards new coordinates is dampened by a factor inversely proportional to the fit error of the reference nodes' coordinates and distances.

2.2.2 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 [16–18], 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.

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 $x_i = x_i + \delta \times (RTT_{ij} - ||x_i - x_j||) \times u(x_i - x_j)$;
```

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.

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.

Dabek *et al.* [3] propose the use of height-vector coordinates, as a variant of Euclidean coordinates, to better model latencies on the internet. A height-vector consists of a Euclidean coordinate augmented with a height. The Euclidean part represents the latencies proportional to the geographic distance, and the height represents the time to travel the access link from the node to the core. If the forces exerted over a node from various other nodes cancel each other out, then the node will not move when in a normal Euclidean space. However, in the height-vector, when that happens, the height forces push the node away or towards the Euclidean plane.

Vivaldi requires no fixed infrastructure or landmarks, thus accomplishing decentralization; it can piggy-back network sampling on the application's traffic, which avoids overhead from extra communication traffic; and is reactive to changes in the network. 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.

2.2.3 Limitations and Discussion

In this section, we discuss known limitations of the coordinate approaches presented above, and how they compare to each other. Then in Section 2.3, we will focus on their security aspects.

Most VCS use the Euclidean space as the virtual geometric space of choice, and, as we referred before, 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, as shown in several studies [16–18], because internet traffic does not always follow the shortest path. In short, VCS systems need to be designed to be "Triangle Inequality Violation", or TIV-aware, since, when in the presence of TIVs, VCS systems make edges shrink and stretch in the geometric space, which results in oscillations and consequently in large prediction errors.

Approaches such as PIC [12], which has a security mechanism relying on a TIV-based test, might degrade the system's performance in benign settings, because of the assumption that triangle inequality holds on the network (as we will detail in Section 2.3.2 when discussing the security approaches in VCSs.

Taking into account the existence of TIVs, Chen *et al.* propose *Pharos* [19, 20], a multi-set coordinates scheme that contributes to a reduction of the impact of TIVs on distance predictions. Pharos aims to improve accuracy in simulation-based algorithms (SBA) like BBS [11] and Vivaldi [3]. An SBA system simulates a physical system, where the nodes' coordinates are then computed by minimizing the energy state of that system. In particular, when Vivaldi operates in the presence of TIVs, it ends up stuck in endless oscillations, with the nodes not converging towards correct and stable coordinates [10]. Chen *et al.* used Vivaldi as a representative SBA system and studied it to find out that the range of distances between peers has a significant impact on the performance of the system. As a solution, Pharos splits the nodes into clusters with different ranges of distances, and create two coordinate sets where one of them has *local coordinates* and the other has *global coordinates*. The former is the most accurate for predicting short distances and the latter for longer distances. This way, when predicting intra-cluster distances, the local coordinates are used, while the global coordinates are for inter-cluster predictions. Furthermore, Pharos uses Vivaldi's algorithm for all distance predictions. Therefore, the authors compared the performance of Pharos with the one of Vivaldi, obtaining results showing that Pharos achieves higher accuracy than Vivaldi. Similarly, Phoenix [21] also addresses TIVs and accuracy. Its authors designed it to be a decentralized and improved version of the landmark-based system IDES [22].

Finally, in order to adapt to topology changes, the nodes of these systems keep updating its coordinates during their stay in the system, with the intuition of minimizing the difference between real latencies and estimated latencies to other nodes. The minimization referred can be executed in several ways. Systems like PIC [12] and PCoord [13, 14], as they are based on multidimensional minimization, can be caught by some local minimum, different from the global one, close to the starting value. This characteristic makes these systems very sensitive to the initial coordinates of the nodes. It allows for the possible attribution of different coordinates to the same node. They are usually slow to converge and sensitive to high error measurements. Concretely, the multidimensional minimization algorithm running is the Simplex Downhill [15]. On the other hand, simulation techniques like spring relaxation used in

Vivaldi [3] are not sensitive to the initial conditions of the system, tolerate high error nodes, and are inexpensive to compute, providing a suitable technique for decentralized systems.

	TIVs Aware	Error Minimization Technique	Year
BBS [11] - Big-Bang Simulation	✗	Minimize Potential Energy of the System	2004
PIC [12] - Practical Internet Coordinates	✗	Simplex Downhill	2004
PCoord [13, 14]	✓	Simplex Downhill	2004, 2006
Vivaldi [3]	✗	Minimize Squared Error	2004
Pharos [19, 20]	✓	Minimize Squared Error	2007
Phoenix [21]	✓	Minimize Weighted Sum of Squared Errors	2007, 2009

Table 2.1: Decentralized VCS - Overview

Table 2.1 contains an overview of the different decentralized virtual coordinate systems we have presented. The column *TIVs Aware* shows if the system is aware, and consequently, can deal with the existence of TIVs in the network. Then, the *Error Minimization Technique* shows the error minimization technique that the VCS uses.

2.3 Security in Virtual Coordinate Systems

One of the most critical limitations of VCS systems is security, especially as the majority of these systems assume honest nodes that fully-cooperate and report correct information to each other. Thus, when in benign settings, VCS systems have good accuracy. However, the system's performance rapidly deteriorates when under attack by nodes acting as *insider attackers*, that is, malicious nodes with access to the same data as legitimate nodes.

We start by presenting known attacks to VCS systems in Section 2.3.1, and in Section 2.3.2 we show existing security mechanisms. In the same way, as with Vivaldi above, we explain Newton in more detail than any other VCS security mechanism, because it is a crucial part of our work. Newton is explained in Section 2.3.3.

2.3.1 Attacks on VCS

In this section 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*, as described previously.

Kaafar *et al.* [23, 24] were the first to identify some basic attacks that are effective against coordinate-based systems and showed how vulnerable Vivaldi is against them. They classified the attacks in four different types: *isolation attack*, which aims at convincing the victim node that it is in an isolated location in the network; *repulsion attack*, which has some lying node trying to reduce its attractiveness, convincing the other nodes that it is far away; *disorder attack* aims at introducing chaos as a form of denial of service (DoS) attack; and *system control*, where an attacker tries to take control of a reference node or to become one, and in this way be able to influence the coordinate maintenance process.

Zage *et al.* [25] also identified some basic attacks, but in a more 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.

Oscillation Attack, like the disorder attack, introduces chaos in the coordinate system. Attackers choose random coordinates to report and randomly delay measurement probes. Consequently, the remaining nodes have to keep updating their coordinates, not converging to their correct positions.

Additionally, Chan-Tin *et al.* [26, 27] 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.

2.3.2 Security Mechanisms

When trying to make VCS systems robust to attacks, there are two kinds of security mechanisms. The first one relies on landmarks, while the second one is decentralized. For landmark-based approaches, the high-level intuition is that their security techniques focus on trying to secure the set of trusted landmarks (reference nodes) used in the coordinates computations. This makes it simpler to reason about their robustness to attacks than in decentralized approaches. However, this comes with a cost in scalability and single points of failure. In contrast, decentralized systems make fewer assumptions and can scale better, but can be more susceptible to attacks, as any node can be an insider attacker.

As already referred, we direct our attention to decentralized systems. As such, for landmark-based security mechanisms [28, 29], we only detail a single system as an example of this type of approach.

Saucez *et al.* [28] define a reputation-based system that associates a reputation with each node of

the system. The reputation of a node is directly proportional to how reliable the node is. It is associated with the node's past behavior and with how long it has been in the system.
Specifically, a node A's reputation is a combination of trust from all the other nodes towards A, where *trust* represents an expectation on the future behavior of A, based on experiences observed in the past of node A.
As a way of evaluating the reputation of every node in the system, two new entities were created: the *Reputation Computation Agent* (RCA), a certificating agent that is used to construct a reliable reputation for any node in the distributed system; and the *Surveyors*, chosen nodes with the job of estimating the reputation of the remaining nodes. Every node in the system have a few surveyors assigned. When evaluating a new reputation for some node A, the RCA uses the trust surveyors have in node A and the trust it has in the set of surveyors that evaluate node A.
This reputation-based approach was then incorporated into Vivaldi, to make it more robust to attacks. The resulting system was named *Reputation-based Vivaldi* (RVivaldi), with the idea of keeping Vivaldi's standard procedure when the neighbor nodes are reliable, but limiting the changes of coordinates when they are not reliable. Despite becoming more robust to attacks, RVivaldi has the drawback of introducing a single point of failure and trust, the RCA. Furthermore, RVivaldi introduces a substantial overhead when calculating a node's coordinates, because when that is happening, the node will also take into consideration its neighbors' reputation, besides their coordinates. After that calculation, this node still needs to contact the RCA to update its own reputation.

Decentralized Approaches:

PIC [12] was one of the first systems to introduce a security mechanism against malicious behavior. To remove malicious nodes from the reference set, PIC relies on a test based on *triangle inequality*. In particular, a malicious node is identified when it is perceived to be causing a violation of the triangle inequality, $d_{AC} \leq d_{AB} + d_{BC}$, where d_{xy} is the distance between the nodes x and y. However, as discussed in Section 2.2.3, the *triangle inequality* is regularly violated by latencies in the internet. As such, the PIC's security mechanism might lower the accuracy of the system in scenarios without malicious nodes.

Regarding *PCoord* [14], as we briefly described in Section 2.2.1, it already incorporates mechanisms to tolerate faulty or misbehaving nodes.

Zage *et al.* [25, 30] explored the performance of decentralized virtual coordinate systems (VCS) in adversarial environments and proposed a solution that uses *outlier detection* to make them more robust against insider (or Byzantine) attacks. More specifically, nodes that report data that is very different from what the other nodes are reporting are likely to be malicious nodes, and therefore their reports should be discarded. For that reason, each node must first learn what good behavior is regarding the behavior of the nodes in the system, so that they can recognize malicious behavior and discard the corresponding updates, which are considered outliers. To this end, the following metrics the are analyzed.
