Secure and Accurate Network Coordinate Systems

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

D. Eric Chan-Tin

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Nicholas Hopper

May, 2011

Acknowledgements

I thank my adviser Nicholas Hopper very much for keeping faith in me over the past five years. Without him, I would never have finished this thesis and degree. Thank you Nick for all your patience and time. I hope I can live up to your expectations.

Thank you also to Yongdae Kim for his insightful comments and encouragements to always push myself. It was a privilege working with Yongdae.

I also thank the two other members of my committee: Tian He and Andrew Odlyzko for their invaluable comments about my thesis.

A huge thanks to all the members, past and present, of the security group *sclab* at the University of Minnesota, for keeping up the humor and various help with ideas and programming.

Obviously, I am thankful for my wife for putting up with me over the years. I am also grateful for the support of my mother and brother. Finally, thanking my father who could not see me finish this work.

Dedication

To Youa, Philippe, Corinne, and Sebastien

Abstract

Network coordinate systems allow a node to estimate the network latency between any pair of nodes on the Internet, without having to directly contact the nodes. Existing network coordinate systems have been shown to be very accurate in predicting network distances, and efficient with low computational and communication overhead. However, a malicious node participating in the system can lie about either its network coordinates or its network latency to other nodes, with the end result being to disrupt the whole system, making it inaccurate in predicting network latencies, or isolating targeted victims from the rest of the network. Over the past few years, several schemes have been proposed to secure network coordinate systems. They can be categorized in two: 1) statistical methods that try to filter out malicious peers, and 2) non-statistical methods such as reputation systems to ensure that nodes' reported coordinates are correct and verified.

The main contributions of this thesis are to 1) introduce a new attack, the *Frog-Boiling* attack that bypasses all the "secure" schemes previously designed, 2) define a security model and realistic threat model, 3) show how insecure network coordinates can be mis-used to attack a real application, such as hijacking the routing layer of the Vuze BitTorrent client, and 4) propose two secure designs; the first one, *Treeple* is provably secure under our model while providing accurate estimations, and the second scheme, *KoNKS* is secure on an average-case but provides a completely decentralized solution to network coordinates, and can be used as a "base" for existing secure network coordinate schemes.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Network coordinate systems [1, 2, 3, 4, 5, 6] assign a *network coordinate* to each node in a distributed system such that the distance between two nodes' coordinates provides a good estimate of the Internet round-trip time (RTT) between the nodes. The ability to estimate the network distance between arbitrary peers is useful in many cases: for example, finding the closest node to download content from in a content distribution network or file-sharing system [7]; choosing peers for routing in a DHT [8]; reducing inter-ISP communication [9, 10]; reducing state in Internet routers [11, 12, 13]; detecting Sybil attacks [14, 15]; improving the Tor [16] router selection algorithm [17]; conducting byzantine leader elections [18]; and online game matchmaking [19]. Early network coordinate systems were shown to have reasonable accuracy and fast convergence, while later schemes added features such as coordinate stability under churn and measurement uncertainty [20, 21, 1].

Unfortunately, Kaafar *et al.* [22, 23] demonstrated that these early network coordinate systems were vulnerable to very simple attacks, in which adversarial nodes disrupt the coordinate system by claiming to have randomly chosen positions and adding random delays to their outgoing messages. In response, several schemes to mitigate these attacks have been proposed [24, 25, 26, 27, 28]. Zage and Nita-Rotaru [24] proposed a mechanism, based on real-time statistical analysis of nodes' coordinates, to detect and discard adversarial inputs. A similar mechanism was proposed and evaluated by [25]. Both methods rely on outlier detection using statistical models – respectively, the Mahalanobis distance and Kalman filters – of coordinate evolution. More recently,

Veracity [26] – a decentralized reputation system, which verifies the self-reported coordinates of peers and aims at preventing peers from delaying their network measurements, was proposed as a "secure" network coordinate system.

Vuze [7] is a popular, open-source, BitTorrent [29] client, with over 2 million concurrent users. It includes an implementation of Vivaldi [1], a popular and widely-cited network coordinate system. Vivaldi is turned on by default and is used in the "distributed" portion of Vuze. When searching for content in Vuze, a user would contact both the central servers and its distributed hash table (DHT) for peer-to-peer downloads. The Vuze DHT is based on the Kademlia DHT [30]. Vuze uses the Vivaldi network coordinates in deciding which peers to contact next when searching for content. Peers that are closest to itself, in terms of network distance, will be contacted first to obtain other peers that are closer to the content location.

The contributions of this thesis are

- Introduce a new attack, called the *Frog-Boiling* attack, which bypasses **all** the secure schemes proposed so far.
- Show that an insecure network coordinate system can exploited to attack the DHT routing infrastructure, that is, hijacking every search lookup in the Vuze BitTorrent network.
- Define a security model and realistic threat model.
- Propose two secure and accurate schemes *Treeple* and *KoNKS*.

1.1 Frog-Boiling Attack

We demonstrate the inherent challenge in designing a secure network coordinate system based on rejecting "bad" inputs that do not conform. We propose the *frog-boiling attack*, where an adversary disrupts the network while consistently operating within the threshold of rejection. This is analogous to the popular account that a frog put in hot water will quickly jump out but a frog placed in cold water that is gradually brought to a boil will not notice the change and boil to death. The adversary sends "small-step" fake updates (inflated RTT or coordinates) to nodes in the network. The "step" is small

enough that it does not trigger either the anomaly detection or the verification methods used by Veracity, but the nodes targeted are still disrupted – their network coordinate is not accurate. Thus, the coordinates of the nodes in the attacked network quickly become very different from the coordinates of the same nodes in the original network. We argue that our attack generalizes since any network coordinate system will have to accept changes in coordinates due to dynamic network conditions. The effectiveness of the attack can also be significantly increased when conducted in conjunction with a *Sybil* attack [14]. We show in this thesis that the frog-boiling attack is effective at disrupting the three aforementioned "secure" schemes [24, 25, 26].

We implemented, and empirically evaluated, three variants of the frog-boiling attack to demonstrate its effectiveness against the current defenses. In the *Basic-Targeted* attack, a single targeted node is pushed to an arbitrary coordinate in the network; in the *Network-Partition* attack the network is divided into two subnetworks in coordinate space, effectively partitioning the whole network into two arbitrary smaller clusters; and in the *Closest-Node* attack the adversary becomes the closest neighbor to a targeted node. Partitioning a network is bad because two nodes which are close to each other in terms of network latency (for example in the same ISP) will appear to be far away from each other. If, for example, an adversary becomes the closest neighbor to every peer in a file-sharing system, the adversary will receive most the requests in that system. All three attacks rely on a simple concept: lying can be harmful but telling consistent, believable lies is even more harmful.

We evaluated our frog-boiling attack on the Mahalanobis distance outlier detection, Kalman filter anomaly detection, and Veracity. Our evaluation on a PlanetLab deployment of Vivaldi shows that even the basic frog-boiling attack is *more disruptive* against the security mechanism proposed in [24] than the random attacks on the mechanism. In particular, with only 5% of attackers in the network, frog-boiling causes a median relative error of 0.28 after 720 update intervals (2 hours) and 0.57 after 5040 update intevals (14 hours). This translates to each malicious node being contacted only twice and 10 times respectively. This shows that our attack is actually relatively fast. Although our attack only works if the attacker nodes are contacted by the victim nodes, this is very likely to happen as in any peer-to-peer system, peers have to frequently contact each other. For example in a file-sharing application like Vuze [7], peers have to frequently

contact other peers in the network to maintain their routing tables and to find content. The same network with no attackers has a median relative error of 0.11, and under Zage and Nita-Rotaru's "random" attack the insecure coordinate scheme has a maximum median relative error of 0.22, even when the fraction of attackers is above 10%. Thus the outlier detection mechanism is completely ineffective against frog-boiling. We note that while the step size of the attack is small – nodes are pushed "little by little" – the result of the attack is neither slow nor small, resulting in similar errors just as quickly as previously known attacks but causing greater damage over time.

While similar attacks on outlier detection mechanisms appear in the literature (including [31, 32, 33]), to our knowledge we are the first to demonstrate the effectiveness of frog-boiling in the context of network coordinate systems. Furthermore, we demonstrate that the attacks are *more disruptive* than previous work and are completely unmitigated by the existing approaches to securing network coordinate systems. These results suggest that new approaches and/or stronger assumptions are needed to construct secure network coordinate systems.

Since the Mahalanobis outlier detection mechanism failed, the natural next step is to use another detection mechanism with trusted peers. Kaafar *et al.* described a Kalman filter-based anomaly detection secure network coordinate system [25]. They also required that a fixed percentage of the network be trusted. These trusted nodes only contact each other and cannot be influenced by non-trusted nodes. We show that our frog-boiling attack is also successful again this scheme. Our network-partition variant of the attack effectively divided the whole network into two independent subnetworks. Moreover, our attack barely increases the median relative error of the network and remains mostly undetected throughout the experiment. We emphasize that our attack is still successful even with the presence of trusted entities – having trusted components in a network coordinate system does not mean that it will be harder to disrupt. Finally, we show that our attack is effective against a distributed reputation network coordinate system named Veracity [26]. Veracity is different from the two other secure schemes in that it does not use any statistical means to accept or reject coordinate updates from peers. However, we show that Veracity is vulnerable to our attack as the attacker can still lie in small steps without being detected – Veracity allows each verifier to accept coordinates from the publisher as long as the error calculated by the majority of the

witnesses is below a certain threshold. Thus the frog-boiling attacker can lie in small steps and still stay below the threshold. From our experimental results, our attack can still partition the network into two separate clusters. Veracity also includes a second verification step in an attempt to prevent nodes from delaying their replies – inflating their network latency. Our attack does not need to bypass this step and is still successful. Thus, the second verification step of Veracity is useless.

The reason why our attack works is that none of these schemes seems to have an explicit security goal; they cannot distinguish whether a variation in coordinates or network latencies is due to noise or malicious peers.

A complete description of our proposed attack is given in Chapter 3.

1.2 Attacking the Vuze BitTorrent Network

We showed how an application using a network coordinate system to improve efficiency, can be attacked. Instead of attacking the actual application, the network coordinate system is targeted such that the eventual outcome of the "indirect" attack is similar to directly attacking the application. However, the cost of targeting the network coordinate system is much cheaper than targeting the application. More specifically, the Vuze DHT routing uses network coordinates to contact closer next-hop peers in an attempt to locate the content faster. Vuze allows five parallel but not independent queries for each search. If a malicious peer is contacted on the first hop, it can advertise that it knows of other (malicious) peers which hold the content being searched for. Thus, if a malicious peer is contacted on the first hop, this search is considered to be hijacked as the malicious peer can return other malicious peers close to the target, and the attacker can eventually return bogus results. Although attacks [14, 34, 35, 36, 37] against distributed hash tables (DHT) [38, 39, 30] to try to hijack searches have been proposed before, this thesis shows that the network coordinate system can be exploited so that searches are hijacked more efficiently. Only 32 nodes are required for our attack to be successful, regardless of the total number of nodes in the Vuze network. Although [37] is effective with a fixed number of malicious peers, it requires 100MB/s of download bandwidth whereas our attack requires only 700KB/s of download bandwidth, more than two orders of magnitude cheaper.

The main point of this attack is that although implementing and using a tool to improve performance can be useful, that tool can also be exploited to abuse the application. Although it has been shown that existing network coordinate systems are insecure [23, 22, 24, 40], all the attacks were on the actual network coordinate systems in an attempt to disrupt the network latency estimations.

Our attack on the network coordinate system is to appear closer to all the peers in the network than anybody else. Since a Vuze node would pick closer peers to send its search queries, if an adversary is very close to every peer, then it will most likely receive a majority of search queries. The goal of the attacker is to be among the first ones contacted during a search query. Once the attacker becomes the first hop, this search is considered to be hijacked. The attacker's plan is thus to 1) lie about its network coordinates such that it appears very close to every peer on the network, 2) remain online such that it becomes the first hop on every search query, and 3) return other malicious peers whenever it is queried.

An attacker trying to directly attack the DHT routing of Vuze can capture at most 2% of all first-hop queries, that is, an attacker can hijack 2% of all searches. However, an attacker performing our proposed attack can capture almost 20% of all first-hop queries, that is, an attacker can hijack 20% of all searches.

A complete description of how to exploit the insecure network coordinate system is given in Chapter 4.

1.3 Treeple

First, we introduce a strong definition of security for latency estimation schemes that is robust to new attack types. Informally, our condition states that an adversary should be unable to influence the estimated distance between two honest nodes, and additionally, should only be able to increase the distance between pairs of nodes involving at least one adversarial node. We then consider whether previous secure network coordinate systems meet our definition. We showed in Chapter 3 that several "secure" schemes are vulnerable to our frog-boiling attack. We additionally show that even schemes that rely on trusted "landmark nodes" [2] can fail by allowing adversarial nodes to reduce their coordinate distance to targeted victim nodes in the network. The common

fault underlying both of these vulnerabilities is that current network coordinate systems ignore the underlying network topology, and thus cannot distinguish between anomalous round-trip times due to adversarial manipulation and anomalies due to the topology and changing conditions in the network.

After defining security, we describe Treeple, our scheme for secure network latency estimation. Node positions in Treeple are not abstract coordinates but instead represent their position in the network graph in a way that allows efficient computation of the estimated latency between a pair of nodes. Assuming a small collection of trusted "vantage points", we prove that Treeple meets our security definition even in the presence of an arbitrary number of adversarial nodes.

To illustrate the idea behind Treeple, suppose we have a system with trusted vantage point A and peers X, Y, and Z. Using techniques similar to traceroute, A can discover the network paths from itself to each of the peers, constructing a tree with routers at the internal nodes and link latencies along the edges. A could then use this tree to compute an upper bound on the latency between X and Y as follows: first identify C, the least common ancestor of X and Y – $lca(X, Y)$ – in the tree. Then compute the sum of the latency between C and X and the latency between C and Y. Since there exists a path of this latency between X and Y, it represents an upper bound on the latency of the actual network path between the nodes. [1] For example, suppose that the tree is built as shown in Figure 1.1, and A is estimating the distance between nodes Y and Z. In this case, $lca(Y, Z) = B$ and the estimated distance is $35 + 15 + 20 = 70$ ms.

Notice that this approach does not suffer from the same issues as network coordinate systems. A malicious node can only affect the distance (real or estimated) between itself and another node, but not between two honest nodes. For example, if node Y from Figure 1.1 was malicious, it could affect the distance estimation between Y and Z. However, node Y cannot affect the distance estimation between nodes X and Z because it will not be on the same network path. Furthermore, using basic techniques

[1] We note that due to the Internet's policy-based routing: (a) the RTTs measured to the intermediate nodes of the traceroutes will not typically reflect the link latencies exactly due to asymmetric return paths, and (b) "triangle inequality violations" [41, 42] can occur, in which case the path X-C-Y is shorter than the actual network path. However, in practice these measurements still provide a good approximation

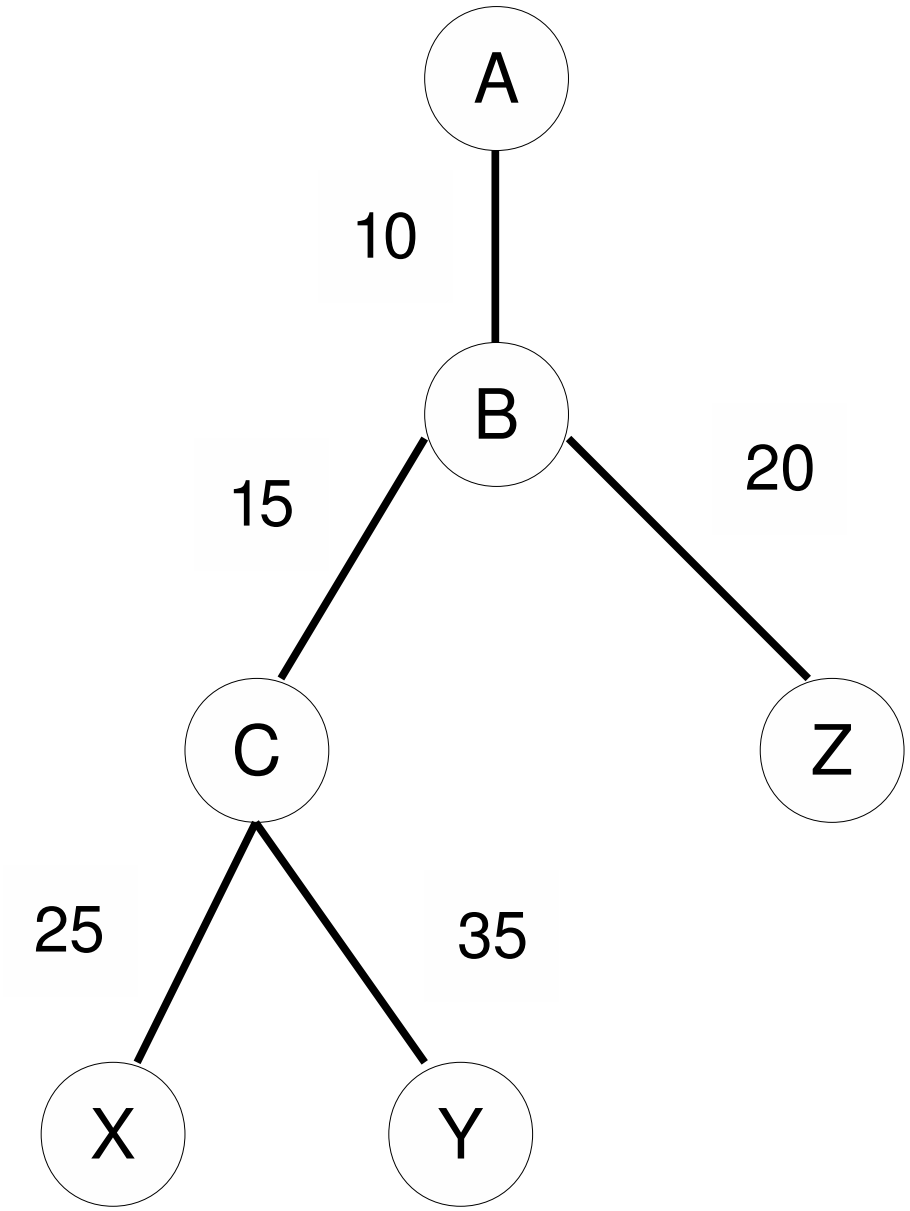

Figure 1.1: A tree built from one source to three destinations with link latency in ms.

like unpredictable nonces when measuring the RTT from A to Y can prevent Y from decreasing the distance estimation to other nodes. We describe the full scheme based on this idea in Chapter 5.

We evaluate Treeple on a large, real-world dataset. For this dataset, Treeple has a median relative error of 0.26; This means that on average, half of the estimated latencies are within 26% of the true latency. For comparison, we simulated Vivaldi [1], a popular but insecure network coordinate system, on the same data set and the resulting coordinate scheme has a median relative error of 0.25. We note that the choice of Vivaldi is not arbitrary: all of the recently proposed "secure" or "stable" network coordinate schemes [24, 25, 26, 27, 28, 20] work by adding additional measures to Vivaldi coordinate calculations that attempt to discard anomalous inputs, and thus would have the same accuracy in the absence of an attacker. This shows that Treeple has accuracy comparable to network coordinate systems while providing provable security.

We additionally demonstrate that Treeple positions are highly stable: positions calculated on the first day of our dataset provide nearly the same accuracy nearly three weeks later. This has several important implications. First, because peers do not need to recalculate their positions the bandwidth overhead is significantly reduced compared to network coordinate schemes. Second, for the same reason, the "centralized" vantage points do not present a significant scaling challenge for Treeple. Finally, trusted vantage points do not present a central point of failure: the system can continue to function with high accuracy even if all vantage points are unavailable for an extended period.

1.4 KoNKS

We then describe a completely decentralized network coordinate system, KoNKS, which is secure under our stated security model. KoNKS – consensus-style network coordinate system – modifies the objective function that each peer follows to update its coordinates. In current network coordinate systems, a peer's goal is to minimize the sum of the prediction errors for all of its neighbors. In contrast, using KoNKS, a peer's goal is to minimize the number of neighbors whose individual relative error is unacceptable – KoNKS puts an upper bound on each neighbor's relative error. The relative error determines how accurate the coordinate system is, thus when there are no attackers,

minimizing the sum of errors should lead to more accurate distance predictions. However, minimizing the sum of prediction errors allows each neighbor to have a significant influence on the position of its peers. This is one of the reasons why the frog-boiling attack works. For example, a malicious neighbor could craft a lie so that its coordinate distance to the peer is much smaller than the measured network distance. In response, the peer would make a significant change to its coordinate because that update seemed to give the minimum total prediction error, even though it adds significant prediction error to every other neighbor.

This example cannot happen in KoNKS because every neighbor of a peer has the same amount of influence on that peer. In a way, KoNKS peers achieve consensus among their neighbors: each neighbor "votes" for a region in which the peer should reside, and the network position with the most "votes" from the neighbors is the one that KoNKS chooses. A malicious neighbor can still choose its reported coordinates and add delay to its RTT, but the push that lie has on the peer is limited, as the latter will have to satisfy its other neighbors as well. At every update, the peer takes into consideration each of its neighbors' relative error. We argue that KoNKS is secure because 1) a malicious node's influence on the coordinate distance between two honest nodes is limited, and 2) a malicious node cannot appear closer than it actually is because its relative error will be higher than the imposed threshold.

We show that KoNKS is as accurate as Vivaldi [1], one of the most popular decentralized network coordinate system (Vivaldi is implemented in Vuze [7] and is the basis for all the "secure" network coordinate systems [24, 27, 26, 25]), and is secure against all the current attacks, including the network-partition frog-boiling attack. More specifically, KoNKS puts an upper bound on the amount of influence an adversary can have on the honest nodes. For example, 10% of attackers can partition a network using KoNKS only so much before their lies do not have any effect anymore because they are outside of the threshold, or the other honest neighbors' influence equals the malicious neighbors' influence. KoNKS with no attack can achieve a median relative error as low as 12%, which is comparable to Vivaldi's median relative error of 10%. Under the "random" attack, even with 30% of malicious nodes, KoNKS' median relative error is increased to only 20%; Veracity's median relative error is increased to 32%; Vivaldi's median relative error is increased beyond 150%. Moreover, KoNKS incurs a very low

overhead, similar to Vivaldi as coordinates can be piggybacked on top of application messages. The processing overhead of each node updating its coordinates is also very small.

A complete description and evaluation results of KoNKS can be found in Chapter 6.

Chapter 2

Background

2.1 Network Coordinate Systems

The first network coordinate systems developed were centralized – trusted infrastructure nodes compute coordinates for all other nodes. Centralized network coordinate systems include IDMaps [43], GNP [2] and NPS [3].

To improve the ease of deployment of network coordinate systems, decentralized network coordinate systems were introduced [1, 5, 6, 44]. A decentralized network coordinate system has no infrastructure nodes. Instead, normal nodes pick peers out of the set of all nodes, and compute their own coordinates with respect to those peers only. Finding potential peers is delegated to the underlying network. Decentralized network coordinate systems are attractive for P2P applications, since they can be deployed alongside the client software. Moreover, decentralized network coordinate systems are scalable as there are no centralized servers which could become overloaded.

2.1.1 Vivaldi

Vivaldi [1] is a decentralized network coordinate system. It provides fast convergence and resilience to changing network conditions such as churn in a P2P environment.

Vivaldi is based on a spring model. Its behavior is analogous to a physical model made of springs and balls, in which each ball represents a network node and the spring connecting any two balls is longer when the latency between those nodes is larger. Over time, such a model reaches a stable equilibrium. A Vivaldi node begins by selecting

an arbitrary set of peers, and sets its initial coordinate to the origin. It then begins an iterative algorithm that pulls it closer to peers with lower latencies, and pushes it away from peers with higher latencies. After many iterations, the coordinate system reaches an equilibrium, and subsequent changes are due only to the changing latency between nodes. Each node will pick 64 other nodes in its reference set – 32 nodes are "close" and 32 nodes are "far". On each iteration, a Vivaldi node sends a probe packet (which could be piggybacked on top of application-level messages) to each of its peers. It receives a response to each probe packet containing the peer's current coordinate and self-reported error estimate (can also be piggybacked on top of application-level messages), and learns its latency to that peer from the RTT of the transaction. It then computes a new position that is closer to the peer if the estimated latency is too large, and farther from the peer if the estimated latency is too small.

The estimated latency between two nodes is calculated by applying a distance function to their coordinates. A modified Euclidean distance function that incorporates a height component on top of Euclidean space is empirically more accurate than a simple Euclidean distance function. The number of dimensions also affects the accuracy of the coordinate system. Self-reported error estimates enhance the accuracy of the coordinate system. Typically, nodes compute an error estimate based on their coordinate history. A node with rapidly fluctuating coordinates should report a large error, since its coordinates at any given moment are unlikely to reflect its latency to other nodes. This error detection mechanism is entirely voluntary.

Vivaldi's coordinate system is n-dimensional. It was shown in [1] that a 2-dimensional space with *height* works well for the topologies they considered. Moreover, Vivaldi boasts a low convergence time, a low reported error, and is shown to be accurate. Churn is an important issue in all P2P networks. Due to the public nature of a P2P network, hosts can join and leave the network at any time and for any duration of time. Vivaldi handles churn well due to its low convergence time. However, Vivaldi was not designed for an adversarial environment and it is simple for an attacker to disrupt the whole network.

2.1.2 Pyxida

Pyxida [45] implements a virtual coordinate network. It is being used in both academia and commercially – to track the coordinates of all the PlanetLab [46, 47] nodes; and in the Vuze [7] BitTorrent client. It is designed to work on a P2P network and implements the Vivaldi algorithm. Pyxida uses a 4-dimensional with height coordinate space. Moreover, it is open-source, enabling easier modification to implement the countermeasures and attacks. We implemented the Mahalanobis distance defense mechanism [24] on top of Pyxida in our experiments since it implements the Vivaldi algorithm, provides a stable network coordinate system, and has been used in a large-scale deployment [20]. A detailed description of Pyxida is given in [21].

Pyxida contains a "gossip" protocol. Every 10 seconds, it will randomly pick a node it knows and send it a "ping" request. When the node replies, Pyxida will add that node to its neighbors list (the list can contain a maximum of n neighbors). When a Pyxida node receives a ping request, it will just send a response – it does not add that node to its neighbors list. After receiving a response, which contains the other node's coordinates, error, and RTT, the Pyxida node will calculate the current force. All the nodes in its neighbors list are inspected (coordinates and error) and the current force calculated from these data. The system coordinates are then updated accordingly. Thus, Pyxida will only update its coordinates when it initiates a request and this is performed every 10 seconds.

2.2 Performance Metrics

We use several metrics to evaluate the performance of a network coordinate system:

- **error**: The median relative error is calculated as $\frac{|RTT_{estimated} - RTT_{actual}|}{RTT_{actual}}$, where RTT_{actual} is the actual RTT between two nodes and $RTT_{estimated}$ is the RTT obtained by taking the difference in the coordinates of the two nodes. The lower this number is, the more accurate the network coordinate system is (each node believes it has the right coordinate). We note that errors are inherent in network coordinate systems because an embedding from a matrix of RTTs (high dimension) to a Euclidean coordinate (low dimension) will contain errors.

- **RRL**,Lua *et al.* [48] Relative Rank Loss: Pyxida calculates the rrl as the pairwise orderings of each neighbor, that is, for every pair of neighbors A and B, if $RTT_A \geq RTT_B$ and the coordinate distance to A is less than the coordinate distance to B, then this counts as a "wrong ordering", that is, if there is a sign mismatch. The rrl is thus the number of "wrong orderings" over the total number of possible orderings. The lower the *rrl* is, the better as there are more "correct orderings". Notice that a ranking that chose randomly between each pair of nodes would have an expected rrl of 0.5, so in some sense this represents the "worst possible" value of rrl.

- **RALP**: The relative application latency penalty [49] – this calculates the percentage of "lost latency" when using a network coordinate system. It is calculated as follows. For every pair of neighbors A and B, if $RTT_A \geq RTT_B$ and the coordinate distance to A is less than the coordinate distance to B, then the *ralp-loss* is $|RTT_A - RTT_B|$. The *ralp* is then the sum of the *ralp-loss* for each pair divided by the sum of the "correct" RTTs for each pair. If $RTT_A \geq RTT_B$, the "correct" RTT is RTT_A. For the other case, the "correct" RTT is RTT_B. A ralp of 0.5 means that it takes 50% longer in terms of latency to get a file, using the network coordinates than using the real RTT.

- **false positive rate**: The false positive rate denotes the number of updates that are rejected, although they should have been accepted. This is used to determine the optimal thresholds for the Mahalanobis distance.

- **intercluster/intracluster ratio**: This is a measure of how far apart two sets of nodes are from each other. This is used to measure the effectiveness of one variant of our frog-boiling attack, which attempts to partition the whole network into two independent subnetworks. An intercluster/intracluster ratio of 2 means that a node from one cluster is twice further apart on average from a node in the other cluster than to a node from the same cluster. A ratio of 1 implies that the network is not partitioned. A higher ratio indicates that our attack is effective. The ratio is calculated as the average coordinate distance of all the nodes of one set to the center of the other set, divided by the average coordinate distance of all the nodes of one set to the center of the same set.

2.3 Existing Attacks

Several attacks on network coordinate systems have been described in the literature [22, 23, 25, 24]. These include the *Disorder attack*, *Repulsion attack*, *Colluding Isolation attack*, *Inflation/Deflation attack*, and the *Oscillation attack*. The Repulsion and Colluding Isolation attacker sends the same coordinates each time in an attempt to move the victim nodes to some location. The Disorder attacker always reports randomly chosen coordinates and a low error. Basically, an attacker can lie about its self-reported coordinates and/or delay its reply so that the measured RTT will be higher than the real RTT. The reader is referred to those papers for a more detailed description of the attacks.

2.4 Countermeasures

Several mechanisms [25, 24, 26, 27, 28] have been proposed to secure network coordinate systems. We give an overview of each of them here. We refer the reader to the respective papers for a more detailed description.

Mahalanobis Distance: The first defense mechanism that we will look at uses statistical means to reject inputs that do not conform to past accepted inputs. Zage and Nita-Rotaru [24] use the Mahalanobis distance, an outlier detection system, to flag "bad" coordinate updates. Based on past good updates, the system determines if a new coordinate update is acceptable or not.

Kalman Filter: The system by [25] also uses a statistical method to determine if a coordinate is acceptable. They implement the Kalman filter [50], an anomaly detection system. The main difference from the previous scheme is that this system requires that some peers in the network are trusted. These trusted nodes only communicate with each other to determine the parameters for the Kalman filter, which other non-trusted nodes use. We show in this thesis that our attack is effective against this scheme.

Veracity: Veracity [26] is a distributed reputation system. A peer's self-reported coordinate can be verified by other peers, and a peer is prevented from delaying its reply to an update in an attempt to inflate its network latency. We show in this thesis that Veracity is also vulnerable to our attacks.

Other schemes: RVivaldi [27] is a reputation system, which assigns weighted trust to

peers and uses the trust metric to accept coordinate updates from these peers. [28] proposed using a two-step verification method to mitigate malicious nodes in the network. The first step is to verify the self-reported coordinates of peers. This is performed by having witnesses attest to each peer's coordinate update. A verifier can then audit that peer to make sure that the peer computed its coordinates correctly. The second step is to prevent nodes from delaying their reply so as to increase the network latency. This step involves using their triangle inequality violation (TIV) detection mechanism [51], as the authors claim that a peer which delays its reply is more likely to be part of a TIV than a honest peer which is not delaying its reply. Since their TIV detection scheme can filter out TIVs, then since they claim a malicious peer is part of a TIV, the scheme should be able to filter out the adversarial node.

2.5 Vuze

Vuze is a popular BitTorrent client with over 2 million concurrent users, which supports additional functionality, such as *distributed tracking* of torrent files. This is achieved through the use of a *distributed database*, which in turn is an implementation of the Kademlia distributed hash table (DHT) [30]. In the original BitTorrent specifications, in order to start a download, the user needs to obtain a torrent file that contains all the necessary information for the download process. This includes: *i)* IP address of one or more trackers, to obtain information about the swarm and other peers *ii)* information about files, such as number and size of pieces *iii)* info-hash, a hash of the file contents to verify that the download was successful. Distributed tracking, on the other hand, allows Vuze clients to start a download without a torrent file. This is a usability feature as well as a backup option to locate extra peers in case the tracker is unreachable. To participate in distributed tracking, Vuze clients register torrents in the DHT by storing their IP and port under a key. The key is a 160-bit identifier obtained by performing SHA-1 hash of the torrent info-hash.

When another user wishes to start a download, but does not have access to the torrent file, he can use a *m*agnet link that contains the key described above. A magnet link is a specially formatted URL that contains a unique identifier of a torrent file, computed using the torrent's info-hash. It is usually provided on the websites and

forums as an alternative to downloading a torrent file, to provide a simple, one-click download process. A simple DHT lookup for the key allows the client to obtain a list of IP:port pairs of other users that are in possession of the torrent file and download it from them. Further peer location is done through standard tracker communication and peer exchange.

2.5.1 Routing Table

Vuze uses a modified version of the Kademlia DHT, which differs from the original in several ways, some of which include Vivaldi coordinate extension, caching along paths and encrypted data transfer. Each Vuze client has an 160-bit unique ID, which is simply a SHA-1 hash of the IP:port pair. The Vuze ID is the main identifier for any node and is used to store nodes in the routing table. The routing table is a tree structure that consists of buckets, each bucket corresponds to a number of Vuze ID prefix matches and contains up to 20 nodes (see Figure 2.1). "Closeness" of any two nodes in the network or the routing table is determined by *XOR distance*. XOR distance is calculated by performing a bitwise exclusive OR operation on two given Vuze IDs, and the result is treated as an integer.

