Securing Virtual Coordinates by Enforcing Physical Laws

Jeff Seibert*, Sheila Becker†, Cristina Nita-Rotaru* and Radu State†
*Purdue University
{jcseiber, crisn}@cs.purdue.edu
†University of Luxembourg
{sheila.becker, radu.state}@uni.lu

***Abstract*—Virtual coordinate systems (VCS) provide accurate estimations of latency between arbitrary hosts on a network, while conducting a small amount of actual measurements and relying on node cooperation. While these systems have good accuracy under benign settings, they suffer a severe decrease of their effectiveness when under attack by compromised nodes acting as insider attackers. Previous defenses mitigate such attacks by using machine learning techniques to differentiate good behavior (learned over time) from bad behavior. However, these defense schemes have been shown to be vulnerable to advanced attacks that make the schemes learn malicious behavior as good behavior.**

We present Newton, a decentralized VCS that is robust to a wide class of insider attacks. Newton uses an abstraction of a real-life physical system, similar to that of Vivaldi, but in addition uses safety invariants derived from Newton's laws of motion. As a result, Newton does not need to learn good behavior and can tolerate a significantly higher percentage of malicious nodes. We show through simulations and real-world experiments on the PlanetLab testbed that Newton is able to mitigate all known attacks against VCS while providing better accuracy than Vivaldi, even in benign settings.

I. Introduction

Numerous distributed protocols use network locality for optimized replica placement [1], multicast tree and mesh construction [2], routing on the Internet [3, 4], and Byzantine fault-tolerant membership management [5]. Virtual Coordinate Systems (VCS) have been proposed as an efficient and low cost service to provide network locality estimations by accurately predicting round-trip times (RTT) between arbitrary nodes in a network. Each node measures the RTT to a small number of other nodes and the VCS then assigns a coordinate to each node. Each node can then estimate the RTT between itself and any arbitrary node by calculating some distance function.

While some VCS are centralized in nature [6], many have been designed as distributed systems [7], where each node maintains and updates its own coordinate by relying on information received from other nodes. Distributed VCS can be classified as landmark-based and decentralized. Landmark-based systems [8]–[12] assume a trusted set of nodes that form the infrastructure by which other nodes can determine their coordinates. Decentralized VCS [7, 13]–[15] assume no such infrastructure; a node updates its coordinate based on measurements and information from a random set of nodes.

Unfortunately, distributed VCSs have been shown [16] to be vulnerable to insider attacks, where compromised nodes delay measurement probes and lie about their coordinates to decrease system performance. As many applications rely on VCS to build robust services, there have been several proposals to secure them. For example, outlier detection [17, 18] and voting [19] were used to detect equivocation of lying attackers. Most of these defense methods ultimately decide if an update from a node is malicious or not by learning good behavior through system observation over time. As a result, these schemes are vulnerable to attacks where through small changes attackers make the defense mechanisms learn malicious behavior as being good behavior. One such attack is the well-known frog-boiling attack where attackers lie by small amounts that accumulate over time and gradually lead to large changes in performance [20]–[22].

A classical approach for designing distributed systems is to use *safety invariants* in order to ensure system correctness. These safety invariants specify states into which the distributed system should never enter. For example, a distributed system that forms a tree of nodes should never have any loops, or a distributed hash table should never form multiple rings, but only one continuous ring. At first glance, VCS do not appear to have such invariants as minimal constraints are imposed on how neighbors are selected or on what coordinates a node can possibly have. We make the key observation that some VCS are designed around an abstraction of a physical system [6, 7, 23] and that physical systems follow physical laws. As these laws are universally true, we can leverage them to identify safety invariants for VCSs based on physical systems.

We present Newton, a decentralized VCS which extends Vivaldi [7] to withstand a wide class of insider attacks by using safety invariants derived from Newton's three laws of motion. Newton relies on the observation that Vivaldi is an abstraction of a real-life physical system and therefore all participating nodes must follow Newton's three laws of motion. As there is a direct mapping between the actions taken by nodes, in reporting their coordinates and RTTs, and the forces that these physical laws govern, any attack in which malicious nodes lie about their coordinates or delay probes will result in the invariants being violated. We leverage this fact to detect attacks and discard malicious updates. Our contributions are:

- We describe how to use Newton's three laws of motion as well as a mapping between forces and virtual coordinates to identify invariants that mitigate a wide range of attacks against Vivaldi. We show how to use the three identified invariants to detect and mitigate the well-studied inflation, deflation, and oscillation attacks, as well as the more recent frog-boiling and network-partition attacks.
- We conduct extensive simulations and real-world experiments on PlanetLab to demonstrate that Newton is able to

mitigate all known attacks against VCSs. We compare Newton with Vivaldi outfitted with Outlier Detection [18] and show that Newton is not vulnerable to the frog-boiling and network-partition attacks. We also find that, even with no attackers, Newton has better performance than Vivaldi, i.e. Newton is 25% more accurate and 68% more stable.

• We consider extreme scenarios where the attackers are present in a much higher percentage, over 50% of nodes in the network are malicious, and also where attackers are conducting attacks from the beginning of the experiment, while the system has not converged yet to a steady state. We show that even under such conditions Newton still performs well.

• We consider adaptive attackers that know how the invariants are used and try to exploit them. Because in real-deployments Newton is not a perfect abstraction of a physical system, an attacker can try to exploit the invariants. We explore a new type of attack, *rotation attack*, where attackers rotate their positions slowly around the origin of the coordinate plane in an attempt to destabilize nodes while remaining undetected. We find that Newton holds up well to such attacks, incurring only slightly decreased accuracy.

The remainder of this paper is organized as follows: We describe Vivaldi in Sec. II and attacks against it in Sec. III. We describe Newton and our invariants in Sec. IV. We show simulation results in Sec. V and PlanetLab experimental results in Sec. VI. We present related work in Sec. VII and our conclusion in Sec. VIII.

II. Vivaldi Coordinate System

Algorithm 1: Node i Coordinate Update

Input: Remote node tuple $\langle x_j, e_j, RTT_{ij} \rangle$
Output: Updated local 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 $\alpha = c_e \times w$
4 $e_i = (\alpha \times e_s) + ((1 - \alpha) \times e_i)$
5 $\delta = c_c \times w$
6 $x_i = x_i + \delta \times (RTT_{ij} - \|x_i - x_j\|) \times u(x_i - x_j)$

Vivaldi [7] is a decentralized VCS where the distance between coordinates represents the estimated RTT between nodes. All nodes start at the origin and periodically update their coordinates based on interaction with a subset of nodes referred to as the neighbor set. A node chooses half of these nodes randomly from all possible nodes and the other half from a set of low-latency nodes. Research [7] has shown that a neighbor set of 64 nodes ensures quick convergence.

In addition to the coordinate value, each node also maintains a local error value which shows the confidence in the coordinate. Algorithm 1 describes how each node i updates its coordinate. Specifically, i will send a request to node j for its coordinate and local error value. When node j replies node i also measures the actual RTT. An observation confidence w is calculated first (line 1) along with the error e_s in comparing the coordinates with the actual RTT (line 2). Node i updates its local error (line 4) by calculating an exponentially-weighted moving average with weight α and system parameter c_e (line 3). Next, i computes the movement dampening factor calculated with another system parameter c_c (line 5) and updates its coordinate by finding how far it should move and then multiplying that by a unit vector (represented by $u(\bullet)$) in the direction it should move (line 6).