A 3-tuple of <remote error, change in remote coordinates, latency> is used to generate spatial outlier statistics and a 5-tuple of <remote error, local error, latency, change in remote coordinates, change in local coordinates> to generate temporal outlier statistics. Then, a *spatial outlier detection* checks if the

metrics reported from one node are consistent with the ones from its neighbor nodes. Every time an update is received from a neighbor, the detection mechanism is executed, given the 3-tuple described above. The last u updates are stored in a queue, with u being equal to the size of the neighbor set. The *Mahalanobis distance* is then calculated between the last observation tuple and the centroid of the last u updates in the queue. For a new update to be accepted, the distance previously computed must remain under a *spatial threshold*.
In addition to detecting spatial outliers, a *temporal outlier detection* checks the consistency of a given node over time. To this end, the outlier detection takes into consideration the previously described 5-tuple. For each neighbor node, a temporal centroid is computed using incremental learning. The newly received metrics are then compared to the mean, standard deviation, and sample count for that node, using a simplified *Mahalanobis distance*. For the update to be accepted, the distance must not exceed a *temporal threshold*.
Both the *spatial outlier detection* and the *temporal outlier detection* are combined to identify outlier updates, and consequently, not including them in the new coordinate computations. An update is accepted if both the *spatial threshold* and the *temporal threshold* are not exceeded.

Wang *et al.* [31] propose a security mechanism that has two different approaches. One for securing coordinate computations and another for securing delay measurements. The authors show that a Byzantine fault detection algorithm can protect the coordinate computation, detecting malicious nodes that lie about their coordinates. To that end, either *Byzantine Fault Tolerance* (BFT) [32] or *Byzantine Fault Detection* (BFD) [33] can be used. For instance, Wang et al. [31] show that the *PeerReview* [33] accountability protocol, which is a BFD techique, can be used to secure how coordinates are computed. Additionally, the authors propose a triangle inequality violation (TIV) detection algorithm to secure the delay measurements. Following the intuition that delaying measurements is likely to result in severe TIVs, and that edges causing them are likely to have a low prediction ratio ($predicted/measured$), Wang et al. [31] recur to a TIV alert technique that uses the prediction ratio to identify malicious nodes.

Veracity [34] is a fully decentralized service designed with the intent of securing network coordinate systems (like Vivaldi [3], for example). As we already explained, in decentralized VCS, any participating node (referred to as the investigator) must securely update its coordinates. To accomplish that, it periodically requests the coordinates from some other node (referred to as the publisher), while measuring the RTT between both.
Unlike these existing approaches, Veracity does not require trusted parties nor *a priori* secrets. In most implementations, the investigator has a fixed neighbor set where publishers have to be pre-assigned. Instead, Veracity has a distributed directory service, implemented using DHTs (Distributed Hash Tables), and it is used by each node to select random publishers on demand. To become robust to malicious nodes, Veracity goes through a two-step verification process:

- The first step is when the publisher coordinates verification, which detects nodes that concurrently report false coordinates. During this step, a verification set (VSet) is assigned to each publisher. Each VSet member independently computes the accuracy of the coordinates the publisher reported, and a majority has to accept the publisher's coordinates so that the investigator does not discard it.

- The second step is the candidate coordinate verification. After the investigator knows that the publisher's coordinates are valid, it must detect if the publisher delayed the RTT probe response. To accomplish that, the investigator computes a candidate coordinate using the publisher's coordinate validated in the first step and the measured RTT. Then, estimation errors among the investigator and a randomly chosen set of peers (RSet) are computed for both the current and the candidate coordinates of the investigator. Finally, the investigator updates its coordinate to the candidate coordinate only if this new coordinate results in a small increase in estimation error (i.e., below some threshold).

One of the positive characteristics of this system is that it does not make a distinction between intentionally falsified coordinates and inaccurate coordinates due to limitations of the embedding process. That ends up preventing the use of inaccurate coordinates.

2.3.3 Newton

Newton [4] 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. The known attacks Newton considers are the *inflation, deflation, oscillation, frog-boiling and network partition* attacks, that were already explained in Section 2.3.1. The main underlying idea is to have all the participating nodes following Newton's laws, 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.

2.3.4 Discussion

In this section, we compare the security mechanisms previously presented, with a particular focus on Newton [4], which we believe to be the most complete and robust approach that we can use in our work. We have briefly described some landmark-based defenses. However, as we previously referred, we are more interested in fully decentralized approaches, since they do not require *a priori* trust in a predefined set of nodes.

An important downside of all the approaches, except PCoord and Newton, is that they remain sensitive to the presence of TIVs in the network. The resistance and damping mechanisms in PCoord reduce the instability from path anomalies, thus allowing the system to be more robust to triangle inequality violations in the network. As for Newton, it leverages the threshold for the **IN3**, in such a way as to not let TIVs reduce Newton's performance in benign settings. Moreover, most mechanisms introduce overheads during the process of computing coordinates. Seibert *et al.* [4] wanted to preserve Vivaldi's characteristics of efficiency and low-cost, so no further network communication is added, and the amount

[1] How stiff the spring is.

of computation and memory needed for the use of the invariants is minimal. This way, Newton does not add significant overhead to Vivaldi's operation.

Chan *et al.* [26, 27], first show that outlier detection is not a good approach to deal with slow attacks like the frog-boiling and network partition attacks that they propose. Concretely, outlier detection-based systems, like [25], would learn the behavior of these attacks as good behavior and fail at dealing with the attacks. Then, Chan *et al.* also showed that Veracity [34], despite not being based on outlier detection, but instead, in a decentralized reputation mechanism, also fails to detect the "small-step" tampered updates.

However, Newton [4] and Becker et al.'s work [35] were designed with the *frog-boiling* and *network partition* attacks into consideration, so they both are capable of mitigating these advanced attacks in addition to the basic ones. Becker *et al.* [35] designed the method to be also capable of resisting complex attack strategies, where several single attacks are launched. In short, Becker et al.'s work [35] detects the attacks with supervised machine learning techniques, such as classification decision trees, that must be trained to learn the difference between benign and malicious data. This need for training raises problems such as the need to re-train for deployment in another system, and the fact that labeled data and ground truth to train the model can be hard to get. Additionally, it is infeasible to use this approach in a real system in a distributed and local manner, as this requires every node to train the algorithm locally and to have a different decision tree, which can be resource consuming and result in different results for different runs. As for Newton, no training is needed, and it can mitigate the attacks by resorting to physical laws, with each node making the computations locally.

Summing up, Newton came to surpass some of the problems of all the above systems. Specifically, Newton and Becker et al. [35] are the tested security mechanisms that, in addition to the basic attacks, also mitigate more advanced attacks like the *frog-boiling* and *network partition* attacks. Wang et al. [31] has a high cost for computing public-key cryptography and some other operations, which introduces a substantially higher overhead than Newton. Consequently, it is also less desirable than Newton.

Table 2.2 contains an overview of the different decentralized security mechanisms we have presented. The *Target VCS*, shows to which VCS the security mechanism might be applied. Instead of a VCS name, the *Generic* term can be used to denote that the security mechanism may be applied to any VCS. In the *TIVs Aware*, we can see if the mechanism helps to deal with TIVs. Then in the *Advanced Attacks*, as all mechanisms can deal with the basic attacks, we can see if they can mitigate the frog-boiling and the network partition attacks.

	Idea	Target VCS	TIVs Aware	Advanced Attacks	Year
PIC security [12]	Test based in TIVs.	PIC	✗	✗	2004
PCoord security [14]	”Damping” mechanism to avoid instability derived from faulty reported information.	PCoord	✓	✗	2006
Zage *et al.* [25]	Outliers Detection.	Generic	✗	✗	2007
Wang *et al.* [31]	Byzantine Fault Detection and TIV detection algorithm.	Generic	✗	-	2008
Veracity [34]	Voting scheme for malicious updates detection.	Generic	✗	✗	2009
Becker *et al.* [35]	Supervised Machine Learning detection techniques.	Vivaldi	✗	✓	2011
Newton [4]	Safety Invariants based on Newton’s laws of physics.	Vivaldi	✓	✓	2014

Table 2.2: Security Mechanisms - Overview (”**-**” is for unclear or unknown cases)

Chapter 3

Implementation

In this chapter, we start by describing our approach in Section 3.1, and in the following ones, we go through the main phases of our implementation process. As we worked in a simulated environment, we present the core concepts of Corten [5], the simulator used, in Section 3.2. Our implementation is then divided into two major parts: the Virtual Coordinate System and the attacks to be performed on this system. From all the VCS we showed in Chapter 2, the one we chose was Newton, and we present the relevant aspects of its implementation in Section 3.3. Lastly, the attacks on VCS and the concept of a new attack strategy for a scenario where some cluster of the network tries to split its coordinates are explained in Sections 3.4 and 3.5, respectively.

3.1 Problem Description

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. From the comparative overview of secure decentralized VCS systems done in Section 2.3.4, we conclude that Newton [4] is, according to our assessment, the most suitable choice for this purpose.

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 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. The measurements and results of these experiments are shown in Section 4.4.

- **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. Section 3.5 presents the design of this new attack and Section 4.5 its evaluation.

3.2 Simulation Environment

The implementation is conducted in a simulation environment, which allows us to remove a lot of the complexity of a real execution, since simulators avoid the details of the network implementation, replacing it with a queue-based model, and presenting a simplified interface that abstracts the network [5]. Consequently, we get a controlled environment where we can focus more on the protocol and algorithms we are working with. This way, we can simulate a networked system, with nodes spread across locations connected through wide-area latency links, running the protocols and attack strategies described in this thesis, as we will demonstrate in our evaluation in Chapter 4.

Regarding the simulator to use, we picked the ***Corten Simulator*** [5], 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. In particular, Corten offers the following main features:

- Network modeling of *latency*, *jitter*, *packet loss*, and *link asymmetry*. Specifically, *latency* is given by a matrix of inter-host internet latencies, *jitter* can be used with a uniform distribution, a lognormal distribution, or it can be disabled and *packet loss* is defined by the percentage of packets lost, with a value between 0.0 (0%) and 1.0 (100%).