A Vuze peer continuously discovers new nodes and updates its state by sending various types of messages to known peers. These are the message types we are interested in: *ping*, *findNode*, and *findValue*. The *findNode* and *findValue* message types are used by Vuze to maintain its routing table and for finding content (Vuze ID key) in the DHT. It is important to note that *ping* message is not an ICMP ping, but a UDP packet containing information such as Vuze protocol version, network version, time, etc. Upon receiving a reply to any of these messages, the node is added to the appropriate bucket in the routing table and marked alive. If the bucket is full, the node is added to the replacement list. The replacement list is another extension to the routing table that allows Vuze to quickly replace dead nodes in the bucket. Each bucket has a separate replacement list containing up to 5 alive nodes. Whenever a node in the bucket fails to respond to a message several consecutive times, it is marked as dead and another node from the replacement list, chosen randomly, is put in its place. Nodes discovered as a result of replies to *findNode* and *findValue* messages are marked as known, but not alive, and a *ping* message will be sent to them in order to establish whether they are

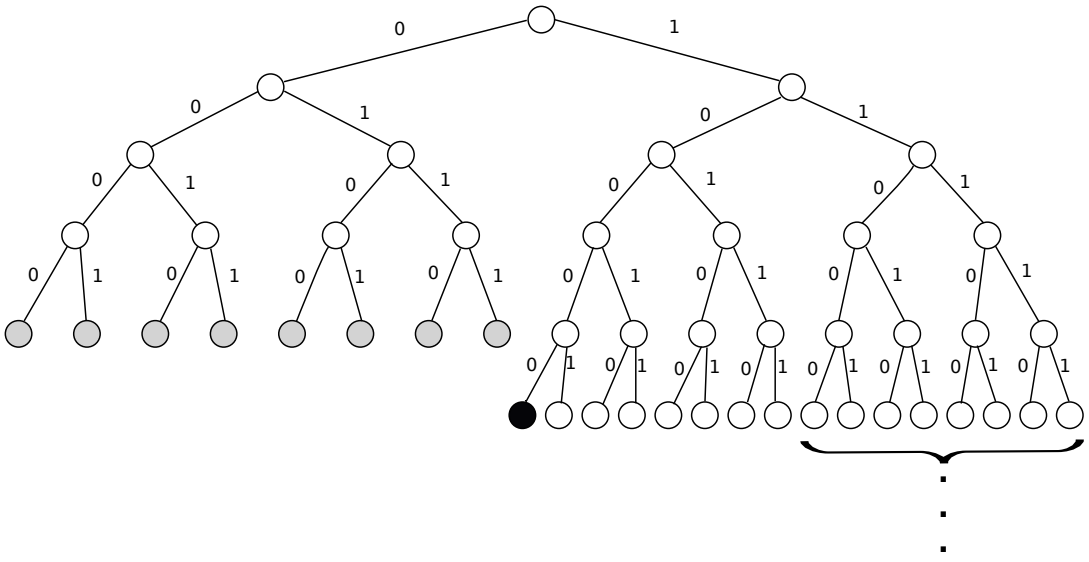

Figure 2.1: Vuze routing table.
Numbers indicate the ID bit match. Gray nodes indicate buckets important to our attack. Black node indicates the bucket where nodes with only 1-bit ID match would reside. Right side of routing table keeps expanding if nodes with more prefix bit matches are found.

alive.

2.5.2 Routing

Lookups in Vuze are parallel and iterative. A search starts by the searcher choosing nodes, closest to the target ID in terms of XOR distance, from its the routing table. A query is then sent to each selected node. Upon receiving a reply, newly discovered nodes are added to the list of nodes to query. The nodes in said list are then sorted according to XOR distance and Vivaldi distance (see Figure 2.2). The query sending process is repeated until the target is found or no closer nodes can be discovered. Here are the exact steps of performing a DHT lookup for target key T in Vuze:

1. Choose 20 nodes closest to T in terms of XOR distance from the appropriate bucket in the routing table and add them to the processing list.
2. Sort first 20 nodes in the processing list by XOR distance to key T.
3. Sort first 10 nodes in the processing list by Vivaldi distance.
4. Send queries to the top 5 nodes.
5. When reply is received, add discovered nodes (usually 10 per reply) to processing list, and perform sorting operations as described in steps 2 and 3.
6. If there are fewer than 5 outstanding queries, send another query.
7. Go to step 5, unless target is reached, no new nodes were discovered, or a timeout occurs.

2.5.3 Vivaldi usage

Vuze maintains Vivaldi [1] network coordinates for each node it knows to be alive in the routing table. Vivaldi coordinates are three-dimensional and include position on a Cartesian plane plus height. The height is used to account for the "last mile distance", a common theory that last hop significantly contributes to the network latency. The end distance between nodes is calculated to be a sum of their heights plus the Euclidean distance of the network coordinates. For more details on the Vivaldi model and latency

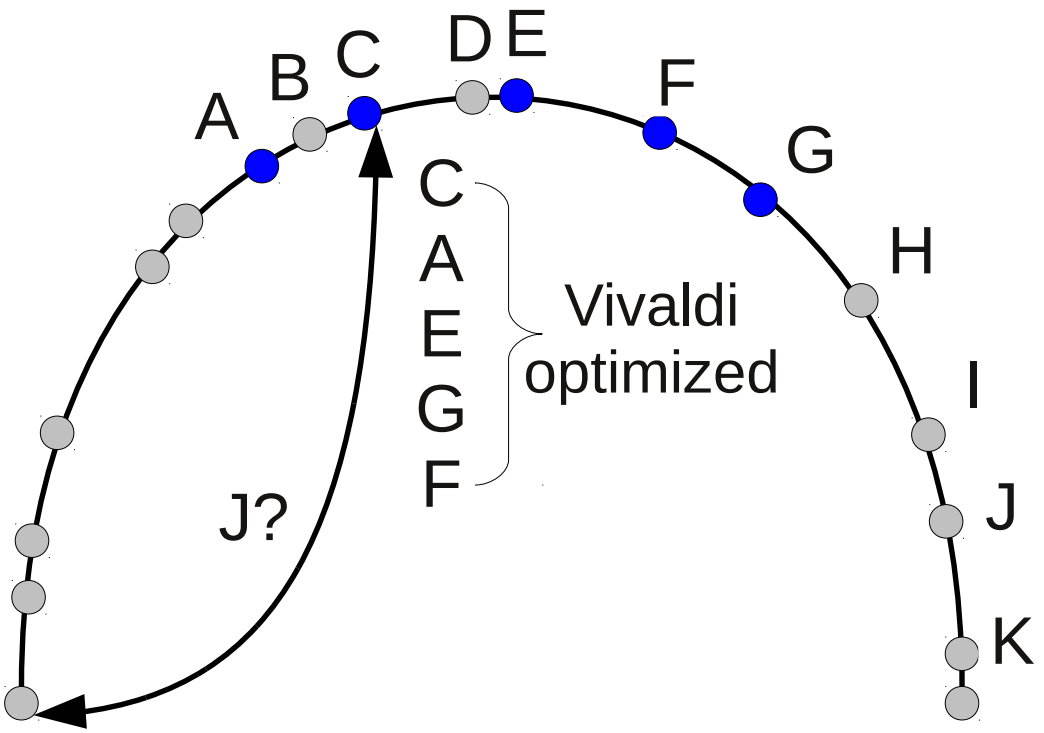

Figure 2.2: Vuze lookup process.
First 10 contacts are optimized using Vivaldi distance.

estimation, see the work by Dabek *et. al.* [1]. In order to keep this information up to date, Vuze occasionally sends *ping* messages to nodes in the routing table to confirm that they are alive and update the network coordinates. Network coordinates are piggy-backed to the *pingReply* messages as well as replies to *findNode* and *findValue* messages. This allows for transmission of up-to-date information about coordinates with minimum additional cost.

Chapter 3

The Frog-Boiling Attack

We first describe our proposed attack, called the *frog-boiling attack*, on the secure schemes [24, 25, 27, 26, 28] outlined in Chapter 2. We demonstrate the inherent challenge in designing a secure network coordinate system based on rejecting "bad" inputs that do not conform. Our frog-boiling adversary disrupts the network while consistently operating within the threshold of rejection. This is analogous to the popular account that a frog put in hot water will quickly jump out but a frog placed in cold water that is gradually brought to a boil will not notice the change and boil to death. The adversary sends "small-step" fake updates (inflated RTT or coordinates) to nodes in the network. The "step" is small enough that it does not trigger either the anomaly detection or the verification methods used by Veracity, but the nodes targeted are still disrupted – their network coordinate is not accurate. Thus, the coordinates of the nodes in the attacked network quickly become very different from the coordinates of the same nodes in the original network. We argue that our attack generalizes since any network coordinate system will have to accept changes in coordinates due to dynamic network conditions. The effectiveness of the attack can also be significantly increased when conducted in conjunction with a *Sybil* attack [14]. We show in this chapter that the frog-boiling attack is effective at disrupting the three aforementioned "secure" schemes [24, 25, 26].

We implement, and empirically evaluate, three variants of the frog-boiling attack to demonstrate its effectiveness against outlier-detection based defenses. In the *Basic-Targeted* attack, a single targeted node is pushed to an arbitrary coordinate in the network; in the *Network-Partition* attack the network is divided into two subnetworks

in coordinate space, effectively partitioning the whole network into two arbitrary smaller clusters; and in the *Closest-Node* attack the adversary becomes the closest neighbor to a targeted node. Partitioning a network is bad because two nodes which are close to each other in terms of network latency (for example in the same ISP) will appear to be far away from each other. If, for example, an adversary becomes the closest neighbor to every peer in a file-sharing system, the adversary will receive most the requests in that system. All three attacks rely on a simple concept: lying can be harmful but telling consistent, believable lies is even more harmful.

The remainder of this chapter is organized as follows. A detailed description of the attacks is given in Section 3.1. The evaluations of our experiments on a wide area network for the Mahalanobis distance based outlier detection [24] are shown in Section 3.2. Section 3.3 shows our attack results on the Kalman filter [25] and Section 3.4 shows our frog-boiling attack evaluations on Veracity [26]. A summary of the attack is provided in Section 3.5.

3.1 Our attacks

3.1.1 Frog-boiling Attack

Previous secure network coordinate schemes create a range of acceptable new coordinates. Updated coordinates are accepted if they fall inside this range and rejected otherwise. These mechanisms correctly identify spurious peers that return random coordinates as these updates will not fit inside the range. Since honest and correctly operating peers are unlikely to change their coordinates faster than average, their updates will fit inside the range of acceptable coordinates and be accepted by the scheme. The Mahalanobis distance and Kalman filter are statistical outlier detection schemes, so honest nodes will be considered normal while randomly behaving nodes will be flagged as outliers. Although Veracity does not use any statistical method, it accepts peers that are within a threshold and rejects peers with computed error greater than the threshold. Since honest peers behave as per the protocol, their computed error will be less than the threshold, while malicious peers' computed error will be greater than the threshold.

However, an intelligent adversary can send "random" data points that still fall within the acceptable range. Thus, the data points will be accepted although they are "wrong".

We call this approach the *frog-boiling attack*. If the adversary lies too much, its peers will not accept its updates. If it lies too little, the attack will not succeed in disrupting the network. The frog-boiling attack can be used to disrupt the whole network by continuously lying to all the nodes.

As a simple example, assume there are only two nodes A and B in the network and they have converged to stable coordinates. An attacker node C is introduced and obtains its coordinates from both A and B. However, each time C receives a request (say from A), it replies with $Coord_C = Coord_C + \delta$, where δ is a small offset. For example, if its coordinates in 2 dimensions are $(120, 100)$, the reported coordinate will be $(120.5, 100.5)$. Since the coordinate reported is not outside of the Mahalanobis thresholds, Kalman filter, or the Veracity error ratio cut-off, A will accept the coordinate and update its own coordinate accordingly. Then whenever B queries A, the response will be a coordinate that is slightly higher than what the "real" coordinate should have been. Thus, B's coordinate changes slightly as well. This process continues with the attacker continuously lying in small increments about its own coordinate. It is conceivable that this whole process might just shift the coordinates, but not affect the estimated distance between any two nodes. However, we show in Sections 3.2, 3.3, and 3.4 that this is not the case; our attack effectively renders the network coordinates ineffective.

3.1.2 Targeted Attack

Our targeted attack relies on the simple concept that consistent lies will not be caught by any of the secure schemes since the attacker's reported coordinates are similar over time. Thus neither the Mahalanobis spatial and temporal thresholds, nor the Kalman filter will be triggered. Moreover, the attack will pass both verification tests for Veracity, as the error would still be below the cut-off. The attacker can also select which nodes to lie to and which nodes not to respond to, since neither of the security mechanisms considered here nor any of the deployed network coordinate system implement any explicit sharing of information about other nodes.

The targeted attack works as follows. The attacker attempts to move some victim nodes (a small fraction of the whole network) to some arbitrary network coordinates. The targeted location in this case is far from the rest of the network. Although those

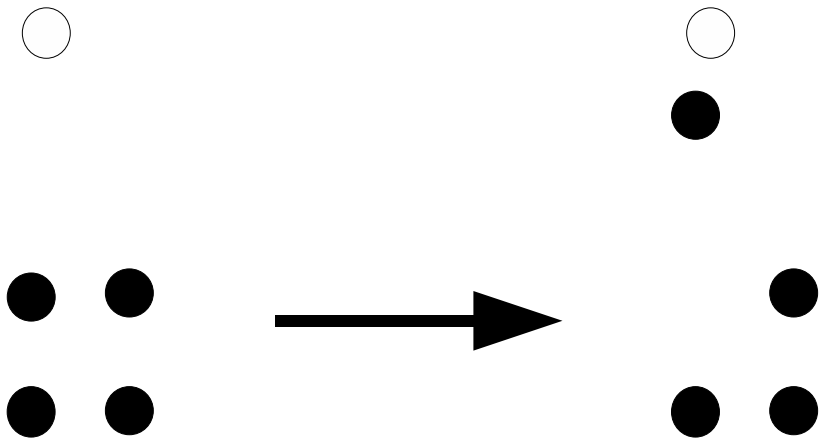

Figure 3.1: Basic-Targeted attack
Moving the targeted nodes to some coordinate.

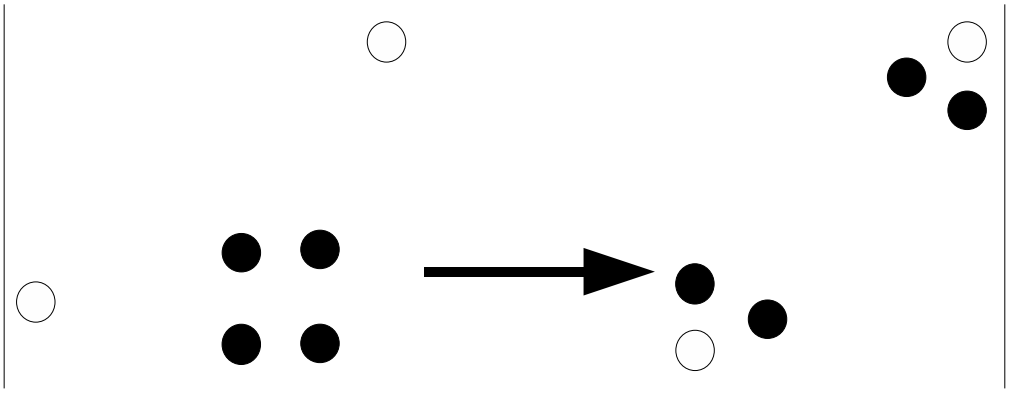

Figure 3.2: Network-Partition attack
Dividing the network into two separate smaller subnetworks.

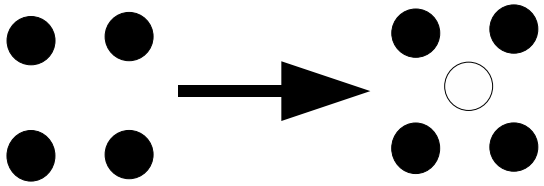

Figure 3.3: Closest-Node attack
Attacker becomes the closest node to the victims.

nodes can still communicate with the rest of the network, they will not be able to calculate a correct coordinate for themselves and will report a "false" coordinate and error to the rest of the network. The three secure mechanisms will flag those nodes as outliers, anomalous, or misbehaving, and will not accept their updates. This effectively isolates the victim nodes from the rest of the network.

Moving a victim node to an arbitrary location with a single update would typically require a force of sufficient magnitude to trigger an outlier filter. In order to avoid this, the victim node will be moved to a target location in small steps. The rest of the network will still accept updates from that victim node if the move is small. Thus, the rest of the network will also be pulled to that location by the victim node. However, since the victim nodes consist of a small portion of the network (less than 5%), the rest of the network will get pulled back together, further from the victim nodes at every update.

One way of performing this attack is for the adversary to consistently report its coordinates to the victim nodes so that the latter end up at location A. Note that the attacker will not be able to pull the victim nodes all the way to A, but the victims will be closer to A than the rest of the network. This is because, although the rest of the network might not accept updates from the victim nodes, the latter will still accept updates from the rest of the network. Thus, the victim nodes are pushed to A by the attacker but also pulled back to the rest of the network. The success of the attack is for the attacker nodes to exert a greater force on the victim nodes than the rest of the network.

In this thesis we evaluate three variants of this attack:

- The **Basic-Targeted** attack described above is shown in Figure 3.1. The shaded nodes are part of the original network while the white nodes denote the attackers.

- The **Network-Partition** attack is an extension of the previous attack, shown in Figure 3.2, where the whole network is partitioned into two subnetworks or clusters. The partition can be arbitrary as the frog-boiling attacker is performing a Basic-Targeted attack to a certain coordinate location on a subset of the network and performing the same Basic-Targeted attack to a different coordinate location on the rest of the network.

- The **Closest-Node** attacker tries to become the closest node (in terms of coordinate distance) to the victim nodes, as shown in Figure 3.3. Becoming the closest node might not be important by itself. However, if the network coordinate system is used with an application such as in Vuze, then the closest node could be used to initiate a file transfer. If the attacker becomes the closest node to a victim node, it will then be the first node that the victim contacts for a file. This can have various implications such as preventing any node in a file-sharing network from being able to download a file.

  This attack is performed in a similar way to the targeted attack. Instead of pulling the victim node to a certain coordinate space, the attacker pushes itself close to the victim node. One way of doing this is for the attacker (after learning the victim's coordinate) to report its network coordinates as being very close to that of the victim's.

3.1.3 Implementation Issue

In most network coordinate systems, a peer V only updates its coordinate after it initiates contact. A malicious peer A cannot contact other peers to force them to update their coordinates. Thus, the malicious peer A needs to be contacted by V before A can affect V's coordinates. However, peers in a network coordinate system need to frequently contact other peers, and the malicious peer will most likely be contacted at some point. There are numerous reasons why a peer needs to contact other nodes in the network: 1) coordinates are constantly changing due to dynamic network conditions, 2) applications using network coordinate systems require frequent exchange of information, for example, in a file-sharing system like Vuze, nodes have to contact each other to find the desired content.

3.2 Mahalanobis Distance

Zage and Nita-Rotaru [24] proposed a countermeasure that uses two statistical filters to ignore peers that report unusually large changes in coordinates or rapidly changing coordinates. The first filter is called the *spatial filter*, while the second is called the

temporal filter. Each node applies both filters to incoming data from its peers, and discards data that do not pass both filters.

The spatial filter uses three variables: the peer's reported error, change in the peer's coordinate since the last iteration, and the latency of the probe packet to that peer. An average of these three variables recorded from all nodes is stored from the last u iterations (we use $u = 64$ in our experiments). Intuitively, we expect all peers to have roughly the same reported error and location change at a given time. Peers that greatly exceed the averages are likely to be lying. The Mahalanobis outlier detection function used by the spatial filter determines if the new spatial vector falls inside an ellipsoid defined by previously-seen vectors. In the case of the spatial filter, we are defining a three-dimensional ellipsoid with axis lengths that are a multiple of the variances of the three variables (the "multiple" is a configuration parameter). Data points that fall inside this ellipsoid are accepted, while data points that fall outside it are discarded.

The temporal filter uses five variables: the remote error, local error, latency, change in the peer's coordinate since the last iteration, and local coordinate change in the last iteration. The ellipsoid for the Mahalanobis outlier detection function is based on these variables, collected over the entire lifetime of the node; since the data set is much larger, a constant-time and constant-space but slightly less accurate variant of the Mahalanobis function is used for this filter. Intuitively, as the network becomes more stable, we expect the peer's error, the peer's coordinate change, the local node's error, and the local node's coordinate change to decrease. Peers that report large errors and coordinate changes while the local node's values are small are likely to be lying.

Since the cost of a false positive is small, nodes can afford to set their thresholds very low. However, if the thresholds are too low, nodes will only accept data points that fit into a small range, leading to inaccurate coordinates. To our knowledge, the correct choice of thresholds to maximize security vs correctness has not been studied. When a peer's data fails either the spatial or temporal filter, there are two consequences. First, that peer's data is not used to update the node's current coordinate. Second, that peer's data is not used as history for the filters in the next iteration. However, there is no permanent blacklist of nodes which failed the filters.

3.2.1 Experimental Setup

To evaluate the impact of the attacks on a network coordinate system with the Mahalanobis distance implemented, we deployed a standalone Pyxida service (see Section 2.1.2) on PlanetLab [47]. Since the original Pyxida code implements the basic Vivaldi coordinate system, the Mahalanobis distance outlier detection mechanism proposed in [24] was added to the Pyxida code using a third-party library [52]. The number of PlanetLab nodes available to our service varied from 200 to 600.

We made some small modifications to Pyxida before deploying it. The neighbor list was modified to contain a maximum of 32 nodes (due to an estimated PlanetLab network size of 400). We used 50 nodes as the common "bootstrap" nodes, that is, all the Pyxida nodes contact those nodes when they first start. We wait until the network stabilizes before introducing any adversaries in the network. Stabilization occurs when the median relative error of the whole network remains constant and when the coordinates of all the nodes in the network do not change by more than a small amount ϵ at each update step.

We note that most of the experiments here were also performed using a simulated network to verify implementation correctness. The results of these simulations are consistent with experimental results and are thus omitted.

Mahalanobis Thresholds

We deployed a modified version of Pyxida, with the Mahalanobis distance implemented and no attackers. As with any other outlier detection, the thresholds used are important. Although [24] reported the spatial and temporal thresholds used as 1.5 (determined experimentally) and 4.0 (allow the four variables to vary by one standard deviation) respectively, those thresholds are not effective for Pyxida when deployed on PlanetLab, due to the differences in network topology.

Increasing the threshold will naturally lower the false positive rate as more updates are accepted by the Mahalanobis distance mechanism but this also increases the relative error as more "bad" updates are accepted. A "bad" update could occur due to a long RTT or the local node taking longer than expected to reply to a request. Thus a trade-off is needed to determine better spatial and temporal thresholds which result in a low

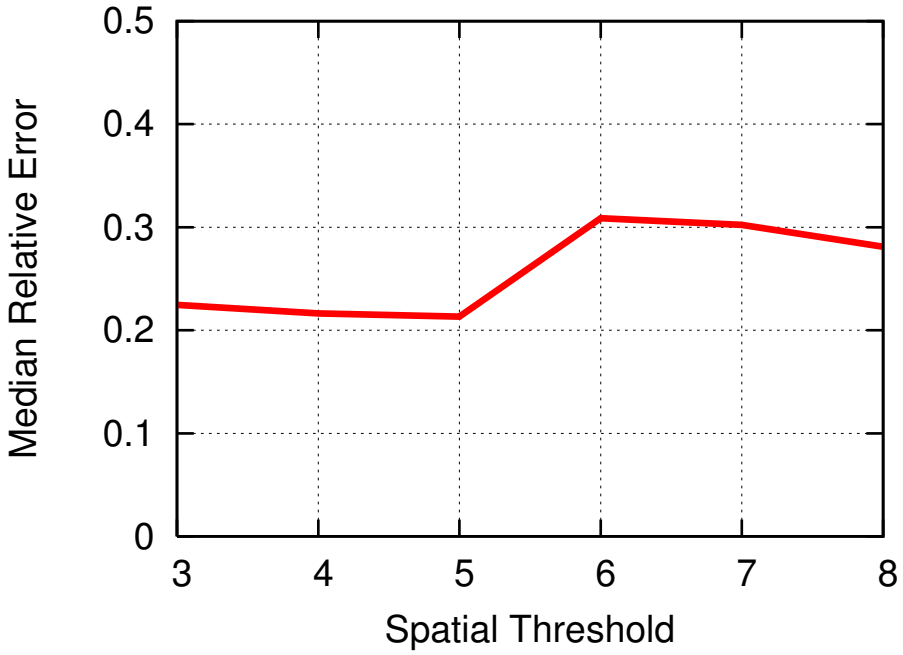

Figure 3.4: The average median relative error for varying spatial thresholds

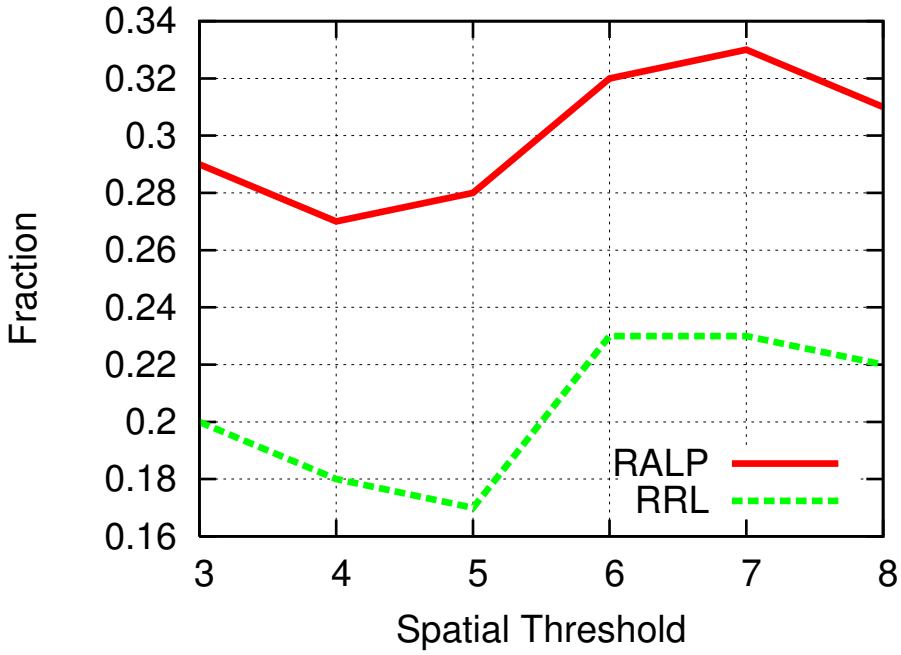

Figure 3.5: The average rrl and ralp for varying spatial thresholds

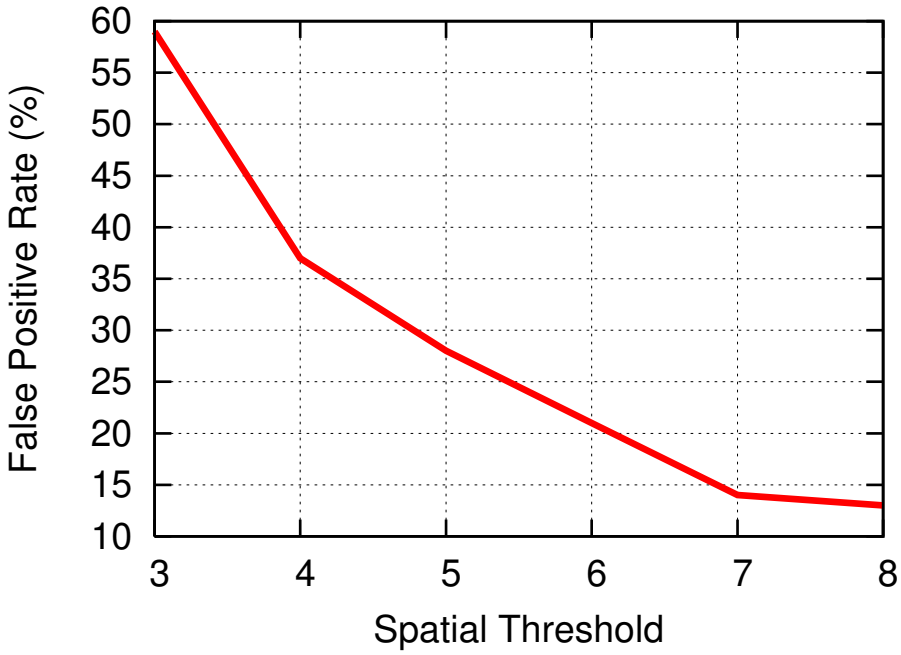

Figure 3.6: The average false positive rate for varying spatial thresholds

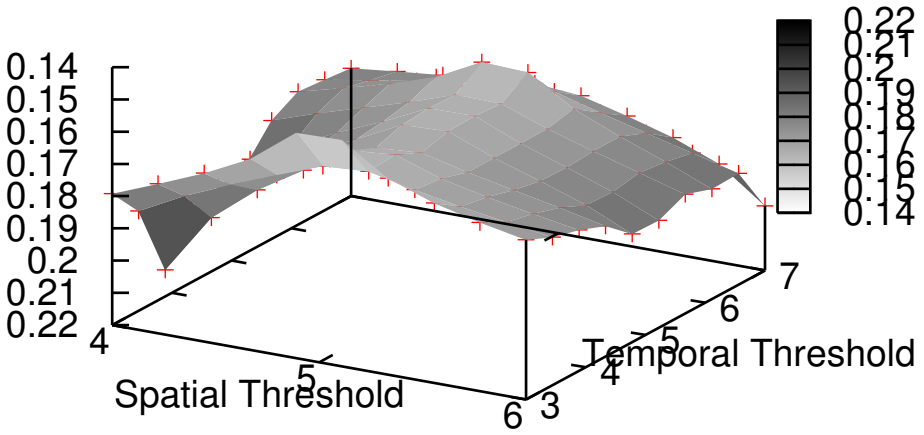

Figure 3.7: The average median relative error for varying spatial and temporal thresholds (note the reverse z-axis)

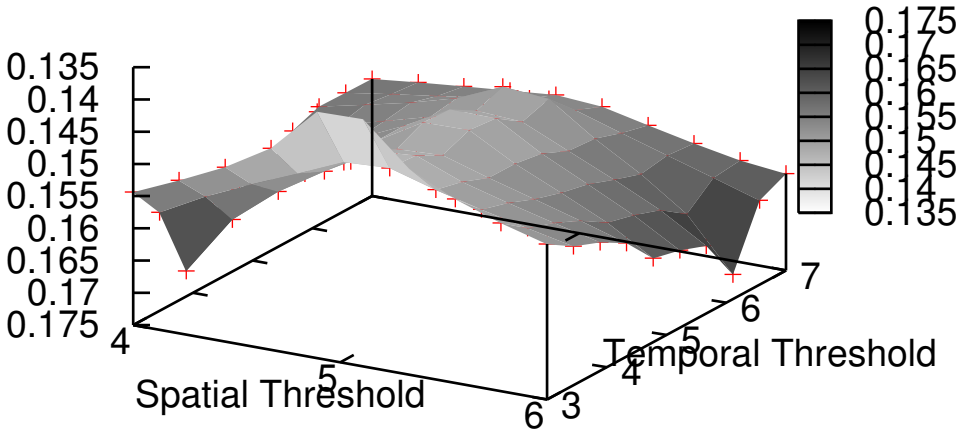

Figure 3.8: The average rrl for varying spatial and temporal thresholds (note the reverse z-axis)

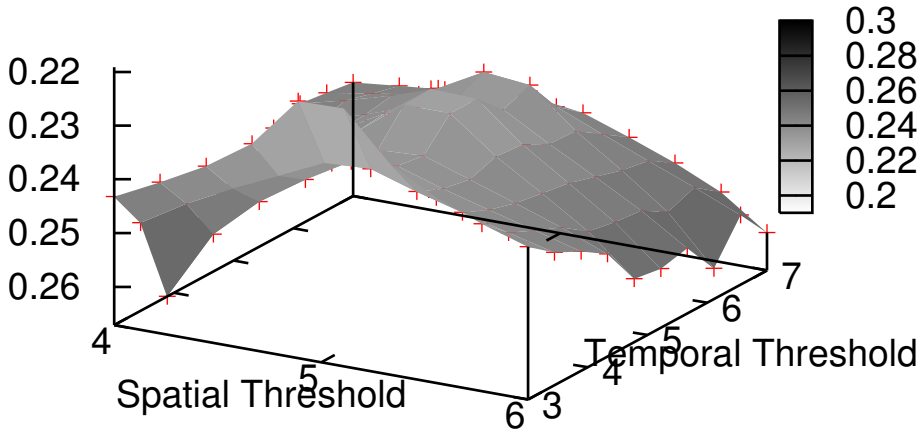

Figure 3.9: The average ralp for varying spatial and temporal thresholds (note the reverse z-axis)

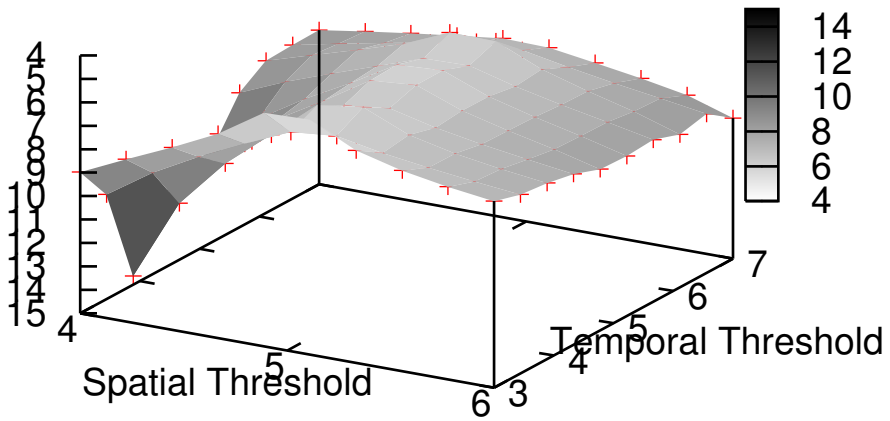

Figure 3.10: The average spatial false positive rate for varying spatial and temporal thresholds (note the reverse z-axis)

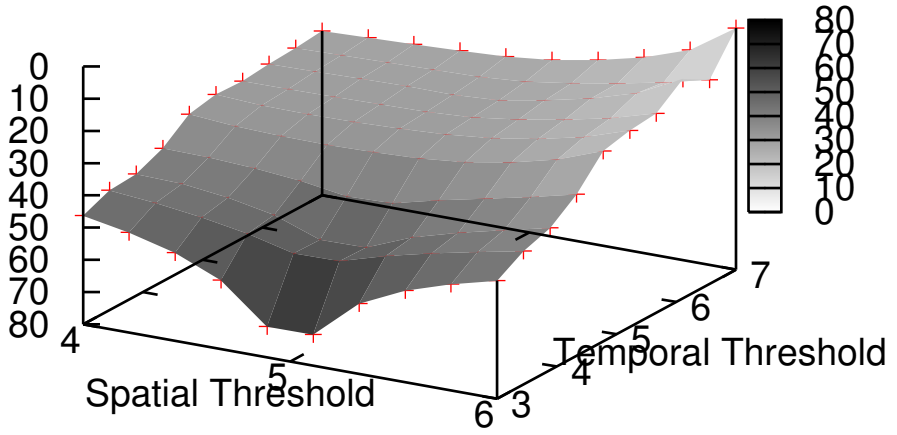

Figure 3.11: The average temporal false positive rate for varying spatial and temporal thresholds
(note the reverse z-axis)

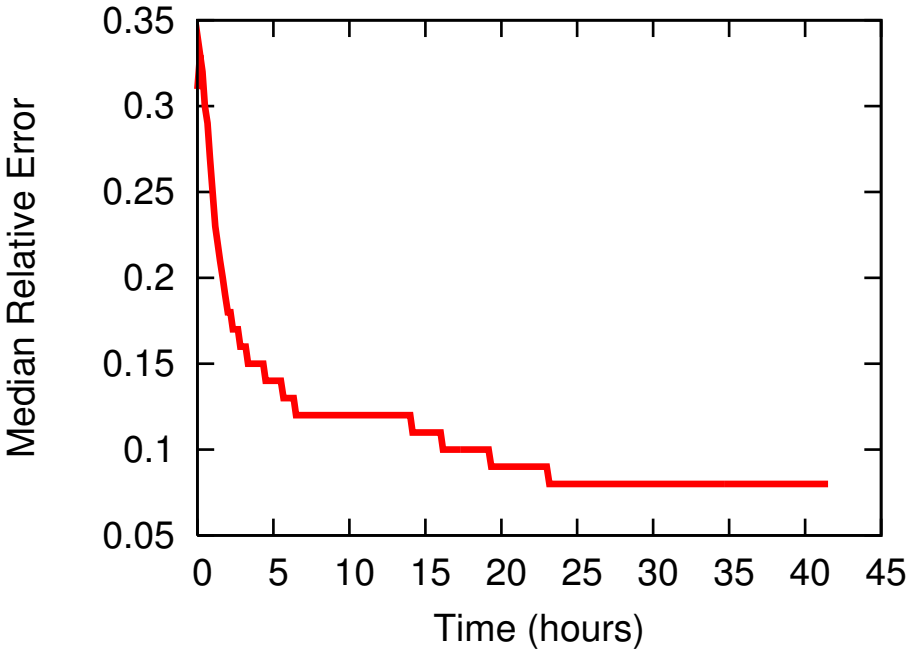

Figure 3.12: The average median relative error over time with no attackers with both a spatial and temporal threshold of 5.