A VCS generally has the system goals of providing accuracy and stability with respect to the coordinates that it produces. Accuracy describes how closely the coordinates reflect the actual RTT between nodes. Stability describes how quickly nodes converge to a set of accurate coordinates and how long a node can be absent from the system and still have accurate coordinates.

Accuracy. We use *prediction error* to measure accuracy: $Error_{pred} = |RTT_{Act} - RTT_{Est}|$, where RTT_{Act} is the measured RTT and RTT_{Est} is the estimated RTT. A small prediction error indicates high accuracy. We report the median of all the prediction errors at a time instant.

Stability. We use *velocity* of a node to measure stability: $Velocity = \frac{\Delta x_i}{t}$, where Δx_i is the change in coordinates for node i (or distance traveled by a node), and t is the amount of time taken to make that change. A small velocity indicates high stability. We report the average of velocity of all nodes at a time instant.

III. Attacks Against VCS

We consider that a bounded number of compromised and colluding nodes act maliciously. To attack Vivaldi, a malicious node can (1) influence the coordinate value computation by lying about its coordinate and local error value or (2) influence the RTT computation by delaying the measurement probe.

An attacker can exploit coordinate and RTT computation to conduct the following *basic attacks*:

• **Inflation:** Attackers lie about having very large coordinates. This pulls benign nodes far away from their correct coordinates and thus is an attack on accuracy.

• **Deflation:** Attackers lie about having small coordinates near the origin. This prevents benign nodes from being able to update to their correct coordinates and therefore is also an attack on accuracy.

• **Oscillation:** Attackers lie by reporting randomly chosen coordinates and randomly delaying measurement probes. This is an attack both on accuracy and stability.

Basic attacks against Vivaldi have been shown to be very effective in reducing accuracy and stability [16]. Moreover, while defenses have been proposed, recent work [20]–[22] identified more *advanced attacks* that are able to bypass all previously proposed defenses [17]–[19]. Advanced attacks are:

• **Frog-boiling:** Attackers lie by small amounts at a time, slowly increasing this amount by moving their coordinates in one direction. Over time though, the attacker ends up reporting coordinates that are far away from their correct coordinate. This results in an attack on both accuracy and stability.

• **Network-partition:** Attackers lie similarly as in the frog-boiling attack, but instead groups of nodes collude together and move in opposite directions, again attacking both accuracy and stability.

IV. Description of Newton

In this section we present our VCS, Newton, which builds upon Vivaldi by implementing invariants derived from physical

laws to defend against all known insider attacks against VCS.

A. Vivaldi as a Physical System

The coordinate update in Vivaldi is actually modeled based on a mass-spring system abstraction, where each pair of nodes have a spring connecting them. Depending on its state, the spring applies a force to the nodes to either push them together or pull them apart. This force is calculated by Hooke's law, $F = -kx$, where k is a spring constant and x is the amount of displacement that a spring currently is from its equilibrium or rest position. Every node has a spring constant k value of 1. To determine displacement, the measured RTT between a pair of nodes is considered to be the length of the spring at its rest position, while the current length of the spring is the estimated RTT. Over time, the system stabilizes when all pairs of nodes minimize the amount of force that is placed upon them.

When updating its coordinate based on information from node j, a node i calculates the magnitude and direction of the force $\vec{f_{ij}}$ that node j is applying to it. The magnitude of the force m_{ij} is determined by the RTT between the two nodes and the distance of the current nodes' coordinates: $m_{ij} = RTT_{ij} - \|x_i - x_j\|$. The direction of the force $\vec{d_{ij}}$ is a unit vector that is calculated based on the two nodes' coordinates: $\vec{d_{ij}} = u(x_i - x_j)$. The force $\vec{f_{ij}}$ is then simply $\vec{f_{ij}} = m_{ij} * \vec{d_{ij}}$. This determines how much the coordinate needs to be updated from the previous value and corresponds to Line 6 in Algorithm 1. Note that Vivaldi is not a perfect physical system and also takes into account the perceived error reported by the node j and its own local error value. We discuss the implications of Vivaldi not being a perfect physical system in Sec. IV-E.

B. Using Physical Laws to Identify Invariants

Detecting insider attacks in distributed systems can benefit from identifying invariants in the system. For Vivaldi, no such invariants appear to exist at first glance since nodes make decisions based on inputs from nodes in their neighbor set and there are no constraints imposed by the system in node selection. We make the key observation that since Vivaldi [7] is built upon an abstraction of a mass-spring system, all nodes must follow physical laws. These laws are universal truths so they represent invariants that all nodes in Vivaldi should globally follow. In particular, nodes must follow Newton's three laws of motion which are:

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 the acceleration is proportional to the force and indirectly proportional to the mass.*

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.*

When an attacker lies about its own coordinate, it is implicitly lying about forces that have previously acted upon it, thus introducing extraneous *indirect forces* into the system. Introducing such forces into the system breaks the first and third laws, as attackers are not acting according to the influences of the outside forces upon them. When an attacker delays a measurement probe or lies about its local error value, it is lying about the force between itself and another node, thus introducing extraneous *direct forces* into the system. Lying about such forces breaks the second law, as nodes do not undergo accelerations that are governed by the forces determined by Hooke's law.

We show how to leverage Newton's three laws of motion to identify three invariants, which we call **IN1**, **IN2** and **IN3**. Nodes can then use these invariants to *locally* detect whether an update that results in a force being acted upon is the result of nodes behaving according to the protocol and thus following physical laws, or the result of a lying attacker. Below we define the invariants and describe how to detect extraneous indirect and direct forces with their help.

C. Detecting Extraneous Indirect Forces

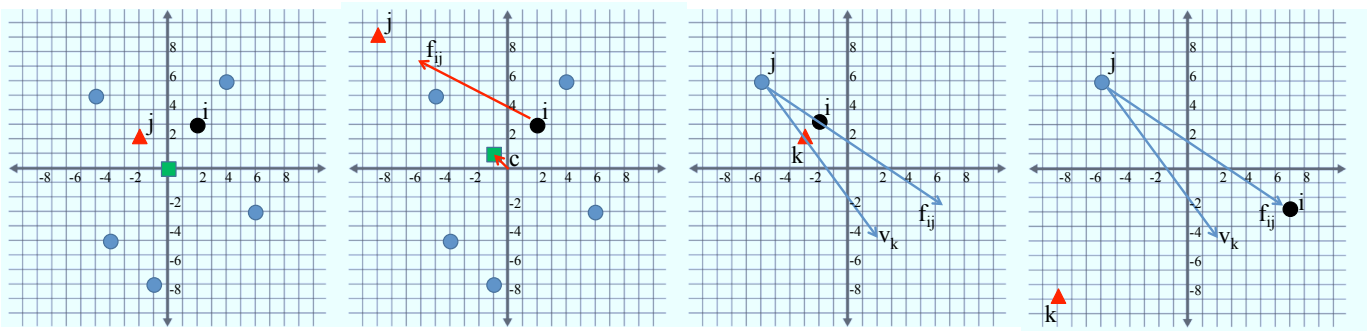
We first focus on how to detect whether a node is lying about the forces that have acted upon it, resulting in maliciously derived coordinates. For ease of exposition, assume each node i is at coordinate x_i and at any moment is applying the force $\vec{f_{ij}}$ onto node j. As described in Sec. II, a node chooses its neighbor set based on two criteria: (1) half are chosen randomly and (2) half are chosen based on if they are physically close. We design two detection schemes, one for nodes that are randomly chosen, and the other for nodes that are physically close.