- *Process Asynchrony*, which can be disabled, according to the authors of [5], is a feature that only Corten offers. It allows for making simulation executions more realistic, simulating the fact that local method calls can be executed before or after the expected time, due to various reasons.

- *Process Churn*, which is useful to investigate application behavior under faults in the network.

- *Checkpointing*, which allows to save the state of the system and re-run it from that point several times.

Among the important characteristics of Corten is the fact that it prevents programmers from creating unrealistic situations, even if accidentally, such as programming applications that allow some process in one node to call local methods from another process/node in an atomic manner (as this would eliminate the time needed for messages to go from one node to another, which is not possible in a real execution). The way to prevent this is to only allow processes to communicate through messages, thus enforcing isolation between processes (applications). Additionally, Corten is intended to be *efficient*, specifically memory and time-efficient, as well as present good scalability, supporting simulations of systems with thousands or even millions of processes in a single machine, like a laptop, which was our case. In order to achieve this, the Rust programming language was used in the implementation of Corten, as it is developed with efficiency concerns, especially regarding memory management. Corten was even tested with up to one million nodes, far more than what we use in our simulated scenarios, and additionally, the authors compared it with a state-of-the-art simulator, PeerSim [36], and revealed that Corten scaled better both in terms of time and memory.

As any simulator should, Corten makes sure that every experience is *reproducible*, and for this, it relies on the ability to reproduce the output of its Random Number Generator (RNG), more specifically using a pseudo-random generator. This RNG receives an initial value, called seed, and generates random values for different seeds, but the same values for the same seed. This makes it possible to run experiments multiple times and get always the same course of action, which, in the context of our work, means that we can, for example, guarantee the same course of updates of virtual coordinates for the same network until the point in time where different attacks occur. With this concept in mind, we take care of using the API made available by Sequeira *et al.* [5], to maintain the reproducibility capabilities.

Lastly, the configuration of the simulations is described in a YAML file, where both the simulation parameters, as well as the user-developed application parameters are defined. The simulation's configuration includes, among others, parameters related to the number of processes/nodes, the simulation time, the time between consecutive probes sent, level of asynchrony, network latency, jitter and packet loss, and seed value for the RNG. In the next section, we will also mention the application's parameters.

3.3 Newton

The Newton protocol [4] was explained in Chapter 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. The core algorithm from Vivaldi [3] was explained in Section 2.2.2 and the changes that turn it into Newton, mainly the *security invariants*, in Section 2.3.3. We next provide more detail on our implementation of these protocols.

This implementation leverages the Corten simulator, which 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. Furthermore, the main parameters of the application are specified in the configuration file mentioned previously in Section 3.2, including the *number of neighbors*, i.e., the number of elements that each node has in its neighbor set, which was explained in Section 2.2.2, and *security invariants*, which allows us to turn on/off the security mechanisms, and consequently run either Vivaldi or Newton.

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.

3.3.1 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. Furthermore, it should be easy to manage these coordinates and make the RTT estimates. As mentioned in [3], the simplest solution is to use n-dimensional coordinates and the standard Euclidean distance function, as per Equation 3.1, where x and y are the coordinates of the two nodes, and n is the number of dimensions.

$$d\left(x,y\right)=\sqrt{\sum_{i=1}^{n}\left(y_i-x_i\right)^2} \tag{3.1}$$

Additionally, Seibert *et al.* [4] 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 The neighbor set was described in Section 2.2.2, where we explained that 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.

In a real setting, however, without the global knowledge of the pairwise latencies of the network, nodes cannot know directly which are the closest nodes to them. Therefore, we propose that both new nodes and all nodes at the beginning of the system's execution use the Vivaldi/Newton algorithm, but only with randomly chosen neighbors. This way, they can compute an estimate of the coordinates, and then update the neighbor set to include the closest nodes found with this estimation. Afterwards, with the continuous updates in the system, the coordinates will be refined.

Coordinate Update Regarding the updating of coordinates in the system, our implementation directly reflects the specification given in Section 2.2.2, with its core rationale in Algorithm 1.

3.3.2 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 (as we explained in Section 2.3.3).

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 3.2, 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.

$$c = \frac{\sum_{p=1}^{n} x_p + f_{ij}}{n} \tag{3.2}$$

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 2.3.3, when discussing the detection of extraneous indirect forces introduced in the system by physically close nodes. However, we must add that, to compute the expected new coordinates for the currently updating remote node, we take into consideration the last forces of all the neighbors (close and random) that the local node has experienced. Specifically, the procedure of IN2 as described in Section 2.3.3 is done for each neighbor of the local node and the resulting projections are summed up, and the result gives us the expected new coordinates for the updating remote node. This value is then compared with the actual reported new coordinates from the remote node, and, if the distance between the two is greater than some IN2 threshold, the local node rejects the update.

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 3.3, where t is the threshold for the number of deviations.

$$m_j > \tilde{f} + t * D \tag{3.3}$$

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 [4]. Hence, the threshold for the close neighbors will be higher than the threshold for randomly chosen nodes.

3.4 Existing Attacks

Next, we provide further detail about how we implemented the concepts behind the different known attacks, possible in the context of VCS (*inflation, deflation, oscillation, frog-boiling* and *network partition* attacks), which were previously presented in Section 2.3.1. There, we also explained that to manipulate other nodes and perform attacks in VCS systems, an attacker can influence the coordinate computation of other nodes by reporting false *coordinates* and/or *local errors*, and influence the *RTT* computed by other nodes by delaying their measurement probes.

The main difference between a benign node and an attacker is the extra logic to perform the attacks. When an attacker receives a probe from another node, it will first check if the current simulation time is higher than the predefined attack starting time. If this is the case, it will do whatever is defined for the current type of attack to perform. The configuration file, which we mentioned in Section 3.2, contains the definition of both the *attack starting time* and the *attack type*, but also the *percentage of attackers*.

We next explain the core of our implementation for each attack to achieve their objectives. Firstly, however, a technique that is common to the five types of attack is the behavior regarding the local error report. Specifically, if an attacker has a high local error, it may lie and report a low error value. The objective is to cause the impression of having stable coordinates and maximize their impact on the computations of the nodes receiving the report. In Algorithm 1, presented and explained in Section 2.2.2, regarding Vivaldi/Newton's updating algorithm, we can see that a lower remote error e_j results in a higher sample confidence w (line 1), which will allow for a higher time-step δ (line 4). Consequently, an updating node receiving these reported values will use a bigger fraction of the force that the attacker is applying to it (line 5).

Now we present the specifics of each attack:

Inflation Attack: Attacker node lies about having coordinates far away from the origin, which pulls benign nodes from their correct coordinates. Specifically, when replying to a probe, the node will generate coordinates with a random direction and positioned further from the origin than any other node. Consequently, it is an attack on accuracy.

Deflation Attack: Prevent benign nodes from updating towards their correct coordinates – as such, this is also an attack on accuracy. To accomplish it, the attacker can report coordinates close to the origin, or coordinates that make the victim think that it does not have to move, i.e., coordinates that result in a distance to the victim's coordinates equivalent to the actual measured RTT. We implemented this attack according to the former, and again the direction from the origin is random.

Oscillation Attack: Attacker node lies about its coordinates and delays probes up to 1 second, both in a random way, intending to create chaos in the system. This one is not only an attack on accuracy, but stability as well.

Frog-Boiling Attack: Another attack on both accuracy and stability. Attacker lies in small amounts, accumulating a large error over time. At the start of the experiments, before any updates, the attacker chooses a random direction. Afterwards, it responds to each consecutive probe with fake coordinates moving along that direction, but just a small amount at a time. Specifically, in our implementation, it moves 0.25ms between consecutive probes.

Network Partition Attack: Same as the Frog-Boiling attack, but with multiple nodes colluding together. Pairs of nodes move in opposite directions and colluding attackers are paired randomly at the beginning, just before the system starts updating. Afterwards, the colluding nodes update each other with their coordinates, so that they can synchronize and start moving in opposite directions, in the same way as in the Frog-Boiling attack.

3.5 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 model an advanced attacker with insider knowledge about Newton's protocols and defense mechanisms, which can leverage that knowledge regarding the calibration of security parameters. Given that Newton is supposed to guard against this level of attacker [4], we set the goal of trying to separate the previously mentioned cluster into multiple groups *to the extent possible*. In other words, even if it is not possible to fully deceive the system and convince that the cluster nodes are blended with the rest of the network, we want to at least minimize the perceived level of clustering of the nodes under control of the attacker.

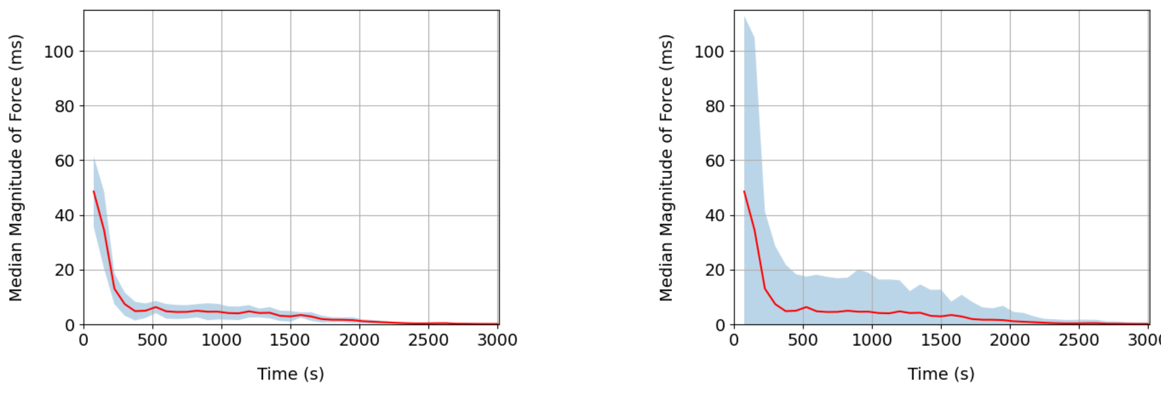
Our approach was devised while taking into account the *security invariants* of Newton, and therefore we try to surpass these one by one. In particular, there are three *security invariants* that Newton tries to validate, and that consequently need to be circumvented by our strategy. With this in mind, 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 3.3.2 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 Figure 3.1, we present the *median magnitude of force* applied to a single node over time, in a benign setting, which we will analyze to understand how to dodge the IN3 most effectively. The grey area in Figure 3.1(a) is defined by the error, in the form of the *median absolute deviation*, and in Figure 3.1(b) the allowed deviation, where the upper bound is the threshold t_r that IN3 uses to detect malicious updates coming from randomly selected neighbors. It is equal to $5 \times D$, where D is the *median absolute deviation* and 5 is the number of deviations allowed defined for IN3 for randomly selected neighbors.