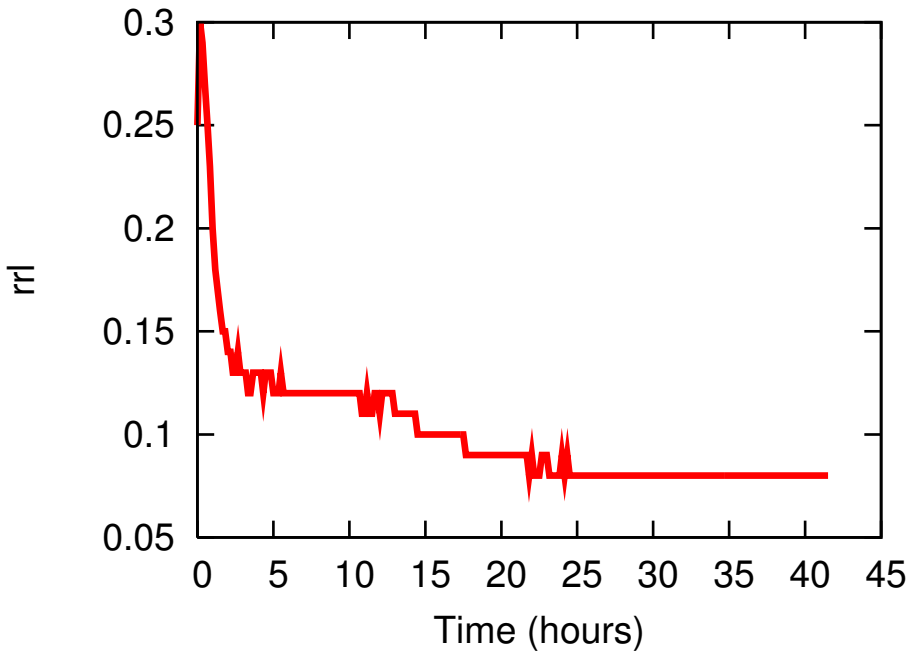

Figure 3.13: The average rrl over time with no attackers with both a spatial and temporal threshold of 5.

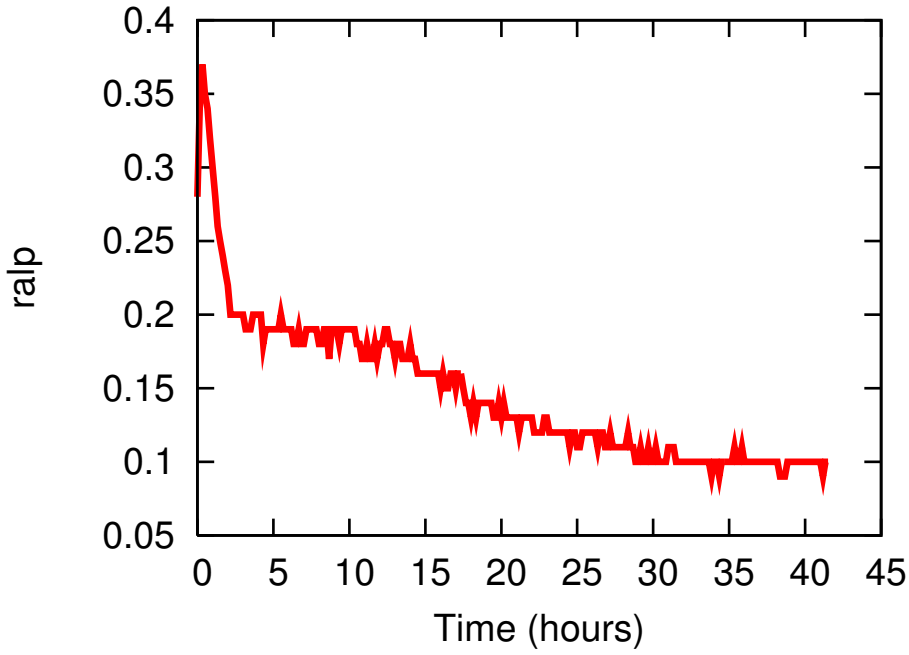

Figure 3.14: The average ralp over time with no attackers with both a spatial and temporal threshold of 5.

false positive rate, error, rrl, and ralp.

We first determine the best spatial threshold – comparing the relative errors, rrl, and ralp. We also attempted to minimize the false positive rate, that is, updates sent to a Pyxida node that are not accepted, because they are outside of the threshold. In a pristine Pyxida environment with no attackers, this should not happen. The main reasons are attributed to the variance in latency between Planetlab nodes, and the variance in the CPU usage of Planetlab nodes. We then determine the best spatial-temporal threshold pair.

Spatial Thresholds. We set the temporal threshold to be a high number so as not to affect the analysis of the spatial threshold, and varied the spatial thresholds from 3 to 8 to determine the spatial threshold value that minimized median relative error, rrl, ralp, and the false positive rate. Figures 3.4, 3.5 and 3.6 show the median relative error, ralp and rrl, and the false positive rate in these experiments. Based on those graphs, we experimentally that a spatial threshold of 5 produces the lowest combinations of relative error, rrl, ralp, and false positive rate.

Temporal Thresholds. Next, we vary both the spatial and temporal thresholds. Since we know that the best spatial threshold is 5, we varied the spatial threshold from 3 to 6. We varied the temporal threshold from 3 to 7. Figures 3.7, 3.8, 3.9, 3.10, and 3.11 show the median relative error, the rrl value, the ralp value, the spatial false positive rate, and the temporal false positive rate respectively. Our results suggest that the optimal temporal threshold is 5. Although a spatial threshold of 5 and a temporal threshold of 3 have a slightly lower relative error, rrl, and ralp than a spatial threshold of 5 and a temporal threshold of 5, the false positive rate (either spatial or temporal) is much higher. Similarly, a spatial threshold of 6 and a temporal threshold of 7 have a lower false positive rate but the relative error is much higher.

Long-running experiment

We use both a spatial and temporal threshold of 5 in the remainder of our experiments. We then started a long-running experiment of about 400 PlanetLab nodes for almost 2 days with no attackers. The median relative error, rrl, and ralp are shown in Figure 3.12, 3.13, and 3.14 respectively. It can be seen that the network starts to stabilize after 2 hours, indicating a low convergence time. Both the median relative error and rrl are

0.1, indicating that the network coordinates are very accurate. Moreover the ralp is 0.15, eventually becoming as low as 0.1. This shows that for our deployment, using a network coordinate system does not impose a high latency penalty.

3.2.2 Attack Evaluations

We evaluate our attacks from Section 3.1, using the spatial and temporal thresholds of 5 as determined in Section 3.2.1. The attackers join the network at time 120 minutes, as this is when the network has stabilized. Note that all numbers reported are at time 500 minutes unless otherwise specified.

Previous Attacks

To establish a baseline for comparison with the effectiveness of our attacks, we implemented the previously proposed "coordinate oscillation" attack [24] (in which attacker nodes report completely random coordinates with low relative error) and measure the performance of the attack against our Pyxida deployment (without the Mahalanobis distance filter enabled). The progress over time of the median relative error with 11% attacker nodes is shown in Table 3.1.

Basic-Targeted Attack

The Basic-Targeted attacker targets a victim node and attempts to change the victim's coordinates in small steps. We attempt to change the coordinate of the victim nodes to be $Loc_T = (2000, 2000, 2000, 2000)$ with height 2000. Initially, for each victim node V (say with coordinate C_V), the attacker node A will report its coordinate to be $C_{A-V} = C_V + \delta$. For each subsequent time that V contacts A, the latter reports its coordinate as $C_{A-V} = C_{A-V} + \delta$, until $C_{A-V} = Loc_T$. Thus, V's coordinate is moved in small steps to the target coordinate.

Recall from Section 2.1.2 that a Pyxida node only updates its coordinate when it has sent a "ping" request. Thus, the victim nodes have to contact the attacker nodes for the attack to work. With 10% of attackers, the victim will contact one attacker node 10% of the time. Once an attacker node becomes a neighbor of the victim, it will stay in the neighbor's list for at least the next 32 iterations, which is long enough for

Table 3.1: The median relative error with 11% of attackers at different time snapshots for the network

Time (mins)	100	250	500	750	1000
Relative Error	0.23	0.21	0.23	0.22	0.2

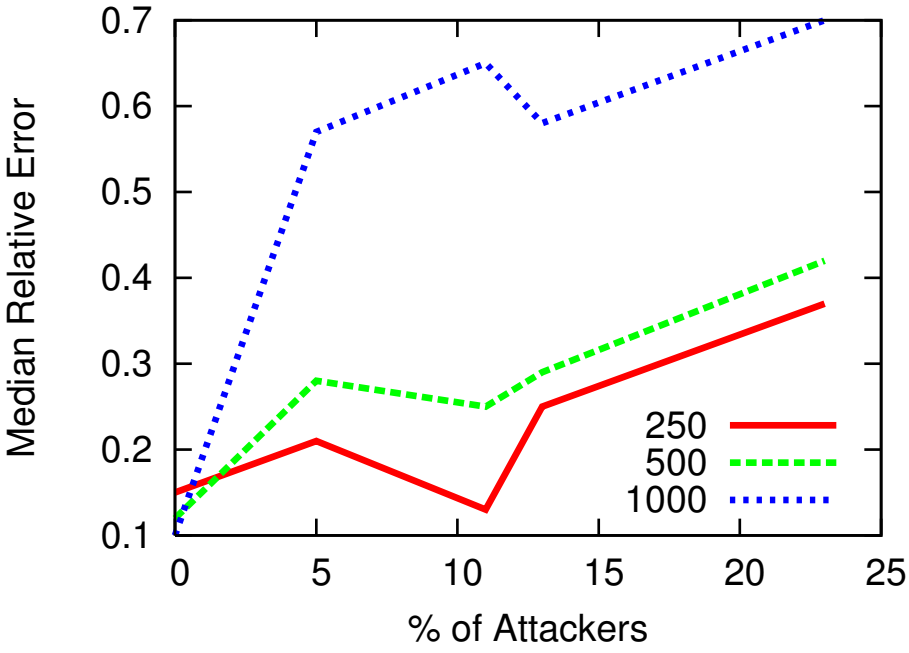

Figure 3.15: The average median relative error for varying % of attackers at different timestamps in the Mahalanobis distance secure mechanism.

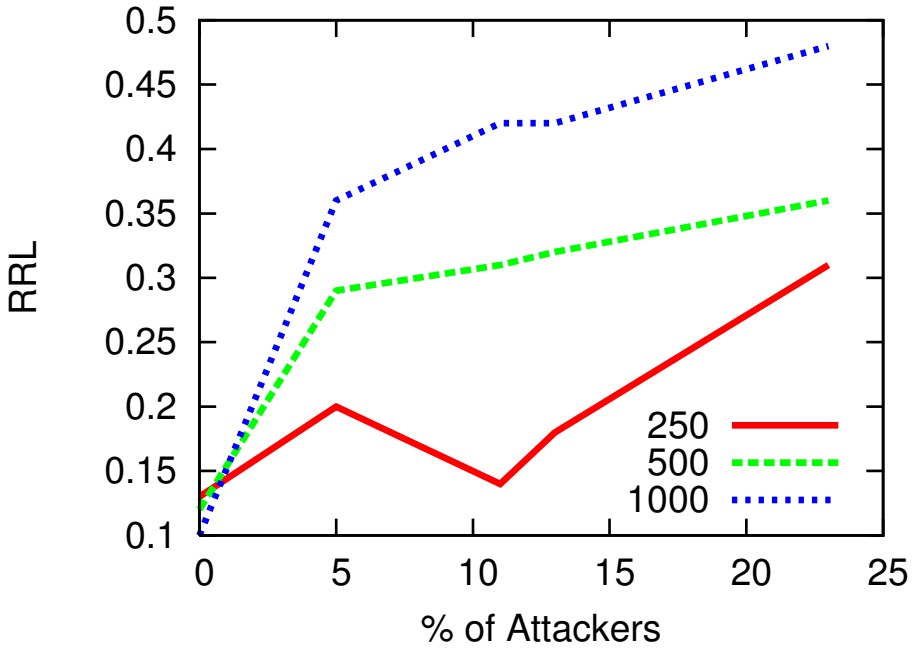

Figure 3.16: The average rrl for varying % of attackers at different timestamps in the Mahalanobis distance secure mechanism.

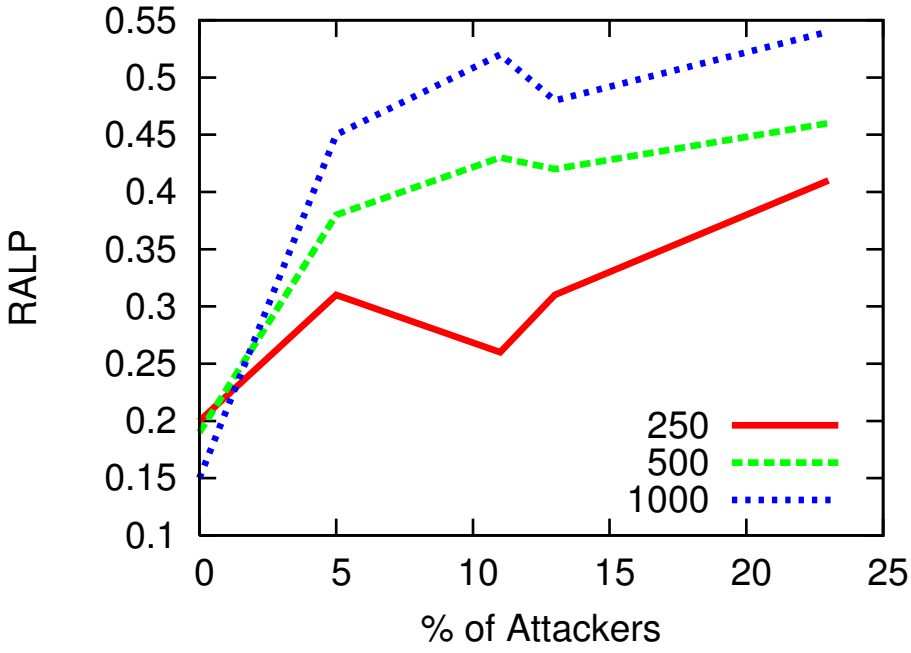

Figure 3.17: The average ralp for varying % of attackers at different timestamps in the Mahalanobis distance secure mechanism.

another attacker to be contacted and added to the list. The probability of an attacker node being part of the neighbor list after 32 iterations is $1 - 0.9^{32} = 96.5\%$. Thus, there is a very high probability that a victim node will have at least one attacker node in its neighbor list. Recall that the neighbor list is used every 10 seconds in Pyxida to calculate the current force. Since the attacker is updating its coordinate to be closer to the target coordinate at each time step, the victim will thus go closer to the target coordinate progressively. The Mahalanobis distance does not work in this case because the attacker is within the thresholds (since δ is small). The attacker only attacks the victim nodes and does not respond to other nodes in the network. Since there is no gossiping in Pyxida, this does not affect the attack.

Figure 3.15 shows the median relative error with varying percentages of attackers. (We note that 20% of attacker nodes may seem high, but many of the applications that implement network coordinate systems are vulnerable to Sybil attacks that make it trivial to control a large fraction of the nodes) The different lines show the error at different times in minutes – 250 minutes, 500 minutes, and 1000 minutes. Adding more adversaries significantly increases the median relative error (by more than 100% with only 11% of attackers). The relative error is increased from 0.12 with no attackers to 0.25 with 11% of attackers, an increase of 108%. The rrl and the ralp are shown in Figures 3.16 and 3.17 respectively. Again, it can be seen that the frog-boiling attack works as nodes are not able to compute a correct coordinate ranking based on RTT (rrl value is high) and they are also experiencing a high percentage of lost latency (high ralp value). With only 5% of attackers, after only 250 minutes (130 minutes after the attackers join the network), the latency loss due to using a network coordinate system is higher than 35% (compared to only 19% with no attackers). Moreover, as time goes, the error, rrl, and ralp increase. After 1000 minutes (a little over 16 hours), it can be seen from these figures that the network coordinate is unusable even with only 5% of the network being malicious – the relative error is greater than 0.5, the rrl value is greater than 40%, and the ralp is greater than 0.45%.

The frog-boiling attack on the Mahalanobis distance-based network coordinate system is as effective as a random attack on the original network coordinate system. At time 500 minutes, the relative error for the random attack is 0.23 while the relative

error for the frog-boiling attack is 0.25 with 11% of attackers. This means that the Mahalanobis distance is not providing any extra countermeasure to a network coordinate system. This supports our hypothesis that an outlier detection system is not suitable to secure a network coordinate system.

Aggressive Frog-Boiling

Our attack works by moving the victims in small steps to some location. In the previous section, the step size δ was $2ms$. In this section, we varied the value of δ to test the effect of a more aggressive attack, which will produce an impact on the network earlier – in other terms, we show how fast our attack can have an impact on the network. Figures 3.18, 3.19, and 3.20 show the median relative error, rrl value, and ralp value with 11% of attackers in the network respectively. The different lines show the different δ values used – 2, 5, and 10. The network starts to stabilize at time 2 hours, which is when the attackers join the network. With δ equal to 2, the relative error stays the same until time 6 hours, so it take 4 hours for the attack to start having an effect. On the other hand, with δ equal to 5 or 10, the relative error starts to increase at time 4 hours – after only 2 hours, the victim's network coordinates start to be disrupted. Two hours translate to 720 update intervals, which means that the malicious nodes were contacted only 36 times by the victims nodes. Thus, our attack is very effective and fast. Moreover, the rrl and ralp values start to show an upward trend as soon as the attackers join the network. This is because the victim's coordinates are being changed progressively and even at the beginning, the ordering of nodes (rrl) is different since the victim is in a different coordinate than it should be – the ralp value also increases due to this.

Using different step sizes (δ) always results in the same ending (except for $\delta = 1$ which is very conservative). The only difference in using different δ values is that the upward trend in relative error starts earlier and the upward trend in rrl and ralp values is more pronounced at the beginning. Thus, our attack is fast and efficient.

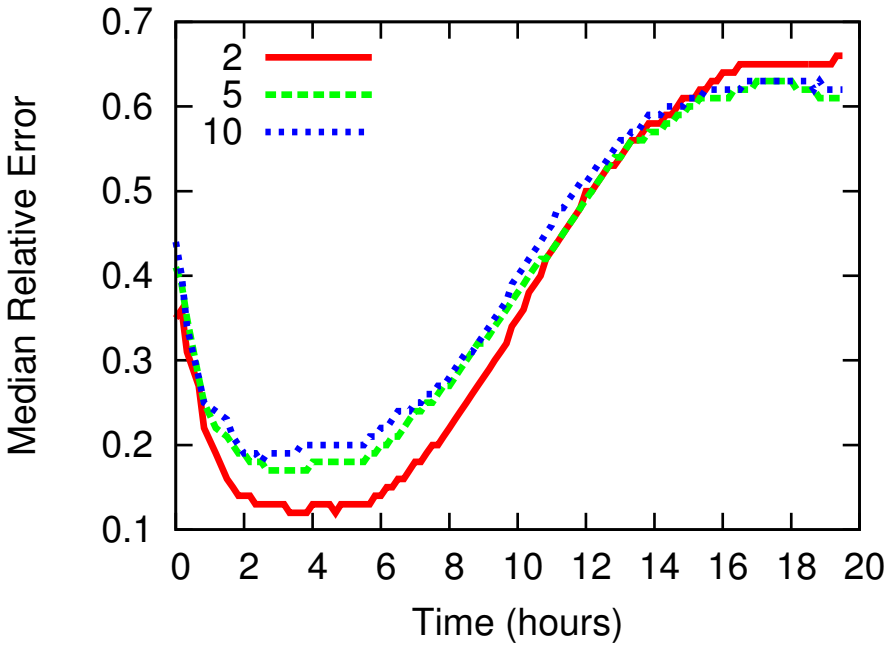

Figure 3.18: Median relative error for the targeted nodes with 11% of attackers over time and with different values of δ for the Mahalanobis distance-based secure scheme.

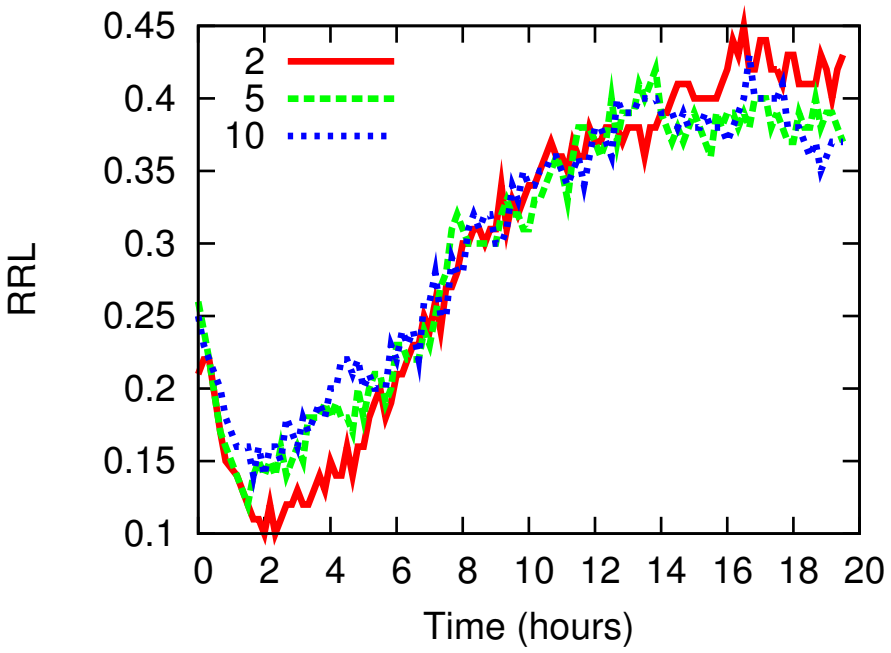

Figure 3.19: rrl for the targeted nodes
with 11% of attackers over time and with different values of δ for the Mahalanobis distance-based secure scheme.

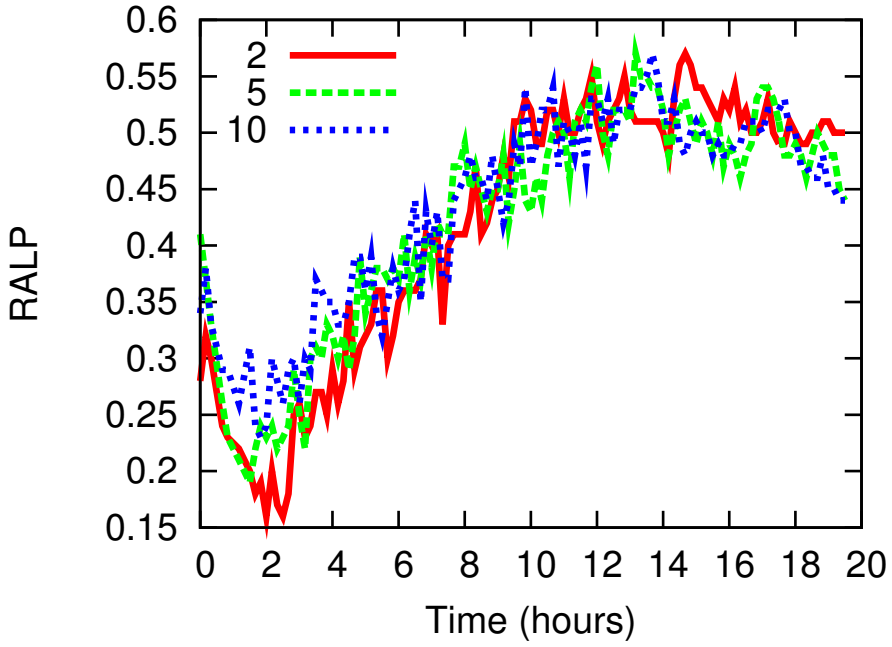

Figure 3.20: ralp for the targeted nodes
with 11% of attackers over time and with different values of δ for the Mahalanobis distance-based secure scheme.

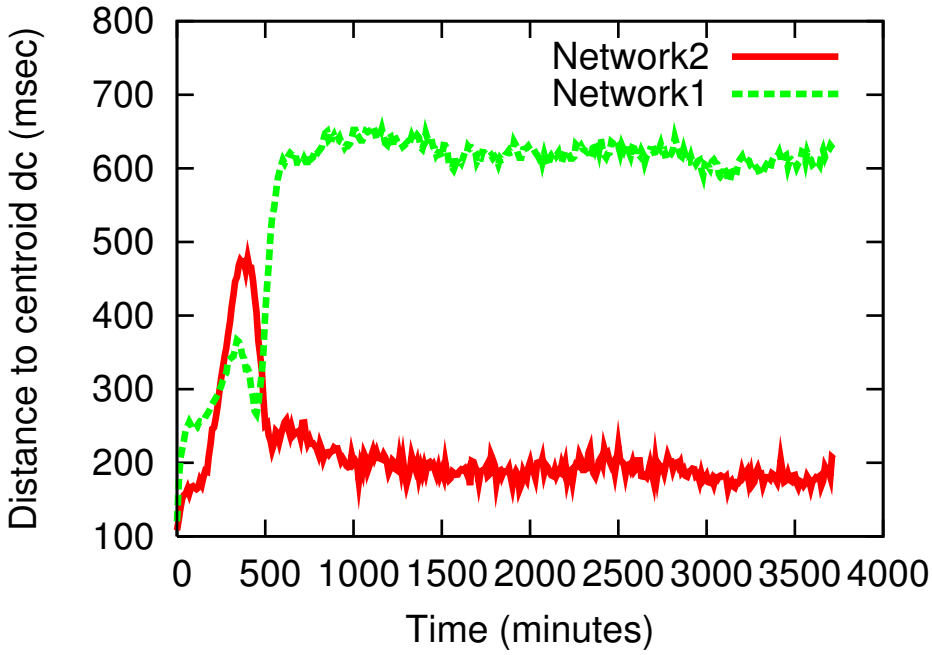

Figure 3.21: The coordinate distance to the centroid for the Network-Partition attack

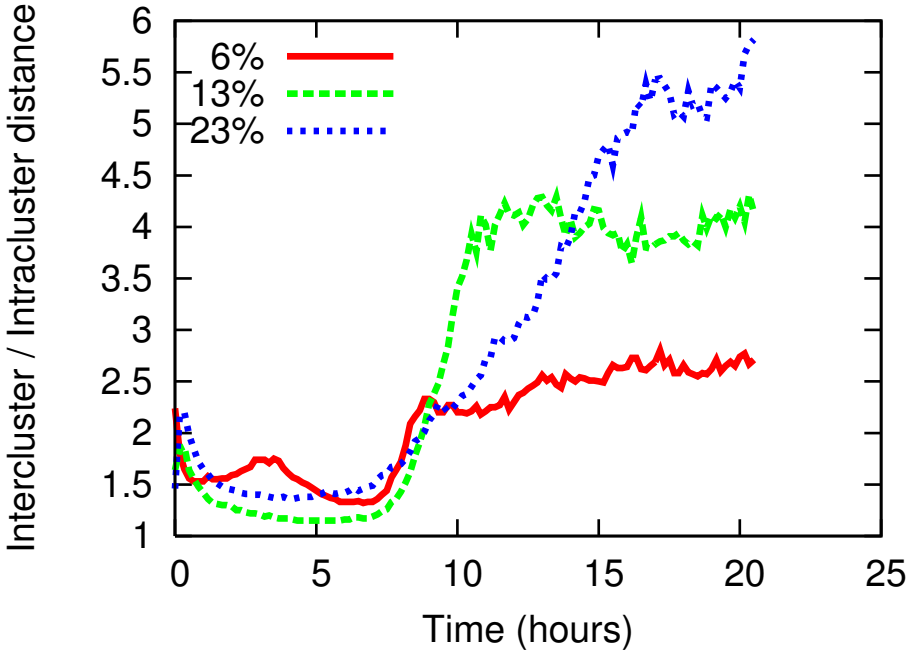

Figure 3.22: The intercluster/intracluster ratio for the Network-Partition attack

Network-Partition Attack

The Network-Partition attack is similar to the Basic-Targeted attack. Instead of just moving the victim nodes (*Network1*) to some far-away coordinate, the rest of the network (*Network2*) is also moved to some other location. This effectively partitions the network into two subnetworks. The targeted coordinate for Network1 was set to $Loc_{N1} = (1000, 1000, 1000, 1000)$ with height 1000 and the targeted coordinate for Network2 was set to $Loc_{N2} = (-1000, -1000, -1000, -1000)$ with height -1000.

In our experiment, 6% of the nodes were adversaries, 37% of the network was assigned to Network1 and 57% of the network was assigned to Network2. Figure 3.21 shows the average distance to the origin of the nodes in Network1 and Network2. At the beginning, the two clusters are close together. At time 500 minutes, which is how long it takes for the attack to have an effect, the two networks start to diverge. Network1 is pushed to 1000 while Network2 is pushed to -1000. Since the two clusters continue to exert some pull on each other, the intended coordinates are not reached, but the network is still effectively partitioned.

Figure 3.22 shows the ratio of the intercluster distance to the intracluster distance. The intercluster distance is the average over all the nodes, of the distance to the centroid of the opposite cluster. The intracluster distance is the average of all the nodes in a cluster to the centroid of that same cluster. The ratio shows how far apart the two clusters are moving from each other. The figure shows that over time, the two networks are getting pulled further apart from each other. The different lines show different percentages of attackers. This shows that our attack effectively partitions the whole network into two smaller networks far apart from each other. We note that our attack could easily be further extended to partition a network into more than two clusters.

Closest-Node Attack

An adversary tries to become the closest node (in terms of coordinate distance) to a victim in the Closest-Node attack. In Vuze, that would mean the attacker is the node that a file transfer would be initiated to and the attacker can control which file to transmit. The attacker node queries the victim nodes constantly to obtain their coordinates. When a victim node V with coordinates C_V queries the attacker node

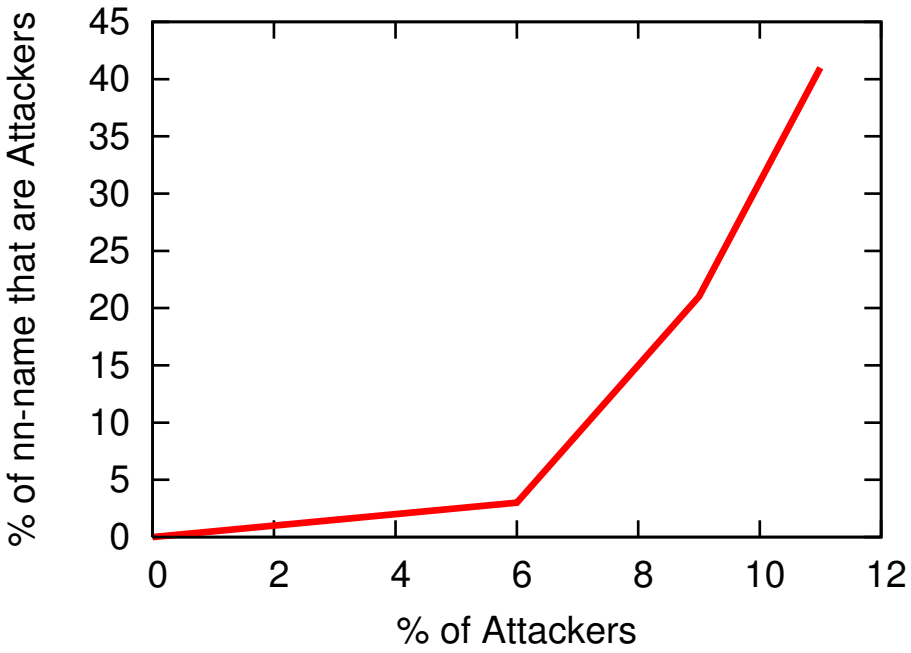

Figure 3.23: % of closest neighbors with varying % of attackers reported by the victim nodes in the network using the Mahalanobis distance that are attacker nodes.

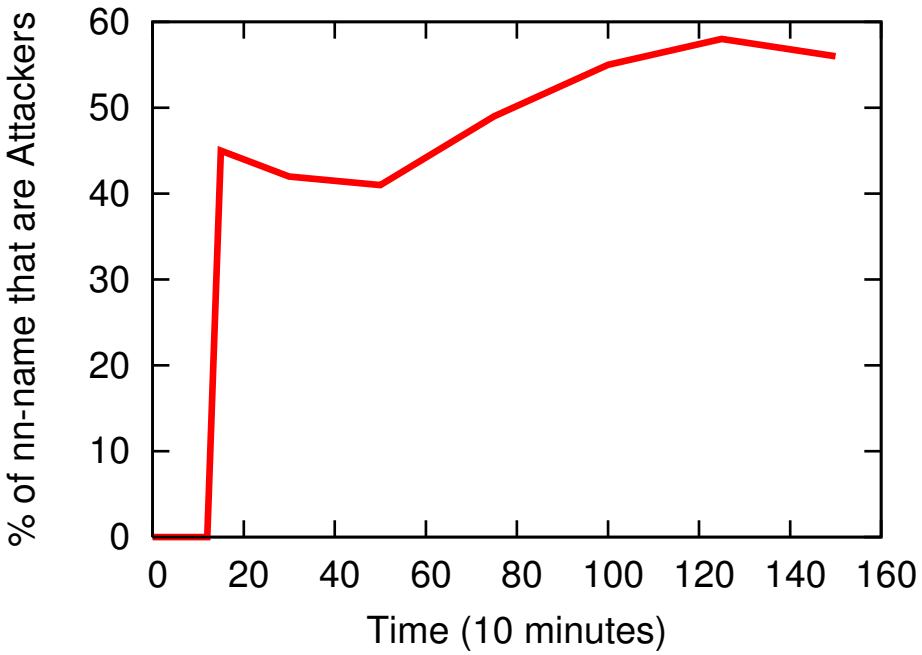

Figure 3.24: % of closest neighbors over time with 11% of attackers reported by the victim nodes in the network using the Mahalanobis distance that are attacker nodes.