Detection for malicious random nodes from the neighbor set: We observe that the third law states that there can be no unbalanced forces in the mass-spring system. An attacker introducing any extraneous indirect force that causes nodes to move will be an unbalanced force by definition of the first law. The third law then implies that an unbalanced force can be detected by finding the centroid of all the node's coordinates, where the centroid is the average of all the coordinates and has the physical analogue of being the center of mass of the mass-spring system. We note that while perfect detection requires knowledge of the coordinates of all nodes, using just the randomly selected nodes also provides a good vantage point from which to calculate an approximate centroid. We summarize our first invariant.

IN1: *If the centroid of a node i and the randomly selected nodes from its neighbor set is at the origin 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 ($\vec{c}$).*

In Figs. 1(a) and 1(b) we illustrate how to use **IN1** to detect attacks. In Fig. 1(a) node i, located at coordinate x_i = (2,3), is the victim and all the other dots are the randomly selected nodes from its neighbor set, including node j. Node i can calculate the centroid c based on its own coordinate and the coordinates of all those neighbors $c = \frac{\sum_{p=1}^{n} x_p}{n}$. Since the third law states that all forces must be balanced, we would expect that the centroid would never move, and thus even during normal operations, would be at the origin. In Fig. 1(a) the green square signifies this calculated centroid, and since no attack has taken place yet, it is at the origin.

In Fig. 1(b), we consider what happens when the attacker, node j, represented by the red triangle at coordinate x_j = (-



(a) Random: no attack (b) Random: attack (c) Physical close: no attack (d) Physical close: attack

Fig. 1: Detecting extraneous indirect forces

2,2), introduces an extraneous unbalanced force. In this case, the attacker moves to coordinate (-9,9). Node i recalculates the centroid, using $c_t = \frac{\sum_{p=1}^{n} x_{p_t} + f_{ij}}{n}$, to be at coordinate (-1,1), which corresponds to $\vec{c}$, the force that moved the centroid from the origin. Node i also experiences a force $\vec{f_{ij}}$, represented by the arrow pushing it towards the attacker. Node i can detect the attack by finding that $\vec{c}$ is non-zero, as described in **IN1**. It can then find which node introduced the unbalanced force, and thus is the attacker. Specifically, for every neighbor node k, i sums up the forces ($\vec{s_{ik}}$) that k has applied to it since k entered its neighbor set and then calculates the vector projection of $\vec{s_{ik}}$ onto $\vec{c}$. The node whose projection has the greatest magnitude is the one who has contributed most to the centroid being moved, thus an attacker, and its force is ignored.

IN1 holds even if a malicious node initially reports an incorrect coordinate because the system always starts in a correct state (all the nodes start at the origin, and so does the centroid).

Detection for malicious physically close nodes from neighbor set: For nodes that are physically close, we observe that because all nodes are connected via springs they will experience very similar forces from the same nodes. We can use the first law, which dictates that a node in a mass-spring system must move if acted upon by an external, unbalanced force. Moreover, the second law implies that we can detect if a node should be moving or not and we can calculate how much it should move. Our second invariant can now be summarized:
IN2: *Nodes i and k are physically close and if node i experiences a force $\vec{f_{ij}}$ from node j, then node i would expect node k to experience a force from j similar to the vector projection of $\vec{f_{ij}}$ onto the vector $u(x_j - x_k)$.*

We use Figs. 1(c) and 1(d) to illustrate **IN2**. Fig. 1(c) shows the nodes before the attack. The black dot at coordinate (-2,3) is node i, the victim, the blue dot at coordinate (-6,6) is node j, and the red triangle at coordinate (-3,2) is the attacker node k. Both i and k experience forces upon them from j. Node i can calculate what it expects the force upon k to be and thus determine that it expects k to update its coordinate to (2,-5).

Fig 1(d) shows the nodes when the attack happens. Node i does move according to the force applied to it to coordinate (7,-3). However, when k attacks by introducing an extraneous indirect force, it moves in a different direction than expected. To detect the attack, node i can calculate the force value for node k as described in **IN2** for every force that is applied to itself and sum up that value ($\vec{v_k}$). Node i will remember the previous coordinate that was reported by k and when it receives a new update from k it calculates the change (Δx_k). This difference and the sum of vector projections $\vec{v_k}$ should be equivalent, if they are not, then k did not move according to the external unbalanced force.

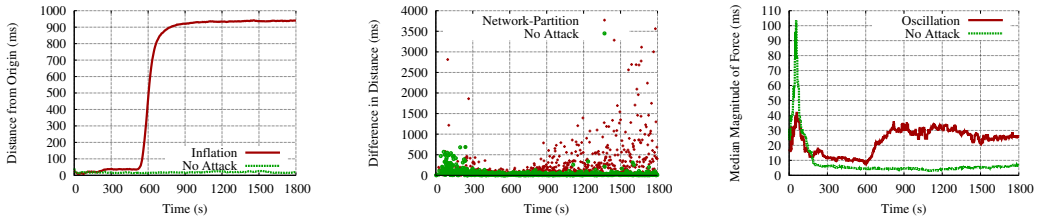
D. Detecting Extraneous Direct Forces

We now focus on how to detect if a force directly acting on a node is extraneous and is caused by a malicious process. To accomplish this, we leverage the second law of motion and Hooke's law. The second law states how much a node should accelerate given the force and mass of a node. In our mass-spring system, the mass of every node is 1, and thus can be ignored. In a mass-spring system, the amount of force applied to a node is controlled by Hooke's law, $F = -kx$ which states that the amount of force on a node is proportional to the spring's current displacement from its rest position. We now state our third and final invariant:
IN3: *As the springs in the physical system stabilize and come closer to their rest position, nodes should decelerate and thus the forces that are applied to them should decrease over time.*

IN3 applies also to joining and leaving nodes. While joining nodes may lie about their initial force, **IN3** obliges a decreasing force over time. Leaving nodes stop moving and the force becomes zero.

One possible detection scheme is to impose a certain rate of decrease on the forces applied to a node, and if the force is larger than expected, offending nodes are considered malicious. However, we have experimentally found that this approach is too strict for real deployments, due to practical aspects of the Internet. First, triangle inequality violations result in nodes stabilizing even though springs are still exerting force on nodes. Thus we can expect forces to never decrease all the way to zero, but rather opposing forces will simply be balanced. Second, **IN3** assumes that latencies do not change as real springs can not change their rest position. However, on the real Internet this will not hold as routes change and mobile nodes move.

We instead take a different approach. A node calculates the median $\tilde{f}$ and median absolute deviation D of the magnitude of the force that each node is applying to it. Then if the magnitude of any force m_j is a few deviations larger than the median $m_j > \tilde{f} + k * D$, the node will ignore it. We use the median and median absolute deviation instead of the average and standard deviation, as the former are more robust to outliers and have been shown to be resilient against frog-



(a) **IN1** with Inflation (b) **IN2** with Network-Partition (c) **IN3** with Oscillation

Fig. 2: Invariants shown through deployments of Vivaldi on PlanetLab.

boiling attacks [24].