(a) Median Absolute Deviation (b) Random IN3 Threshold - 5 Median Absolute Deviations

Figure 3.1: Median Magnitude of Force

As the execution starts, all nodes have their virtual coordinates in the origin and will begin updating them, with bigger movements at first, and then, when approaching the correct location, with smaller and smaller movements just to refine the coordinates. We can verify in Figure 3.1 that the curve decreases over time, with the biggest drop in the median magnitude during the first 400 seconds. Regarding the *median absolute deviation*, it also decreases over time, as shown in Figure 3.1(a), and consequently, the allowed deviation from the median magnitude, shown in Figure 3.1(b), gets smaller over time too, with the upper bound of the grey area getting closer and closer to the curve of the magnitude, which leaves an ever smaller margin for introducing extraneous forces into the system. This implies that it gets harder and harder to push the coordinates from their real position before benign nodes start ignoring the updates. Concluding from this brief analysis, our design for this attack takes advantage of the higher force magnitude allowed by the IN3 threshold at an early stage of execution, and, for maximum efficiency, this attack must be performed from the beginning of the system's execution, or at least from the beginning of the execution of the specific target nodes.

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. They will start to lie to the benign nodes from the start of the execution when the IN3 threshold is more tolerant, as we saw in Figure 3.1, and 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.

In Section 4.5 we will iterate over these ideas and gather results showing what this strategy can accomplish against the security mechanisms of Newton. Specifically, we will start by showing if the attack can indeed deal with the different invariants, with the approaches explained in this section, and then verify if this attack strategy is effective against Newton.

Chapter 4

Measurements and Results

In this chapter we present the results of our experimental evaluation of the proposed attack strategies, and of the overall effectiveness of Newton as a safeguard against certain types of Sybil attacks against blockchain networks. We start by describing the experimental setup and the main metrics used in Section 4.1. In Section 4.2, we demonstrate the comparison between the results of tests performed in [4] by the creators of Newton and the results of those same tests recreated by us, in order to validate our code. Afterward, and to prepare for our experiments, we present in Section 4.3 our procedures on the preparation of the latency data, i.e., the network, we use. Sections 4.4 and 4.5 present our experiments, results, and analysis while trying to find weak points on Newton's protocol. Especially regarding attack strategies performed by attackers that form a cluster. Finally, in Section 4.6 we present a summary of the evaluation done in this chapter.

4.1 Experimental setup and metrics

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 [4], 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 [4] 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.

In all simulations, we use a Euclidean Coordinate Space with two dimensions, as we had already mentioned in Section 3.3.1, where 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 2 seconds, 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 4.1, 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).

$$pred\_error = |RTT_{actual} - RTT_{predicted}| \tag{4.1}$$

- *Velocity* is given by Equation 4.2, 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.

$$v = \frac{\triangle x}{t} \tag{4.2}$$

4.2 Validation

We begin by reporting the set of experiments that validate our implementation of Newton and the know attacks in *virtual coordinate systems*. This is important to validate the quality of our implementation and to check the correctness of the components of this work.

The first type of validation was unit testing. As we were developing the code, we wrote and ran unit tests for our methods/functions in an incremental way, trying to guarantee the correctness of the building blocks of more complex behavior, until we end up with Vivaldi implemented. (As explained in Section 3.3, we first implemented Vivaldi and then the security invariants that turn it into Newton.)

After finishing the implementation of the Vivaldi algorithm, we created a small network that allows us to visualize the behavior of the VCS, and check if it was working properly. In Figure 4.1 we have the geometric representation of the original network we created, with only 15 nodes, where each node chooses a neighbor set with 8 elements, and the distances between the various coordinates are intended to represent the RTTs between nodes. Then, in Figure 4.2, we present the output of Vivaldi at 3 different points in time. Figure 4.2(a) shows the system's starting point, with all nodes in the origin of the Euclidean space, as all the nodes join the network at the beginning of the experiment in a flash-crowd scenario. Figure 4.2(b) presents a subsequent point in time, where the coordinates are being updated but the system has not stabilized yet. Lastly, in Figure 4.2(c) we show the representation of the network after the coordinates of the nodes stabilize. Regarding the accuracy of the system, we achieved a *prediction error* of 122.0 ms when the simulation starts, 50.76 ms at the time of Figure 4.2(b), and 0.58 ms in Figure 4.2(c). Regarding the *prediction error* obtained when the virtual coordinates stabilize, we consider that it represents a good accuracy from our implementation of Vivaldi. In particular, when we look at both the original coordinates and the ones generated by Vivaldi, we can easily identify the same 3 groups of nodes ([1 - 6, 10 - 14], [0, 8, 9], and [7]) in both. The only node that misses its location, relative to the other nodes, in the geometric representation is number 4, and there are slight differences in the distances between the 3 groups, which are all explained by the fact that the network has 15 nodes but the neighbor set of each one has 8 nodes to update with. This highlights an intrinsic characteristic of Vivaldi, as it is not feasible for every node to update with every other node in the network.

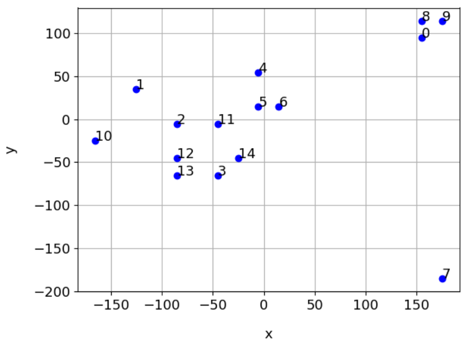
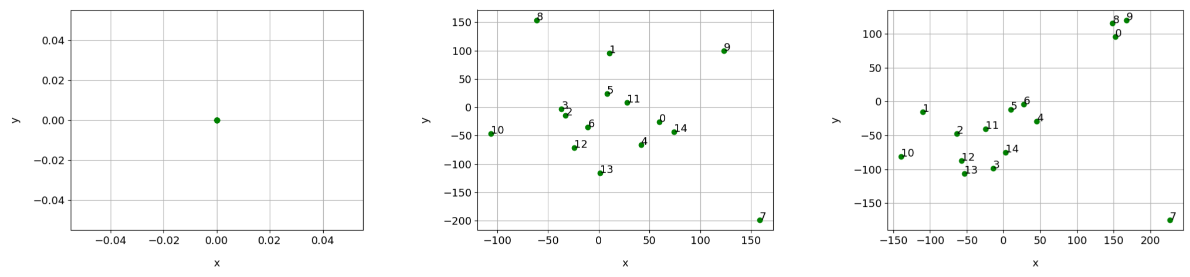


Figure 4.1: Original network geometric representation



(a) Beginning (b) Mid-progress (c) Stabilized

Figure 4.2: Vivaldi generated geometric representation

The next step is to validate the complete implementation of Newton, now with the security invariants, and to verify if our implementation of the attacks has the intended effect on the VCS system. For that, we recreate the simulation tests done in [4], to which we compare our results. In order to have the same conditions as the authors, we first had to take into account the latency data used by them, which they obtained from the King data set [37], containing pairwise measurements between 1740 nodes, with an average RTT of 180ms and a maximum RTT of 800ms. Hence, we used the King data set as well. Additionally, the malicious nodes only start their attacks after the system stabilizes, at about one third of the time, and we vary their percentage in the network between 10% and 30%, just like the authors have also done. We present our results and the results obtained in [4] for the same experiments in Figures 4.3 through 4.7, which show the accuracy, represented by the prediction error of the system, for different attack scenarios. This prior set of results includes Newton and Vivaldi under various attacks, Vivaldi under no attack[1] serving as a baseline, and, specifically in the data from [4] (a and c), an *OutlierDetection* curve that is not relevant for our work and we just ignore it. Furthermore, for each attack scenario we show the case for 10% of the nodes being attackers (a and b) and for 30% (c and d), where the attackers are chosen randomly from all nodes.

[1] Represented as *NoAttack*

We now compare the results from our implementation with the previously published results, focusing on three main points:

- We first make a comparison between our baseline and theirs (*NoAttack*), which behave similarly, both stabilizing with a prediction error below 20 ms, concretely in our results around 16 ms. This can be verified, for instance, in Figure 4.3(a) and Figure 4.3(b).

- Secondly, we validate our implementation of the five attacks. To this end, we analyze the effects that the different attacks have on the accuracy of the system with nodes running the insecure Vivaldi. The way accuracy reacts and changes when the system is under attack lets us know if our implementations of the attacks (*b* and *c*) has the intended behavior shown in the results from [4] (*a* and *b*). Additionally, when comparing these results, we can spot some differences in some of the prediction error values between them. We hypothesize that these are due to minor differences in the configuration of the system, and possibly in the attacks and the simulations between us and Seibert *et al.* [4], as well as the use of different simulators (P2PSim [2] vs Corten [5]). However, the results validate the quality of our implementation, since we are mainly interested in the trend of the accuracy curve, and not so much in smaller deviations from the exact values obtained in [4].

  Inflation Attack: Figure 4.3 shows the accuracy under an inflation attack. We can see that, in both cases, the prediction error of Vivaldi rapidly increases after the attack has started and then remains almost constant for the rest of the time. Additionally, it gets higher with the increase in the percentage of attackers.