A, A will reply back with coordinates $C_V + \delta$. The attacker node does not reply to other nodes in the network. The more attacker nodes in the network, the more likely for an attacker node to become the closest node. Moreover, since the attacker nodes constantly query the victim nodes for their coordinates, the attacker nodes will know when the victim nodes' coordinates change and update their replies accordingly.

We examined the reported closest neighbor of every victim node at 10 minute intervals. We took a snapshot at 500 minutes and determined how many times one of the attacker nodes was reported as being the closest neighbor of a victim node (this reporting is done every 10 minutes). Since the attackers are introduced at time 120 minutes, we do not consider the first 120 minutes of the experiments. Figure 3.23 shows the fraction of nodes for which the attacker was the closest as the percentage of attackers varies. As expected, with more attackers, it is more likely for an attacker to become the closest node. The attacker is not able to become the closest node 100% of the time because the victim nodes have to query the attacker nodes first before they are added to the neighbor list. The closest neighbor is only reported every 10 minutes, so the attacker could have been the closest neighbor for 9 minutes before it was discarded from the neighbor list.

With only 11% of attackers, we find that an attacker is able to become the closest neighbor to a victim node 41% of the time. We next hypothesize that the longer the experiment is run, the higher the probability that the closest neighbor of a victim will be one of the attacker nodes. Figure 3.24 shows a long-running experiment with 11% of attackers. Notice that the time axis is in 10-minute intervals. This confirms our hypothesis that over time, the attacker nodes have a higher chance of being the closest neighbor to a victim node. After one day, 11% of attackers have almost a 60% chance of becoming the closest neighbor to a victim node.

3.3 Kalman Filter

Kaafar *et al.* [25] propose to implement a Kalman filter [50] to detect outlier hosts in the network, that is, hosts that are lying or behaving strangely. The Kalman filter works by comparing the measured relative error and the predicted relative error for each node that replies to a coordinate update. If the difference is bigger than a threshold, the

update is rejected.

The authors of the paper focus their work on NPS [2, 3]. They introduce the notion of a trusted surveyor node (which is very similar to a landmark). The authors estimate that about 10% of the network have to be surveyor nodes. In the context of NPS, this will work since there are hosts which are deployed as landmarks and those hosts can also be used as surveyor nodes. The paper also implemented the anomaly detection on Vivaldi, but again, some hosts in Vivaldi act as trusted surveyor nodes.

As mentioned before, this defense scheme makes use of trusted peers. Those peers only contact each other to obtain the RTT and coordinates of the other trusted nodes. The trusted nodes then run an Expectation-Maximization step to determine the optimal parameters to be used for the Kalman filter. These parameters are then used by the non-trusted nodes to calculate the predicted error and predicted error variance, which in turn, are used to accept or reject updates from other nodes. The non-trusted peers contact the closest trusted node to obtain the parameters. The non-trusted peers contact both the trusted peers and the other non-trusted peers to initiate a coordinate update. If a normal peer A contacts another normal peer B, B replies with its coordinates and RTT. A first calculates the predicted relative error based on its past relative errors and the parameters obtained from the trusted node. A then calculates the difference between the predicted relative error and the measured relative error. If the difference is bigger than the threshold (based on the error variance), then the update is rejected.

3.3.1 Implementation

We implemented the Kalman filter described in [25] on top of Vivaldi. We set 8% of the network to be trusted. These trusted nodes contact each other until they stabilize, that is, their coordinates stop fluctuating and their reported relative error is constant. Then they each run the expectation-maximization step. After each trusted node obtains its parameters, the non-trusted normal nodes join the network. Each normal node contacts the closest trusted node in terms of network latency to obtain the parameters. The system then continues with each normal node randomly contacting another normal node or a trusted node to initiate a coordinate update. To confirm correctness of our implementation of the Kalman filter, we ran a simulation on the King [53] dataset, the same that the authors used, using the same initial parameters from the paper, and

obtained a similar result both for the experiment without any attackers and for the experiment with varying percentages of the "random" attacker.

3.3.2 Attack Evaluations

Here, we will show that the frog-boiling attack is effective against the Kalman filter defense mechanism. Although we could have implemented all three variants of the frog-boiling attack, we only performed experiments with the network-partition variant. The basic-targeted variant is a weaker version of the network-partition variant as the attacker only lies to a small subset of the network. Thus, the basic-targeted attack should be at least as effective as the network-partition attack. We did not implement the closest-node attack as there was no notion of "closest node" in our implementation of the Kalman filter whereas a Pyxida node explicitly records its closest node.

Figures 3.25 and 3.26 show the frog-boiling attack simulation results on partitioning the network. We found that $\delta = 10$ seems to provide a good lie step for the malicious nodes. Figure 3.25 shows that the median relative error remains unchanged even with 20% of attackers. Moreover, the number of malicious updates which were flagged as "malicious" is also very small. This leads us to conclude that the frog-boiling attack is not detected by the Kalman filter. However, Figure 3.26 shows that the intercluster/intracluster ratio increases. It is clear from this graph that our attack is successful. Moreover, contrary to our experiment with the Mahalanobis distance, our attack starts having an immediate effect on the network. By the end of the experiment, with 20% of malicious nodes, the intercluster/intracluster ratio increases to 2.2. We also plotted the "distance to origin" graphs, shown in Figures 3.27 and 3.28. Even with 10% of attackers, the two networks are being pushed further apart from each other. All the graphs show the average over 100 simulation runs.

3.4 Veracity

Veracity [26] is a distributed reputation system, which can be used on top of a decentralized network coordinate system such as Vivaldi. Similar to the scheme by [28], Veracity uses a two-step method to mitigate attacks on network coordinate systems.

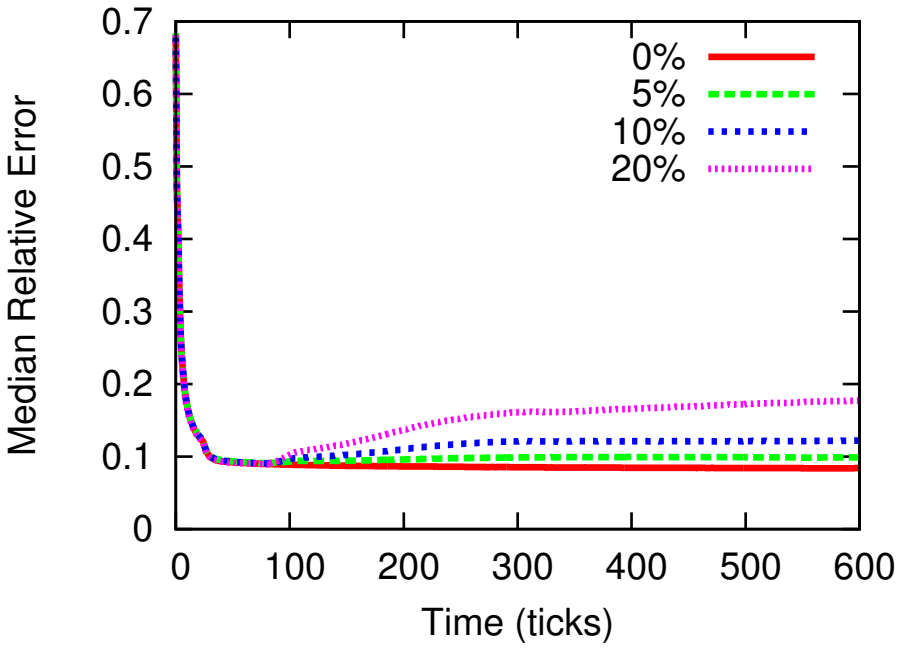

Figure 3.25: The median relative error for the attack on the Kalman filter-based secure scheme

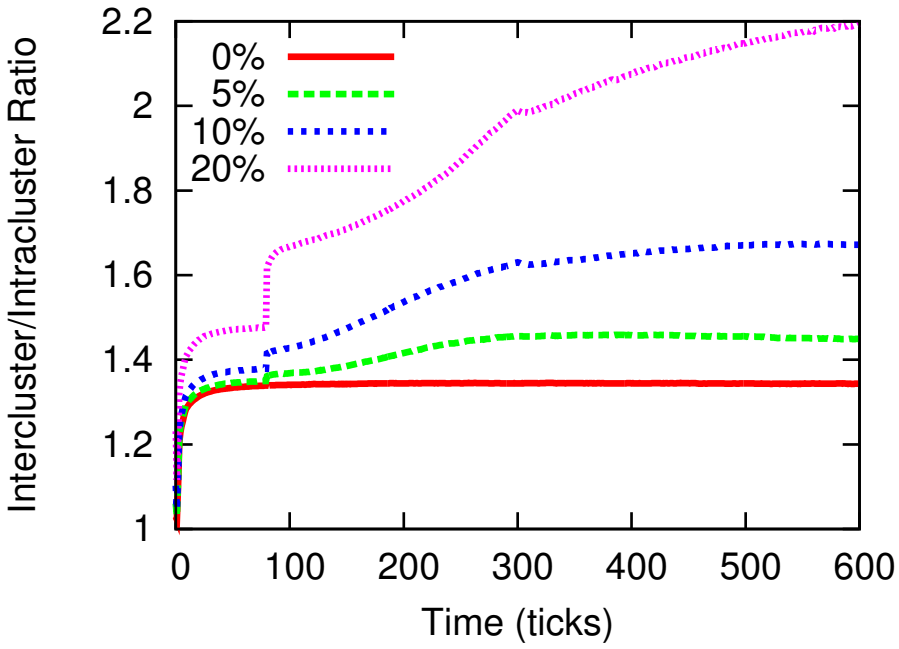

Figure 3.26: The intercluster/intracluster ratio for the attack on the Kalman filter-based secure scheme

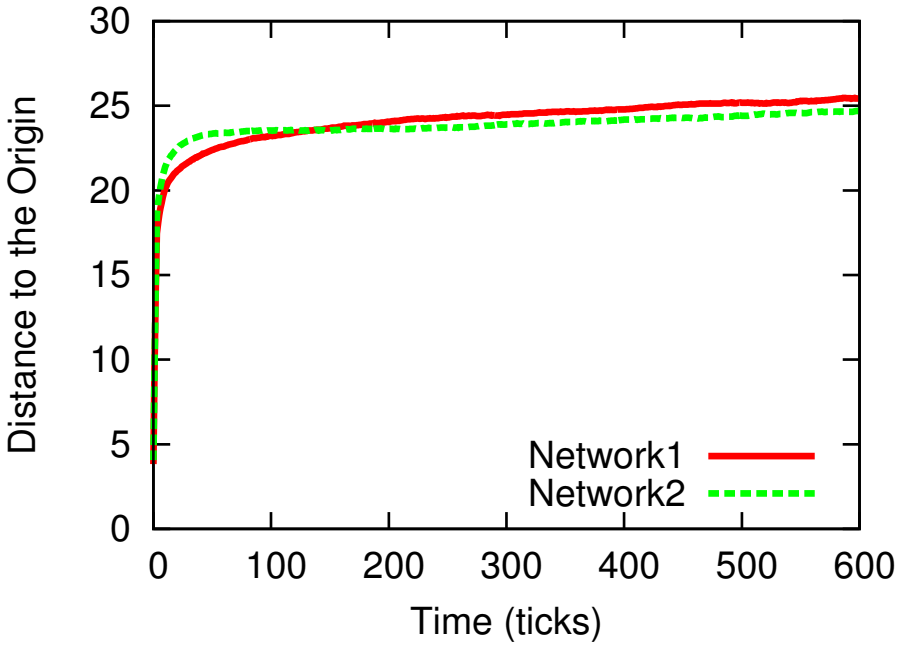

Figure 3.27: The distance to the origin with no attackers

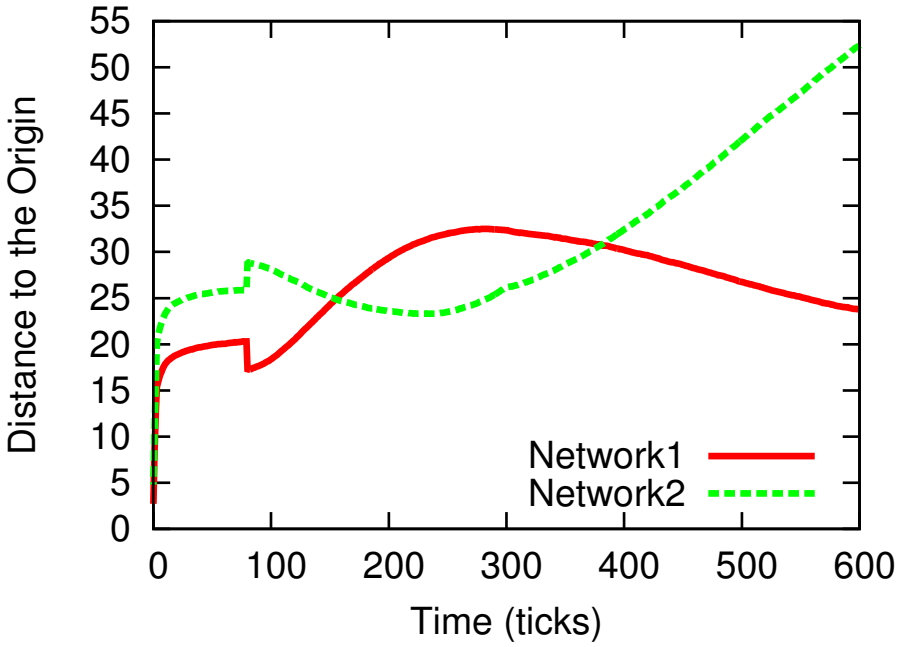

Figure 3.28: The distance to the origin with 10% attackers

The first step is to verify that the self-reported network coordinate of every peer is correct. Veracity uses a distributed hash table (DHT) and every peer is assigned a unique global identifier. Each node also has a verification set VSET, which consists of $\Gamma = 7$ members. The members of each node's VSET can be deterministically calculated and found using the DHT's global identifier. At every update step, every peer will report to its VSET its new coordinate. Each member of the VSET will measure its network latency to the peer, and calculate the error. A verifier node V will accept the peer's coordinate as valid if the majority $R \geq 4$ of that peer's VSET members computed an error less than $\hat{\delta} = 0.4$.

After a peer P passes the coordinate verification test, a random set of $\Lambda = 7$ peers in the DHT are contacted by the verifier V. This test prevents a peer from delaying its reply so as to inflate the network latency between P and V. This random set of nodes is called the RSET and changes at every coordinate update. V first calculates the average error ratio of its current coordinate with respect to the RSET. Then V calculates the new average error ratio of its new coordinate, if V were to accept P's update, with respect to the same RSET. If the two error ratios differ by more than $\Delta = 20\%$, then the update is rejected. Else, the update is accepted and V updates its network coordinate.

3.4.1 Implementation

Our experiments on Veracity [26] were fairly straightforward as we were able to obtain the source code used by the authors. Moreover, the authors performed their evaluations both on the simulator portion of their code and on PlanetLab. It was thus fairly simple for us to modify the code to add our frog-boiling attack. We performed both simulations and PlanetLab experiments but as they were similar, we only show the PlanetLab experimental results. Finally, we used the exact same parameters reported in [26].

3.4.2 Attack Evaluation

We chose to perform the Network-Partition attack when evaluating Veracity's defence mechanisms. As with our evaluation of the Kalman filter-based defense mechanism, we

did not perform the basic-targeted attack or the closest-node attack on Veracity. As a base for our experiments we used the implementation described by the authors of [26], which uses the Bamboo DHT [54, 55] as an underlying layer of communication. We have performed both simulations of the attacks as well as PlanetLab experiments; however we will present only PlanetLab results, since they provide more accurate estimation of practical deployment and both the simulation and experimental results were similar.

Through simulation we found the step size $\delta = 15$ to provide noticeable effect, and yet maintain a relatively low error. Figures 3.29 and 3.30 show the results from our PlanetLab experiments, where the number of nodes varied slightly between 460 and 480 over the course of the experiment, while the rest of the network-wide parameters were fixed to default values, as described in the original paper: $\hat{\delta} = 0.4$, $\Delta = 20\%$, $\Gamma = 7$ and $\Lambda = 7$.

The details of the attack are as follows. Each attacker chooses whether to shift its reported coordinate by $+\delta$ or $-\delta$ based on the victim's unique global identifier, GUID. If the most significant bit of the GUID is 1, the attacker chooses the positive shift; otherwise, the attacker shifts its reported coordinate by $-\delta$. This allows us to split the network into two roughly equal parts, because the GUID is the hash of the IP address and is evenly distributed. Whenever a node queries the attacker for his coordinate, before replying with the forged coordinate, the attacker sends out messages to the VSET members claiming that his coordinate has changed and provides the shifted coordinate. In our experiment we have found it necessary to compromise only the first step of coordinate verification. Thus we avoid delaying RTT probes and do not interfere with the RSET step of coordinate verification. As seen in Figures 3.29 and 3.30, the attack starts after the network has stabilized, at time = 50 minutes. The effect is seen almost immediately, as the two networks begin to drift apart. With 15% of malicious nodes the intercluster/intracluster ratio increases to 3 by the end of the experiment. We note that the ratio keeps showing an upward trend. Since the lie δ was small, it never caused any VSET or RSET verification to fail. Thus, our attack on Veracity remains completely undetected.

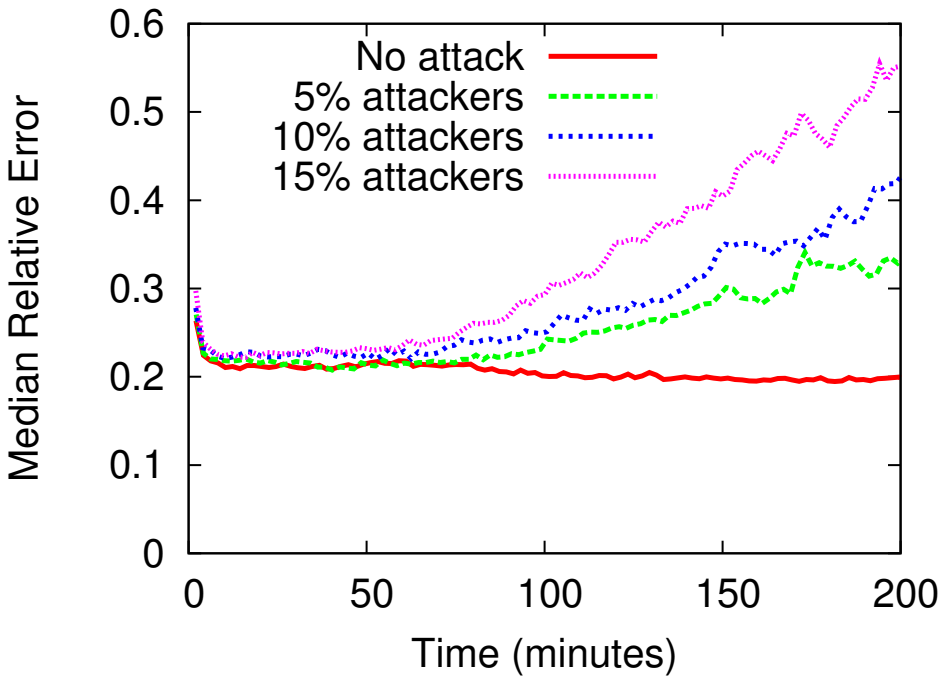

Figure 3.29: The median relative error Veracity
Attack starts at time = 50 minutes.

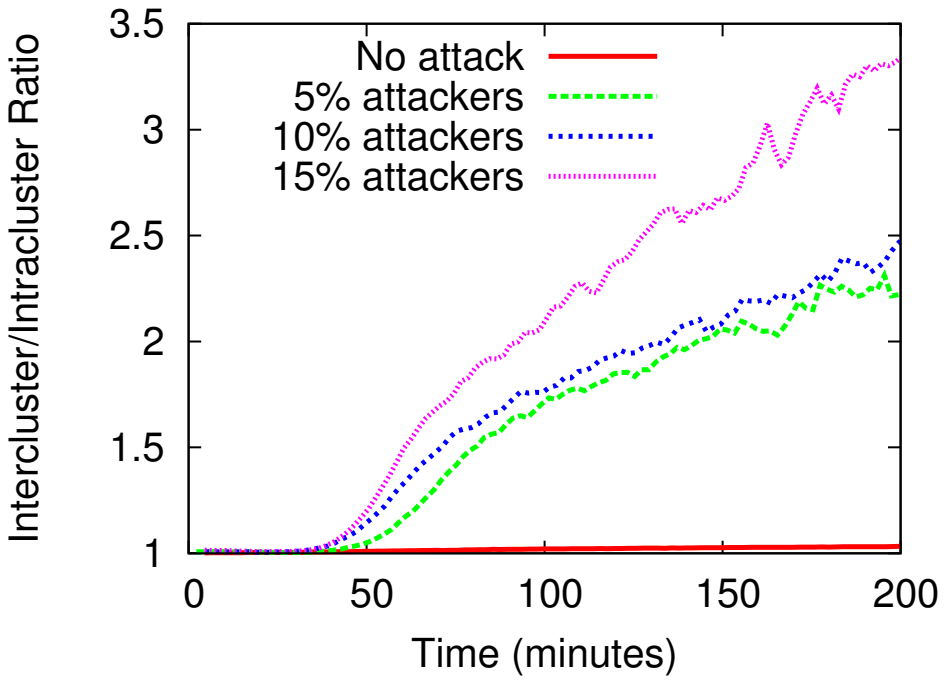

Figure 3.30: The intercluster/intracluster ratio for Veracity Attack starts at time = 50 minutes.

3.5 Summary

A stable, decentralized network coordinate system could potentially provide a beneficial service for many Internet applications. However, early systems provide no protection against malicious participants: even a single adversary can cause the entire coordinate system to fail. One apparent solution to such a dilemma is to add an anomaly detection mechanism to the coordinate system. Previous studies have shown that such a mechanism can prevent adversaries from disrupting the network. However, protection against more complicated adversaries is fraught with difficulty.

Consider a node in a network coordinate system that has some outlier detection mechanism. In order for the node to determine its coordinates, it must learn about the coordinates of its peers; that is, it must accept some updates. The range of updates it accepts must be based on recent history, since network conditions can vary widely. However, under these two assumptions an adversary can slowly expand the range of data accepted by the node by influencing the node's recent history. We call this attack the *frog-boiling* attack. In this Chapter we have introduced three variants of the frog-boiling attack and empirically demonstrated that the attack effectively disrupts the Vivaldi network coordinate system to a greater extent than previous attacks, and that the attack is completely unmitigated by Mahalanobis distance-based outlier detection, Kalman filter-based anomaly detection, or Veracity, the distributed reputation system. Thus, our attack is effective against three of the recently proposed "secure" mechanisms for network coordinate systems. We argue that the other similar schemes [28, 27] are also vulnerable to our attack.

The task of securing a distributed network coordinate system against adversaries seems very challenging. One of the problems is that the current distributed network coordinate system mechanisms (secure or not) rely only on a node's local view of the network. Errors are also inherent in network coordinate systems as there is no perfect embedding of the matrix of round-trip times into Euclidean coordinates. Moreover, network conditions on the Internet are dynamic and network coordinates and errors change over time. Because of this, it is a challenge for a node to know whether a reported coordinate and RTT is correct or faked. Thus, a secure network coordinate system will need to provide some mechanism to verify a node's reported coordinates and/or RTTs.

The success of the frog-boiling attack demonstrates that outlier or anomaly detection is not a secure mechanism to provide this service. Moreover, the attack is also successful against a distributed reputation system. This leads us to conclude that many of the proposed secure network coordinate schemes are vulnerable to more clever attacks, and the construction of a secure system will require careful evaluation.

Chapter 4

Attacking the Vuze Network

This chapter shows how an application using a network coordinate system to improve efficiency, can be attacked. Instead of attacking the actual application, the network coordinate system is targeted such that the eventual outcome of the "indirect" attack is similar to directly attacking the application. However, the cost of targeting the network coordinate system is much cheaper than targeting the application. More specifically, the Vuze DHT routing uses network coordinates to contact closer next-hop peers in an attempt to locate the content faster. Vuze allows five parallel but not independent queries for each search. If a malicious peer is contacted on the first hop, it can advertise that it knows of other (malicious) peers which hold the content being searched for. Thus, if a malicious peer is contacted on the first hop, this search is considered to be hijacked as the malicious peer can return other malicious peers close to the target, and the attacker can eventually return bogus results. Although attacks [14, 34, 35, 36, 37] against distributed hash tables (DHT) [38, 39, 30] to try to hijack searches have been proposed before, we show that the network coordinate system can be exploited so that searches are hijacked more efficiently. Only 32 nodes are required for our attack to be successful, regardless of the total number of nodes in the Vuze network. Although [37] is effective with a fixed number of malicious peers, it requires 100MB/s of download bandwidth whereas our attack requires only 700KB/s of download bandwidth, more than two orders of magnitude cheaper.

The main point of this attack is that although implementing and using a tool to

improve performance can be useful, that tool can also be exploited to abuse the application. Although it has been shown that existing network coordinate systems are insecure [23, 22, 24, 40], all the attacks were on the actual network coordinate systems in an attempt to disrupt the network latency estimations.

Our attack on the network coordinate system is to appear closer to all the peers in the network than anybody else. Since a Vuze node would pick closer peers to send its search queries, if an adversary is very close to every peer, then it will most likely receive a majority of search queries. The goal of the attacker is to be among the first ones contacted during a search query. Once the attacker becomes the first hop, this search is considered to be hijacked. The attacker's plan is thus to 1) lie about its network coordinates such that it appears very close to every peer on the network, 2) remain online such that it becomes the first hop on every search query, and 3) return other malicious peers whenever it is queried. A detailed description of our attack is in Section 4.1.

An attacker trying to directly attack the DHT routing of Vuze can capture at most 2% of all first-hop queries, that is, an attacker can hijack 2% of all searches. However, an attacker performing our proposed attack can capture almost 20% of all first-hop queries, that is, an attacker can hijack 20% of all searches. Section 4.2 provides more information about our experimental results.

4.1 Attack on Vuze

4.1.1 Overview

Our attack is a practical application of the *Closest node attack* described in [40]. The objective of the attack is to capture the DHT lookups performed by other Vuze clients. With successful distribution of colluding attacker nodes in the Vuze ID space, it is possible to capture a majority of lookups, giving us complete control of the network. To achieve this we exploit the lookup mechanism in Vuze, in particular the ordering of nodes according to the network coordinate distance when choosing the next hop. In our attack, the attacker claims to be the closest node to the victim in terms of network coordinate distance, which causes the victim to send a query for each search lookup to the attacker. It is important to note that our attack is significantly different from

previous attacks on DHTs, such as Sybil attacks [14] or Eclipse attacks [34]. This is due to the fact that we do not directly target the routing layer of the DHT, but rather target the network coordinate system, which indirectly influences the routing of lookups. Although the same outcome is obtained, the means and cost of achieving that outcome is very different.

4.1.2 Execution

Let us describe a step by step execution of the attack. First, it is important for the attacker to become a member of the victim's routing table. In order to do so, the attacker A sends a *ping* message to the victim V to indicate that A is a node in the network. This achieves two important goals: V adds A to its routing table (or replacement list) and tells A its current network coordinates C in the *pingReply* message. We know we were successful in penetrating V's routing table when V sends a *ping* message to the attacker. This happens shortly after V adds A to the routing table in order to verify that A is still alive. If the attacker did not receive such message, he simply needs to continue sending *ping* messages to the victim, since the attacker might have become a member of the replacement list and needs to remind V that he is still alive in the network. This leads us to the second step of the attack: after receiving a *ping* message, A replies with falsified coordinates $C' = C + \delta$ piggybacked on top of *pingReply* message to V, making it appear as if A was indeed close to V. This lie allows the attacker node A to become the closest peer to V. Now the attacker must continue to perpetuate the lie. In order to do so, the attacker periodically sends any one of the messages discussed in Section 2.5.1 to obtain an up-to-date knowledge of the victim's network coordinate position. Similarly, when A receives any of such messages from V, he must respond with newly forged coordinates $C' = C + \delta$, where C is the last known position of V.

To summarize, below is what the attacker needs to do in order to hijack every search query for the whole Vuze network:

1. Send ping messages to each victim to get into their routing table.

2. Once in a victim's routing table, keep sending ping messages to get the latest network coordinate of that victim. This allows the attacker to lie in a more effective manner so that it will always be the closest node to each victim.

3. Remain active and participate in the Vuze network.

4. When receiving a query, return other malicious peers closer to the target key T.

4.1.3 Analysis

Our attack allows the attacker to capture the lookups in Vuze DHT at a low cost. The execution of the attack does not require a large number of sybil identities. The only requirement is that attacker IDs are uniformly distributed across the 160-bit ID space, to cover all the buckets in the victim's routing table. This is easy to achieve, since changing the port of the attacker node will result in a new Vuze ID. Our goal is to place the attacker nodes into the 8 buckets marked in Figure 2.1, since a large portion of the lookups performed are for keys that have no first-bit prefix match with the ID of the searcher. Such placement will allow the attacker to become the first hop for over 50% of the lookups, giving us control of a large portion of the network. Note that the other 4 parallel queries have little effect on this attack, since the attacker can always provide more attacker nodes close to target key T in his reply, rendering other queries' responses unappealing. Due to the way the Vuze search lookup works (see Section 2.5.2), the attacker actually needs to have two entries in each top-level bucket, since each bucket contains 20 Vuze nodes and Vuze sorts the first 10 nodes by Vivaldi distance. For example, for bucket 0000, the attacker needs to create two Vuze IDs 00001... and 00000.... This ensures that the attacker can hijack all the search queries in that bucket. With 8 top-level buckets, the attacker needs to create 16 Vuze IDs. However, it needs double that number to also hijack the other 50% of the network (the victims with Vuze IDs with different first bit). A total of 32 attacker nodes with carefully crafted Vuze IDs is required to hijack 50% of the search queries for the whole Vuze network.

We note that the success of our attack depends on two factors:

1. Ability to get into routing table.

2. Ability to be the closest node to each victim.

Our attack has very low network and computation cost. Each *ping* message sent in order to penetrate the victim's routing table is only 42 bytes (*pingReply* is 80 bytes), thus

bandwidth consumption is very low. Processing overhead is minimal as well, since are no computationally or memory intensive tasks involved. Despite its low cost, our attack has serious implications to those using Vuze for file sharing. As we mentioned above, Vuze uses DHT lookups to locate a suitable torrent file to start a download. However, if a user is a victim of our attack, his or her lookups can be redirected to another malicious node, which will serve a bogus torrent file causing the user to perform a futile download. In addition, any research applications [56, 57] relying on correctness of the Vuze DHT might suffer from such malfunction.

4.2 Experimental Results

4.2.1 Experimental Setup

We downloaded the source code from the Vuze [7] website in August 2010. We only made minor modifications to the code to make sure that our experiments are as close to regular users' version as possible. Thus, our experiments are applicable to the real Vuze network. The modifications made include extra logging and extra code for our attack. All our experiments were performed on PlanetLab [46]. Although all the Vuze clients were connected to the real Vuze network, our malicious nodes only attacked the victim PlanetLab nodes.

To simulate search lookups, every 30 minutes, each victim Vuze node performs a search lookup for a randomly generated target key. The hijacking succeeds if one of the five first-hop queries for each key is sent to an attacker node. This is because the attacker can reply with closer (in terms of Vuze ID distance) nodes to the target key than other normal Vuze nodes. Those closer nodes will, of course, be under the control of the attacker.

4.2.2 Analytical Estimates

Recall from Section 2 how the Vuze routing algorithm works. The goal of our attack is to hijack all the queries (search requests) for every node on the Vuze network. We exploit the insecurity of the network coordinate system used by Vuze in order to hijack the queries. In order for our attack to be successful, our malicious nodes need to be in

the victim's routing table, preferably, at the top-level buckets. Let's assume a victim V. Obviously, if every entry in V's routing table points to a malicious node, then the hijacking is trivial. However, hijacking a node's routing table and maintaining the entries (the entries have to route requests and respond to queries) is very expensive. In our attack, the malicious nodes only need to maintain two entries for each of the 16 top-level buckets, for a total of 16 routing table entries instead of 160.

Due to the way the Vuze routing algorithm works, the malicious nodes have to "create" a Vuze ID so that it can "insert" itself into the victim's top-level buckets. To do that, one of the malicious node Vuze ID first bit should be 1 and the other Vuze ID first bit should be 0. Since the search lookup algorithm sorts the first 10 nodes in its bucket by network distance, the attacker has to insert two of its nodes into each top-level bucket. For example, for the 0000 bucket, the attacker has to create two nodes with Vuze IDs 00001... and 00000... so as to hijack every query for that bucket. Recall that we assume that once a query is hijacked, the whole search lookup is hijacked as the attacker can return its own peers closer to the target key T. Thus, the attacker only needs 32 malicious nodes with carefully chosen Vuze IDs to hijack 50% of search lookups for the whole Vuze network.

The success of our attack depends on the malicious nodes being added to the victims' routing tables. This is not easy as a victim's routing table, especially the top-level buckets, are usually full. The malicious peer has to be added to the replacement list of each bucket first, and then when an entry in the bucket is considered dead (does not respond to messages), then a random entry in the replacement list is chosen to replace the dead entry. Since this is completely random, the only way to get into victims' routing tables is to keep sending *ping* messages to them and hoping to be randomly added at some point. This takes time, but the attacker can create 32 malicious peers participating in the Vuze network for months and eventually, all 32 nodes will be added to every victim's routing table.

To determine the expected percentage of hijacked search lookups for a Vuze client V, we need to know the number of attackers added to V's top-level routing table. If the number is, say, 4, meaning that the top-level buckets contain four malicious peers, then the attacker will be able to hijack $\frac{4}{16} * 100 = 25\%$ of queries.

4.2.3 Experimental Baseline

To determine the efficiency of our proposed attack, we ran an experiment on PlanetLab with no manipulation of network coordinates. This would be a similar attack as the Sybil attack [14], where many malicious peers keep sending *ping* messages to the victims in an attempt to be added to their routing table. We consider this as our *baseline*. The only difference between this experiment and our attack experiment is that in the attack experiment, malicious peers lie about their coordinates when responding to requests from the victims.

4.2.4 Results

We ran both the baseline and attack experiments a few times on PlanetLab. Each experiment was run for at least five days. We also had two sizes of experiments: a "large" one which consisted of rougly $1,000$ nodes, and a "small" one consisting of roughly 500 nodes. All the nodes were connected to the real Vuze network but only the PlanetLab nodes were the victims. For each experiment, about 10% of the network was malicious, and the rest of the network were victims.

Figure 4.1 shows the results of our experiments. For the small experiment, the baseline was able to capture 0.83% of search queries whereas our attack was able to hijack 10.7% of search queries. This is lower than our theoretical analysis of 15.6% of search queries hijacked. Our theoretical analysis is very conservative as it assumes that every top-level bucket is full and the Vuze IDs in each bucket evenly distributed. For the large experiment, the baseline hijacked 1.27% of search queries and our attack hijacked 18.8% of search queries. Again, our attack result is lower than our analytical analysis of 37.5% of captured search queries. Our theoretical analysis depends on the number of attackers that were able to be added to victims' routing tables. On average, 2.5 out of the 160 routing table entries for each victim pointed to a malicious node for the small experiment. This number increased to 6 for the large experiment, which explains the higher percentage of hijacked search queries.