E. Using IN1, IN2, and IN3 to Design Newton

We use **IN1**, **IN2**, and **IN3** combined with Vivaldi to create Newton. We investigate if these invariants hold on real deployments of Vivaldi on PlanetLab. While Vivaldi models a mass-spring system, the actual protocol, and more importantly, any network on which it runs, will not perfectly emulate a physical mass-spring system. Thus, we expect some discrepancy between the ideal physical system and the real deployed system. We investigate these discrepancies and use the results to calibrate Newton.

We use results of Vivaldi on PlanetLab deployments of 500 nodes to investigate the invariants. We implement all 5 attacks (inflation, deflation, oscillation, frog-boiling, and network-partition) and plot results relevant to each invariant. As inflation and deflation share similar characteristics, with inflation being a more damaging attack, and network-partition is a stronger variant of the frog-boiling attack, Fig. 2 shows results for inflation, oscillation, and network-partition. Each attack starts at 600 seconds into the experiment and are conducted where 10% of nodes are attackers.

IN1. In Fig. 2(a), we plot the distance from the origin to the centroid of the coordinates of randomly chosen neighbor nodes, averaged for all nodes in the system. We expect this distance to be zero or very small. When there is no attack, we find the centroid to be less than 20 ms away from the origin. However, during an inflation attack, the value increases drastically as nodes start to lie about their coordinate. We select a threshold of 20 ms to detect an attack.

IN2. In Fig. 2(b) we plot the difference in distance from where physically close nodes were expected to have their coordinates located at, versus where they actually reported themselves to be. We expect this value to be zero or very small. When there is no attack on Vivaldi, we find these values to be small, with most less than 50 ms. When under a network-partition attack, these values increase dramatically, especially the further a node has moved from its correct coordinate. To find a good value for the threshold we conducted a sensitivity study by varying it between 10 and 50 and then finding the true positive rate (TPR) and the false positive rate (FPR) when classifying updates. We show the results in Table I and found a good threshold for detecting the attack to be 35 ms, which trades-off discarding some benign updates for better detection of malicious nodes.

IN3. Fig. 2(c) depicts the median of the magnitude of the force applied to a single node over time. We see that while there is not a strictly decreasing line as one would expect in an ideal system, the general trend is present. Also, in an oscillation attack we see that this value quickly grows and is inconsistent with benign behavior. To detect attackers, we have found that it is best to calculate the median separately for randomly chosen nodes and physically close nodes. This is due to physically close nodes having smaller force values, but deviate more from the median, while randomly chosen nodes have the opposite characteristics. Thus we choose a threshold of 8 absolute deviations for physically close nodes and 5 for randomly chosen nodes.

TABLE I: Sensitivity on threshold for IN2

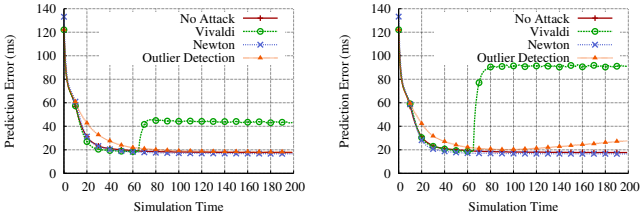
Threshold (ms)	FPR	TPR
10	0.57	0.98
15	0.37	0.97
20	0.27	0.95
25	0.19	0.91
30	0.14	0.90
35	0.11	0.84
40	0.09	0.83
45	0.08	0.75
50	0.06	0.61

Implementation. To implement Newton, we started with the base code of Vivaldi and then added the invariants. In Newton, every node checks the invariants after receiving an update from another node. If at least one invariant is violated the update is discarded.

Thresholds. Because Newton uses thresholds that rely on a fixed point as a reference, such as the origin, they are more difficult to exploit by an attacker. Nevertheless, an attacker can still try to exploit these thresholds by staying under their values.We discuss scenarios where an attacker can exploit these thresholds in Sec. V-D and show that Newton is robust even under such scenarios.

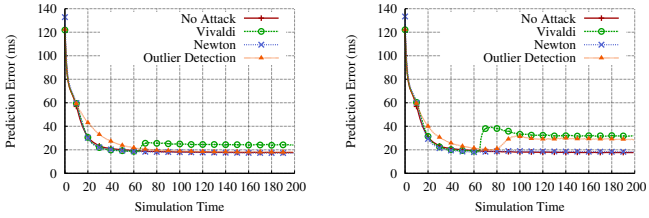
Overhead. As Vivaldi is an efficient and low cost service for latency estimation, we also aimed to preserve that goal in designing Newton. As such, we do not add any extra network communication, as the use of our invariants do not require it, and the added computation and memory usage are very small.

Non-Euclidiean spaces. Since Newton is based on physical laws found in our Euclidean-based world, we investigate if Newton works in non-Euclidean spaces. We show results for Newton in hyperbolic spaces in Sec. V-C. Furthermore, as the most general form of non-Euclidean spaces are Riemannian manifolds, and as the Nash embedding theorem says that any m dimensional Riemannian manifold can be embedded isometrically in some Euclidean space, we see that the defined invariants would still hold in non-Euclidean spaces. However, the construction of this isometrical embedding is not straightforward and if pseudo-Riemannian manifolds are used for virtual coordinates then no such embedding might exist.



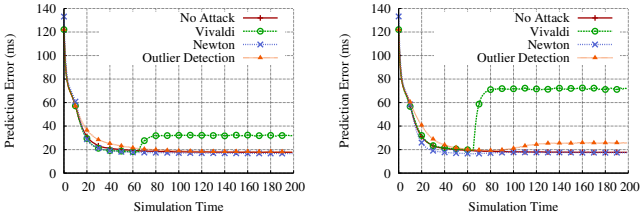
(a) Accuracy - 10% attackers (b) Accuracy - 30% attackers

Fig. 3: Simulation results - inflation attack



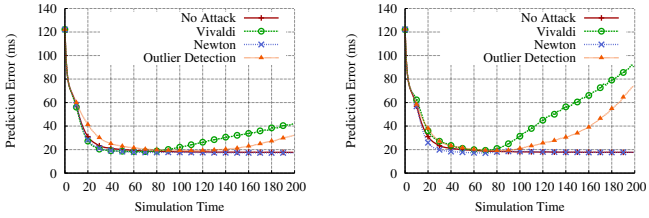
(a) Accuracy - 10% attackers (b) Accuracy - 30% attackers

Fig. 4: Simulation results - deflation attack



(a) Accuracy - 10% attackers (b) Accuracy - 30% attackers

Fig. 5: Simulation results - oscillation attack



(a) Accuracy - 10% attackers (b) Accuracy - 30% attackers

Fig. 6: Simulation results - frog-boiling attack

V. Simulation Results

We show through simulations, using the p2psim simulator [25], how effective Newton is in defending against attacks. We compare Newton against the unsecured Vivaldi and also Vivaldi outfitted with Outlier Detection [18], referred to as *Outlier Detection*. We also include Vivaldi when no attackers are present, referred to as *No Attack*, as a baseline comparison.