  Deflation Attack: This attack have a smaller impact on Vivaldi than the inflation attack, as is the case for both us in Figure 4.4(b) and 4.4(d), and Seibert *et al.* in Figure 4.4(a) and 4.4(c). Once again, the prediction error increases with the increase of attackers.

  Oscillation Attack: In Figure 4.5 we have the accuracy under the oscillation attack, and we can make the same observations as in the inflation attack results, simply with slightly lower prediction errors.

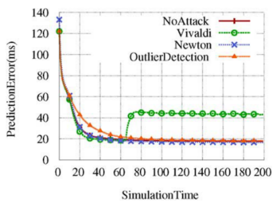
  Frog-Boiling Attack: As one of the *advanced slow attacks*, as explained in Section 2.3.1, this has a slow but constantly increasing impact on the accuracy of Vivaldi. When comparing Figure 4.6(a) and 4.6(c) to Figure 4.6(b) and 4.6(d), respectively, we observe a similar increase of the prediction error, with the exception of a higher increasing rate of our prediction error in the scenario with 10% attackers.

  Network Partition Attack: The exact same comparison as the one we made for the frog-boiling attack is valid here. The only slight difference is a smaller increasing rate of the prediction error. The results of the different systems under network partition attack are showed in Figure 4.7.

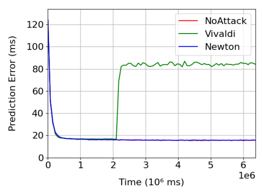
  The conclusion from these comparisons is that we believe that our implementation of the five attacks is correct and, since it displays the intended behavior when trying to degrade the performance of the system.

[2]https://pdos.csail.mit.edu/archive/p2psim/

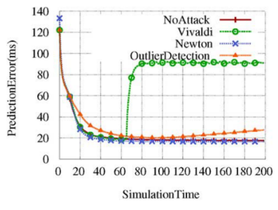
- Lastly, we want to highlight that, just as in the original implementation [4], our implementation of Newton is able to successfully mitigate the five attacks while providing good performance for both percentages of attackers. Specifically, it can keep the error low and match the baseline prediction error (*NoAttack*), with a value around 16 ms in both the original and our tests, even after the attacks start.



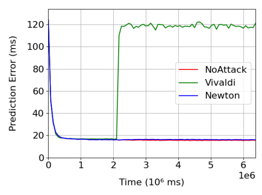
(a) [4] - 10% attackers



(b) Our - 10% attackers

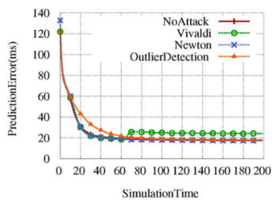


(c) [4] - 30% attackers

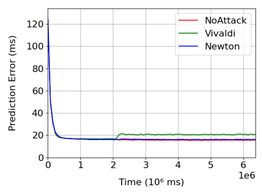


(d) Our - 30% attackers

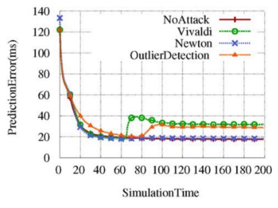
Figure 4.3: Accuracy - Simulation results from [4] (a, c) and from our implementation (b, d), under the *Inflation Attack*



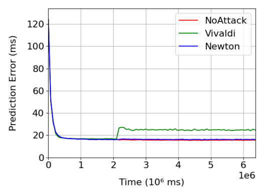
(a) [4] - 10% attackers



(b) Our - 10% attackers

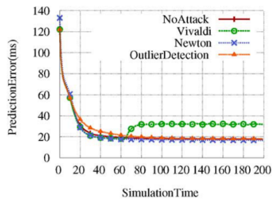


(c) [4] - 30% attackers

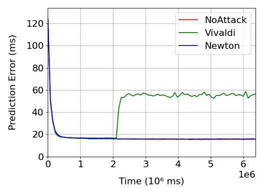


(d) Our - 30% attackers

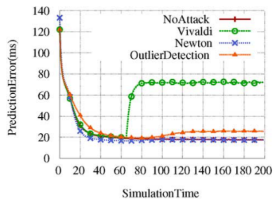
Figure 4.4: Accuracy - Simulation results from [4] (a, c) and from our implementation (b, d), under the *Deflation Attack*



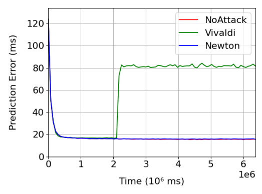
(a) [4] - 10% attackers



(b) Our - 10% attackers

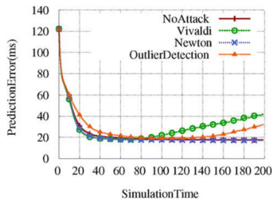


(c) [4] - 30% attackers

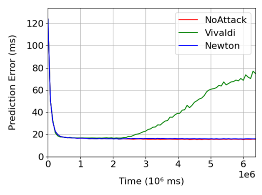


(d) Our - 30% attackers

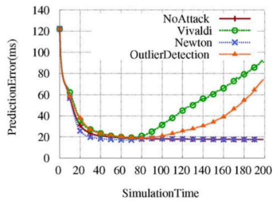
Figure 4.5: Accuracy - Simulation results from [4] (a, c) and from our implementation (b, d), under the *Oscillation Attack*



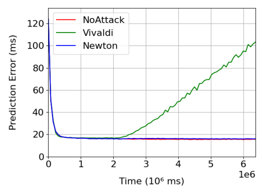
(a) [4] - 10% attackers



(b) Our - 10% attackers

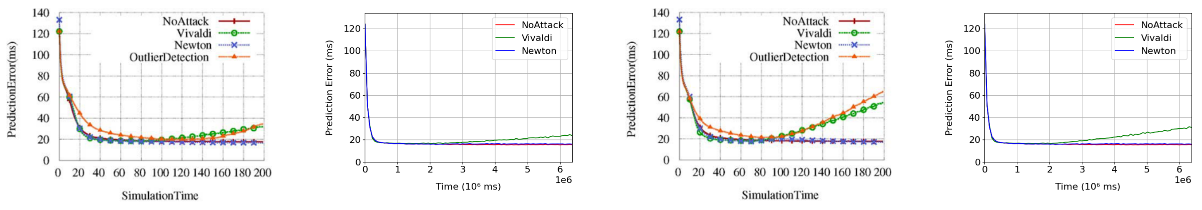


(c) [4] - 30% attackers



(d) Our - 30% attackers

Figure 4.6: Accuracy - Simulation results from [4] (a, c) and from our implementation (b, d), under the *Frog-Boiling Attack*



(a) [4] - 10% attackers (b) Our - 10% attackers (c) [4] - 30% attackers (d) Our - 30% attackers

Figure 4.7: Accuracy - Simulation results from [4] (a, c) and from our implementation (b, d), under the *Network Partition Attack*

4.3 Network Setup

As mentioned in Section 3.2, the Corten Simulator [5] will need a matrix of inter-host internet latencies of the network in the simulations. The data sets used by most wide-area simulations are outdated, given that several years passed after their creation, as is the case of the King data set [37], one of the most used data sets in virtual coordinate systems experiments from the past several years. For that reason, we use a data set containing pairwise measurements of latencies taken from 226 PlanetLab nodes, which was used by the authors of Corten in 2019.

Additionally, there are two relevant points to address, the dimension, or the number of nodes, of the network and the creation of the cluster that will be performing the attacks, as explained in Section 3.1. Hence, we describe our procedure next.

Procedure

We want to create and modify a network to use in our simulation environment, and as we said above, we use the data set of the 226 PlanetLab nodes. However, this data set contains the measurements of latencies between the nodes, but in order to use the network represented by this dataset as a basis and modify it, we need to obtain a geometric representation of these 226 nodes. For this, and having already validated the implementation of Newton [4], we used it to generate a geometric representation from the latency matrix (data set), and thus obtain coordinates for the 226 nodes.

Dimension: As a 226 node network would not be very relevant in the context of blockchain today, due to having a small number of nodes, we have expanded the network to obtain a network with a minimum [3] of 1000 nodes and thus have more relevant results in our experiments, which are presented in Section 4.4 and Section 4.5. To this end, for each node in the original network, a number n of co-located new nodes were created. With n being determined by Equation 4.3, where $threshold$ is the minimum number of nodes we want for the network (before adding the cluster of malicious nodes), and $current\_num\_nodes$ is the current number of nodes of the network being expanded.

$$n = \lceil \frac{threshold - current\_num\_nodes}{current\_num\_nodes} \rceil \tag{4.3}$$

[3] It will have more nodes when the cluster is created.

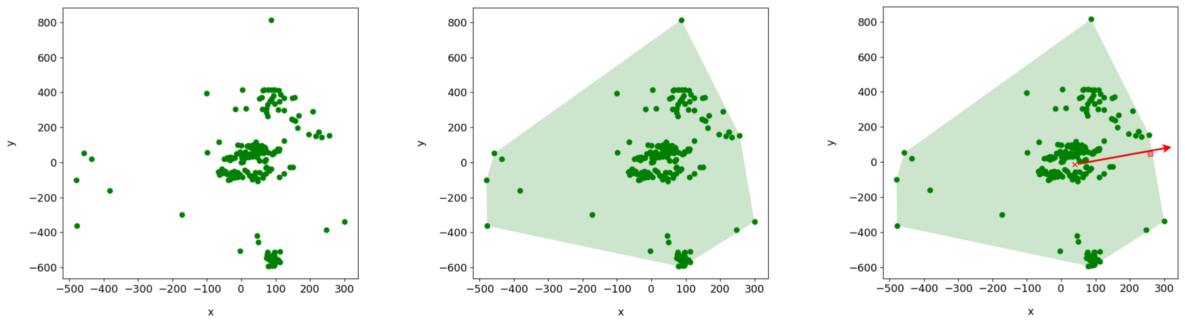
In the particular case of the experiments of our evaluation, solving the Equation 4.3, we have $n = \lceil \frac{1000-226}{226} \rceil = \lceil \frac{774}{226} \rceil = \lceil 3.42 \rceil = 4$, where $threshold = 1000$ and $current\_num\_nodes = 226$, and consequently we added 4 nodes near every one of the 226 nodes, with random directions.