The success of our attack depends on 1) the ability to get into victim's top-level buckets, and 2) lying to victims about the network coordinates such that the attacker is the closest peer to each victim. We reran the small experiment but with 20% of

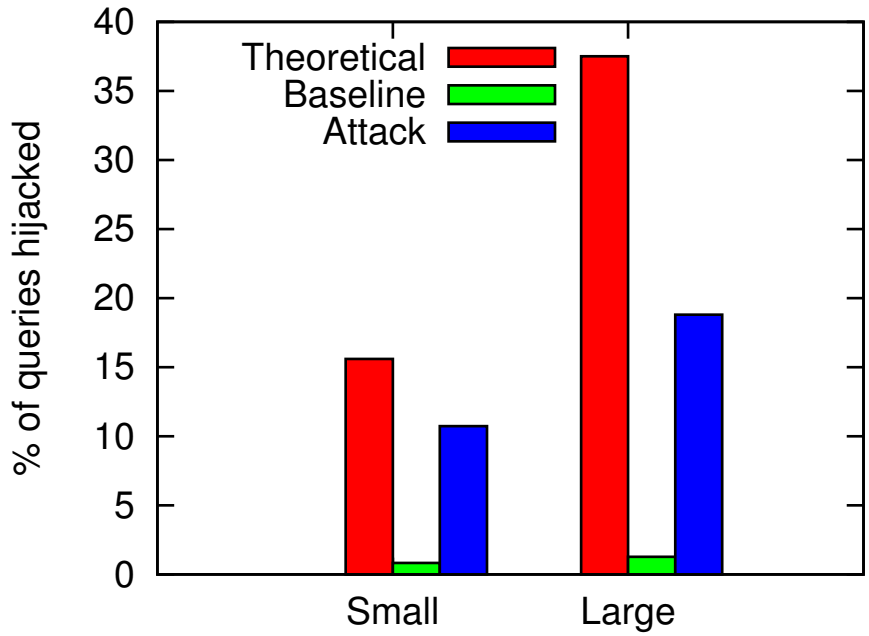

Figure 4.1: The percentage of hijacked search queries
for both the large and small experiments obtained by analytical measurement, baseline, and our attack.

attackers instead of 10%. As expected, the average number of malicious entries in the top-level routing table of victims rose to 4.3, which allows the attacker to hijack 16.5% of search queries, when compared to a theoretical hijack rate of 27%.

Our attack is less effective than the expected theoretical values. This is because our attacker is not the closest peer to each victim; thus the first-hop search queries are sent to other peers. The reason for this can be depicted in Figures 4.2, 4.3, and 4.4. The figure shows the coordinates of three victim peers over time. Since it is hard to picture a 3-dimensional coordinate with height, the figure shows the distance to the origin for each peer. The figure shows that for all three peers, their coordinates change drastically over time. Thus, the coordinate reported by a malicious node at time t to become the closest node to a victim V, is no longer the best coordinate at time $t' \neq t$. The attacker needs to keep lying about its coordinate. The frequency of possible lies depends on how often the victim contacts the attacker. Since this is beyond our control, there are gaps when the attacker node is no longer the closest peer to the victim. This explains why our attack is less effective than expected.

In an attempt to improve the effectiveness of our attack, we implemented a "smarter" version of our attack. Instead of the attacker nodes reporting themselves to be the closest node to the victims all the time, they try to position the victim nodes such that they will be the closest nodes for a long period of time. When an attacker node receives a *ping* message from a victim node, it compares the first bit of its Vuze ID to the first bit of the victim's Vuze ID. If the first bits are different, then that means the attacker node is in the top-level buckets of that victim's routing table. Then the attacker replies with a *pingReply* message with its coordinate being close to the victim's coordinate. This is similar to the original attack. However if the first bits are the same, then the attacker reports a coordinate far away from the victim in an attempt to isolate the victim node. In that way, when an attacker node in the victim's top-level routing buckets is contacted next, that attacker node will be the closest node to the victim for a long time. We ran a few experiments for three days each; Figure 4.5 shows the result. The percentage of hijacked queries is 10.5% for our attack, compared to 12% for the theoretical result (average of 1.9 attacker nodes in top-level buckets of victims). Our attack is thus more effective.

We emphasize that our attack is very efficient. Each attacker only needs to send one

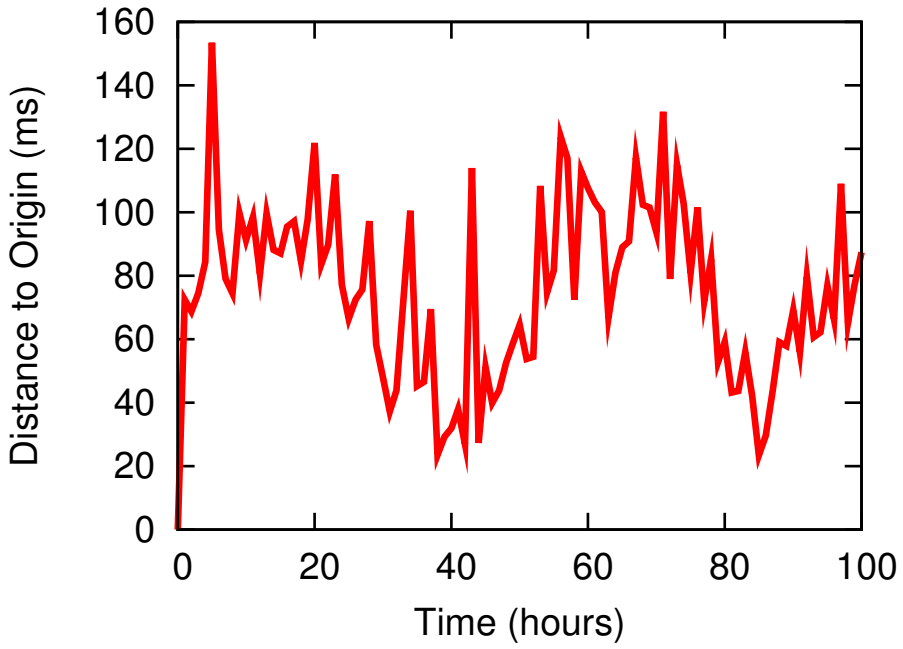

Figure 4.2: The coordinates of a PlanetLab Vuze node over time.
The graph shows that the coordinates vary over time, making it harder for the attacker to be the closest node in terms of Vivaldi distance all the time.

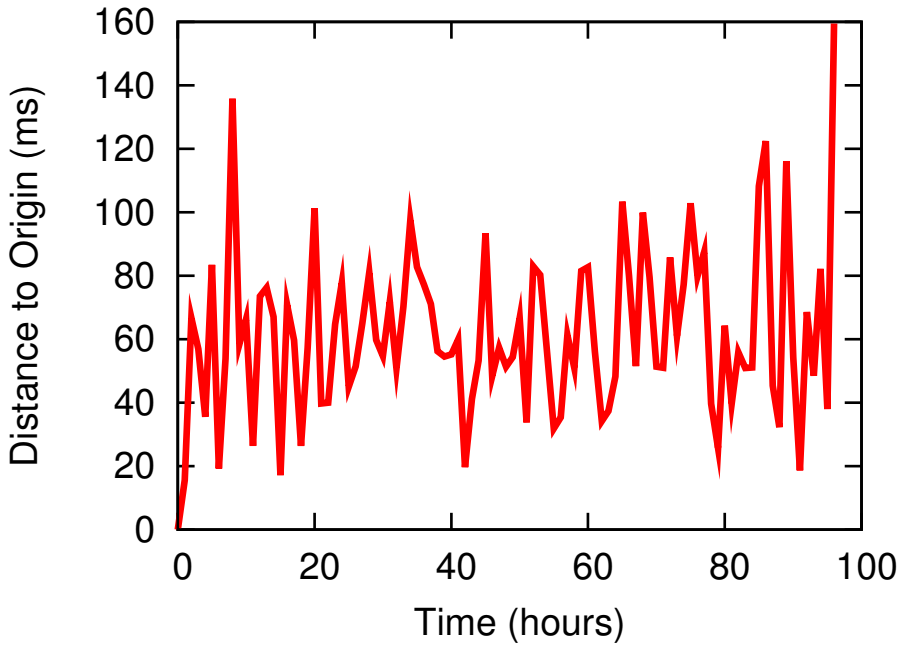

Figure 4.3: The coordinates of a PlanetLab Vuze node over time.
The graph shows that the coordinates vary over time, making it harder for the attacker to be the closest node in terms of Vivaldi distance all the time.

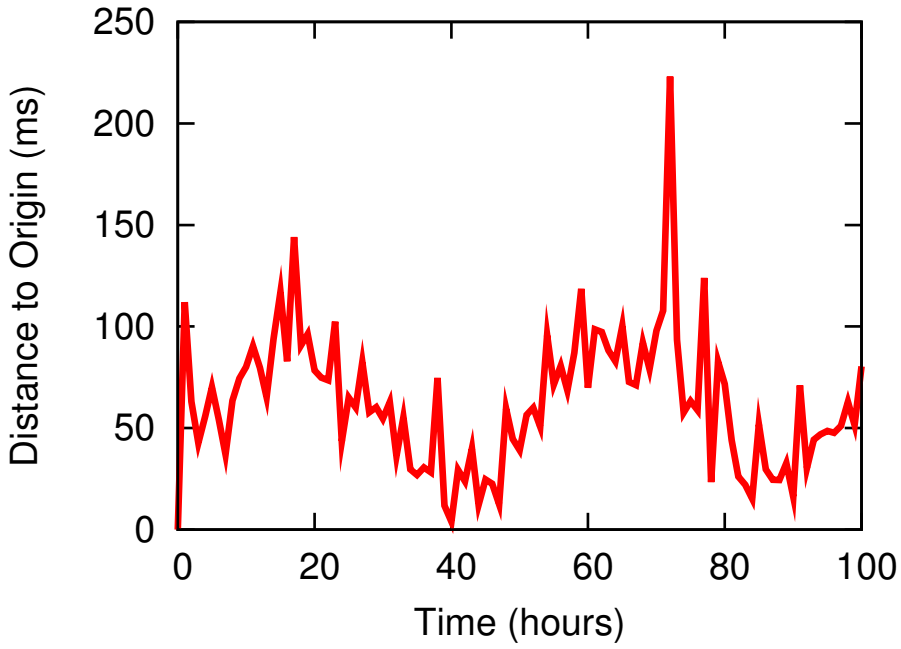

Figure 4.4: The coordinates of a PlanetLab Vuze node over time.
The graph shows that the coordinates vary over time, making it harder for the attacker to be the closest node in terms of Vivaldi distance all the time.

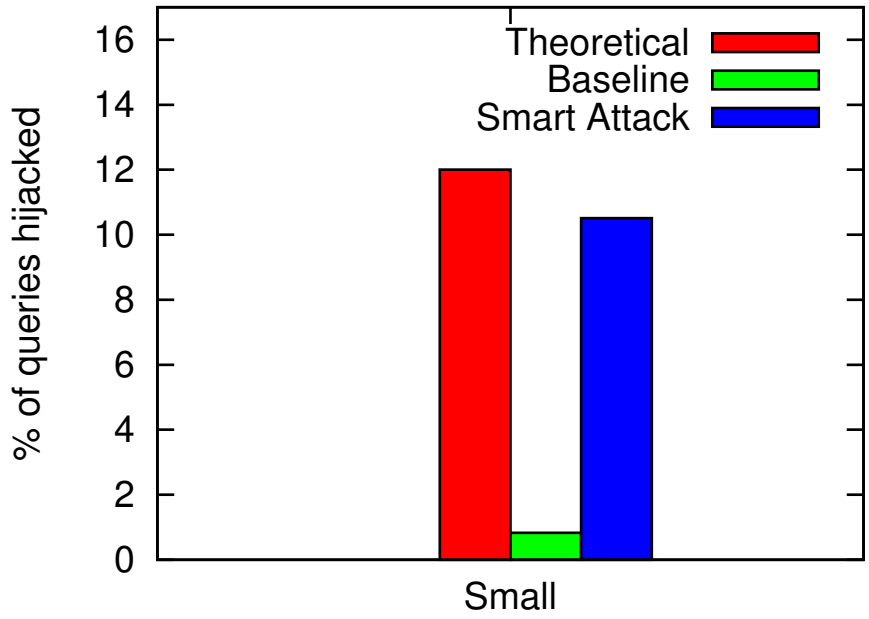

Figure 4.5: The percentage of hijacked search queries obtained by analytical measurement, baseline, and our "smart" attack.

ping message every 2 hours to try to get into each victim's routing table. Each ping request is 42 bytes and each ping reply is 80 bytes. With a 2 million nodes Vuze network, that's about 12 KB per second uplink and about 22 KB per second downlink. Since we need 32 attackers, that translate to 371 KB per second of uploaded bandwidth and 711 KB/s of download bandwidth. This is much cheaper than the 100MB/s required for [37]. The cost to maintain the routing table is also fairly cheap. An average user can run at least one Vuze instance on its home machine. A determined attacker can easily run 32 Vuze instances on a more powerful machine. Each Vuze attacker instance uses around 55 MB of memory and less than 3% of one core of a quad-core 2.67Ghz Intel Xeon W3550 processor. If our attacker were to use Amazon EC2 [58] to conduct the attack, this would cost less than $650 per month.

As mentioned earlier, our attack's success depends on two things: 1) getting into victims' routing table, and 2) being the closest peer. To have more attackers be added to a victim's routing table, the attack needs to be run for longer, in the order of weeks and months, or to have a bigger initial pool of attacker nodes. After the victims' routing table has been infiltrated, the extra attacker nodes can be shut down. To always be the closest node, the attacker needs to obtain the victims network coordinate more often, and thus requires more *ping* messages to be sent.

4.3 Summary

Vuze is a popular BitTorrent file-sharing client. It uses a network coordinate system to improve the efficiency of its search lookups. However, the network coordinate system implemented is insecure and can be easily attacked. By becoming the closest peer, in terms of network coordinate distance, to every victim node in the Vuze network, an attacker can theoretically hijack every single search query. Experimentally, we were able to hijack almost 20% of all search queries. The reason for our lower success rate is due to the instability of the victims' network coordinates. Our attack is directly applicable to the real Vuze network and can be deployed any time. Moreover, our attack is very cheap to launch.

To mitigate our attack, Vuze should not be using an insecure network coordinate system in an attempt to improve its performance as it actually makes its system more

vulnerable to easier and cheaper types of attacks. Alternatively, Vuze could implement a secure network coordinate system such as Treeple [59].

Chapter 5

Treeple

In this chapter, we introduce a strong definition of security for latency estimation schemes that is robust to new attack types. Informally, our condition states that an adversary should be unable to influence the estimated distance between two honest nodes, and additionally, should only be able to increase the distance between pairs of nodes involving at least one adversarial node. We then consider whether previous secure network coordinate systems meet our definition. We demonstrated in Chapter 3 that several "secure" schemes are vulnerable to a variant of the "Frog-Boiling" attack [40], where an attacker injects a sequence of small inaccuracies – each of which appear plausible – that cumulatively result in the partitioning of the whole network into two independent clusters. We additionally show that even schemes that rely on trusted "landmark nodes" [2] can fail by allowing adversarial nodes to reduce their coordinate distance to targeted victim nodes in the network. The common fault underlying both of these vulnerabilities is that current network coordinate systems ignore the underlying network topology, and thus cannot distinguish between anomalous round-trip times due to adversarial manipulation and anomalies due to the topology and changing conditions in the network. We describe the common faults of current systems in more details in Section 5.1.3.

After defining security, we describe Treeple, our scheme for secure network latency estimation. Node positions in Treeple are not abstract coordinates but instead represent their position in the network graph in a way that allows efficient computation of the

estimated latency between a pair of nodes. Assuming a small collection of trusted "vantage points", we prove that Treeple meets our security definition even in the presence of an arbitrary number of adversarial nodes.

We evaluate Treeple on a large, real-world dataset in Section 5.3. For this dataset, Treeple has a median relative error of 0.26; This means that on average, half of the estimated latencies are within 26% of the true latency. For comparison, we simulated Vivaldi [1], a popular but insecure network coordinate system, on the same data set and the resulting coordinate scheme has a median relative error of 0.25. We note that the choice of Vivaldi is not arbitrary: all of the recently proposed "secure" or "stable" network coordinate schemes [24, 25, 26, 27, 28, 20] work by adding additional measures to Vivaldi coordinate calculations that attempt to discard anomalous inputs, and thus would have the same accuracy in the absence of an attacker. This shows that Treeple has accuracy comparable to network coordinate systems while providing provable security.

We additionally demonstrate that Treeple positions are highly stable: positions calculated on the first day of our dataset provide nearly the same accuracy nearly three weeks later. This has several important implications. First, because peers do not need to recalculate their positions the bandwidth overhead is significantly reduced compared to network coordinate schemes. Second, for the same reason, the "centralized" vantage points do not present a significant scaling challenge for Treeple. Finally, trusted vantage points do not present a central point of failure: the system can continue to function with high accuracy even if all vantage points are unavailable for an extended period.

5.1 Security

5.1.1 Threat Model

We model a network as a collection of N *end-hosts* connected by M *routers*, forming together a set of $N + M$ nodes. At any time t (we assume synchrony for simplicity only) there is a *network condition* $\chi(t)$ which assigns a *route* $route_\chi(n_1, n_2)$, a *return route* $rroute_\chi(n_1, n_2) = route_\chi(n_2, n_1)$ and a resulting round-trip time $rtt(n_1, n_2)$ to every pair of nodes (n_1, n_2). We allow each end-host n to measure both $rtt(n, n')$ and $route(n, n')$ for arbitrary hosts n'. We may extend our model to allow $\chi(t)$ to assign other conditions to peers and routers as well (for example, to model peer churn or packet

loss), but we omit these details for clarity of presentation.

We allow an adversary to control an arbitrary number of end-hosts, but no routers. Thus an adversary can send messages to arbitrary hosts, with arbitrary apparent origination, deviate from protocols in arbitrary collusive fashion, and arbitrarily inflate the measurement of $rtt(n, m)$ when m is adversarially controlled; however the adversary cannot effect the measurement of rtt between honest end-hosts, and cannot intercept, drop, or delay communication between honest end-hosts.

It may seem at first that excluding routers from adversarial control is a strong assumption. We argue, however, that excluding routers from adversarial control is reasonable in this setting; if an adversary could arbitrarily delay or redirect packets between any pair of peers (and thus affect the round-trip time between honest nodes) then a scheme to estimate network latency cannot succeed, since *any* latency estimate can be invalidated by the adversary. We note that under the current Internet architecture, an adversary that controls a *single* BGP speaker can exploit longest-prefix matching to receive traffic directed to arbitrary hosts. Furthermore, as recently shown by Goldberg, *et al.* [60], the Internet's policy-based routing is such that there exist single attackers that can intercept over 90% of all routes even when constrained by SBGP to use only existing routes. Goldberg *et al.* showed that their strategy was suboptimal, meaning the actual fraction of routes that a single attacker can intercept under SBGP may be even higher. Since we seek to provide provable security for the current Internet architecture, we must therefore assume that any attacker that controls a router is the worst-case attacker and can intercept and delay *every message sent* between peers.

We note additionally, that while the most desirable situation would be to resist such attacks, both the attacks in the current literature, and the attacks on existing schemes that we describe, fit within our threat model (and in fact, do not fully exploit the abilities we ascribe to an adversary.) Thus our threat model is *strictly stronger* than that considered in every previous work on the topic, and a scheme that provably resists our threat model will already rule out all of the known methods of attack.

5.1.2 Definitions

Latency Estimation Scheme

A *latency estimation scheme* for a set Peers of end-hosts consists of four distributed protocols:

- A global initiation protocol GlobalInit that initializes the global parameters of the scheme.
- An interactive protocol LocalInit which initializes the state of a peer.
- An interactive protocol Update(P) in which the peer P uses its local state and global parameters to compute a new value for its *position* $\rho(P)$, possibly after interacting with other peers.
- An algorithm Distance(ρ_1, ρ_2) which computes an estimated latency between positions ρ_1 and ρ_2.

Peer P runs LocalInit on joining the system, and then at regular time intervals τ, it calls Update(P) to update its position. The most important functional goals for a latency estimation scheme are:

Accuracy. Informally, a scheme is accurate if two nodes that compute positions ρ_1 and ρ_2 have network latency that is close to Distance(ρ_1, ρ_2). The typical measure of a scheme's accuracy used in the literature is the *median relative error* of peer P,

$$\underset{P' \in \mathsf{Peers}}{\text{median}} \frac{|\mathsf{Distance}(\rho(P), \rho(P')) - rtt(P, P')|}{rtt(P, P')} .$$

We note that this measure inherently compares a scheme's accuracy to the "ground truth:" stating that a scheme has median relative error c is stating that on average, 50% of the estimates are within a factor c of the true latency, while 50% are not. Thus, lower values for median relative error equate to better estimates. Ideally, a scheme would achieve relative error of 0 on all estimates. However, it is not hard (logically) to construct a network that has incompressible latencies, so that any scheme to represent positions by strings of length $o(N)$ must be incorrect on at least a constant fraction of the estimates.

Stability. A latency estimation scheme is *stable* if positions computed at time t still provide good accuracy at time steps $t' > t$. This can be computed by computing the distance in coordinates at time t and comparing it to the latency at time t', in the calculation of median relative error.

Efficiency. A latency estimation scheme is not very useful if the bandwidth required to transmit positions exceeds the $O(N^2)$ bandwidth required to simply have all nodes measure pairwise RTTs, and similarly if the distance computations from positions is inefficient. Ideally, the size of positions and the time required to compute distances should be essentially independent of the number of peers.

We note that all of these aspects of a scheme may depend to a large extent on the topology of the underlying network. In Section 5.3, we use a large set of Internet measurements to compare the predictions of Treeple to measured latencies, measure the stability of Treeple positions, and evaluate the size of Treeple positions, along with the average computational load.

Triangle Inequality Violations. We note that several measurement studies [41, 42] have reported that as many as 5% of all node "triangles" (N_1, N_2, N_3) violate the triangle inequality, that is, $rtt(N_1, N_3) > rtt(N_1, N_2) + rtt(N_2, N_3)$; we call these triangles "triangle inequality violations" or TIVs. These occur due to the fact that Internet routing is policy-based, rather than distance-based: each autonomous system chooses among possible routes to a given destination based primarily on the cost it will incur by sending packets along the various routes. Thus any system for estimating latencies that satisfies the triangle inequality – including Euclidian distances as in Vivaldi and GNP, and tree distance as in Treeple – must be inaccurate on at least one pair of nodes in each TIV. However, this does not preclude having acceptable accuracy on the remaining 95% of node pairs; indeed, previous studies have shown that in the absence of attacks, Vivaldi achieves low median and 90th percentile relative errors, while we show in Section 5.3 that the same is true of Treeple.

Security Goal

To motivate our Security definition, we consider an hypothetical ("ideal") system in which we can instantaneously ask any node to measure its RTT to another node. In

this setting, an adversary is unable to alter the RTT between any pair of honest nodes. On the other hand, any measurement in which at least one of the nodes is an adversary can be increased, but if the query includes some challenge value, it cannot be decreased below the actual network latency. Since an adversary can *always* increase apparent RTTs involving an adversarial node by delaying responses, this hypothetical scheme would provide the best security we could hope to provide without complete knowledge of the underlying topology. However, since an explicit goal of Treeple is to have short position strings, we will relax this notion slightly to only consider how the adversary can influence Treeple's latency estimates. We will consider a latency estimation scheme to be secure if an adversary cannot influence the estimated latency between honest nodes, and by deviating from the protocol, can only increase the estimated latency between a pair involving at least one malicious node.

Formally, we define security as follows. Let Π be a latency estimation scheme. Fix a network, a sequence of network conditions, and a set $\mathcal{A}$ of adversarial nodes. We will compare random variables $\mathcal{H}$ and $\mathcal{M}$, where $\mathcal{H}$ is an execution trace of Π in which all peers behave according to Π at all time steps; whereas in the execution trace $\mathcal{M}$ the nodes in $\mathcal{A}$ behave arbitrarily, subject to computational restrictions. Each execution trace consists of the set of all messages sent, *rtt* and *route* measurements taken, and positions $\rho_{i,t}$ computed by a peer. We note that in the adversarial trace, misbehaving nodes are not constrained to use a new position at each time step. We compare the sequence of positions $\mathcal{H}.\rho_{i,t}$ and $\mathcal{M}.\rho_{i,t}$ held by each peer i at each timestep t in the traces $\mathcal{H}$ and $\mathcal{M}$. We say that Π is *secure* if the following properties hold with all but negligible probability:

1. For all times t, for all $i, j \in \mathsf{Peers} \setminus \mathcal{A}$,

$$\mathsf{Distance}(\mathcal{H}.\rho_{i,t}, \mathcal{H}.\rho_{j,t}) = \mathsf{Distance}(\mathcal{M}.\rho_{i,t}, \mathcal{M}.\rho_{j,t}) \ .$$

2. For all times t, for all i, j such that $i \in \mathsf{Peers}$ and $j \in \mathcal{A}$, there exists time $t' \in (t - \tau, t]$ such that $\mathsf{Distance}(\mathcal{M}.\rho_{i,t}, \mathcal{M}.\rho_{j,t}) \geq \mathsf{Distance}(\mathcal{H}.\rho_{i,t}, \mathcal{H}.\rho_{j,t'})$.

Informally, condition 2 relaxes our intuitive notion so an adversary can use positions computed at a different time within the update period, since there is no way to prevent an adversary from choosing when it updates its position.

Relationship between security and accuracy. We note that under our definition, security and accuracy are orthogonal. In particular, a scheme may be accurate but not secure: schemes like Vivaldi, GNP, Big Bang, and so on can produce latency estimates that are within 10%-20% of the actual latency, depending on the evaluation, but fall to trivial attacks. On the other hand, it is trivial to produce a scheme that is secure but not accurate: if we assign each node a distinct position and then predict that the latency between any pair of distinct nodes is 100ms, then the adversary clearly cannot influence latency estimates between any pair of nodes, trivially satisfying our definition. Of course this scheme will be highly inaccurate; the challenge lies in simultaneously achieving security and acceptable accuracy. We note that once a scheme is *both* secure, and accurate with no attackers, then it will continue to accurately estimate pairwise latencies between honest nodes when under attack. This definition therefore allows us to evaluate security and functionality in a modular way: given a proof that a scheme is secure, it is sufficient to evaluate its accuracy with no adversarial nodes.

5.1.3 Failures of Previous Network Coordinate Systems

Network coordinate systems can be categorized into centralized schemes with trusted nodes [3, 2] and decentralized schemes [1, 6]. Centralized network coordinate systems consist of some trusted nodes which communicate with each other to compute their coordinates; other nodes can contact a subset of those trusted nodes to obtain their coordinates. In a decentralized scheme, each node contacts a different set of peers to compute and update its coordinates. Here we briefly demonstrate that (1) *having trusted nodes does not ensure that a scheme meets our security definition.*

Manipulating coordinates in GNP

GNP [2] is a landmark-based network coordinate scheme; the authors do not claim security against attacks but it might seem intuitively appealing that if landmarks' coordinates are digitally signed, and nodes obtain a digitally signed messages from each landmark attesting to RTT measurements, the scheme could be secure. Although this would seem to prevent an adversary from influencing the estimated distance between honest nodes, however, we show that it is possible for an adversary to influence the

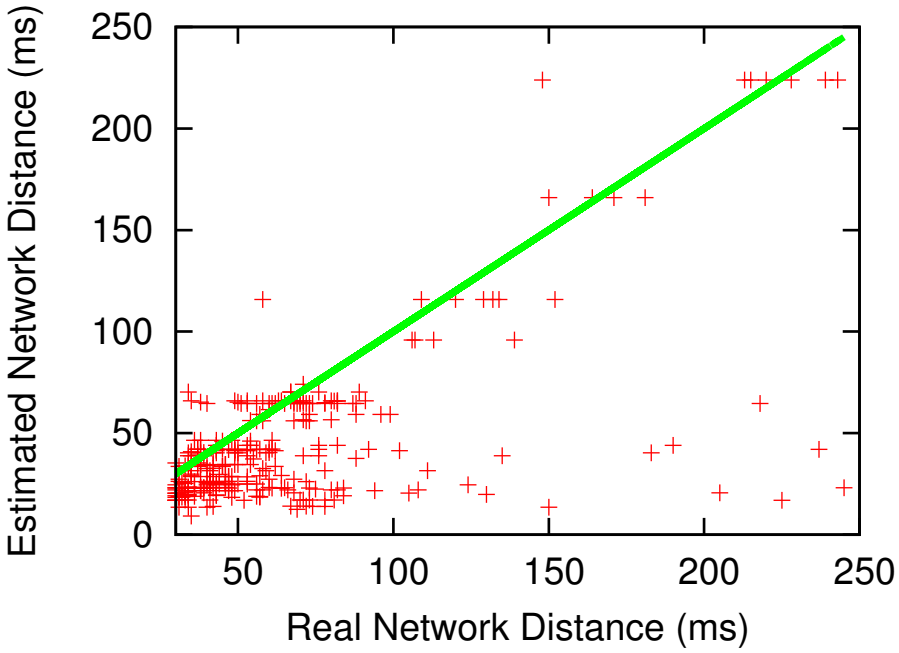

Figure 5.1: Results of the targeted close node attack on GNP
Each point represents one simulated attack, and compares the resulting coordinate distance to the underlying network distance between the attacker and the targeted node.

protocol to artificially decrease the estimated distance between an adversarial node and a targeted honest node.

We implemented a very simple attack that demonstrates the feasibility of this goal. In our attack, the adversarial node A knows the coordinates of the node T it wants to target, so A can compute the distance between T and each landmark node. Then when a given landmark Λ_i measures $rtt(\Lambda_i, A)$, A attempts to make the result as close to $rtt(\Lambda_i, T)$ as possible, subject to the constraint that the adversary cannot cause the measurement to be smaller than the underlying network distance. We repeatedly simulated this process using the code and matrix topology from [61]. For each simulation run, we randomly picked one victim node and one attacker node from a set of 101 nodes with 10 of these nodes as landmarks, subject to the constraint that the victim was not already "close" to the attacker (RTT less than 30ms). Figure 5.1(a) shows that this simple attack is very successful: in nearly all cases, A receives a coordinate that is closer to T than the underlying network distance.

5.2 Description

Suppose that we have a single trusted vantage point and wish to build a secure network latency estimation scheme for the current Internet. A very simple way to incorporate network topology into Treeple positions is to have the trusted node measure the network path to each peer, for example using repeated calls to traceroute, and measure the RTT to each router along the path. The position that the vantage point assigns to each peer would then be the signed path from the trusted node to the peer, including the RTTs to each node along the path. To compute the distance between two peers A and B given their positions, we could find the last "common ancestor" C on each of the paths and then estimate that the distance between the peers is the distance between A and C plus the distance between C and B, since a path of this distance exists between A and B.[1] It is easy to see that while it may be inaccurate, this scheme meets our stated security goal, assuming that malicious peers cannot interfere with the routing infrastructure: in this case the paths from the trusted node to any two honest nodes

[1] Due to the complexities of Internet routing this path is unlikely to be used, but it may still represent a good approximation.

Define Treeple.GlobalInit:
For each trusted $T_i \in \{T_1, \ldots, T_k\}$:
T_i: choose $(vk_i, sk_i) \leftarrow \mathsf{Gen}(\lambda)$.
T_i: **send** (i, vk_i) to $\{T_1, \ldots, T_k\}$.
Output: $\langle vk_1, \ldots, vk_k \rangle$.

Define Treeple.LocalInit(N):
set $\mathsf{pos}_{\mathtt{N}} \leftarrow \mathsf{getPosition}(\mathtt{N}, time())$.

Define Treeple.Udpate(N, t):
if $length(\mathsf{pos}_{\mathtt{N}}) < k$
or $\mathsf{pos}_{\mathtt{N}}$ is stale:
set $\mathsf{pos}_{\mathtt{N}} \leftarrow \mathsf{getPosition}(\mathtt{N}, t)$.

Define Treeple.getPosition(N, t):
N: choose $rid_t \leftarrow_R \{0,1\}^\lambda$. Set $\mathsf{pos}_{\mathtt{N}}.\mathsf{rid} = rid_t$.
Foreach $T_i \in \{T_1, \ldots, T_k\}$ do:
N: choose $rid_i \leftarrow_R \{0,1\}^\lambda$.
N: **send** `pos-request`(rid_t, rid_i, t) to T_i
T_i: **on** `pos-request`(rid_t, rid_i, t) from N:
T_i: set $\mathsf{rt} \leftarrow \langle\rangle$
T_i: $\mathsf{rt}.rid \leftarrow rid_i$.
T_i: $\mathsf{rt}.t \leftarrow t$.
T_i: if $|t - time()| > \tau$: **abort**.
T_i: compute $\langle r_1, \ldots r_\ell \rangle = T_i.route(\mathtt{N})$.
T_i: for each $r_j \in \langle r_1, \ldots, r_\ell = \mathtt{N} \rangle$:
T_i: $\mathsf{rt}.\mathsf{host}_j \leftarrow r_j$.
T_i: $\mathsf{rt}.\mathsf{rtt}_j \leftarrow T_i.rtt(r_j)$
T_i: $\mathsf{rt}.\mathsf{sig} \leftarrow \mathsf{Sign}_{sk_i}(\mathsf{rt}.\mathsf{host}, \mathsf{rt}.\mathsf{rtt}, rid_t, rid_i, t)$.
T_i **send** `route-reply`(rt) to N.
N: **on** `route-reply`(rt) from T_i:
N: if $\mathsf{Verify}_{vk_i}((\mathsf{rt}.\mathsf{host}, \mathsf{rt}.\mathsf{rtt}, rid_i, rid_t, t), \mathsf{sig})$:
N: set $\mathsf{pos}_{\mathtt{N}}.\mathsf{route}_i = \mathsf{rt}_i$.
N: else, set $\mathsf{pos}_{\mathtt{N}}.\mathsf{route}_i = \perp$.
Output: $\mathsf{pos}_{\mathtt{N}}$.

Define Treeple.Distance($\mathsf{pos}_A, \mathsf{pos}_B$):
set $dist \leftarrow \infty$.
if $(|\mathsf{pos}_A.t - \mathsf{pos}_B.t| \leq \tau)$:
For each $T_i \in \{T_1, \ldots, T_k\}$ do:
if $(\mathsf{verify}(i, \mathsf{pos}_A)$ and $\mathsf{verify}(i, \mathsf{pos}_B))$:
set $lca \leftarrow \mathsf{find\text{-}lca}(\mathsf{pos}_A.\mathsf{route}_i, \mathsf{pos}_B.\mathsf{route}_i)$.
set $d_i \leftarrow (\mathsf{pos}_A.\mathsf{route}_i.\mathsf{rtt}_A - \mathsf{pos}_A.\mathsf{route}_i.\mathsf{rtt}_{lca}) +$
$(\mathsf{pos}_B.\mathsf{route}_i.\mathsf{rtt}_B - \mathsf{pos}_B.\mathsf{route}_i.\mathsf{rtt}_{lca})$.
Update $dist \leftarrow \min(dist, d_i)$.
Output: $dist$

Define $\mathsf{find\text{-}lca}(rt_1, rt_2)$:
Output: $max\{j | rt_1.\mathsf{host}_j = rt_2.\mathsf{host}_j\}$.