We use the King data set [26] which contains Internet pairwise measurements between 1740 nodes (average RTT is 180 ms and maximum RTT is 800 ms). Simulations last for 200 time units, where each time unit is 500 seconds. Each node joins at the beginning of the simulation in a flash-crowd scenario and remains for the entire duration. We use a typical setting for Vivaldi [7], where every node has a neighbor set of 64 nodes, with half randomly chosen and the other half being nodes with low RTT (also referred to as physically close nodes). The attackers are chosen randomly from all nodes. Unless otherwise stated, malicious nodes start their attack at one-third of the way through the simulation. This is to give a fair comparison for Outlier Detection, as it needs to learn what good behavior is. Outlier Detection uses spatial and temporal thresholds of 1.25 and 4, respectively, as described in [18]. Newton uses the thresholds described in Sec. IV-E which any Internet-wide deployment could use. For the coordinate space, we use a Euclidean distance and gradient function in 2 dimensions, unless otherwise stated.

A. Attacks Mitigation

We vary the percentage of nodes that are attackers between 10%, 20%, and 30%. For lack of space and similarity of results we show only the 10% and 30% cases.

Inflation. Figs. 3(a) and 3(b) show the accuracy under an inflation attack. We can see that when under attack Vivaldi has very poor accuracy, which gets increasingly worse with the percentage of attackers. Both Outlier Detection and Newton are able to effectively keep the error low after the attack starts. However, as the percentage of malicious attackers increase, Outlier Detection's prediction error also increases as time progresses, while Newton is able to match the baseline prediction error. We attribute Newton's performance to its ability to detect that the attacker nodes are introducing unbalanced forces and thus shifting the centroid far away from the origin.

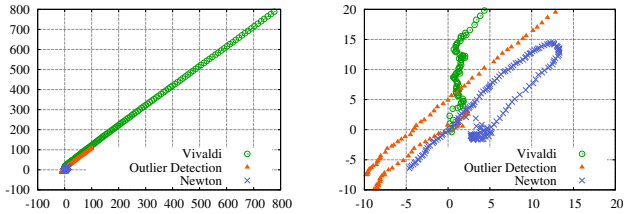
Deflation. Results for the impact of the deflation attack on accuracy are in Figs. 4(a) and 4(b). The deflation attack does not have as great of an impact on Vivaldi as inflation, but the opposite is true of Outlier Detection. However, we see again that Newton is able to successfully mitigate the attack.

Oscillation. The oscillation attack is different from the previous two attacks in that while attackers lie about their coordinates in a random way, they also delay measurement probes up to 1 second. We show the results of how the different systems handle the attack and the impact on accuracy in Figs. 5(a) and 5(b). Outlier Detection is able to withstand the attacks until there are 30% attackers, when the prediction error increases to 26 ms. However, Newton continues to provide good performance for all percentage of attackers. We attribute this to **IN3**, requiring forces to decrease over time.

Frog-boiling. The frog-boiling attack, has been shown in [20]–[22] to be an effective attack against VCS defenses that must learn over time what good behavior is. We now show the impact of the attack on accuracy in Figs. 6(a) and 6(b). Similar to previous works, we see that Outlier Detection indeed does not protect against such an attack. Newton, though, is able to successfully protect against the frog-boiling attack.

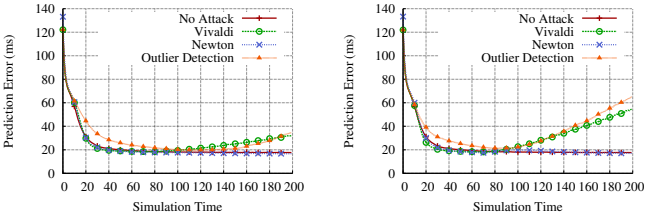
We give insights about how Newton works in Fig. 7(a) showing how the centroid moves over time on the coordinate plane when under attack (10% attackers). Vivaldi's centroid moves far away from the origin. Outlier Detection's centroid does not move as far, but still it moves close to (100,100). To be able to see how Newton's centroid moves, we show a zoomed in picture in Fig. 7(b). Newton's centroid also initially moves away from the origin, until it almost reaches coordinate (13,15). At this point individual nodes calculate that the centroid is near 20 ms away from the origin, thus triggering the detection mechanism. The honest nodes can then determine who the attackers are and ignore their updates.

Network-partition. The network-partition attack is similar to the frog-boiling attack, except multiple groups of attackers move in opposite directions, trying to split the network. We consider four groups of nodes moving in four different direc-



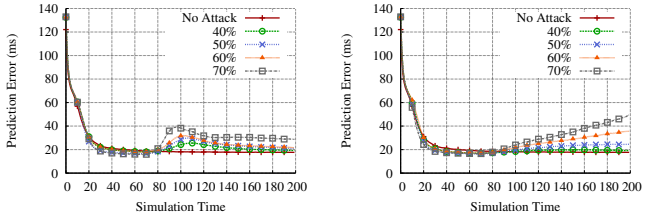
(a) Centroid (b) Centroid of Newton

Fig. 7: Centroid over time for frog-boiling attack



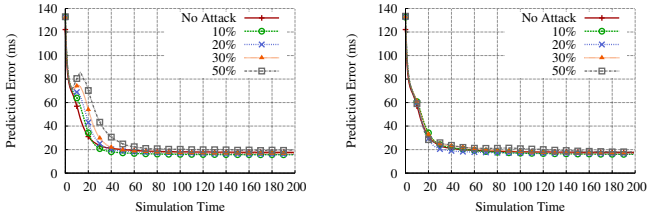
(a) Accuracy - 10% attackers (b) Accuracy - 30% attackers

Fig. 8: Simulation results - network-partition attack



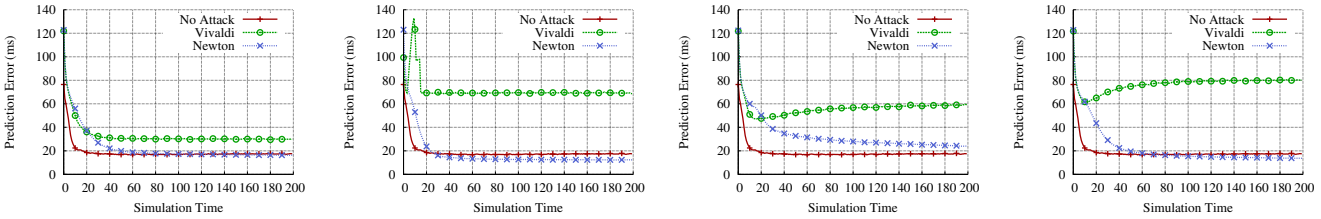
(a) Frog-boiling (b) Network-partition

Fig. 9: Simulation results - high percentage of attackers



(a) Oscillation (b) Frog-boiling

Fig. 10: Simulation results - attacks start at the beginning



(a) Deflation attack (b) Oscillation attack (c) Frog-boiling attack (d) Network-partition attack

Fig. 11: Simulation results - accuracy when using 4 dimensions in hyperbolic space with 30% attackers

tions. Figs. 8(a) and 8(b) show the accuracy for the different systems under attack. This attack is successful against Outlier Detection, while Newton still performs well under attack. This is even though groups of attackers moving in different directions give the illusion that they are actually acting according to balanced forces by not moving the centroid, thus making it difficult to detect this attack using **IN1**. However, in this case, attackers that are physically close can still be detected by **IN2** and all types of attackers can be detected by **IN3**.