Cluster: Regarding the creation of the malicious cluster, we wanted to get a generic way to position it, allowing for increasing its isolation as it gets further away from the rest of the network. The procedure we describe next allows us to do just that, by systematically controlling the way we create and position the cluster relative to the rest of the network. We present Figure 4.8 as a visual representation, aimed at helping understand the procedure.

- *Input values*: the **percentage** of nodes of the network that belong to the cluster, and the **distance** between the cluster and the *convex hull* of the rest of the network. In particular, these two variables allow us to characterize the cluster for the different experiments we run in Section 4.4 and Section 4.5.

- Compute the *convex hull* of the set of nodes in the network[4]. In Figure 4.8(b) the contour of the green shadow is the *convex hull*, which is the smallest convex set that contains all the nodes in the network, and the algorithm used to compute it was QuickHull, since it is the one implemented in the library *geo*$\_0.14.2$ [5] from Rust, the programming language we used. The use of the *convex hull* is relevant mainly for the next step, where we compute the position of the cluster.

- The vector from the centroid of the network to its closest point on the *convex hull* gives us the direction in which we move the cluster (Figure 4.8(c)). Along this direction, we position the cluster outside the convex hull, with a pre-defined distance away from it. This distance, as implied above, is chosen according to different scenarios.



(a) Network (b) Convex Hull (c) Direction to move cluster

Figure 4.8: Cluster creation and positioning

- Finally, the nodes in the cluster are created and positioned randomly around the previously computed location. The number of nodes in the cluster, represented by m, can be computed by solving $f = \frac{m}{m+b}$, where b is the number of benign nodes (nodes in the network excluding the ones from the cluster) and f is the percentage of nodes from the network that belong to the cluster, which

[4]Network after adding co-located nodes.
[5]https://docs.rs/geo/0.14.2/geo/algorithm/convexhull/trait.ConvexHull.html#tymethod.convex_hull.

are known *a priori*. Equation 4.4 shows how to compute m, allowing us to know how many nodes to create for the cluster.

$$\begin{aligned} f &= \frac{m}{m+b} \\ m &= fm + fb \\ m(1-f) &= fb \\ m &= \frac{fb}{1-f} \end{aligned} \tag{4.4}$$

Finally, after extending the network and adding the cluster, we generate the matrix of latencies between all nodes. To do that, the important thing to know is that computing the latency between two nodes corresponds to calculating the distance between the corresponding coordinates and then divide it by 2. The reason for that, as explained in Section 2.2.2, is that the distance between two nodes represents an estimate of the RTT between the nodes, and the Corten Simulator assumes that each latency in the matrix is the one-way communication time between two nodes.

4.4 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.* [4], and by ourselves in Section 4.2. 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, and, unlike all the previous simulations, we now set the time between probes to 2500 ms. The reason for this is that the maximum RTT of the network we set up in Section 4.3 was 1408.25 ms and, during the oscillation attack, attackers can delay probes up to one second, thus increasing the maximum RTT to 2408.25 ms. Hence, to have the same configuration across all the different attack scenarios, probes are sent every 2500 ms in every experiment of this section.

We vary the percentage of nodes from the network that are attackers between 10% and 30%, as in the previous section. We create the network as explained in Section 4.3 and, before adding the attacker nodes, the network has 1130 nodes, with a subsequent total of 125 attacker nodes in the 10% scenario and 484 attackers nodes in the 30% scenarios being added. The former will result in a total of 1255 nodes, and the latter in 1614 nodes.

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 *convex hull* of 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:

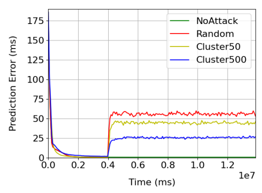
Inflation Attack: To report fake coordinates far away from the origin, an attacker randomly chooses a distance between 900 and 1000 ms, which is far enough from the origin to be beyond every correct node in the network. Observing Figures 4.9(a) and 4.9(c), we realize that the cluster scenarios have less impact on the virtual coordinates than the random one. Hence, it is of no surprise that Newton is able to keep the prediction error matching with the baseline, since the cluster, even far away, has no advantage over the randomly distributed attackers, which we had already seen in Section 4.2 that Newton could withstand. 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 4.9(b) and 4.9(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 4.10(a) and 4.10(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 4.10(b) and 4.10(d) that Newton is able to successfully mitigate the attack once again.

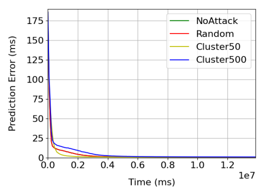
Oscillation Attack: As we explained in Section 3.4, an attacker not only lies about its coordinates but also delays probes by up to 1 second. Regarding the coordinates reported, it randomly generates fake coordinates between 0.1 and 1000 ms of distance to the origin in any direction. Just like in the inflation attack, the cluster attackers have a smaller impact than the random attackers on the system's accuracy, as can be seen in Figures 4.11(a) and 4.11(c). Once again, as in the previous attacks, Newton is able to match the baseline accuracy in all scenarios, as shown in Figures 4.11(b) and 4.11(d), which we attribute to the IN3 ability to detect when forces do not decrease over time.

Frog-Boiling Attack: As we mentioned in Section 3.4, the small deviations that the attacker reports in each update have a length of 0.25 ms. Figure 4.12 presents the prediction error of the system under the frog-boiling attack. When there are 30% of attackers, we can see in Figure 4.12(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.

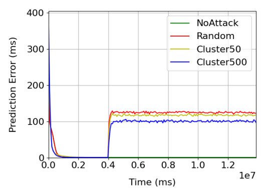
Network Partition Attack: Like in the Frog-Boiling attack, the small inaccuracies that the attacker reports in each update that is sent increment its fake coordinates by 0.25 ms. In Figure 4.13 we have the accuracy results, which allow us to see, when Vivaldi is under attack, that the network partition 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.



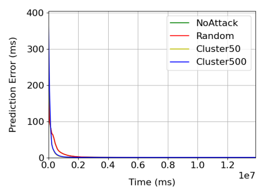
(a) Vivaldi - 10% attackers



(b) Newton - 10% attackers

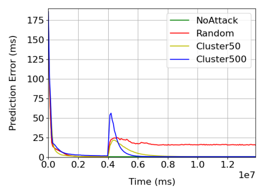


(c) Vivaldi - 30% attackers

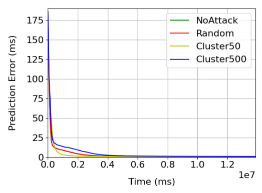


(d) Newton - 30% attackers

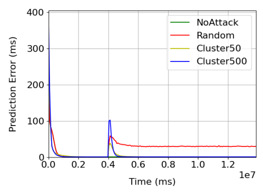
Figure 4.9: Accuracy - *Inflation Attack*



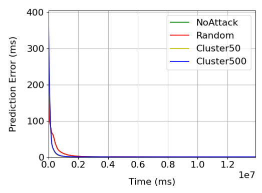
(a) Vivaldi - 10% attackers



(b) Newton - 10% attackers

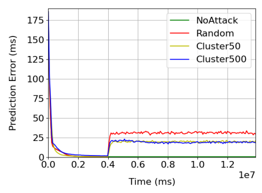


(c) Vivaldi - 30% attackers

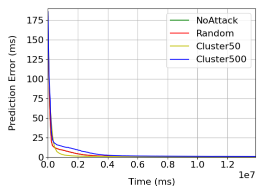


(d) Newton - 30% attackers

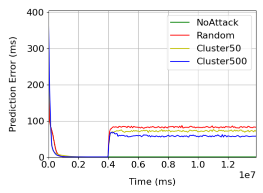
Figure 4.10: Accuracy - *Deflation Attack*



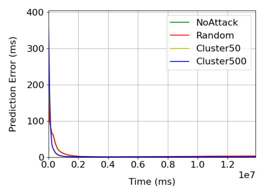
(a) Vivaldi - 10% attackers



(b) Newton - 10% attackers

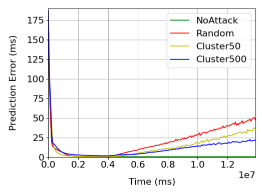


(c) Vivaldi - 30% attackers

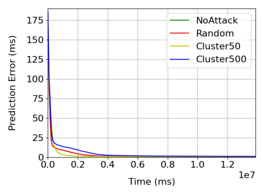


(d) Newton - 30% attackers

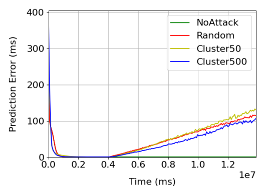
Figure 4.11: Accuracy - *Oscillation Attack*



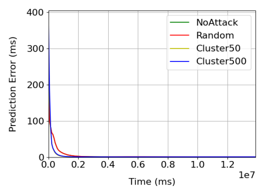
(a) Vivaldi - 10% attackers



(b) Newton - 10% attackers



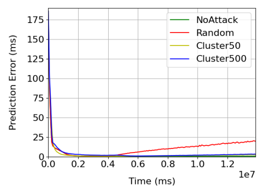
(c) Vivaldi - 30% attackers



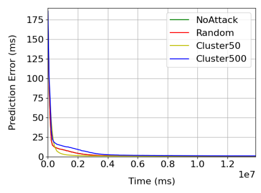
(d) Newton - 30% attackers

Figure 4.12: Accuracy - *Frog-Boiling Attack*

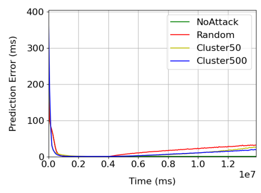
Stability: We have just seen the accuracy results obtained by Newton under all the different attacks we consider, and how it is able to match the baseline accuracy and maintain it over time. Thus, it is of no surprise that we learn from Figure 4.14, Figure 4.15 and Figure 4.16, that it obtains stable 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



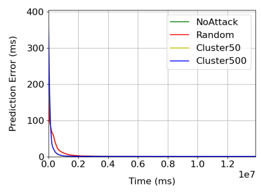
(a) Vivaldi - 10% attackers



(b) Newton - 10% attackers



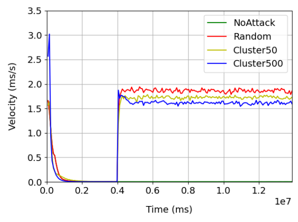
(c) Vivaldi - 30% attackers



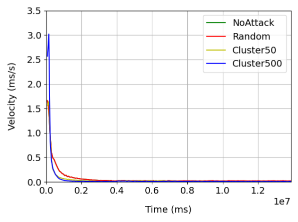
(d) Newton - 30% attackers

Figure 4.13: Accuracy - *Network Partition Attack*

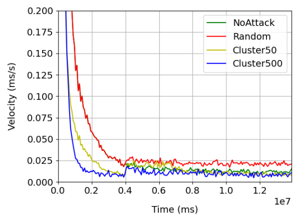
than the random attackers during the frog-boiling attack, whereas the opposite happens in the network partition attack. However, Newton can deal with both. Additionally, we present the zoomed-in velocity axis for the three attacks, which highlights that, in all scenarios, Newton reaches velocity values with minor, almost insignificant deviations from the baseline.