Define $\mathsf{verify}(i, \rho)$:
if $\rho.\mathsf{route}_i = \perp$: **output** False
else:
set $m_i \leftarrow \langle \rho.\mathsf{route}_i.\mathsf{host}, \rho.\mathsf{route}_i.\mathsf{rtt}, \rho.\mathsf{route}_i.\mathsf{rid}, \rho.\mathsf{rid}, \rho.t \rangle$.
output $\mathsf{Verify}_{vk_i}(m_i, \rho.\mathsf{route}_i.\mathsf{sig})$.

Figure 5.2: Treeple algorithms

Assuming the existence of a digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ and a pre-selected set of trusted vantage points $\{T_1, \ldots, T_k\}$. Here a *position* consists of a time t, a global random identifer *rid*, plus an indexed array route, where each entry route_i is either $\perp$ or a record consisting of indexed arrays host and rtt along with a signature field and a local identifier.

would not involve malicious nodes, and only the final hops to malicious nodes could be impacted, by delaying responses to the trusted node's traceroute request.

An alternate view of this system is that the trusted node is computing the tree of shortest paths between itself and other peers, and "embedding" the network into this tree metric. It is clear that for some pairs of nodes the tree distance could be larger than the actual network distance, because many network links will not be included in the trusted node's shortest path tree.

To address this, we choose k topologically distinct vantage points to repeat this process: a peer's coordinate becomes an ordered k-tuple of signed routes (one from each trusted vantage points), and the distance between A and B becomes the minimum of the distances computed from each of the k trees. Again, it is easy to see that the security of the scheme holds for this variant: no adversarial node can interfere with the route and RTT measurements involved in computing honest nodes' positions; so all honest node coordinates will be the same regardless of adversarial behavior. And since adversarial nodes can only increase RTT measurements they cannot appear closer to other nodes by deviating from the protocol. Intuitively, adding the extra vantage points increases the probability of discovering the network links used by the actual route between two nodes, thus improving the accuracy of latency estimates.

5.2.1 Complete Description

For completeness, pseudocode for the component algorithms of Treeple is shown in Figure 5.2. Treeple assumes the existence of a signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ that is existentially unforgeable against chosen message attack: any efficient program given access to a verification key vk and a signing oracle for the corresponding signing key sk cannot produce a correct (message, signature) pair with a message that was not a previous signing oracle query, except with negligible probability in the security parameter. Additionally we assume that each end-host `h` has access to two functions: $\texttt{h}.route(\texttt{g})$ returns a list of routers along the path from `h` to `g`; and $\texttt{h}.rtt(\texttt{g})$ returns the round-trip time between `h` and `g`. For clarity, the "system parameters" generated in GlobalInit are passed as implicit arguments to other procedures. Finally, we assume a fixed set of k trusted vantage points whose addresses are included in the global parameters, but are not chosen by the protocols. These nodes serve as a "root of trust" in the scheme. We

briefly discuss algorithms for choosing from among several possible vantage points in Section 5.3.

5.2.2 Security

Theorem 1. *Assuming the set* $\{T_1, \ldots, T_k\}$ *of vantage points are honest,* Treeple *is a secure network latency estimation scheme.*

Proof. Fix a network graph and condition sequence χ, along with a set of adversarial peers $\mathcal{A}$ such that $\mathcal{A} \cap \{T_1, \ldots, T_k\} = \emptyset$. Suppose that there exists a pair of peers (A, B) that violate the security condition of section 5.1; we will show that, except with negligible probability, the existence of this pair must imply a signature forgery. We let α denote the total number of messages sent by adversarial nodes in the adversarial trace.

First, notice that for any honest peer h, at any time t, $\mathsf{pos}_{\mathtt{h}}$ must be identical in both the adversarial and non-adversarial execution traces, except with negligible probability. This is because the "request identifier" rid_i generated in getPosition(h), combined with signature verification on received route messages, ensures that h only updates its coordinates when it receives authentic responses to its own requests, from trusted nodes. Since calls to getPosition(h) and are only initiated by h and our adversarial model excludes dropping and interception of honest messages, it follows that in any pair of execution traces with identical network conditions, an honest node will have equivalent positions.

Specifically, an honest node only changes its position when it receives a message that is signed and contains its most recent, randomly chosen identifier. Since adversarial nodes do not see the requests generated by honest nodes, the probability of correctly guessing a request identifier in α attempts is at most $\frac{\ell n \alpha}{2^\lambda}$ (where ℓ is the length of the trace, n is the number of peers, and λ is the length of request identifiers). Given that the adversary does not correctly guess request identifiers, it can only cause an honest node to accept a position that is different than the non-adversarial trace by generating a response message route′ that is apparently signed by one of the trusted vantage points. Notice that given the sequence of network conditions and a signing oracle, it is easy to generate all the messages a set of adversarial nodes would see in the adversarial execution (because the honest nodes follow the protocol), and thus it follows that this route′ and its signature would constitute a forgery. If we denote the (negligible) probability of a

forgery with $n\ell + \alpha$ signature queries by ϵ, then the probability of this event is at most $k\epsilon$ by the standard reduction that guesses which trusted party the adversary will forge against.

Thus, if the pair (A, B) is honest, the presence of adversaries is irrelevant: $\mathsf{Distance}(\mathsf{pos}_A, \mathsf{pos}_B)$ will be the same in both traces. On the other hand, consider the variable pos_A assigned to an adversarial node A at time t. Since the adversarial node initiates the same requests in both traces, it can only manipulate its position by either manipulating the measurement of $T_i.rtt(A)$ for some T_i (by assumption the measurement of $route(T_i, A)$ and $T_i.rtt(x)$ for $x \neq A$ are not vulnerable to manipulation) or by substituting a different value for some route_i. By assumption on the rtt functionality, the first type of manipulation will only inflate the distance between A and its "last hop", which will inflate the distance to other nodes (uniformly). So the only remaining option to violate the security condition is to replace some route_i. The requirement that all route messages in a position have the same request identifier prevents "mix-and-match" substitution of routes between requests. Dropping any route message that does not give the minimal distance to a particular position will not affect the distance calculation, while dropping the minimal distance route will only increase the distance. Finally, producing a signed route_i with different rtt or host entries would constitute a forgery, and thus an adversary would again successfully produce this forgery with probability at most $k\epsilon$. □

5.3 Evaluation

5.3.1 Experimental Setup

To evaluate our approach, we used the iPlane [62] dataset. This data set contains the results of periodic traceroutes from 250 Planetlab [46] nodes to all other Planetlab nodes and thousands of other IP addresses. We note that the iPlane dataset is "live"; every day, each of the PlanetLab nodes performs multiple traceroutes to over 100,000 IP addresses and publishes the results. We downloaded the traceroute datasets from Dec 1st, 2009 to Dec 22nd, 2009. The dataset presented 250 possible trusted nodes to choose from; each trusted node contacted more than $130,000$ IP addresses on average, and each tree constructed from the traceroutes contained on average $200,000$ unique nodes (including intermediate nodes). In constructing the paths from trusted nodes to

peers, we always used the minimum RTT measured at each hop, and when repeated traceroutes resulted in different routes, we selected the shorter route.

To determine the accuracy of a given set T of vantage points, we considered all pairs of nodes (A, B) such that (i) all vantage points in the set had successfully completed a traceroute to both A and B, allowing us to compute positions pos_A, pos_B; and (ii) we had measured the RTT between A and B. For each such pair, we computed the relative error,

$$\frac{|\mathsf{Distance}(\mathsf{pos}_A, \mathsf{pos}_B) - RTT_{A,B}|}{RTT_{A,B}},$$

to measure the accuracy in estimating the latency between A and B using T; lower relative error indicates a better estimate of the latency. Because the iPlane measurement aparatus does not retry traceroutes that fail, the size of the evaluation set will vary across sets T; in all cases the size was over 200,000. In order to compare between these evaluation sets, we used the median relative error for each. Additionally, for our "best choice" of 20 vantage points T, we also measured the size of the coordinates pos_A and pos_B and the number of node comparisons required to estimate the distance.

5.3.2 Selecting a set of vantage points

Determining the accuracy of Treeple is not as straightforward as for a network coordinate system. The relative error is still used to determine whether an estimated network distance is accurate – the lower the error, the more accurate the estimation. However, using the iPlane dataset, there are 250 possible vantage points. It is clear that, in general, different vantage point sets will produce different accuracy. An important question, both for evaluation and eventual deployment, is how to select a good set of vantage points.

Determining the "best possible" single vantage point is relatively straightforward – compute the median relative error for each pair of end hosts' network distance estimation, for each of the 250 vantage points, and the one with the lowest median relative error is the most accurate. When $k = 2$, we could pick the very best combination of **any** two vantage points such that the combination would result in the lowest median relative error among all the possible combinations (total of $250 \times 249 = 62,250$ combinations). However, this approach is not scalable, as when $k = 3$, there are a possible

$250 \times 249 \times 248 = 15,438,000$ combinations to choose from. Although this approach provides the very best combination of vantage points to produce the lowest median relative error, it does not scale past $k = 3$, and worse, would not give confidence in the future performance of this set, due to overfitting.

In this thesis, we selected vantage sets of different sizes k using a *greedy sampling* algorithm, which works as follows. First, we chose at random a set S of $1,000$ pairs (A, B) between which we had measured latencies. We start by picking the best vantage point T_1 for the pairs in S among the 250 possible choices. Then we pick the best second vantage point T_2 that combined with T_1 would produce the lowest median relative error on S, and so on until k vantage points have been chosen. Using this algorithm to select k of n possible vantage points requires $O(nk)$ steps, as compared to n^k for the previous approach. Furthermore, because we evaluate only on a sample, we avoid overfitting vantage points to our test set.

Although it is clear that the "greedy" approach is scalable for arbitrary k, it remains to be seen whether it produces a good result, that is, whether it can pick vantage points that can accurately estimate network distances between any pair of nodes. Figure 5.3 shows the CDF for the relative error (amongst all pairs) using greedy sampling to select vantage sets of size $k \in \{1, 2, 3\}$. For comparison, the figure also includes the CDF of relative error for the k-node vantage sets that achieve the best median relative error on the iPlane data set from December 1, 2009. The labels $t = 2$ and $t = 3$ represent the optimal result, that is, picking the very best 2 and 3 vantage points respectively. The median relative error is similar for both algorithms for various k. The primary difference in accuracy can be seen at the $90th$ percentiles. The $90th$ percentile relative error for the optimal set for $k = 2$ is 1.7 compared to 2.3 for the greedy approach – a difference of 35.2%. The $90th$ percentile relative error for $k = 3$ for the optimal set is 1.45, compared to 2.15 for the greedy algorithm – a difference of 48.3%.

From Figure 5.3, it is clear that the greedy algorithm is nearly as accurate as the optimal algorithm. In the remainder of this section, we use results from the greedy sampling algorithm.

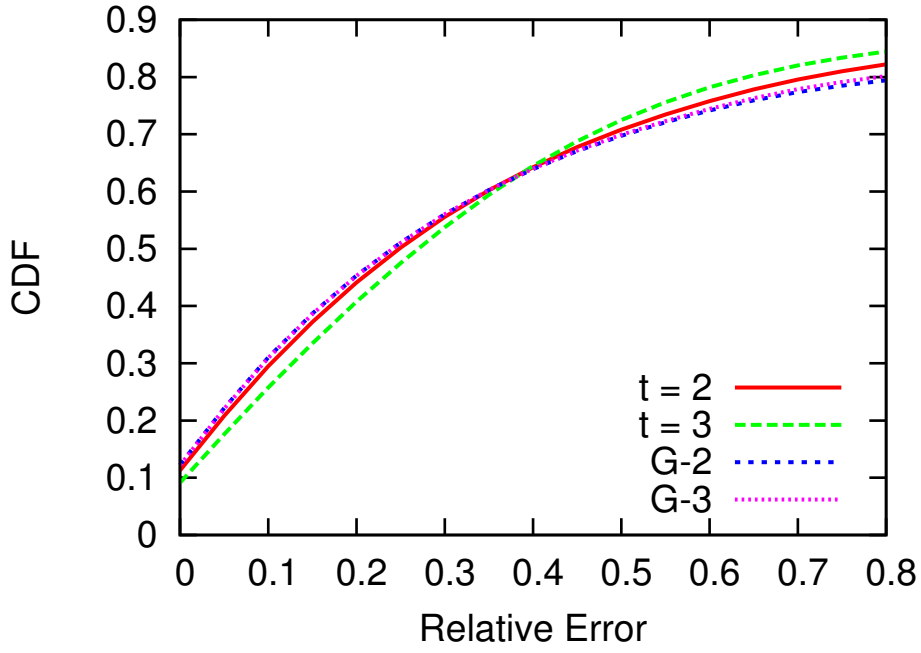

Figure 5.3: The CDF for the relative error of Treeple
for the estimations for the "best" ($t = 2, 3$) and "greedy" sets, with varying k.

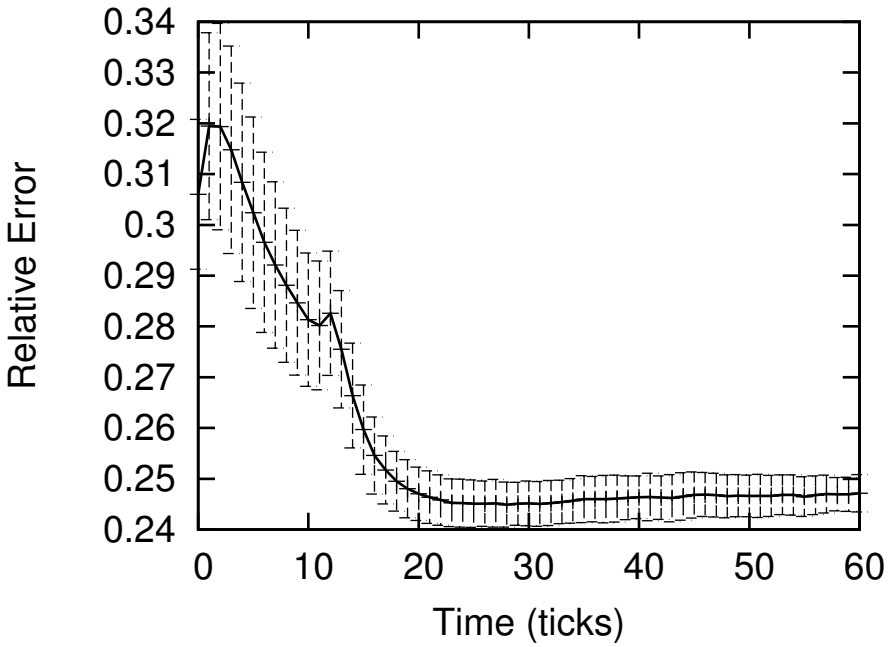

Figure 5.4: The Vivaldi simulation
100 runs for our 250x250 dataset with error bars representing the standard deviation.

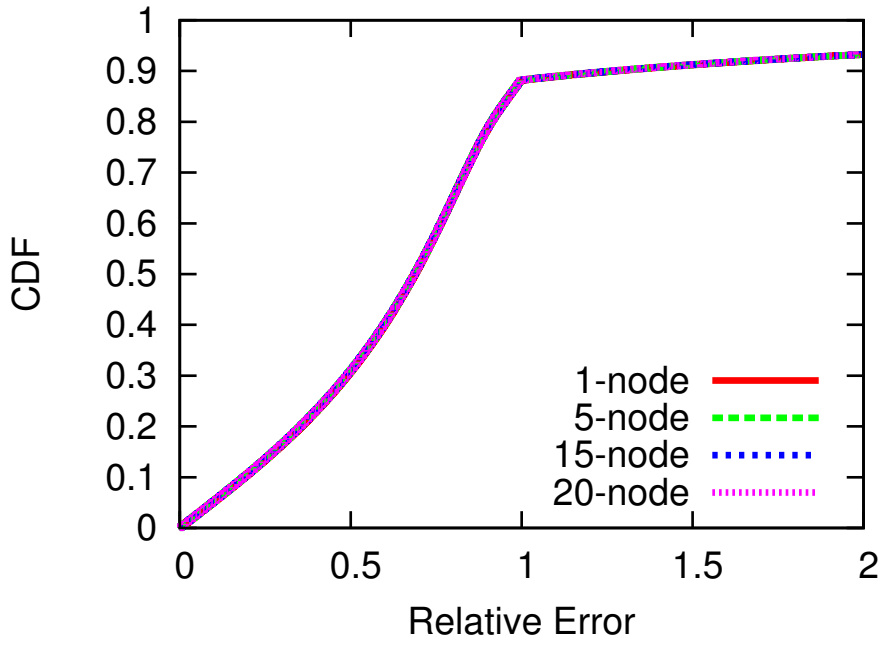

Figure 5.5: Trivial secure scheme: "star topology"
CDF of relative error for the "star topology" scheme.

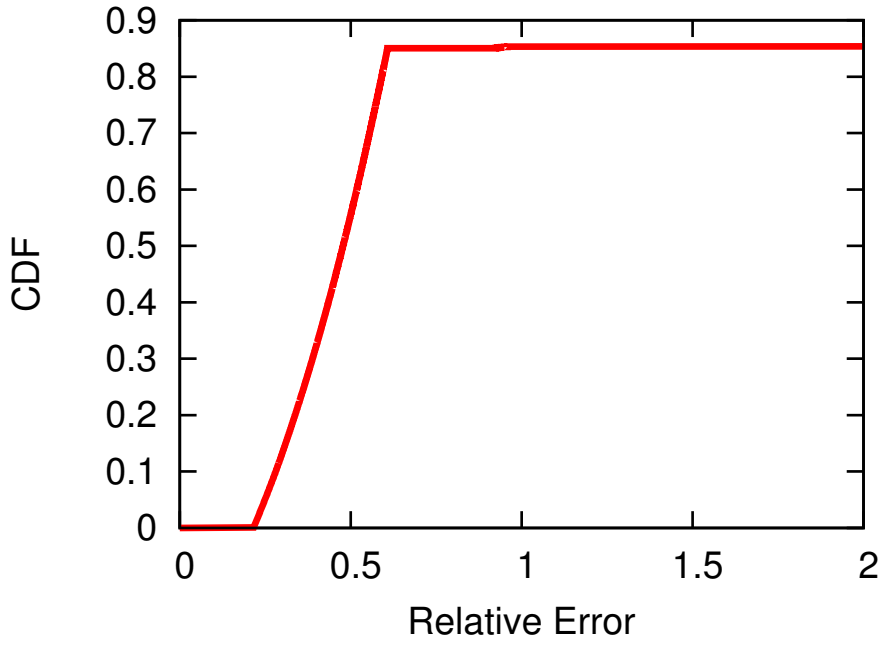

Figure 5.6: Trivial secure scheme: "always predict median RTT"
CDF of relative error for the "always predict median RTT" scheme.

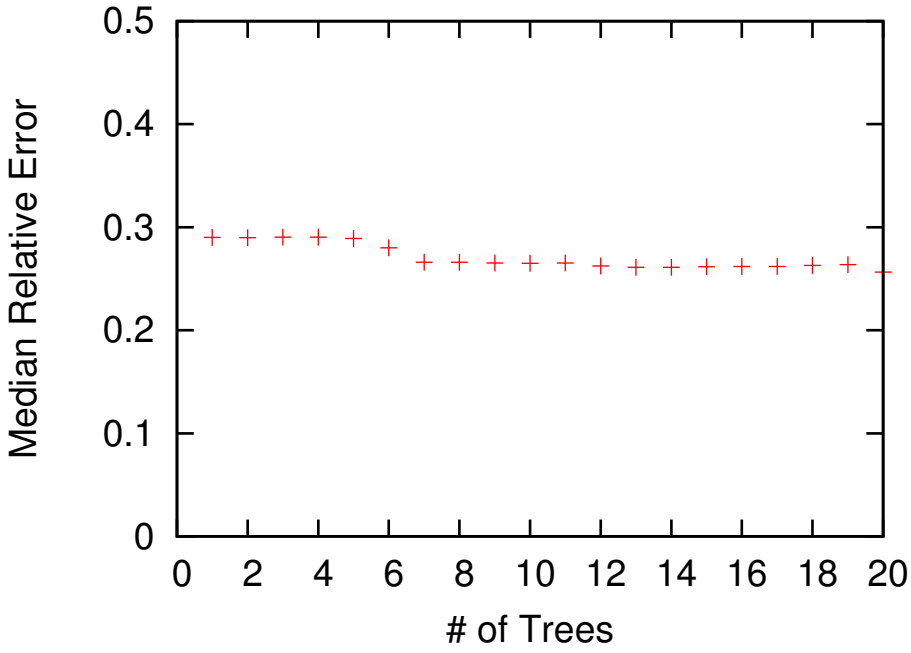

Figure 5.7: Median relative error when varying k for the greedy approach

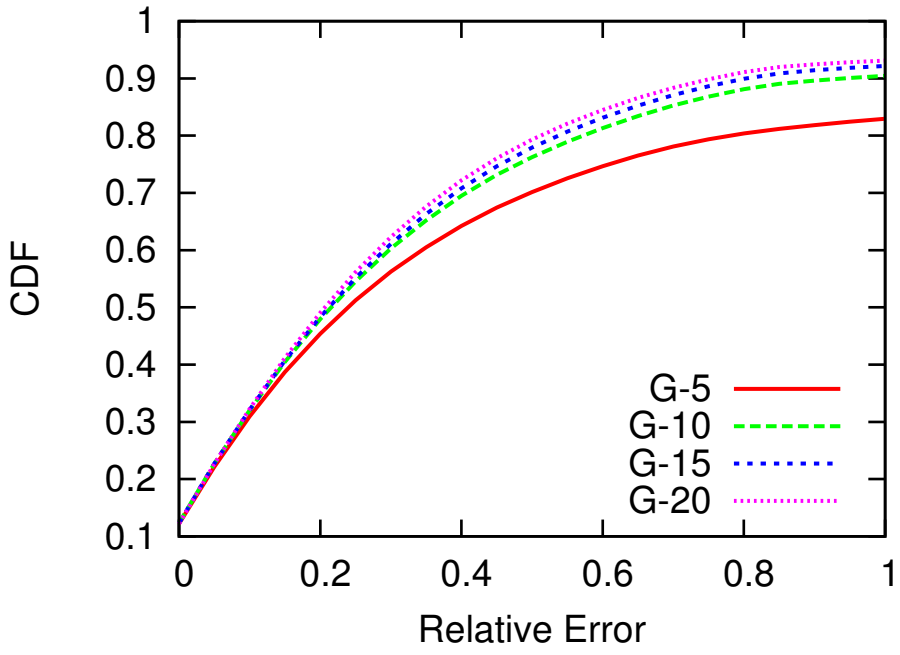

Figure 5.8: CDF for the relative error for $k = 5, 10, 15, 20$ when using the greedy approach

5.3.3 Baselines: Vivaldi, Star Topology, and Median

To establish a comparative baseline for the accuracy of Treeple, we evaluated three "basic" schemes on the same dataset used to evaluate the performance of Treeple.

Vivaldi [1] is a popular but insecure network coordinate system that is implemented in the Vuze file-sharing network [7] and used as the basis for computing coordinates in the various "secure" network coordinate systems [26, 24, 25]. We evaluate Vivaldi's performance on our dataset via simulation. Figure 5.4 shows the median relative error of all the nodes in our Vivaldi simulation over time. To obtain a non-sparse matrix of RTTs, we only used the 250 trusted nodes as source and destination, thus obtained a matrix of 250×250 entries. The median relative error for Vivaldi was 25% at the end of the experiment. The figure shows the error bars representing the standard deviation and the mean over 100 runs.

As further point of comparison, we evaluated two simpler, provably secure schemes. In the first scheme, which we refer to as the "star topology," we assume a set of trusted vantage points (as in Treeple) but assign coordinates to peer A based solely on the (signed) distance rtt between the vantage point and A. When peer A wants to estimate its distance to peer B using vantage point C, it calculates $rtt(C, A) + rtt(C, B)$. The scheme extends to multiple vantage points in the same way as Treeple. The results are shown in Figure 5.5: for our data set, regardless of the number of vantage points used, the star topology yields a median relative error of 0.68. The second trivial scheme we evaluate is the "median" scheme, in which the predicted distance for every pair of distinct nodes is simply the median observed RTT for the data set. The scheme is trivially secure, but as shown in Figure 5.6, also provides poor accuracy, acheiving a median relative error of 0.5.

5.3.4 Accuracy

Figure 5.7 shows the median relative error for vantage sets of varying size k, computed as in the previous section. As k is increased, the median relative error decreases from 0.29 to 0.26, which is comparable to the median relative error obtained when using the Vivaldi network coordinate system. Figure 5.8 shows the CDF for the relative error of the estimations for varying number of vantage points. The different lines indicate the

number of vantage points used. Using fewer than 5 vantage points produces the same accuracy. A gain in accuracy is obtained when $k > 6$. There is a significant difference in the $90th$ percentile between $k = 5$ and $k = 10$. Increasing $k > 10$ provides minimal gain in accuracy (either median relative error or $90th$ percentile relative error). These two figures show that increasing the number of vantage points used for estimations also increases the accuracy of the system. From our experiments, having only 20 trusted nodes perform network measurements is enough to accurately estimate the network distance between two nodes.

5.3.5 Stability

If Treeple's accuracy depends on frequent updates to a node's position, then the vantage points may become a central point of failure, since they would need to be constantly available. Thus it is important to know how the accuracy of Treeple positions changes over time. We used the best 20 vantage points identified by the greedy method on 12/01/2009 to estimate the network distances for end hosts from 12/01/2009 to 12/21/2009 (three weeks). Figure 5.9 shows the median and $90th$ percentile relative errors when using the 12/01/2009 positions to estimate the network distances for other days. That figure shows that Treeple's accuracy remains nearly constant over time. Thus, a side effect of using Treeple is that frequent network measurements are not needed – the same positions can be used for long time periods τ. The spike in relative error on the $20th$ day is attributed to some possible faulty network measurements from the iPlane dataset.

5.3.6 Overhead

In order to show that Treeple imposes acceptable communication and computational costs, we measure these quantities across all nodes in our data set using the "greedy 20" set of vantage points. Figure 5.12 shows the Treeple positions of nodes X and Y. In this example, A is the vantage point, and B, C', and C'' are the routers on the Internet. The "size" of each position is the total number of routers along the 20 paths from each T_i to X or Y. Computing the distance under a given trusted node T_i is straightforward: we start at the first node of each Treeple position. They should be the same since it would

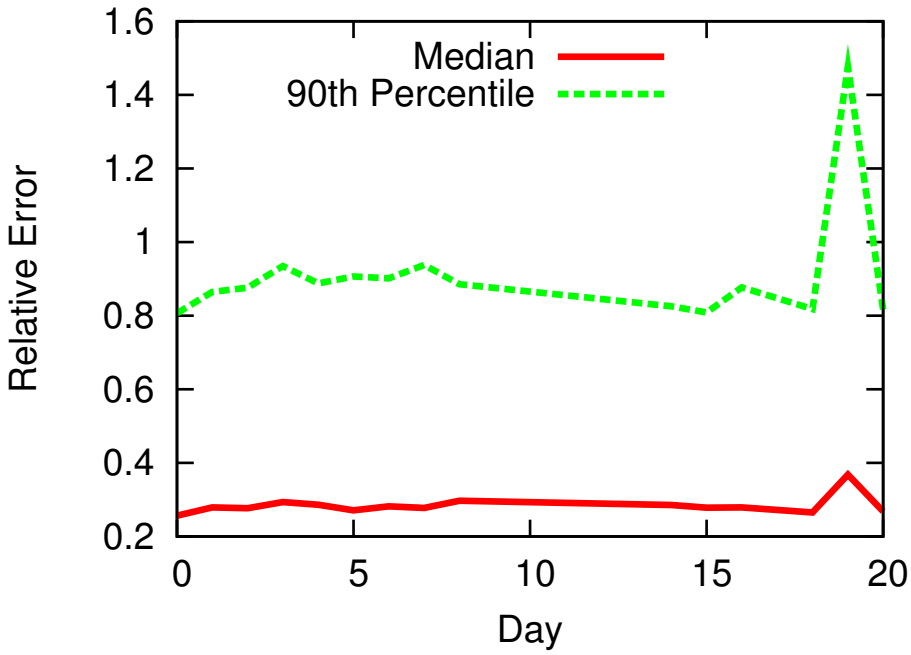

Figure 5.9: Median and 90th percentile relative error showing stability
Using 12/01/09 coordinates for estimation measurements through 12/21/09, by day.

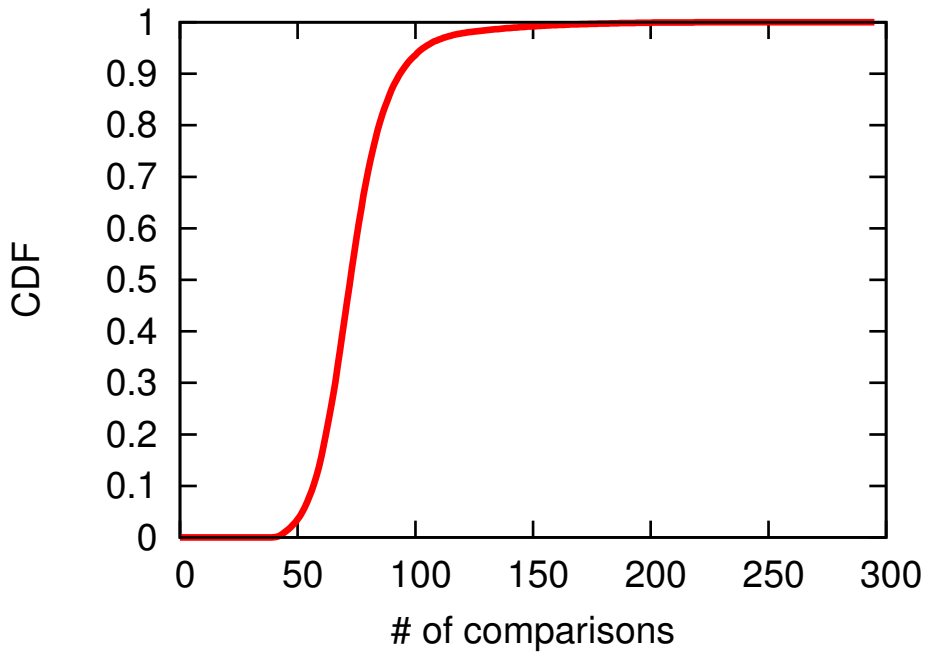

Figure 5.10: CDF for the number of comparisons required to estimate the network latency between two nodes.

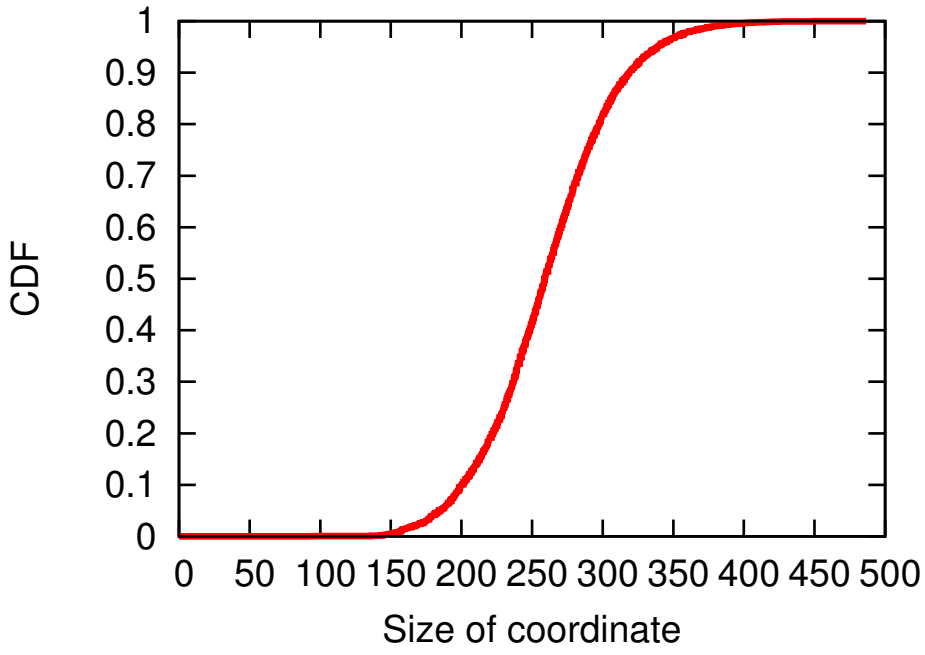

Figure 5.11: CDF for the total number of nodes in each assigned coordinate.

be the trusted node. From that first node, we repeatedly advance to the next nodes until the two corresponding nodes in X and Y's positions are different. In the example shown, node C' is different from node C''. The estimated distance between nodes X and Y can then be calculated as $(rtt_{A,X} - rtt_{A,B}) + (rtt_{A,Y} - rtt_{A,B})$. To compute the distance in this case, we had to compare three pairs of nodes.

Figure 5.10 shows the CDF for the number of comparisons required to compute a network latency estimate among all eligible pairs. The median number of comparisons is 73, with 20 coordinates, this means that each computation requires 3.65 comparisons on average. Figure 5.11 shows the CDF for the total number of nodes for the 20 coordinates of each end-host. The median total coordinate size for each end-host is 260. Each coordinate then contains $260/20 = 13$ hops. Each hop consists of an IP address (32 bits) and a RTT in ms (12 bits). Each hop size is then 44 bits. Each node's total coordinate size is then $44 \times 260 = 11,440$ bits or roughly 1.4 KB.

We note that there is a substantial opportunity for reduction of coordinate size using compression. In particular, it is not necessary to label the individual hops in a coordinate's path with globally unique names: as long as it is possible to determine the position along the path at which two coordinates diverge, the distance can be computed. By examining the distribution on node degrees in the trees within our trusted vantage points, we found that on average each next hop identifier can be encoded with only 2 bits. Similarly, it is not necessary to encode the full RTT between the vantage point and each hop; using difference encoding on the RTTs, we found that on average each hop's RTT can be encoded with 6 bits. These techniques can reduce the size of a coordinate to 260 bytes.

Although this reduced coordinate size is still substantially larger than that of Vivaldi, Treeple requires only one single interaction to compute a peer's coordinate, which can then be used for weeks. On the other hand, Vivaldi requires the constant exchange of coordinates. Thus, over any reasonable period of time, Treeple's bandwidth overhead is smaller than that of Vivaldi.

It can be argued that the cost of the vantage points to perform traceroutes to other nodes on the Internet is very high. However, the vantage points only need to performed this measurement once every few weeks, as we showed in Section 5.3.5. We also note that iPlane is performing these measurements on a daily basis on the PlanetLab nodes.