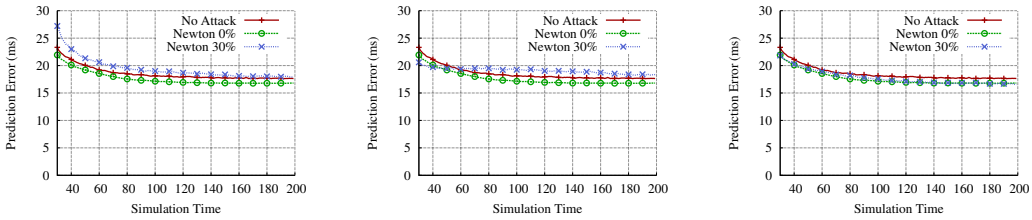
B. Extreme Attack Scenarios

High percentage of attackers. We also show extreme scenarios where Newton must face an increasing percentage of attackers. We show the advanced attacks in Fig. 9, results were similar for the basic attacks, but we did not include them due to space constraints. Overall, we see that Newton is able to handle 50% attackers without losing significant accuracy. However, under 60% and 70% of attackers accuracy starts to degrade, particularly for the network-partition attacks. We point out that each node updates its coordinate based on a set of 64 nodes, thus the high-percentage of malicious nodes results into a lower percentage in the neighbor set. For example, using the analysis from [18], when there are 70% malicious nodes in the entire network, about 54% of nodes will be malicious in the neighbor set and thus able to manipulate the median that is used to detect extraneous direct forces.

Attacks before system converges to steady state. In previous simulations, we showed performance when there was a period before attacks started to allow Outlier Detection to learn good behavior. Newton does not need such period since it is based on invariants. We show results only for oscillation and frog-boiling attacks as results for the other attacks were similar. Fig. 10 shows results when attacks start from the beginning of the simulation. As can be seen, Newton mitigates the attacks. Under the oscillation attack, as the percentage of attackers increase, it takes slightly longer for coordinates to stabilize and become accurate. This is because we do not enforce a strict rate of decrease on the amount of force between two nodes and instead use the median force to detect nodes. Nodes must first sample a number of forces before they can calculate the correct median. Thus, in Newton an honest node cannot immediately detect if a node is artificially increasing the force between itself and another node.

C. Newton in Higher-dimensional and Hyperbolic Space

So far we have shown that Newton works well in simple 2-dimensional Euclidian coordinate spaces. However, more complex spaces have been shown in the past to improve prediction error. For example, Ledlie *et al.* [27] have shown through a Principal Component Analysis that 4 dimensions are appropriate for Internet-scale network coordinates. Hyperbolic spaces also have been proposed as an alternative to Euclidean spaces as they better represent the structure of the Internet [28]. Several works have applied Vivaldi to such spaces and have shown that it does produce an accurate embedding [29, 30]. Modifying Vivaldi and Newton to work in hyperbolic spaces simply involves changing the distance and gradient function. We implement these functions as described in [30]. Hyperbolic spaces also have a curvature parameter that describes how much a line deviates from being flat. We experimentally found that a value of 60 provides good accuracy in benign environments. We ran simulations in hyperbolic space in 4



(a) Attackers push the **IN1** threshold (b) Attackers push the **IN2** threshold (c) Attackers push the **IN3** threshold

Fig. 12: Simulation results - attackers (30%) push the limits of the thresholds used by **IN1**, **IN2**, and **IN3**

dimensions. We find that for 10% and 20% attackers, Newton performs better than the baseline. Newton continues to work well even under 30% attackers, which we show in Fig. 11.

D. Invariants under Attack

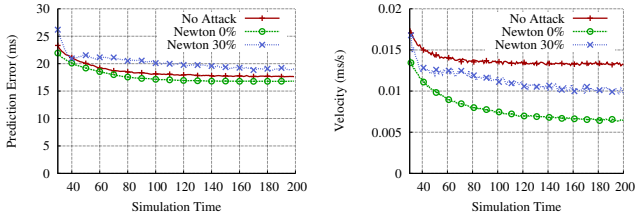
Because in real-deployments Newton does not behave exactly like a physical system, it uses thresholds for the three invariants. We note that Newton's thresholds use as a reference a fixed point such as the origin, while Outlier Detection's thresholds use as a reference a moving point (the centroid of metrics derived from all nodes in the neighbor set), allowing attacks such as frog-boiling to move it. Thus, Newton's thresholds are more difficult to exploit by an attacker. However, an adaptive attacker can still exploit the values of the thresholds used by Newton to his advantage.

We conduct three tests, one for each invariant, where the attacker tries to remain undetected, yet come as close to the threshold as possible. For **IN1**, the attackers push the centroid to right below the 20 ms threshold. For **IN2**, attackers initially move as the forces dictate, but then always shift just below 35 ms away from this position. Finally, for **IN3**, attackers delay probes only enough to stay beneath the deviation threshold. The results of these tests are shown in Fig. 12, where we zoom in on the results of the steady state performance to see the effects. We compare the normal baseline of Vivaldi when no attack occurs, Newton when no attack occurs, labeled *Newton 0%*, and also Newton when there are 30% attackers, labeled *Newton 30%*. We find that even 30% attackers can not significantly increase the prediction error.

Attackers can also conduct a new attack, which we call the *rotation attack*, where the goal is not necessarily to disrupt accuracy, but rather stability. In this attack, colluding nodes rotate around the origin in the same direction at a slow rate. This attack will not trigger **IN1**, and if done slowly enough, will bypass the thresholds of **IN2** and **IN3**. We implement this attack and show the results in Fig. 13 (notice the zoomed-in y axis scale). We find the accuracy in Fig. 13(a) to only be slightly raised over our baseline. Stability, as shown in Fig. 13(b), is also raised over Newton's normal levels, but is not yet worse than the baseline.

VI. Experimental Results

We evaluate Newton in real-life experiments on the PlanetLab testbed. We use 500 nodes and run each experiment for 30 minutes, unless otherwise stated. Every second a node chooses one of its neighbors to probe and gets their coordinate update. We use Newton configured for a Euclidean coordinate space. Due to PlanetLab being an Internet-scale testbed, we use 4 dimensions as suggested by Ledlie *et al.* [27]. Malicious nodes



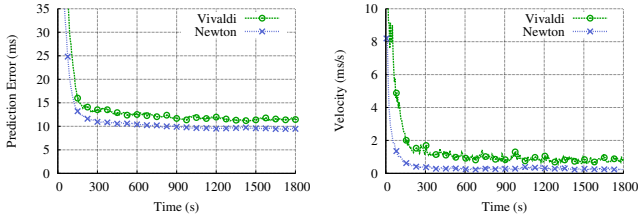
(a) Accuracy (b) Stability

Fig. 13: Simulation results - attackers (30%) rotate around the origin at a slow rate

start performing attacks immediately once the experiment starts. All other parameters are the same as in the simulations. We compare Newton with Vivaldi under attacks and consider as baseline Vivaldi with no attacks.