(a) Vivaldi - 30% attackers

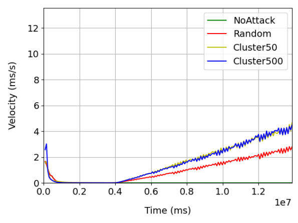


(b) Newton - 30% attackers

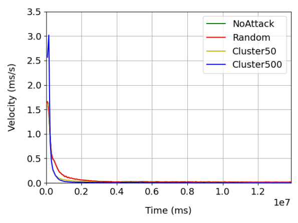


(c) Newton - 30% attackers (zommed in)

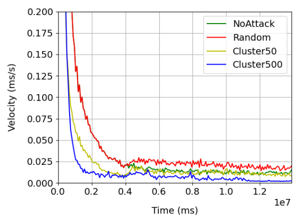
Figure 4.14: Stability - *Oscillation Attack*



(a) Vivaldi - 30% attackers

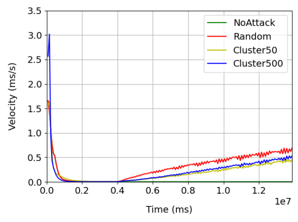


(b) Newton - 30% attackers

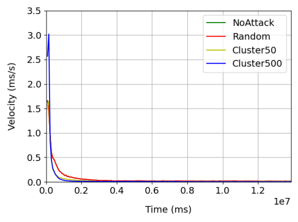


(c) Newton - 30% attackers (zoomed in)

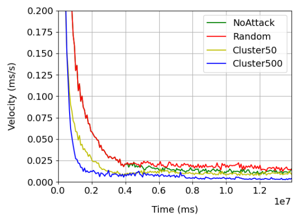
Figure 4.15: Stability - *Frog-Boiling Attack*



(a) Vivaldi - 30% attackers



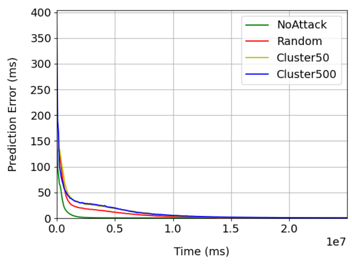
(b) Newton - 30% attackers



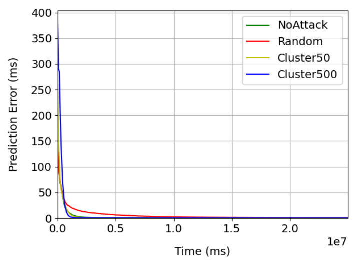
(c) Newton - 30% attackers (zommed in)

Figure 4.16: Stability - *Network Partition Attack*

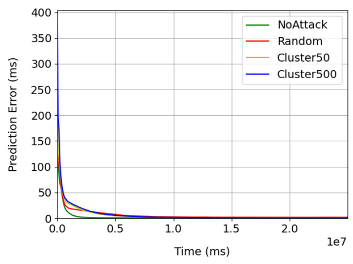
Start Attack from Beginning: In an attempt to make the attacks more effective against Newton, we ran each one of them from the beginning of the system execution, with 30% of attacker nodes in the network. We present the prediction error results of these experiments in Figures 4.17 and 4.18, which run for 2.5×10^7 ms, unlike all the previous experiments in this section. This is to give enough time for Newton to stabilize, since, when nodes are attacked when not yet near their correct positions, they will take longer to stabilize, due to the number of (malicious) updates ignored. As we can observe, Newton mitigates all five attacks. The main observation is that during the network partition attack, the system takes longer to match the baseline accuracy, specifically in the random attacker scenario.



(a) *Inflation*

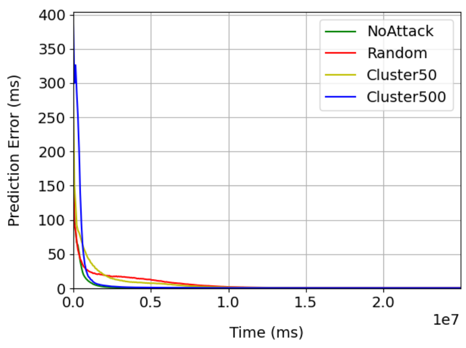


(b) *Deflation*

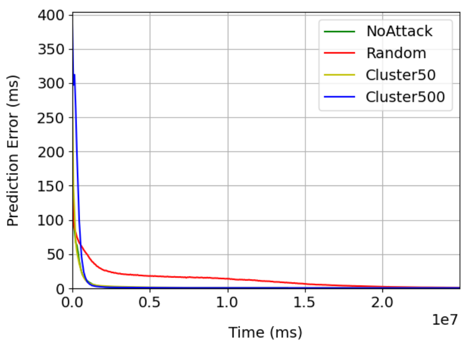


(c) *Oscillation*

Figure 4.17: Accuracy - Newton with 30% of attackers. *Simple attacks* from beginning.



(a) *Frog-Boiling*



(b) *Network Partition*

Figure 4.18: 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.

4.5 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, since the max RTT in the network (Section 4.3) is 1408.25 ms and attackers do not delay probes in this attack scenario.

As we explained in Section 3.5, 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:

- 10% Attackers:
  - Randomly distributed attackers: 1446
  - Cluster at centroid of network: 210
  - Cluster at 50 ms: 0
  - Cluster at 500 ms: 0
- 30% Attackers:
  - Randomly distributed attackers: 3797
  - Cluster at centroid of network: 244
  - Cluster at 50 ms: 0
  - Cluster at 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 convex hull of the rest of the network.
Lastly, we have **IN3**, which, at this point, is the last line of defense of Newton, as we witness in Figure 4.19 with results from the scenario where the cluster is at 50 ms. We observe that, under the *network partition attack*, Newton is unable to identify malicious updates from the nodes of the cluster, with IN1 and IN2 active (curve *IN1 + IN2*). However, with only IN3 active (curve *IN3*), it keeps the Prediction Error of the system low.

To deal with IN3, as explained in Section 3.5, 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 4.20, where 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

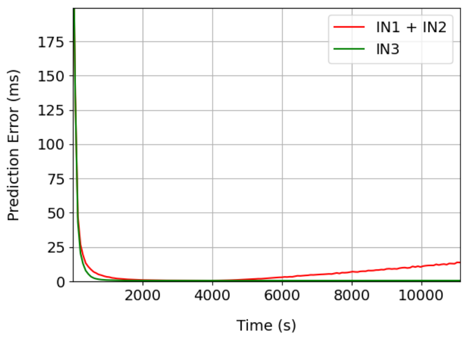


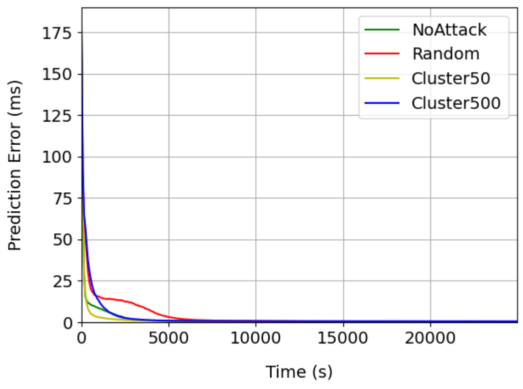
Figure 4.19: Network Partition - Cluster at 50 ms with 30% attackers

scenarios with the attackers forming a cluster, whose distance to the rest of the network convex hull was varied between 50 ms and 500ms.

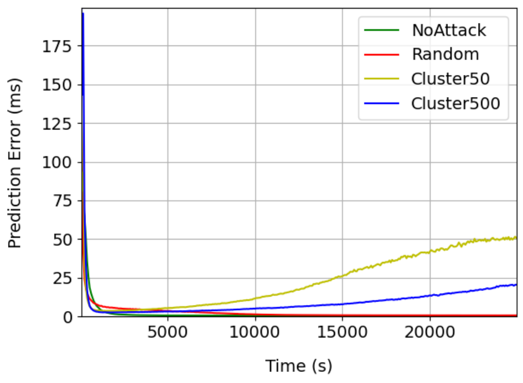
We first observe that in order for the attack to be effective, the attackers must represent at least 30% of the nodes in the network, since we have a clear increase of the error in Figure 4.20(b), but not in Figure 4.20(a). In fact, Newton was able to maintain the accuracy of the system during the three 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 4.20(b), that randomly 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. In contrast, randomly positioned attackers around the network have a lot of their updates checked by the IN2, as they are selected as close neighbors quite often, as we said earlier in this section.