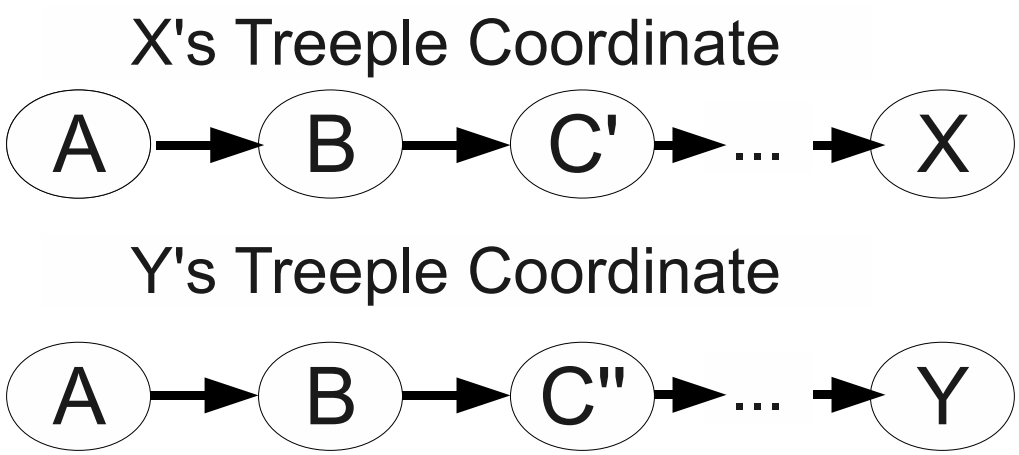

Figure 5.12: An example showing the positions of two end hosts X and Y

5.3.7 Summary

On the large, real-world iPlane dataset, Treeple performs essentially as well as the Vivaldi network coordinate system. The median relative error when using 20 trusted nodes (note that GNP, a centralized network coordinate system, uses 20 trusted landmarks) is 0.26 for our system and 0.25 for Vivaldi. Thus, our system's latency estimation performs roughly as well as that of previous network coordinate system, while being the first system to provide provable security, meeting our goal of simultaneously providing security and accuracy. As expected, using more vantage points improves accuracy, at the cost of relying on more trusted nodes. The communication and processing overheads for Treeple are also relatively small. Finally, Treeple positions can be used to accurately predict network distances over long periods of time, including the full 21-day period of data used in our evaluation.

5.4 Other Related Work

We note that while there have been other systems published [43, 63, 10, 64] that do not use network coordinates and find close nodes, none of these systems can estimate the distance between arbitrary nodes. Several other projects have produced services that allow one peer to estimate its latency to another. IDMaps [43] was one of the first systems to estimate the network latency between two hosts. In IDMaps, hosts perform measurements to the central nodes called *tracers*, and the hosts can then approximate their latency to other hosts without having to contact those hosts directly. IDMaps cannot be used by a third party to predict the latency between pairs of hosts.

Sequoia [65] approximately represents Internet latencies and bandwidth as a tree metric. The authors showed that, for a small data set, using this metric allows accurate prediction of latency as well as bandwidth, with latency prediction comparable to Vivaldi. However, Sequioa relies on a complete view of the reconstructed tree metric; every node must know the complete tree, and in order to be assigned a position in the tree a node must participate in the protocol.

iPlane [62] and iPlane Nano [66] attempt to map the whole Internet to produce an atlas of link-to-link latency, bandwidth, and loss rate. In iPlane, a central database server uses the atlas to respond to path queries from Peers. In iPlane Nano, the atlas is

compressed to about 7MB and distributed to end hosts; end hosts then download about 1MB of deltas produced each day. Hosts use this atlas to compute predicted latencies using shortest-path algorithms. Thus the computational and bandwidth overhead associated with using iPlane Nano are quite high.

We note that all of these systems essentially construct a "global map" of the peer topology by relying on the peers to report pairwise latency and traceroute measurements. None of these systems considers security in any way; essentially the measurements taken by all peers when constructing the maps are trusted. In contrast, Treeple takes security as a starting point and produces "local" positions for each peer, while trusting only a small set of end-hosts.

5.5 Discussion & Summary

Traceroute Complications

It is well-known that several issues such as multiple interfaces, load balancers, multi-protocol label switching, and non-responding hosts can interfere with the accuracy of paths returned by traceroute; in our evaluation, we did not use any special techniques to deal with these issues. We note that none of these issues presents a *security* challenge for Treeple, but that resolving them could well improve the accuracy of Treeple: for example, consistently resolving aliases would place the common ancestor later in a path and thus decrease the estimated distance between two nodes. Thus the performance evaluation in Section 5.3 can be seen as conservative in this sense. We naturally expect such measures to improve the accuracy of Treeple.

Vantage Point Migration

Although we treat vantage points as fixed in our theoretical treatment of Treeple, we point out that in practice, the set of vantage points $\{T_1, \ldots, T_k\}$ is amenable to standard mechanisms used to manage migration of trusted servers for services such as DNS, DHTs, and Tor: it is straightforward to implement the Distance function so that it is robust to additional or missing vantage points – identify vantage points by verification key, and compute distances using only the positions computed by vantage points in common – and executables can eventually phase out support for discarded vantage points

when sufficiently old versions are not supported, while peers running older executables will continue to function with slightly reduced accuracy.

Router Compromise

As we have discussed in section 5.1, under the current Internet architecture compromise of even a single BGP router is sufficient to render meaningless the measurements or estimates of any latency estimation scheme. Thus: *on the current Internet, network coordinate systems (and schemes based on them [11, 12, 13, 9, 10, 15, 18]) are insecure against compromised routers.* However, since router compromises are possible in practice, it would be desirable to consider whether, in some future network that secures path discovery and does not use policy-based routing, it is possible to reason about the security Treeple (or any other latency estimation scheme) can offer against router compromise.

To be concrete, we imagine a network (not the current Internet) in which a corrupted router can only intercept or delay traffic between a small fraction of the pairs of peers. In this case, it might be possible to use the trusted vantage points in Treeple to build a simple reputation system for router edges: if a particular edge exhibits variable behavior, as measured by a vantage point, then paths containing that edge could be marked as untrustworthy. As long as some paths remain, the influence of these adversarial edges would be limited. Similarly, individual nodes can maintain a "local" reputation which calculates whether a given path consistently leads to poor performance and if so, drop the path from their coordinates (replacing it with $\perp$). Of course, the exact nature of the security guarantees these mechanisms can provide would clearly depend on the nature of the routing protocol in this hypothesized future network architecture.

Summary

We propose Treeple, a provably secure network distance estimation service, where the estimated network distance differs from the real network distance by 26%. The accuracy of the system is comparable to using a network coordinate system, where the median relative error was 25%. In addition, Treeple is provably secure, whereas all previously proposed schemes are vulnerable to attacks within the Treeple threat model. Moreover,

because Treeple positions are extremely stable, they can be maintained with very low overhead compared to traditional network coordinate schemes.

Chapter 6

KoNKS

We describe a completely decentralized network coordinate system, KoNKS, which is secure under our stated security model (see Section 6.1 for more details about our security model). KoNKS – consensus-style network coordinate system – modifies the objective function that each peer follows to update its coordinates. In current network coordinate systems, a peer's goal is to minimize the sum of the prediction errors for all of its neighbors. In contrast, using KoNKS, a peer's goal is to minimize the number of neighbors whose individual relative error is unacceptable – KoNKS puts an upper bound on each neighbor's relative error. The relative error determines how accurate the coordinate system is, thus when there are no attackers, minimizing the sum of errors should lead to more accurate distance predictions. However, minimizing the sum of prediction errors allows each neighbor to have a significant influence on the position of its peers. This is one of the reasons why the frog-boiling attack works. For example, a malicious neighbor could craft a lie so that its coordinate distance to the peer is much smaller than the measured network distance. In response, the peer would make a significant change to its coordinate because that update seemed to give the minimum total prediction error, even though it adds significant prediction error to every other neighbor.

This example cannot happen in KoNKS because every neighbor of a peer has the same amount of influence on that peer. In a way, KoNKS peers achieve consensus among their neighbors: each neighbor "votes" for a region in which the peer should reside, and the network position with the most "votes" from the neighbors is the one that KoNKS

chooses. A malicious neighbor can still choose its reported coordinates and add delay to its RTT, but the push that lie has on the peer is limited, as the latter will have to satisfy its other neighbors as well. At every update, the peer takes into consideration each of its neighbors' relative error. We argue that KoNKS is secure because 1) a malicious node's influence on the coordinate distance between two honest nodes is limited, and 2) a malicious node cannot appear closer than it actually is because its relative error will be higher than the imposed threshold. A more detailed description of the design of KoNKS can be found in Section 6.2.

We show in Section 6.3 that KoNKS is as accurate as Vivaldi [1], one of the most popular decentralized network coordinate system (Vivaldi is implemented in Vuze [7] and is the basis for all the "secure" network coordinate systems [24, 27, 26, 25]), and is secure against all the current attacks, including the network-partition frog-boiling attack. More specifically, KoNKS puts an upper bound on the amount of influence an adversary can have on the honest nodes. For example, 10% of attackers can partition a network using KoNKS only so much before their lies do not have any effect anymore because they are outside of the threshold, or the other honest neighbors' influence equals the malicious neighbors' influence. KoNKS with no attack can achieve a median relative error as low as 12%, which is comparable to Vivaldi's median relative error of 10%. Under the "random" attack, even with 30% of malicious nodes, KoNKS' median relative error is increased to only 20%; Veracity's median relative error is increased to 32%; Vivaldi's median relative error is increased beyond 150%. Moreover, KoNKS incurs a very low overhead, similar to Vivaldi as coordinates can be piggybacked on top of application messages. The processing overhead of each node updating its coordinates is also very small.

6.1 Security Model

We first present our threat model, then define the security goals that we will use to evaluate KoNKS.

The network consists of n nodes, of which $0 \leq m < \frac{n}{4}$ are malicious. The number of malicious peers needs to be limited to at most 1/4 of the network so that satisfying a majority of the nodes gaurantees that the median node is honest. Clearly, a botnet

or Sybil [14] could invalidate this assumption, but dealing with such attacks is beyond the scope of this thesis. We note that an adversary controlling a botnet can likely cause more serious problems for an overlay than disrupting its network coordinate system. We allow for a powerful adversary that can compromise any node in the network (up to 1/4 of the network), but cannot adaptively compromise nodes, so that if a peer chooses its neighbors randomly, with high probability only 1/4 of them will be malicious. Moreover, we allow the adversary to have global knowledge of the network, that is, it knows every peer's coordinate, real network distance to other peers, and neighbors. The adversary can also inflate its network distance to other peers by delaying its responses, and can report any coordinate it chooses. However, we do limit the adversary's power in the following sense: we assume that a malicious peer cannot decrease its real network distance to another peer. This can be achieved by using an unpredictable unique nonce in every message. Thus, a malicious peer cannot reply to a message faster than the network conditions allow.

We next describe how a network coordinate system functions. Every node computes its own k-dimensional Euclidean coordinates (for simplicity we assume Euclidean space). The real network distance between any two nodes A and B is denoted as $\mathsf{rtt}(\mathsf{A}, \mathsf{B})$, and the coordinate distance between any two nodes A and B is denoted by $\mathsf{distance}(\mathsf{A}, \mathsf{B})$. Embedding a higher-dimensional space (pairwise network distances) into a low-dimensional Euclidean space (for example 5-dimensional Euclidean coordinates) inherently produces *errors*, that is, it is in general not possible to obtain a perfect embedding from a high-dimensional space to a low-dimensional space. A good embedding minimizes the error produced.

We also assume that each node knows of some other nodes in the network through some bootstrapping mechanism. Each node will periodically contact a randomly chosen node in the network (which can be piggybacked on top of regular application messages), and receive that node's coordinates and the corresponding latency. Based on the information received, the querying node will then update its own coordinate, in an attempt to minimize the prediction error. The prediction error between two peers A and B is defined as $|\mathsf{distance}(\mathsf{A}, \mathsf{B}) - \mathsf{rtt}(\mathsf{A}, \mathsf{B})|$. We also define the relative error as

$$\frac{|\mathsf{distance}(\mathsf{A}, \mathsf{B}) - \mathsf{rtt}(\mathsf{A}, \mathsf{B})|}{\mathsf{rtt}(\mathsf{A}, \mathsf{B})}$$

A low relative error implies a low prediction error, but the converse is not necessarily true. We thus use relative error for our evaluation results. To calculate the accuracy of the whole network, the median relative error for all the nodes is usually used. The lower the relative error, the more accurate the network coordinates as the coordinate distance closely matches the real network distance. We define the network as having *converged* when the median relative error for all the nodes remains unchanged, that is, peers' coordinates are relatively unchanged.

We now define the two goals that any network coordinate system should meet in order to be secure.

1. The coordinate distance between two honest nodes should be mostly unaffected whether there are malicious nodes participating in the network.

2. An adversary cannot appear closer than it actually is, that is, its coordinate distance cannot be smaller than its "true" coordinate distance.

The first security goal is to prevent a "cascading" effect, where an attacker can greatly influence one honest node, which in turn, influences another honest node. Two honest nodes should be able to "embed" each other and calculate their real coordinate distance even if malicious nodes are present. If an attacker can affect the coordinate distance between two honest nodes, then a honest node will not be able to accurately calculate the coordinate distance to any other honest node.The second security goal is to ensure that an adversary cannot appear closer to another peer than it actually is. This prevents the attacker from being close to all the peers in the network. In certain applications such as a peer-to-peer system or closest-server selection, the closest peer is usually queried. Since we use the relative error and not the prediction error, a node's coordinate could be closer than the network distance; however, this is within our stated model and definition of relative error.

6.2 KoNKS Design

In this section, we introduce the design for KoNKS. We then argue that it is secure under our threat model and describe the implementation details.

The objective function of current network coordinate schemes is that each node minimizes its overall prediction error to all other nodes. This is required for the network coordinates to be accurate, as a low prediction error implies that the coordinate distance is close to the network distance. In these schemes, every node maintains a list of neighbors – a subset of all the peers in the network. Each node will, regularly, pick a node from that list of neighbors to "contact" so as to update its own coordinate. For each coordinate update, a node will update its coordinate so that it eventually **minimizes the sum of prediction errors for all of its neighbors** as the end goal. Although this objective function allows a node to select a coordinate that is accurate (low relative error), it allows for "outliers" in the list of neighbors. For example, the sum of prediction errors might be small, with most neighbors' prediction error being very small and one neighbor's prediction error being large. In trying to minimize the sum of errors, the large error of the one neighbor can be decreased. However, this might double the small errors of all the other neighbors. Thus, an adversarial neighbor might be able to have a big influence on the updated coordinate of a peer.

To mitigate possible attacks and to reduce the influence of any one neighbor, we modify the objective function so that each node will **minimize the number of neighbors whose individual relative error is greater than the threshold** T. All the neighbors are considered at each coordinate update. However, let's say a coordinate C can be chosen so that most of the neighbors' individual relative error is less than T, except one neighbor N whose error is greater than T. If another coordinate can be computed such that this neighbor's (N) error is less than T, but if this coordinate change would make other neighbors' individual error be greater than T, then the previous coordinate C would be chosen instead. We note that this neighbor is not blacklisted but will be considered again at the next update step as its coordinate and/or network latency might have changed. This objective function then limits the influence each neighbor can have on a peer. More specifically, each neighbor has the same amount of influence as each other.

In addition to modifying the objective function for calculating the optimal coordinate, KoNKS should have the following functional properties:

Distributed implementation with no centralized entities. KoNKS should be completely decentralized with no central or trusted servers. This makes it more

appealing for implementation in many applications – such as peer-to-peer systems for content distribution – that inherently lack a central server.

Security under the threat model of Section 6.1. Security is important so that an attacker cannot disrupt the network coordinate system or modify the network latency estimate to its own advantage.

Accuracy. We will measure the accuracy of KoNKS using the median relative error. Ideally, KoNKS should provide accuracy comparable to other secure network coordinate systems. Since these other schemes are implemented as additional mechanisms on top of Vivaldi, we compare the accuracy of KoNKS to the accuracy of Vivaldi.

Low computational and communication overhead. If KoNKS is to be used in practice, it cannot incur a high overhead. Coordinate updates should be computed quickly and the size of update messages should be small.

Next we describe the KoNKS algorithm and implementation details.

6.2.1 Algorithm

As stated before, the network consists of n nodes and the set of nodes is represented as $\mathcal{P}$. Every node $P_i \in \mathcal{P}$ maintains a list of neighbors $\mathcal{N}_i$. Each node will also store the coordinate-rtt pair for each neighbor $N_j \in \mathcal{N}_i$ (denoted as $\mathsf{Pair}(N_j)$). At a regular interval, P_i will contact (this can be piggybacked on top of application messages, but for simplicity we separate them) a randomly-selected neighbor $N_j \in \mathcal{N}_i$ by invoking SendUpdate. Upon receiving the message, the neighbor will reply back with its coordinate and the rtt between them, by invoking ReplyUpdate. The peer P_i will then update the corresponding Pair with the received coordinate-rtt pair (ReceiveUpdate). The top half of Figure 6.1 shows the communication algorithm for KoNKS.

The bottom half of Figure 6.1 shows the algorithm outlining how a peer will update its coordinate. Update gets invoked after receiving the reply from the "ping" message. The coordinate space is defined as CoordSpace – it represents every possible coordinate, bounded by some very large coordinate (positive and negative). Each peer will choose its coordinate to maximize the number of its neighbors that are "satisfied" – the individual relative error for each neighbor is less than the threshold T. To find the optimal coordinate, the peer will iteratively search the coordinate space. The details of how to perform an efficient search and how to deal with some exceptions are described in

Section 6.2.3.

6.2.2 Why is KoNKS Secure?

We argue that KoNKS is secure – it meets our two security goals from Section 6.1. We assume that it is possible to obtain a T-embedding for all the honest nodes in the network. This means that it is possible to assign coordinates to honest nodes such that the coordinate distance between any pair of honest nodes will be different from the real network distance between that pair of nodes by a factor of at most T. In other words, the error for the network latency estimate will be at most T. In Section 6.3.2, we show that experimentally, $T = 0.25$ is adequate for the Internet. In KoNKS, this means that all honest neighbors can be satisfied (the individual relative error of every honest neighbor will be less than the threshold T). We also assume that there are more honest nodes than malicious nodes in the network and in every peer's neighbor list (n total nodes and m malicious nodes, where $0 \leq m < \frac{n}{4}$). Since neighbors are chosen randomly, every peer has an equal probability of being picked as a neighbor, so an adversary cannot perform a targeted attack whereby it controls the majority of neighbors for a certain peer.

KoNKS satisfies the first security goal due to multiple reasons. First, each peer is influenced by its list of neighbors and if the attacker is not in that list, it cannot influence the peer's coordinate. Second, even if the neighbor list contains some adversarial nodes, the honest neighbors outnumber the malicious neighbors. Thus, the influence that the malicious neighbors can exert on the peer is limited. In the worst case, $\frac{1}{4}$ of a peer's neighbors are malicious. Thus, the peer can satisfy $\frac{3}{4}$ of its neighbors. If all the malicious neighbors are part of the satisfied neighbors, then half of the satisfied neighbors are still honest. Half of the neighbors will have an error less than T, thus the relative error of the peer will be less than T.

KoNKS trivially satisfies our second security goal because if a peer tries to craft its reported coordinate so that it appears closer, the relative error for that neighbor will be higher than the threshold, and the adversarial neighbor will not be considered when searching for the optimal coordinate. We note that a peer can be closer due to the definition of relative error but this is acceptable as the peer is still within the threshold.

For each $P_i \in \mathcal{P}$
Choose $N_j \in \mathcal{N}_i \subseteq \mathcal{P}$
Define $\mathsf{Pair}(N_j) = (\mathsf{Coord}_{N_j}, \mathsf{rtt}(P_i, N_j))$

Procedure SendUpdate(N_j):
Send Coord_{P_i} to N_j

Procedure ReplyUpdate(N_j):
Send Coord_{P_i} to N_j
Send $\mathsf{rtt}(P_i, N_j)$

Procedure ReceiveUpdate($N_j, \mathsf{Coord}_{N_j}, \mathsf{rtt}(P_i, N_j)$):
$\mathsf{Pair}(N_j) = (\mathsf{Coord}_{N_j}, \mathsf{rtt}(P_i, N_j))$
Update()

Procedure Update():
Set $bestCoordinate \leftarrow null$
Set $bestNumSat \leftarrow 0$
for C $\in$ CoordSpace:
  Set $numSat \leftarrow \mathsf{NumSatisfied}(C)$
  if $numSat > bestNumSat$:
    $bestNumSat \leftarrow numSat$
    $bestCoordinate \leftarrow C$
Output: $bestCoordinate$

Procedure NumSatisfied(coordinate):
Set $satisfiedNeighbors \leftarrow 0$
For each $N_j \in \mathcal{N}_i$ do:
  if $\mathsf{relative\_error}(\mathsf{coordinate}, \mathsf{coord}_{N_j}) \leq T$:
    Increment $satisfiedNeighbors$ by 1
Output: $satisfiedNeighbors$

Figure 6.1: KoNKS algorithm

6.2.3 Implementation

To improve readability, the algorithm from Figure 6.1 omits some implementation details, which we now describe.

Initial coordinates: Instead of setting the initial coordinates of each newly bootstrapped peer to be the origin as in Vivaldi, each peer will randomly select its coordinates from a range $[-\delta, \delta]$, where δ is small. This improves the convergence time of the nodes.

Searching algorithm: We do not perform an exhaustive search of the coordinate space to find the optimal coordinate for each peer, as this might take an unreasonable amount of time. Instead, we use a variant of the hill-climbing search algorithm. At each search step, we consider incrementing or decrementing by one ms each of the dimensions of the coordinate to determine the next best coordinate. In a 5-dimension coordinate, there are thus 10 possible choices at each search step. The objective function is to maximize the number of satisfied neighbors (individual relative error less than the threshold T). The optimal coordinate (global maximum) is found when all of the neighbors have been satisfied. It is possible to reach a local maximum in the hill-climbing search algorithm. If a maximum is reached, but the coordinate is not optimal yet, the algorithm will restart with a randomly chosen coordinate.

No perfect coordinate: It is possible that there is no perfect coordinate, or that it will take too many search steps to find the optimal coordinate. In this case, after a certain amount of "restarts" and search steps, the best location that has been found so far is chosen. That location is the one that satisfied the largest number of neighbors.

6.3 Evaluation Results

6.3.1 Setup

We implemented a simulator for our KoNKS algorithm and evaluated its accuracy and resistance to known attacks. For our simulation we used the King dataset [53] for latencies between nodes. Each peer in the network computes its own 5-dimensional coordinate. Every 10 seconds, each node will randomly select another peer in the network to SendUpdate. Each peer maintains 50 neighbors. All attacks start after time 80 ticks, to allow the network to stabilize – that is, reach a stable median relative error.

This translates to about 13 minutes. All of our simulations were run on a quad-core 2.67GHz Intel Xeon W3550 processor.

We also performed our experiments on PlanetLab [46], with the same implementation details as for the simulator. Attacks start after 10 hours. Each experiment consisted of $400 - 550$ nodes.

6.3.2 Simulation

Picking the threshold T

: In the previous section, we mentioned the threshold T that all neighbor's individual relative error must satisfy. Recall that the relative error between two peers A and B is defined as $\frac{|distance(A,B)-rtt(A,B)|}{rtt(A,B)}$. A low relative error means that the system is more accurate. We now experimentally set the value of T and explain why this value produces accurate latency predictions. Figure 6.2 shows the median relative error for all the nodes in the network over time. The different lines indicate the different values of T for KoNKS. The figure shows that decreasing the threshold improves the accuracy of KoNKS – when $T = 0.2$, KoNKS's accuracy is comparable to Vivaldi's accuracy. This is expected as the lower the threshold, the lower the neighbor's relative error, and this implies that the median relative error should be lower than T. We note that the median relative error is **much lower** than T. When $T = 0.4$, the median relative error converges to 0.15. This means that the network latency prediction has an error of 15%, that is, the coordinate distance differs from the real network distance by 15%. The figure also shows that the lower the threshold, the longer KoNKS takes to converge to a stable equilibrium. From these observations, we picked $T = 0.25$ as our threshold – each neighbor's coordinate distance can differ from the real network distance by a factor of 0.25. Intuitively, a 25% error when estimating network latencies on the Internet is acceptable in practice. Moreover, from Figure 6.2, we see that when $T = 0.25$, the error is actually only 12%.

The median relative error does not show the whole picture. Figure 6.3 shows the percentage of neighbors satisfied – their individual relative error is less than T. The higher the threshold, the more likely it is to satisfy neighbors, and the lower the threshold, the harder it is to choose a coordinate that can satisfy all neighbors. At $T = 0.25$, 90% of

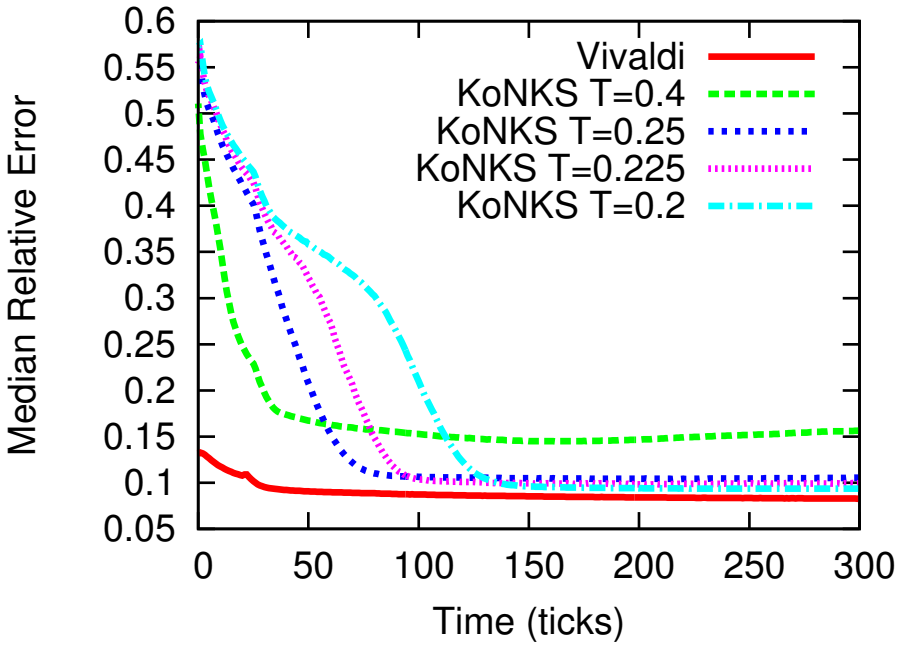

Figure 6.2: The median relative error for Vivaldi and KoNKS with different thresholds

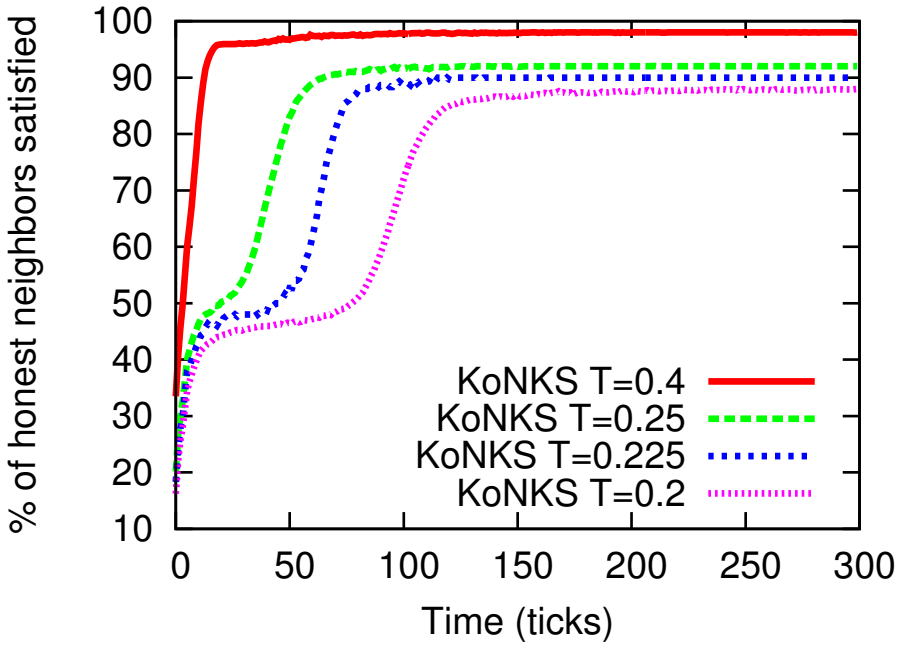

Figure 6.3: The average % of honest neighbors whose individual relative error $<= T$

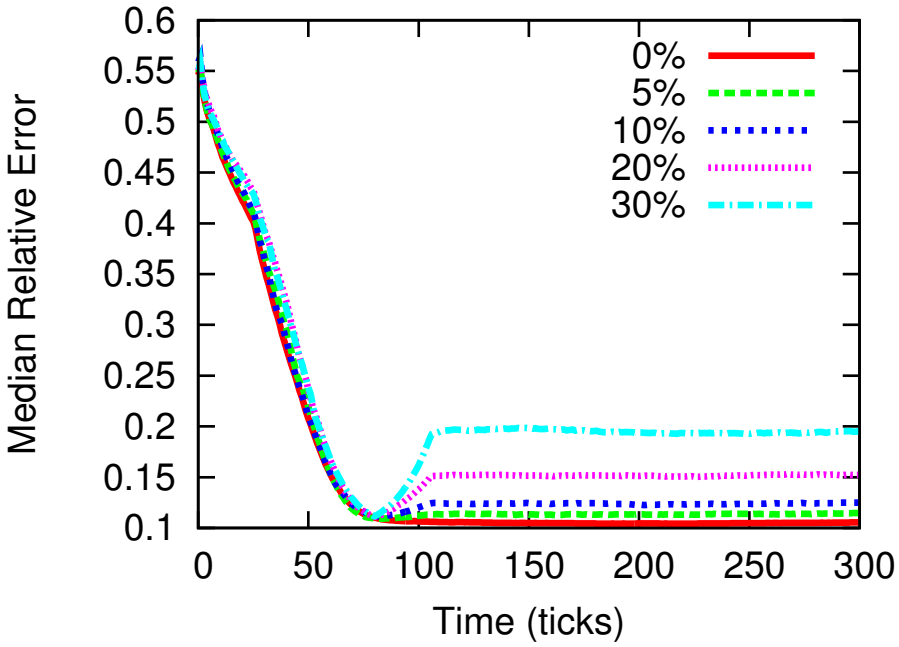

Figure 6.4: The median relative error for KoNKS with $T = 0.25$ under different % of attackers randomly lying about both their rtt and coordinates.

the neighbors can be satisfied. Although 10% of the neighbors cannot be satisfied and are not considered when computing the optimal coordinate, this does not mean that these neighbors are "bad". These same peers might be satisfied neighbors for another node. It is also expected that it is hard to satisfy all the neighbors because we are not doing an exhaustive search of the coordinate space.

6.3.3 Security

To empirically support our security argument, we implemented three of the previously proposed attacks – the "random" attack, the "inflation/deflation" attack, and the frog-boiling attack [40]. Due to space constraints, we only show our results for the "random" and frog-boiling attacks and omit the attacks on Vivaldi since it has already been shown that Vivaldi is insecure. The current "secure" network coordinate systems showed that they can successfully mitigate the random attack. Our variant of the random attack is where an attacker reports randomly chosen coordinates and randomly delay its replies to honest nodes. The malicious nodes will report different coordinates and round trip times each time they are contacted.

Random attacks

Figure 6.4 shows KoNKS' median relative error for the random attack. Recall that the attack starts at time = 80. For 5% and 10% of attackers, the error is mostly unaffected. Even with 30% of attackers (This is higher than the limit of 25% of malicious nodes we assumed before; our analysis is loose and more attackers can be tolerated in practice), the median relative error is increased by a factor of two. A relative error of 20% on the Internet is still accurate and can provide reliable network distance estimation. The threshold used was $T = 0.25$, thus we expected the median relative error to be less than 0.25. The influence that the attacker can have on honest nodes, regardless of the percentage of malicious neighbors, is limited. This leads us to claim that KoNKS can mitigate the random attack as effectively as previous "secure" schemes.

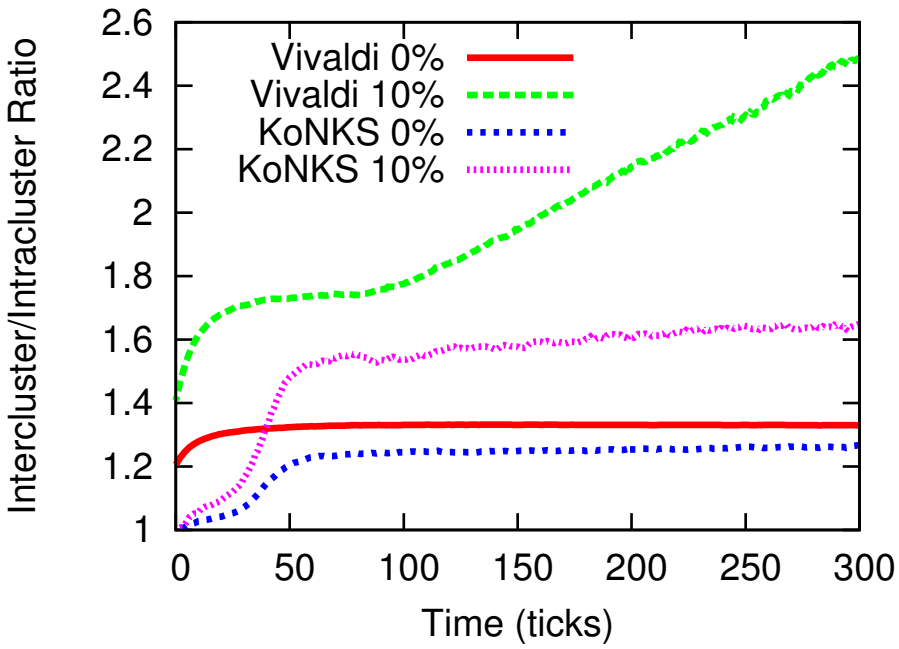

Figure 6.5: The intercluster/intracluster ratio for both Vivaldi and KoNKS with 0% and 10% of frog-boiling attackers.