A. Performance in Benign Networks

We first show the results when there are no attackers in Fig. 14. Accuracy is shown in Fig. 14(a) where the prediction error is lower in both Vivaldi and Newton for PlanetLab than the simulations. This is most likely due to the smaller number of nodes involved as the error needs to be minimized for a fewer number of nodes. Furthermore, Newton only has a resulting prediction error of 9 ms, while Vivaldi has one of 12 ms. The difference in stability has also increased over the simulations, as shown in Fig. 14(b). Vivaldi has a resulting velocity of 0.8 ms/s, while Newton is only 0.25 ms/s. This increase in accuracy and stability is due to Newton being less sensitive to probes that get delayed occasionally as the result of benign occurrences such as queueing delays on routers.



(a) Accuracy (b) Stability

Fig. 14: PlanetLab results - no attackers

Adapting to changes in the network. In real deployments, such as on PlanetLab, route changes will take place, potentially having an effect on **IN3**. To show that Newton can withstand such changes, we run Newton for *four days* on 350 nodes on PlanetLab. For this particular experiment we reduce the frequency of how often a node sends a probe to a neighbor to 5 seconds, all other parameters remained the same as before. We performed traceroutes between all-pairs of nodes before and after the experiment to estimate the number of routes changed. We conservatively only count routes as changed if they contain

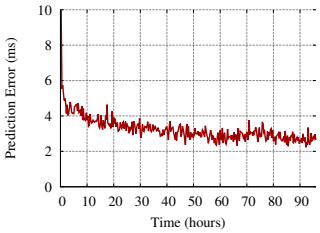


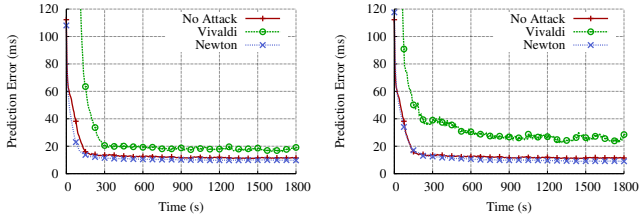
Fig. 15: PlanetLab results - Accuracy of Newton for 4 days

different routers and also have a difference in RTTs greater than 10 ms. We find that 12% of all routes changed.

Fig. 15 shows the results. Initially, Newton is able to stabilize within an hour to 6 ms of error. We attribute this smaller error, compared to the 9 ms seen earlier, to the smaller number of nodes that must embed coordinates. However, over time, Newton reduces the error even further to 3 ms. We also investigate in more details what happens when routes change. We find that in many cases the resulting change is not so large that **IN3** is violated. However, there are cases in which **IN3** is violated for a short period of time, for one of the two nodes. This is due to when a single path between routers change, it often affects many end-to-end routes for one node, thus causing RTTs to multiple neighbors to change simultaneously. Thus, one node will realize that many neighbors are putting extra force on it, and change its coordinate accordingly.

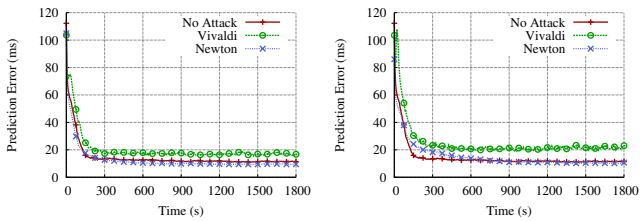
B. Attack Mitigation

Inflation and deflation. Figs. 16 and 17 show accuracy under inflation and deflation attacks respectively, for 10% and 30% attackers in the system. We find that the inflation attack is not as effective against Vivaldi in these experiments as it is in the simulations, even though in the experiments we increased the amount that attackers lie about so that they have larger coordinates. The deflation attack is also not as effective as in simulations. In both cases, Newton is able to handle such attacks while having better accuracy than in the benign setting.



(a) Accuracy - 10% attackers (b) Accuracy - 30% attackers

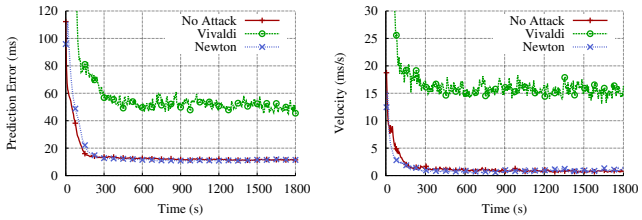
Fig. 16: PlanetLab results - inflation attack



(a) Accuracy - 10% attackers (b) Accuracy - 30% attackers

Fig. 17: PlanetLab results - deflation attack

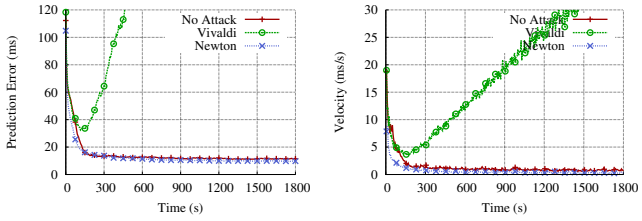
Oscillation. For the rest of the attacks, we show just 30% attackers. We conducted experiments with lower percentages of attackers, but we did not include them because of lack of space and similarity. Fig. 18 shows accuracy and stability under the oscillation attack. This attack proves to be more damaging in the experiments than the simulations for Vivaldi. Newton though, because it is taking advantage of **IN3**, is able to mitigate such attacks easily.



(a) Accuracy - 10% attackers (b) Stability - 30% attackers

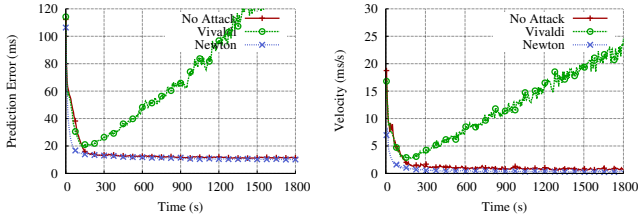
Fig. 18: PlanetLab results - oscillation attack

Frog-boiling and network-partition. Results for frog-boiling are shown in Fig. 19, which while we find it to be the most effective attack on Vivaldi, for reasons previously explained, it has no effect on Newton. Unsurprisingly, we find that the network-partition attack, which is similar to the frog-boiling attack but nodes move in different (four in our case) directions, has similar results to it. We plot the effects of this attack in Fig. 20.



(a) Accuracy - 10% attackers (b) Stability - 30% attackers

Fig. 19: PlanetLab results - frog-boiling attack



(a) Accuracy - 10% attackers (b) Stability - 30% attackers

Fig. 20: PlanetLab results - network-partition attack

VII. Related Work

Much research has been conducted to find detection and mitigation techniques against attacks [16] in VCS.

Landmark-based defenses: Kaafar *et al.* [17] propose to model the behavior of trusted landmark nodes using a Kalman filter, this provides an outlier detection scheme by which nodes learn good behavior and can then filter out malicious updates. Their technique requires 8% of all nodes to be trusted, which could be non-trivial to obtain given a large deployment. Similarly, Saucez *et al.* [31] define a reputation based system that leverage trusted nodes and a reputation certification agent to calculate the other nodes reputation. Treeple [22], while not strictly coordinate based, provides secure latency estimation, using landmarks as vantage points for providing traceroutes on the Internet. In Treeple,

landmarks perform traceroute measurements to peers, which the landmarks can then digitally sign and provide for nodes to compute the network distance themselves. As landmark-based defenses have stronger assumptions, as they require *a priori* trusted nodes, we do not compare Newton to them as Newton is a decentralized defense and does not require trusted nodes.