As we mentioned in Section 3.5, 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 4.21, 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 4.21(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 noteworthy aspect of this evaluation is the following: in prior work [4], 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.

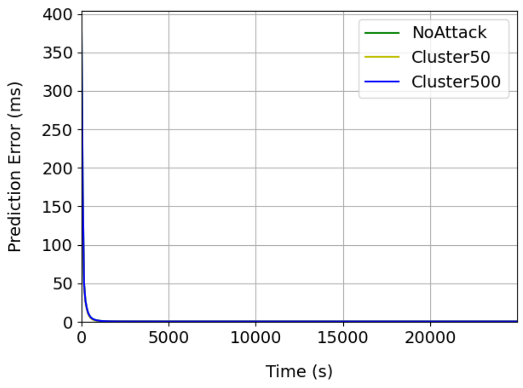


(a) 10% Attackers

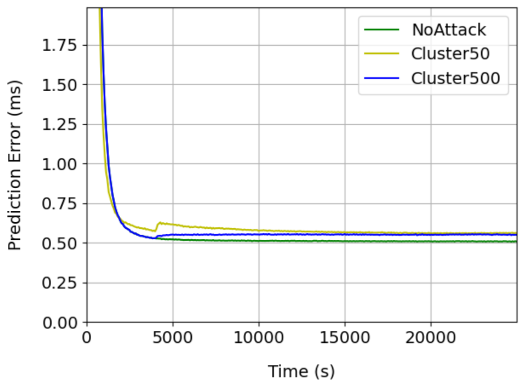


(b) 30% Attackers

Figure 4.20: Split Cluster Attack - Accuracy



(a) 30% Attackers



(b) 30% Attackers (Zoomed in)

Figure 4.21: Split Cluster Attack - Accuracy with attack starts at 4000 seconds

4.6 Summary of Evaluation

To close this chapter, we summarize the main observations from the experiments we have conducted. We first validated our implementation of Newton and the known attacks on VCS systems, and prepared the latency data, i.e., the network, to use in our experiments. Afterward, we evaluated Newton under the known attacks (inflation, deflation, oscillation, frog-boiling and network partition attacks) but in a novel scenario, which to the best of our knowledge has not been tested before, where attackers form a cluster at different distances from the rest of the network. We also tested Newton against a new attack strategy we designed, called Split Cluster Attack. After analyzing the experimental results, we reached the following conclusions:

Newton is able to mitigate all the attack scenarios presented in Section 4.4, even matching the baseline accuracy, what takes us to confirm that it is a reliable Virtual Coordinate System even under attack. However, we managed to disrupt its security with our Split Cluster Attack, which design we explained in Section 3.5. As can be seen in Section 4.5, for the 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. This implies that Newton is less robust than previous studies suggest in attack scenarios for blockchain systems. That said, it is still an important first line of defense that could be useful in practice.

Chapter 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.

These virtual coordinate systems need to be robust and resilient to attacks, and therefore, in our work, we started by making a comparative review of various decentralized Virtual Coordinate Systems, with the goal of understanding which one seems to be the most complete and secure. As a result of this review, we selected Newton as the one we believed most capable.

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 4.4, 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 [4].

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.

5.1 Future Work

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 [4], 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 [7] or Ethereum [8] 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. Subsequently, an evaluation of the blockchain will compare a baseline version, where no VCS was applied, with the result of the blockchain system with Newton incorporated. Even though this evaluation might be less exhaustive due to the greater difficulty in running real-world experiments, it would still be relevant to evaluate both the resilience to attacks and the overheads that are introduced by our techniques.

Bibliography

[1] A. E. Gencer, S. Basu, I. Eyal, R. Van Renesse, and E. Sirer. *Decentralization in Bitcoin and Ethereum Networks*, pages 439–457. 12 2018.

[2] I. Eyal and E. Sirer. Majority is not enough: Bitcoin mining is vulnerable. volume 8437, 11 2013. doi: 10.1007/978-3-662-45472-5_28.

[3] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. *Computer Communication Review*, 34(4):15–26, 2004. ISSN 01464833.

[4] 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, 2014. ISSN 10636692.

[5] I. Sequeira. Large Scale Distributed Algorithms Simulator. *IST Dissertation*, 2019.

[6] M. Pilkington. *Blockchain Technology: Principles and Applications*. 01 2016.

[7] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[8] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. 2014.

[9] Ethereum. A next-generation smart contract and decentralized application platform. Last edit Jun. 2019.

[10] B. Donnet, B. Gueye, and M. A. Kaafar. A survey on network coordinates systems, design, and security. *IEEE Communications Surveys and Tutorials*, 12(4):488–503, 2010. ISSN 1553877X.

[11] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in Euclidean space. *IEEE/ACM Transactions on Networking*, 12(6):993–1006, dec 2004. ISSN 10636692.

[12] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical internet coordinates for distance estimation. In *Proceedings - International Conference on Distributed Computing Systems*, pages 178–187, 2004.

[13] L. W. Lehman and S. Lerman. PCoord: Network position estimation using peer-to-peer measurements. *Proceedings - Third IEEE International Symposium on Network Computing and Applications, NCA 2004*, pages 15–24, 2004.

[14] L. W. Lehman and S. Lerman. A decentralized network coordinate system for robust Internet distance prediction. *Proceedings - Third International Conference onInformation Technology: New Generations, ITNG 2006*, 2006:631–637, 2006.

[15] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7 (4):308–313, 1965. ISSN 0010-4620.

[16] H. Zheng, E. Lua, M. Pias, and T. Griffin. Internet routing policies and round-trip-times. volume 3431, pages 236–250, 03 2005.

[17] J. Ledlie, P. Gardner, and M. Seltzer. Network coordinates in the wild. 01 2007.

[18] G. Wang, B. Zhang, and T. Ng. Towards network triangle inequality violation aware distributed systems. pages 175–188, 01 2007.

[19] Y. Chen, Y. Xiong, X. Shi, B. Deng, and X. Li. Pharos: A decentralized and hierarchical network coordinate system for Internet distance prediction. In *GLOBECOM - IEEE Global Telecommunications Conference*, pages 421–426, 2007. ISBN 1424410436.

[20] Y. Chen, Y. Xiong, X. Shi, J. Zhu, B. Deng, and X. Li. Pharos: Accurate and decentralised network coordinate system. *IET Communications*, 3(4):539–548, 2009. ISSN 17518628.

[21] Y. Chen, X. Wang, X. Song, E. K. Lua, C. Shi, X. Zhao, B. Deng, and X. Li. Phoenix: Towards an accurate, practical and decentralized network coordinate system. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 313–325, 2009. ISBN 978-3-642-01398-0.

[22] Y. Mao, L. K. Saul, and J. M. Smith. IDES: An internet distance estimation service for large networks. *IEEE Journal on Selected Areas in Communications*, 24(12):2273–2284, 2006. ISSN 07338716.

[23] M. A. Kaafar, L. Mathy, T. Turletti, and W. Dabbous. Real attacks on virtual networks: Vivaldi out of tune. *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense, LSAD'06*, 2006:139–146, 2006.

[24] M. A. Kaafar, L. Mathy, T. Turletti, and W. Dabbous. Virtual networks under attack: Disrupting internet coordinate systems. In *Proceedings of the 2006 ACM CoNEXT Conference*, CoNEXT '06, pages 12:1–12:12, 2006. ISBN 1-59593-456-1.

[25] D. J. Zage and C. Nita-Rotaru. On the accuracy of decentralized virtual coordinate systems in adversarial networks. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 214–224, 2007. ISSN 15437221.

[26] E. Chan-Tin, D. Feldman, N. Hopper, and Y. Kim. 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:448–458, 2009. ISSN 18678211.

[27] E. Chan-Tin, V. Heorhiadi, N. Hopper, and Y. Kim. The frog-boiling attack: Limitations of secure network coordinate systems. *ACM Trans. Inf. Syst. Secur.*, 14(3):27:1–27:23, Nov. 2011. ISSN 1094-9224.

[28] D. Saucez, B. Donnet, and O. Bonaventure. A reputation-based approach for securing Vivaldi embedding system. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4606 LNCS:78–85, 2007. ISSN 03029743.

[29] M. A. Kaafar, L. Mathy, C. Barakat, K. Salamatian, T. Turletti, and W. Dabbous. Securing Internet Coordinate Embedding Systems. *Computer Communication Review*, 37(4):61–72, 2007. ISSN 01464833.

[30] D. Zage and C. Nita-Rotaru. Robust decentralized virtual coordinate systems in adversarial environments. *ACM Trans. Inf. Syst. Secur.*, 13(4):38:1–38:34, Dec. 2010. ISSN 1094-9224.

[31] G. Wang and T. S. Ng. Distributed algorithms for stable and secure network coordinates. *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pages 131–144, 2008.

[32] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, 1999. ISBN 1-880446-39-1.

[33] A. Haeberlen, P. Kouznetsov, and P. Druschel. Peerreview: Practical accountability for distributed systems. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 175–188, 2007. ISBN 978-1-59593-591-5.

[34] M. Sherr, M. Blaze, and B. T. Loo. Veracity: practical secure network coordinates via vote-based agreements. *Usenix Atc*, page 13, 2009.

[35] S. Becker, J. Seibert, C. Nita-Rotaru, and R. State. Securing Application-Level Topology Estimation Networks: Facing the Frog-Boiling Attack. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6961 LNCS:201–221, 2011.

[36] A. Montresor and M. Jelasity. Peersim: A scalable p2p simulator. pages 99–100, 09 2009. doi: 10.1109/P2P.2009.5284506.

[37] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *SIGCOMM Internet Measurement Workshop 2002*, 2002.