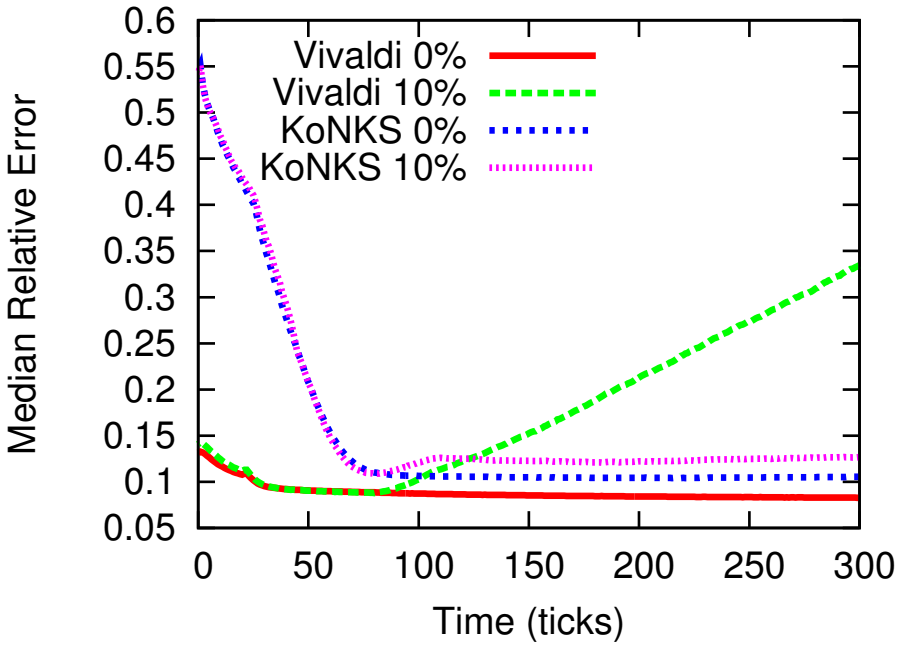

Figure 6.6: The median relative error for both Vivaldi and KoNKS with 0% and 10% of frog-boiling attackers.

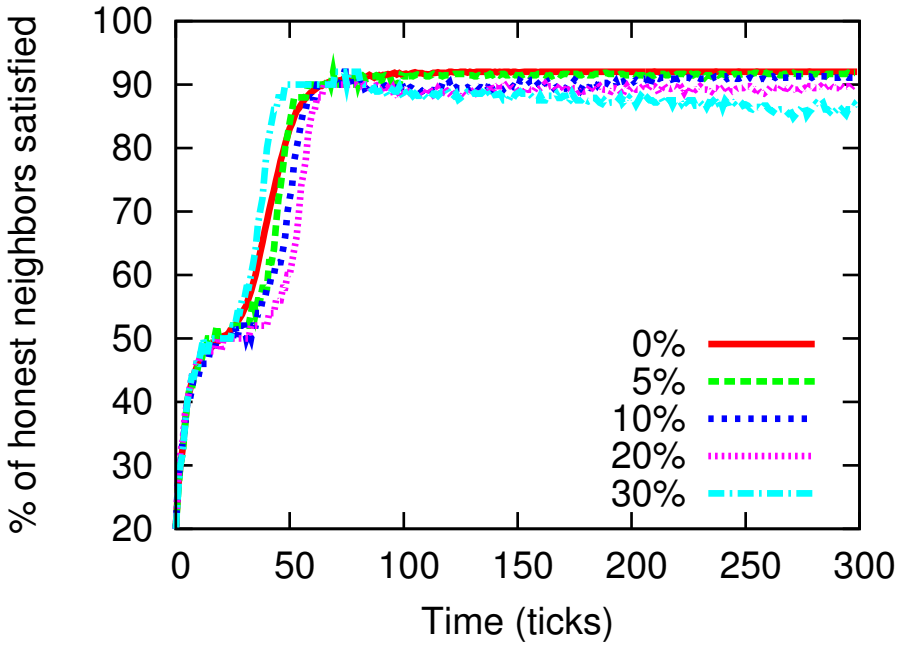

Figure 6.7: The average % of honest neighbors satisfied for honest nodes under attack in KoNKS with $T = 0.25$.

Frog-boiling attack

We also implemented the frog-boiling attack from [40] in an attempt to partition the KoNKS network into two independent subnetworks. The malicious nodes pick the first "half" of the network N_1 to "move" to the coordinate $[-1000, -1000, -1000, -1000, -1000]$ and the second "half" of the network N_2 to move to $[1000, 1000, 1000, 1000, 1000]$. If successful, this attack partitions the whole network into two independent subnetworks N_1 and N_2. The malicious nodes behave normally and compute their best coordinate just like any normal honest peer. However, they lie about their location when they report it to the honest nodes. When a malicious node receives a SendUpdate request from an honest node, it will first determine which "half" that node falls into. If the honest node has not previously contacted the attacker, it will reply with its current best coordinate. If the honest node has previously contacted the attacker, it will reply with the last reported coordinate to that honest node $\pm\delta$ ($-\delta$ if the honest node falls into N_1 and $+\delta$ if the honest node falls into N_2). As [40] showed, each lie is small and not detected by the anomaly detection, but cumulatively, the lies add up such that the network can be effectively partitioned. We set $\delta = 20$ in our experiments. The higher δ is, the faster the attack, but the higher the chance of being detected. We expect that KoNKS will not be affected by the frog-boiling attack.

Figure 6.5 shows the intercluster/intracluster ratio for both KoNKS and Vivaldi with no attackers and 10% of malicious nodes. The intercluster/intracluster ratio indicates how far apart the two networks are. A ratio of two means that the two networks are twice further apart from each other, than they are to their own center. The higher the ratio, the further apart the two networks are. With no attack, both KoNKS and Vivaldi stabilize to a ratio of 1.2. The ratio is not 1 as we always take the same half of the network to be N_1 and the same other half to be N_2. With 10% of attackers, the ratio for Vivaldi keeps increasing over time. Although the intercluster/intracluster ratio for KoNKS increases from 1.2 to 1.6, the ratio remains stable over time. This reinforces our argument that the attacker's influence on honest KoNKS nodes is limited. The malicious neighbors can affect their honest peers only so much before they stop having a malicious effect. Figure 6.6 shows the corresponding median relative error. Vivaldi's median relative error keeps increasing with 10% of malicious nodes, whereas KoNKS' median relative error remains mostly unchanged even under attack. We obtained a

similar result for higher percentages of frog-boiling attackers.

So far, we have shown that KoNKS is secure against all the known attacks. Figure 6.3 showed the percentage of neighbors which can be satisfied. We want to show that even under attack, honest KoNKS peers can still satisfy a high percentage of honest neighbors, while not satisfying the malicious neighbors. Figure 6.7 shows the percentage of honest neighbors satisfied for varying percentages of malicious nodes in the network. We observe that whichever attack is used or the percentage of malicious peers, honest KoNKS peers can still satisfy most honest neighbors while not satisfying malicious neighbors.

6.3.4 Overhead

As mentioned before, the communication overhead is small. Coordinates can be piggy-backed on top of application messages. The processing overhead (the search algorithm to find the optimal coordinate) is also small. Figure 6.8 shows the CDF for the number of search steps performed at each coordinate update to find the best coordinate. The figure shows a median of 697 search steps – this is fairly small as it takes only 697 coordinate changes to find the best coordinate. As Figure 6.9 shows, the median time for a peer to pick its best coordinate at each update is 10ms. The code is not multi-threaded, thus it is expected that the update time can be further reduced. Thus, KoNKS can compute its coordinate efficiently.

6.3.5 Experiments

We performed the same experiments on PlanetLab [46] to confirm our simulation results. The graphs show the average values over a few experiments. Figure 6.11 shows the median relative error for both Vivaldi and KoNKS. Similar to the simulation results, a threshold $T = 0.25$ seems adequate for KoNKS. We note that Vivaldi is a bit less stable than KoNKS – this is attributed to the latencies which vary much more on PlanetLab than in our simulation. Figure 6.12 shows the median relative error for KoNKS with $T = 0.25$ under both the "random" and frog-boiling attacks with different percentages of malicious nodes. We note that in this case $\delta = 50$ for the frog-boiling attack – we found that this value of δ is more effective. The figure shows that KoNKS

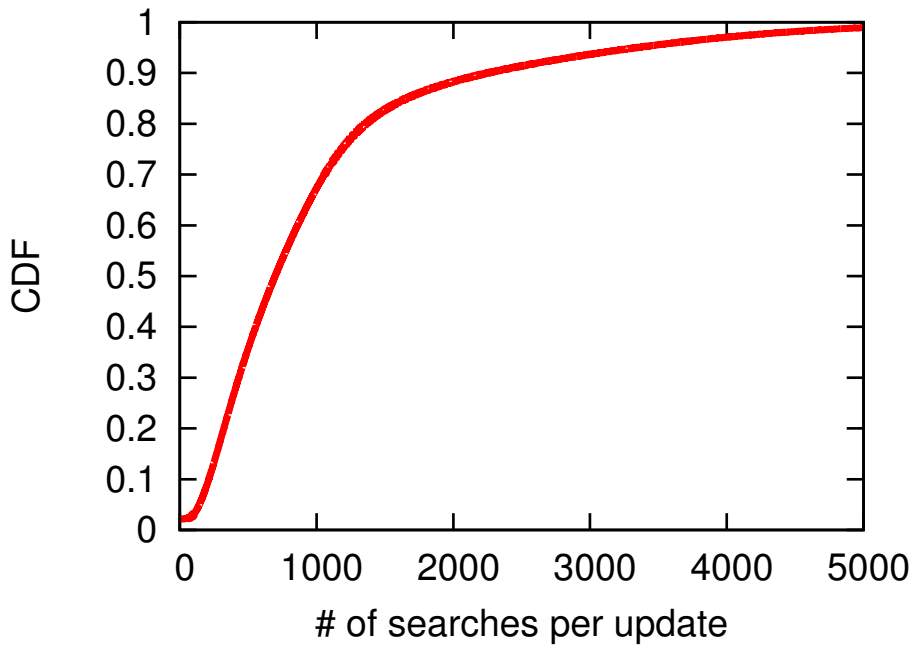

Figure 6.8: The CDF for the number of searches performed by our KoNKS algorithm for each coordinate update.

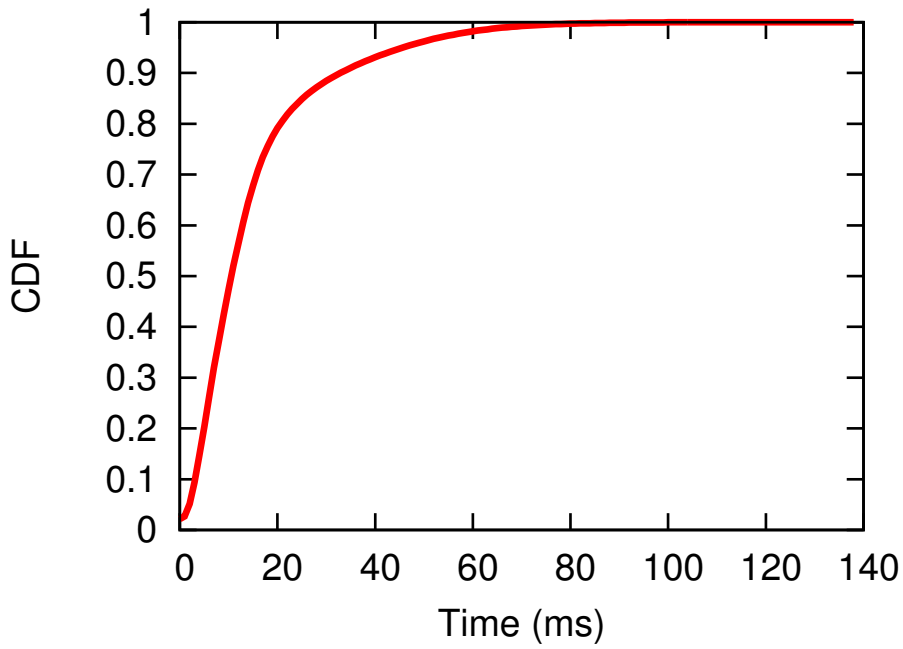

Figure 6.9: The CDF for the time required to complete the search algorithm to find the best coordinate for each update.

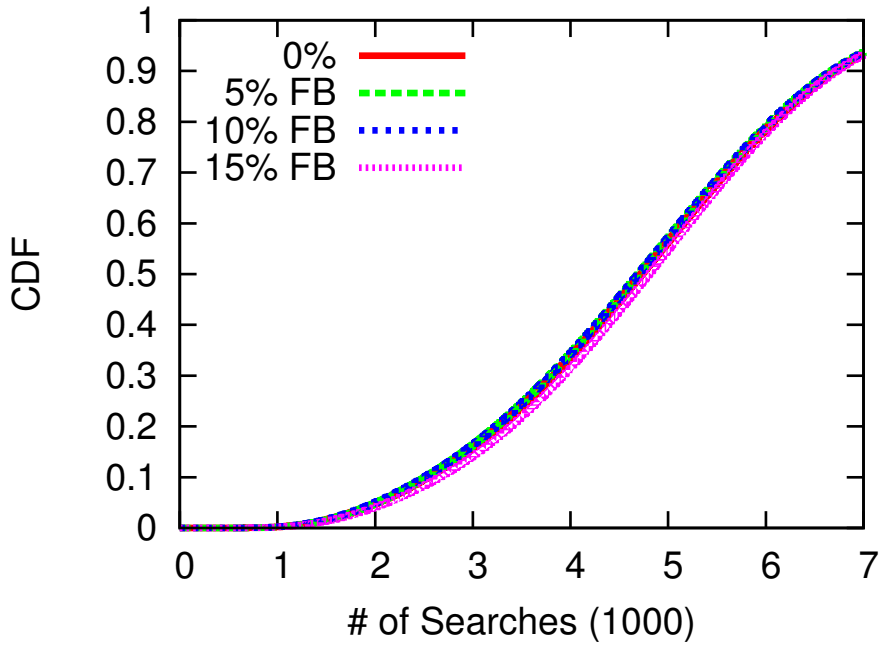

Figure 6.10: The CDF for the number of searches performed by our KoNKS algorithm for each coordinate update on PlanetLab with and without frog-boiling attackers.

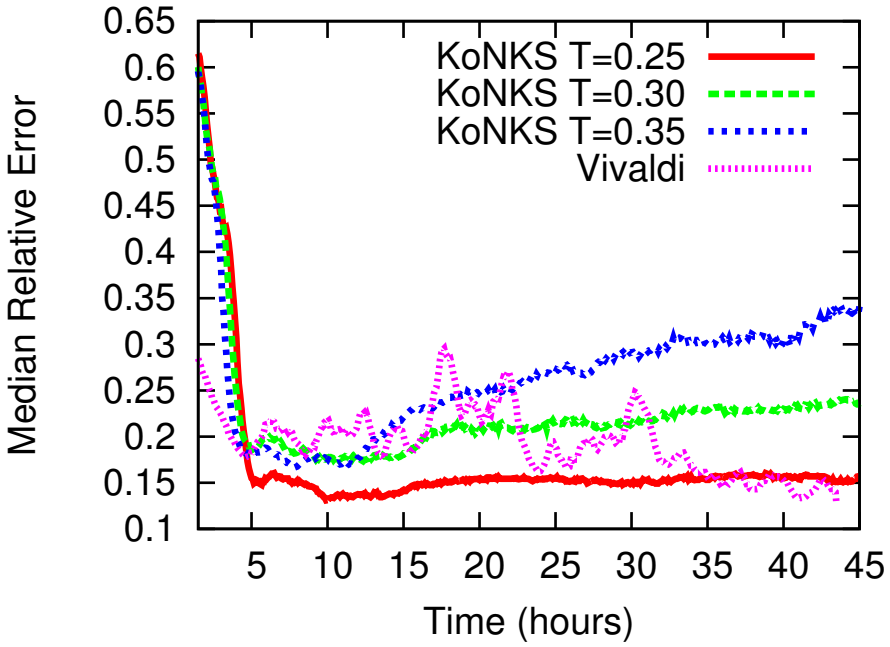

Figure 6.11: The median relative error for Vivaldi and KoNKS on PlanetLab.

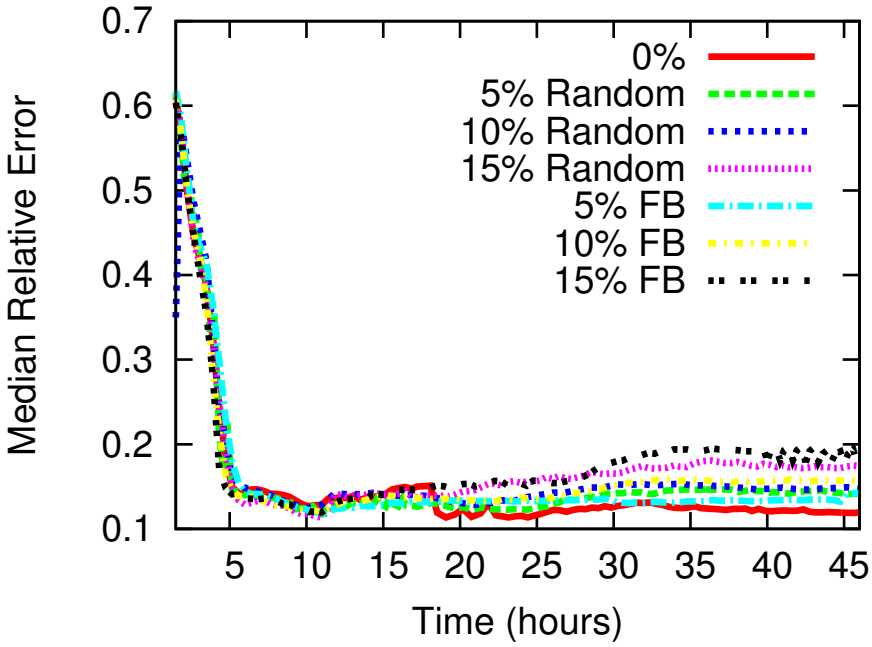

Figure 6.12: The median relative error for KoNKS under attacks on PlanetLab.

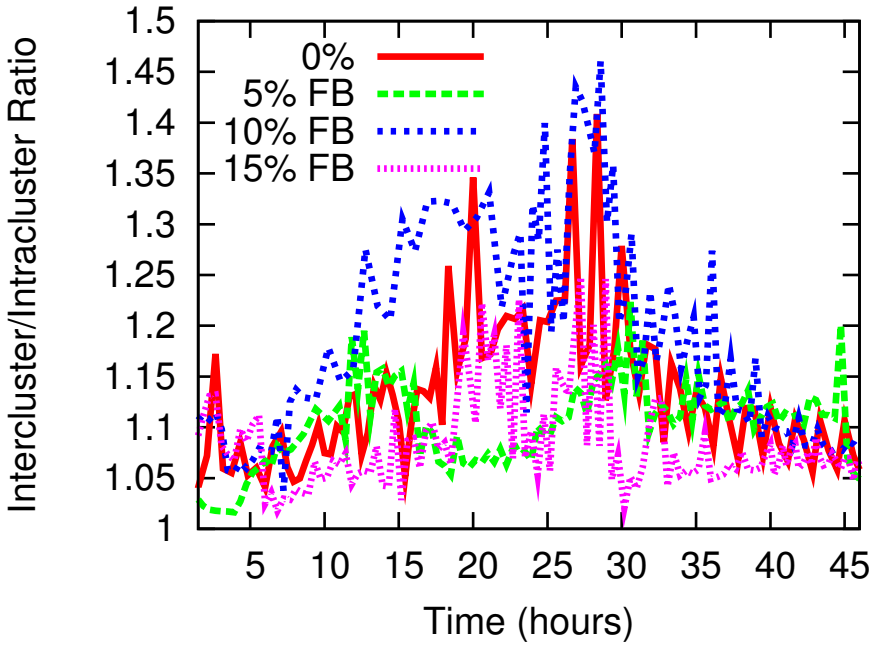

Figure 6.13: The intercluster/intracluster ratio for KoNKS under the frog-boiling attacks on PlanetLab.

is mostly unaffected by the known attacks on network coordinate systems. Figure 6.13 shows the intercluster/intracluster ratio for KoNKS with different number of adversarial frog-boiling nodes. The spikes in the graphs are attributed to network conditions on PlanetLab. In general, it can be seen that the ratio stays stable at around 1.2, even when the frog-boiling attackers are trying to partition the network.

Figure 6.10 shows the computational overhead of running KoNKS on PlanetLab. It shows the CDF of the number of search steps needed for each node to update its coordinate. The number of steps is much higher than for our simulation. We believe this is just an artifact of PlanetLab. We did not show the time taken to perform each search step since PlanetLab would exhibit a much slower time than for a regular machine. However, the time can be extrapolated from Figure 6.9 to be less than 100ms.

As mentioned in Section 6.2.3, choosing the right initial coordinates can improve convergence time. Figure 6.14 shows the median relative error for KoNKS with $T = 0.25$ when using different values for the initial coordinates R. The convergence time can be reduced from 5 hours to 3 hours.

Summary: All the previous experimental results show KoNKS is either unaffected by the attack or that the attacker's influence on KoNKS is limited. This leads us to conclude that KoNKS is secure experimentally under all currently known attacks on network coordinate systems.

6.4 Summary

Although network coordinate systems can accurately predict the network latency between two peers, current systems are not secure in the sense that an adversary can disrupt the whole network by increasing the error, which in turn means that the network latency prediction is no longer accurate. It was previously shown that even the "secure" schemes are vulnerable to the frog-boiling attack. We introduce a new decentralized network coordinate system, KoNKS, and argue that it will be secure not only against known attacks, but also against future attacks that work within our threat model. KoNKS aims to achieve consensus among all the neighbors of a peer, such that, the individual relative error of each neighbor is less than the threshold $T = 0.25$.

We found experimentally that setting the threshold to 0.25 produced a low relative

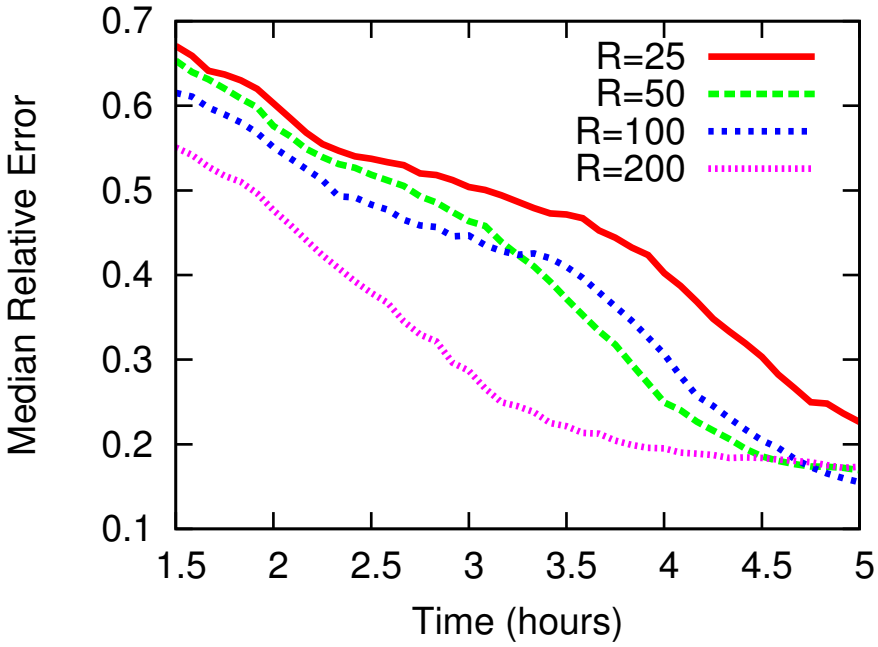

Figure 6.14: The median relative error for KoNKS with varying initial coordinates values

error, comparable to Vivaldi, and allows the network to converge quickly. The median relative error for KoNKS with no attacker is 0.12, compared to 0.10 for Vivaldi. This means that the network distance prediction differs from the real network distance by 12%. Moreover, under the random attack with 30% of malicious nodes, the error is increased to 0.2. KoNKS is also resistant against network partition using the frog-boiling attack. With 10% of malicious nodes, the intercluster/intracluster ratio – a measure of how far away from each other the two networks are – increases from 1.2 to 1.6, but remains **stable**, contrary to the frog-boiling attack on Veracity or Vivaldi, where the two networks keep getting further apart from each other as time goes. We then claim that the influence that an attacker can have on KoNKS is very limited. We also show that each honest KoNKS peer can always satisfy at least 90% of its honest neighbors regardless of the percentage of malicious neighbors and the type of attack.

KoNKS only modifies the objective function of finding the optimal coordinate; it does not perform any verification of coordinates. One of the "secure" schemes previously mentioned such as Veracity [26] can be implemented on top of KoNKS to perform coordinate verification. KoNKS is used to compute the coordinate of peers while Veracity can be used to ensure that neighbors are reporting consistent coordinates. This scheme will prevent attackers from propagating inconsistent coordinates to the overlay, reducing the impact of attacks that rely on being "close" to all nodes in the system.

Chapter 7

Conclusion

Network coordinate systems are very useful for many applications. However, security is a major issue as attackers can easily partition the network, isolate nodes, or disrupt the network coordinate scheme to make it inaccurate. Although a few schemes have been proposed to secure current network coordinate systems, we introduce a new attack, the frog-boiling attack, which bypasses all these schemes and still attack the network coordinate system. We then defined a formal security model and a realistic threat model. Furthermore, we described two secure network coordinate schemes Treeple and KoNKS. Treeple is a centralized system which is provably secure while being very accurate, exhibit low overhead, and computing stable network coordinates which can be used for weeks. KoNKS is completely decentralized and is secure on average. It allows peers to reach a consensus about their network coordinate. KoNKS is also accurate and contains a very small computational overhead. It can be used as the base for other network coordinate schemes. Finally, we showed that applications should not use an insecure network coordinate system. The network coordinate system can be exploited such that the application is indirectly attacked. For example, the Vuze BitTorrent routing layer can be indirectly attacked in an attempt to hijack all the search lookups, by exploiting the insecure network coordinate system used.

As future research directions, a more diverse dataset or widespread deployment is needed to ensure Treeple and KoNKS accuracy and stability. Moreover, both Treeple and KoNKS could be further improved, by mitigating the complications from using traceroutes, or by implementing a better optimization algorithm. Finally, a network

coordinate system can be deployed and used on more latency-sensitive applications.

References

[1] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, New York, NY, USA, 2004. ACM.

[2] T. S. Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *IEEE INFOCOM*, pages 170–179, 2001.

[3] T. S. Eugene Ng and Hui Zhang. A network positioning system for the internet. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, page 11, Berkeley, CA, USA, 2004. USENIX Association.

[4] Manuel Costa, Miguel Castro, Antony Rowstron, and Peter Key. PIC: Practical Internet Coordinates for Distance Estimation. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 178–187, 2004.

[5] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for Scalable Distributed Location. *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[6] Y. Shavitt and T. Tankel. Big-Bang Simulation for embedding network distances in Euclidean space. *IEEE INFOCOM*, 2003.

[7] Vuze. `http://azureus.sourceforge.net`, Accessed 2011.

[8] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a DHT for low latency and high throughput. In *Proceedings of the 1st NSDI (Symposium on Networked Systems Design and Implementation)*, pages 85–98, 2004.

[9] C. Lumezanu, D. Levin, and N. Spring. Peer wise discovery and negotiation of faster path. *In Proceedings of HotNets*, 2007.

[10] David R. Choffnes and Fabián E. Bustamante. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. *SIGCOMM Comput. Commun. Rev.*, 38(4):363–374, 2008.

[11] Ittai Abraham and Dahlia Malkhi. Compact routing on euclidian metrics. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 141–149, New York, NY, USA, 2004. ACM.

[12] R. Gummadi, R. Govindan, N. Kothari, B. Karp, Y. J. Kim, , and S. Shenker. Reduced state routing in the internet. *HotNets*, 2004.

[13] Jonathan Ledlie, Michael Mitzenmacher, and Margo Seltzer. Wired geometric routing. *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[14] John R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[15] Rida A. Bazzi and Goran Konjevod. On the establishment of distinct identities in overlay networks. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 312–320, New York, NY, USA, 2005. ACM.

[16] Tor. `http://www.torproject.org`, Accessed 2011.

[17] Micah Sherr, Matt Blaze, and Boon Thau Loo. Scalable link-based relay selection for anonymous routing. In *PETS '09: Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, pages 73–93, Berlin, Heidelberg, 2009. Springer-Verlag.

[18] James Cowling, Dan Ports, Barbara Liskov, Raluca Ada Popa, and Abhijeet Gaikwad. Census: Location-Aware Membership Management for Large-Scale Distributed Systems. *In the proceedings of USENIX Annual Technical Conference*, 2009.

[19] Sharad Agarwal and Jacob R. Lorch. Matchmaking for online games and other latency-sensitive P2P systems. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 315–326, New York, NY, USA, 2009. ACM.

[20] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network coordinates in the wild. In *Proceeding of USENIX Symposium on Networked Systems Design and Implementation (NSDI)07*, 2007.

[21] Jonathan Ledlie, Peter Pietzuch, and Margo Seltzer. Stable and accurate network coordinates. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 74, Washington, DC, USA, 2006. IEEE Computer Society.

[22] Mohamed Ali Kaafar, Laurent Mathy, Thierry Turletti, and Walid Dabbous. Real attacks on virtual networks: Vivaldi out of tune. In *LSAD '06: Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, pages 139–146, New York, NY, USA, 2006. ACM.

[23] Mohamed Ali Kaafar, Laurent Mathy, Thierry Turletti, and Walid Dabbous. Virtual networks under attack: disrupting internet coordinate systems. In *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, pages 1–12, New York, NY, USA, 2006. ACM.

[24] David John Zage and Cristina Nita-Rotaru. On the accuracy of decentralized virtual coordinate systems in adversarial networks. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 214–224, New York, NY, USA, 2007. ACM.

[25] Mohamed Ali Kaafar, Laurent Mathy, Chadi Barakat, Kave Salamatian, Thierry Turletti, and Walid Dabbous. Securing internet coordinate embedding systems. *SIGCOMM Comput. Commun. Rev.*, 37(4):61–72, 2007.

[26] Micah Sherr, Matt Blaze, and Boon Thau Loo. Veracity: Practical Secure Network Coordinates via Vote-based Agreements. In *USENIX Annual Technical Conference*, 2009.

[27] Damien Saucez, Benoit Donnet, and Olivier Bonaventure. A reputation-based approach for securing vivaldi embedding system. In *EUNICE'07: Proceedings of the 13th open European summer school and IFIP TC6.6 conference on Dependable and adaptable networks and services*, pages 78–85, Berlin, Heidelberg, 2007. Springer-Verlag.

[28] Guohui Wang and T.S. Eugene Ng. Distributed algorithms for stable and secure network coordinates. In *IMC '08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 131–144, New York, NY, USA, 2008. ACM.

[29] BitTorrent. `http://bittorrent.com`, Accessed 2009.

[30] P. Maymounkov and D. Mazíeres. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS*, 2001.

[31] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 255–264, New York, NY, USA, 2002. ACM.

[32] Dorothy E. Denning. An intrusion-detection model. *IEEE Trans. Softw. Eng.*, 13(2):222–232, 1987.

[33] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, New York, NY, USA, 2006. ACM.

[34] Atul Singh, Miguel Castro, Peter Druschel, and Antony Rowstron. Defending against eclipse attacks on overlay networks. In *EW11*, 2004.

[35] E. Sit and R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In *IPTPS*, 2002.

[36] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, 2002.

[37] Peng Wang, James Tyra, Eric Chan-Tin, Tyson Malchow, Denis Foo Kune, Nicholas Hopper, and Yongdae Kim. Attacking the kad network. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, SecureComm '08, pages 23:1–23:10, New York, NY, USA, 2008. ACM.

[38] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2001.

[39] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[40] Eric Chan-Tin, Daniel Feldman, Yongdae Kim, and Nicholas Hopper. The Frog-Boiling Attack: Limitations of Anomaly Detection for Secure Network Coordinates. *SecureComm*, 2009.

[41] Han Zheng, Eng Keong Lua, Marcelo Pias, and Timothy Griffin. Internet Routing Policies and Round-trip Times. *Passive and Active Measurement Workshop*, 2005.

[42] Cristian Lumezanu, Randy Baden, Neil Spring, and Bobby Bhattacharjee. Triangle inequality variations in the internet. In *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 177–183, New York, NY, USA, 2009. ACM.

[43] Paul Francis, Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. IDMaps: a global internet host distance estimation service. *IEEE/ACM Trans. Netw.*, 9(5):525–540, 2001.

[44] Li-wei Lehman and Steven Lerman. Pcoord: Network position estimation using peer-to-peer measurements. In *NCA '04: Proceedings of the Network Computing and Applications, Third IEEE International Symposium*, pages 15–24, Washington, DC, USA, 2004. IEEE Computer Society.

[45] Pyxida. `http://pyxida.sourceforge.net`, Accessed 2009.

[46] PlanetLab. `http://planet-lab.org`, Accessed 2011.

[47] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating system support for planetary-scale network services. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association.

[48] Eng Keong Lua, Timothy Griffin, Marcelo Pias, Han Zheng, and Jon Crowcroft. On the accuracy of embeddings for internet coordinate systems. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, page 11, Berkeley, CA, USA, 2005. USENIX Association.

[49] Peter Pietzuch, Jonathan Ledlie, and Margo Seltzer. Supporting network coordinates on planetlab. In *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*, pages 19–24, Berkeley, CA, USA, 2005. USENIX Association.

[50] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[51] Guohui Wang, Bo Zhang, and T. S. Eugene Ng. Towards network triangle inequality violation aware distributed systems. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 175–188, New York, NY, USA, 2007. ACM.

[52] CommonSense. `http://www.kimvdlinde.com/professional/programming/statistics/commonSe`
Accessed 2008.

[53] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: estimating latency between arbitrary internet end hosts. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 5–18, New York, NY, USA, 2002. ACM.

[54] Bamboo DHT. `http://bamboo-dht.org`, Accessed 2009.

[55] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. Opendht: a public dht service and its uses. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 73–84, New York, NY, USA, 2005. ACM.

[56] Roxana Geambasu, Tadayoshi Kohno, Amit Levy, and Henry M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proc. of the 18th USENIX Security Symposium*, 2009.

[57] Roxana Geambasu, Amit Levy, Tadayoshi Kohno, Arvind Krishnamurthy, and Henry M. Levy. Comet: An active distributed key/value store. In *Proc. of OSDI*, 2010.

[58] Amazon EC2. `http://aws.amazon.com/`, Accessed 2011.

[59] Eric Chan-Tin and Nicholas Hopper. Accurate and Provably Secure Latency Estimation with Treeple. *Network and Distributed System Security (NDSS) Symposium*, 2011.

[60] Sharon Goldberg, Michael Schapira, Peter Hummon, and Jennifer Rexford. How secure are secure interdomain routing protocols? In *SIGCOMM'10: Proceedings of the ACM SIGCOMM 2010 Conference on Data Communication*. ACM, 2010.

[61] GNP Simulator. `http://www.cs.rice.edu/~gw4314/ncs-configurable.tar.gz`, Accessed 2010.

[62] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An Information

Plane for Distributed Services. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[63] Bernard Wong, Aleksandrs Slivkins, and Emin Gün Sirer. Meridian: a lightweight network location service without virtual coordinates. *SIGCOMM Comput. Commun. Rev.*, 35(4):85–96, 2005.

[64] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-Aware Overlay Construction and Server Selection. In *Proceedings of INFOCOM*, 2002.

[65] Venugopalan Ramasubramanian, Dahlia Malkhi, Fabian Kuhn, Mahesh Balakrishnan, Archit Gupta, and Aditya Akella. On the treeness of internet latency and bandwidth. In *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 61–72, New York, NY, USA, 2009. ACM.

[66] Harsha V. Madhyastha, Ethan Katz-Bassett, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane nano: path prediction for peer-to-peer applications. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 137–152, Berkeley, CA, USA, 2009. USENIX Association.