Decentralized defenses: Zage *et al.* [18] propose the usage of spatial and temporal properties of nodes to learn what good behavior is, then by using outlier detection detect anomalous coordinate updates. Veracity [19] uses a voting scheme to verify potentially malicious coordinate updates by using a subset of nodes. Each node maintains a verification set where several other nodes attest to whether a particular update increases their estimation error above a certain threshold, and if so, ignores it. Suspected nodes are tested based on their error to the verifying nodes; nodes with large errors are considered malicious.

Although these decentralized defenses differ in the way they secure virtual coordinate systems, they both, along with [17], suffer from the frog-boiling attack [20]–[22]. A few works have been proposed to defend against the frog-boiling attack. Wang *et al.* [20] proposes detecting attackers that lie about coordinates by using the PeerReview [32] accountability protocol. Since, if implemented, this approach would have higher costs than our method (i.e. bandwidth, storage for a tamper-evident log, and computation for public-key cryptography), we do not compare Newton with them. Becker *et al.* [33] propose a method for detecting frog-boiling by using a machine learning approach, where through a training data set the system learns what normal and abnormal data is. In contrast, our approach has no need to train the system and can detect abnormal behavior directly due to the applied physical laws. Furthermore, while [33] can detect attacks are occuring but not find and discard the updates that are causing it, Newton is able to do both.

VIII. Conclusion

We introduced Newton, a new approach to providing a secure VCS by going back to the abstraction that Vivaldi is based on, a physical mass-spring system. In accordance with the abstraction, our defenses are based on the three laws of motion as put forward by Newton.We have explained in depth how the laws provide invariants for our system and how they are leveraged to mitigate basic attacks such as inflation, deflation, and oscillation but also more advanced attacks like frog-boiling, and network-partition attacks. Through simulations and experiments on the PlanetLab testbed we showed that Newton outperforms Vivaldi even in benign settings and is able to mitigate the advanced attacks that remained undetected by Outlier Detection. Newton can also cope with advanced attackers that might leverage insider knowledge about calibration specific parameters used by Newton. Newton is immune to such attacks, since the calibration of the defense mechanism is relying on robust and fixed, time independent thresholds.

References

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM*, 2001.

[2] Y. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," in *Proc. of SIGMETRICS*, 2000.

[3] J. Ledlie, M. Mitzenmacher, and M. Seltzer, "Wired geometric routing," in *IPTPS*, 2007.

[4] R. Gummadi and R. Govindan, "Reduced state routing in the internet," in *ACM HotNets Workshop*, 2004.

[5] J. Cowling, D. R. K. Ports, B. Liskov, R. Ada, and P. A. Gaikwad, "Census: Location-aware membership management for large-scale distributed systems," in *USENIX*, 2009.

[6] Y. Shavitt and T. Tankel, "Big-bang simulation for embedding network distances in euclidean space," in *INFOCOM*, 2004.

[7] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: a decentralized network coordinate system," in *Proc. of ACM SIGCOMM*, 2004.

[8] T. Ng and H. Zhang, "A network positioning system for the internet," in *Proc. of USENIX*, 2004.

[9] E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *Proc. of INFOCOM*, 2002.

[10] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: A Global Internet Host Distance Estimation Service," *IEEE/ACM Trans. Netw.*, vol. 9, p. 525, 2001.

[11] L. Tang and M. Crovella, "Virtual landmarks for the internet," in *Proc. of SIGCOMM*, 2003.

[12] M. Pias, J. Crowcroft, S. Wilbur, S. Bhatti, and T. Harris, "Lighthouses for scalable distributed location," in *Proc. of IPTPS*, 2003.

[13] M. Costa, M. Castro, R. Rowstron, and P. Key, "PIC: practical Internet coordinates for distance estimation," in *Proc. of ICDCS*, 2004.

[14] L. Lehman and S. Lerman, "A decentralized network coordinate system for robust internet distance," in *Proc. of ITNG*, 2006.

[15] ——, "Pcoord: Network position estimation using peer-to-peer measurements," in *Proc. of NCA*, 2004.

[16] M. A. Kaafar, L. Mathy, T. Turletti, and W. Dabbous, "Virtual networks under attack: Disrupting internet coordinate systems," in *Proc. of CoNext*, 2006.

[17] M. A. Kaafar, L. Mathy, C. B. K. Salamatian, T. Turletti, and W. Dabbous, "Securing internet coordinate embedding systems," in *Proc. of SIGCOMM*, 2007.

[18] D. Zage and C. Nita-Rotaru, "On the accuracy of decentralized virtual coordinate systems in adversarial networks," in *Proc. of CCS*, 2007.

[19] M. Sherr, M. Blaze, and B. T. Loo, "Veracity: Practical secure network coordinates via vote-based agreements," in *Proc. of USENIX ATC*, 2009.

[20] G. Wang and T. S. E. Ng, "Distributed algorithms for stable and secure network coordinates," in *IMC*, 2008.

[21] E. Chan-tin, D. Feldman, N. Hopper, and Y. Kim, "The frog-boiling attack: Limitations of anomaly detection for secure network coordinate systems," in *SecureComm*, 2009.

[22] E. Chan-Tin and N. Hopper, "Accurate and provably secure latency estimation with treeple," in *NDSS*, 2011.

[23] M. Mamei, F. Zambonelli, and L. Leonardi, "Co-fields: A physically inspired approach to distributed motion coordination," *IEEE Pervasive Computing*, vol. 3, no. 2, pp. 52–61, 2004.

[24] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S. Lau, S. Rao, N. Taft, and J. D. Tygar, "Antidote: understanding and defending against poisoning of anomaly detectors," in *IMC*, 2009.

[25] p2psim: A simulator for peer-to-peer protocols, http://pdos.csail.mit.edu/p2psim.

[26] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *Proc. of ACM SIGCOMM-IMW*, 2002.

[27] J. Ledlie, P. Gardner, and M. Seltzer, "Network coordinates in the wild," in *Proc. of USENIX NSDI*, 2007.

[28] Y. Shavitt and T. Tankel, "On the curvature of the internet and its usage for overlay construction and distance estimation," in *INFOCOM*, 2004.

[29] C. Lumezanu and N. Spring, "Measurement manipulation and space selection in network coordinates," in *ICDCS*, 2008.

[30] Y. Fu and Y. Wang, "Hyperspring: Accurate and stable latency estimation in the hyperbolic space," in *ICPADS*, 2009.

[31] D. Saucez, B. Donnet, and O. Bonaventure, "A reputation-based approach for securing vivaldi embedding system," *Lecture Notes in Computer Science*, vol. 4606, p. 78, 2007.

[32] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: practical accountability for distributed systems," in *SOSP*, 2007.

[33] S. Becker, J. Seibert, C. Nita-Rotaru, and R. State, "Securing application-level topology estimation networks: Facing the frog-boiling attack," in *RAID*, 2011.